Language Assistance in Reinforcement Learning in Dynamic Environments

Master's Thesis

Sander van Leeuwen

Language Assistance in Reinforcement Learning in Dynamic Environments

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Sander van Leeuwen
born in Tilburg, The Netherlands

TUDelft

Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
www.ewi.tudelft.nl

Language Assistance in Reinforcement Learning in Dynamic Environments

Author: Sander van Leeuwen
Student id: 4717988

Abstract

Language is an intuitive and effective way for humans to communicate. Large Language Models (LLMs) can interpret and respond well to language. However, their use in deep reinforcement learning is limited as they are sample inefficient. State-of-the-art deep reinforcement learning algorithms are more sample efficient but cannot understand language well. This research aims to study whether RL agents can improve learning by utilizing language assistance and how LLMs can help them. A sentence describing the agent's environment is fed into an LLM to create a semantic embedding, which is consumed by a recurrent Soft Actor-Critic (SAC) agent to create an agent that can listen to natural language. This research shows that the best method for the agent to consume the embedding is concatenating it to each observation. Also, LLM-based embeddings lead to faster and more stable learning than non-LLM-based embeddings. The agent is sensitive to noise in the embedding but not to the embedding's dimensionality. The agent can generalize well across sentences that have a similar meaning to sentences seen during training but are formulated differently, but it can not generalize as well across sentences with unknown subjects and needs the subjects of the sentences to be grounded in training. Lastly, this research shows that the proposed architecture supports scaling language assistance to more complex environments.

Thesis Committee:

Chair:	Dr. M. T. J. Spaan, Faculty EEMCS, TU Delft
University supervisor:	Dr. J. W. Böhmer, Faculty EEMCS, TU Delft
Daily supervisor:	J. A. de Vries, M.Sc., Faculty EEMCS, TU Delft
Committee Member:	Dr. J. Yang, Faculty EEMCS, TU Delft

Preface

Before you lies the thesis that concludes my studies in the field of computer science at the Delft University of Technology. In the past months, I had the privilege and honor to do research in its algorithmics research group, which invited me to its team meetings and reading groups, engaging me with inspiring research and people.

As I have been interested in robotics since a young age, I enjoyed working on this thesis. This work aims to lay a foundation for developing more advanced language-assisted reinforcement learning agents. Language assistance in reinforcement learning is not only important for intuitive human-computer interaction, but it also opens up reinforcement learning agents to utilize large language models for more informative decision-making, adding to the evolving landscape of artificial intelligence.

I want to extend my sincere gratitude to Joery de Vries, who was always available for in-depth discussions and from whom I learned a lot about doing reinforcement learning research. Furthermore, I would like to thank dr. Matthijs Spaan and dr. Wendelin Böhmer, for your constructive criticism and excellent guidance. I extend a special thanks to my dad, Rik van Leeuwen, for providing invaluable feedback on the draft versions of this thesis. Lastly, I want to thank my friends and family for their unwavering support throughout this project.

Sander van Leeuwen
Delft, the Netherlands
December 11, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Since the invention of computers and robots, people have sought to use them to make their lives easier. To do so, interaction between these robots and people is necessary. People often use language to communicate, abstract, and transfer knowledge of their environment and decision-making [113]. Therefore, the ability to manipulate language could be an intuitive way for people to interact with robots and computers as it not only lowers the bar for interfacing with them [19, 92] but also provides the possibility of improving user satisfaction [142, 109].

The field that studies how to make computers understand language is called *Natural Language Processing* (NLP). This field has made tremendous progress since the invention of the Transformer [148], which has led to the creation of the *Large Language Model* (LLM) that revolutionized language fields like translation [148], text generation [122], text understanding [34, 47, 125], and speech-to-text conversion [163]. More generally, LLMs gained popularity because it allowed the creation of ChatGPT [1] which took the world by storm in the spring of 2023. It provided a revolutionary way to interact with a computer as anyone could ask any question, for which the system would create a human-like helpful answer.

However, this progress in intuitive interaction between people and computers has not yet fully transferred to the field of *Reinforcement Learning* (RL), which is concerned with learning how agents (for example, robots) should behave in an environment. As agents in some circumstances need to interact with people, it would be helpful to understand language to some degree. An example would be telling a robot that should bring food to customers: *'watch out, a glass fell and there is a puddle on the floor'*. The robot should understand the sentence and act accordingly.

LLMs cannot be directly used to learn an RL problem with language assistance, as they are sample inefficient: they require a lot of data before they become powerful [25, 77], which is often not available in RL settings. RL algorithms are more sample efficient, however, they cannot make sense of language as well as LLMs. They have been combined in earlier research, mostly by using language to instruct the agent on reaching its goal.

[1] `https://openai.com/blog/chatgpt`

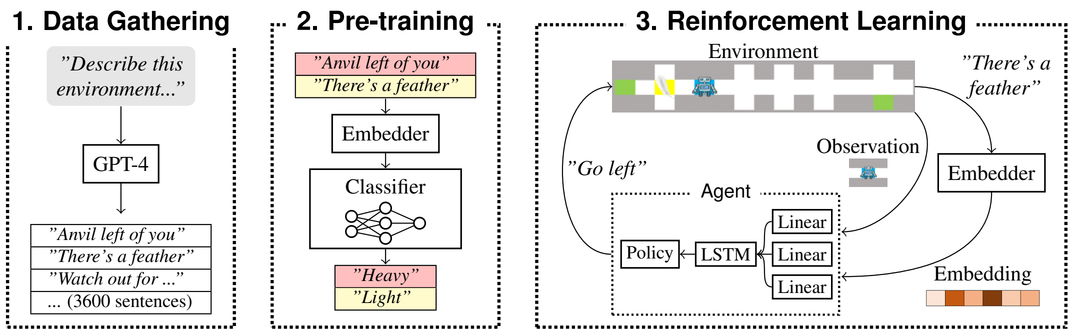


Figure 1.1: High-level systematic view data collection, and the training regime of the agent. The pre-training step trains the classifier and is not used for every experiment.

However, little research has been done on using language to inform the agent about its environment and let the agent figure out the optimal path itself.

This research investigates how an RL agent can best respond to assistance about its environment through natural language. This is done by using LLMs from the NLP domain to interpret the language by summarizing the meaning of a sentence into a vector. This vector is called a semantic embedding. This embedding is used as input to an existing state-of-the-art RL model. An overview of this flow is given in fig. 1.1. This study determines how to enter the embedding into the agent and which existing method can best create these embeddings. It assesses how the agent responds to different amounts of noise and information in the embedding, and lastly, it evaluates to which degree the agent can generalize over unseen objects and sentence formulations.

1.2 Research Goals

This research aims to study whether RL agents can improve learning by utilizing language assistance and how LLMs can help them. Semantic embeddings are used to interface between LLMs and RL agents, which means these embeddings must be entered into (or have to be *consumed* by) the RL agent. Firstly, the research aims to determine the best method for an RL agent to consume an embedding. Next, this research aims to discover how semantic embeddings can best be created and specifically evaluates if LLM-based embeddings lead to better RL performance than state-of-the-art embeddings before LLMs. These two experiments investigate how to architect a language-assisted RL agent.

This agent is assessed on how it responds to changes in the amount of noise and information in the embedding. This aims to provide insight into how embeddings could be improved and how to scale up to more complex environments. It aims to determine how much the embedding properties (such as vector dimensionality, noise, and amount of relevant information) impact the agent's learning ability.

Lastly, a well-performing agent should not only be able to react to sentences it has seen before but also to sentences it was not trained on. The agent is evaluated for generalization across unseen objects and linguistic formulations to find out if the agent just remembers how to respond to specific sentences or whether it interprets sentences in a way that demonstrates an understanding of underlying concepts and can apply that knowledge to novel situations. In the case of generalization across unknown objects, this research is interested in whether the agent can identify a *property* of the object that objects in the training set also have.

This leads to the following research questions.

RQ_1 **How can semantic sentence embeddings best be consumed by the RL agent?**

RQ_2 **Which existing approach can best generate semantic sentence embeddings?**

RQ_3 **How do changes in the amount of information and noise in an embedding affect the agent's performance?**

RQ_4 **How well can the agent generalize over unseen objects with similar properties?**

RQ_5 **How well can the agent generalize over unseen linguistic formulations?**

1.3 Thesis Outline

This thesis will follow the structure of fig. 1.2. Chapter 2 explains the theory behind RL to prepare for the problem formalization and agent architecture, it will explain the state-of-the-art in semantic embeddings, and give an overview of related research.

Next, chapter 3 will introduce the methodology, the problem formalization, data collection, hypotheses, the experiments to construct the agent, as well as experiments to evaluate its performance. Chapter 4 will present the results of the experiments and attempt to confirm or reject the hypotheses, after which chapter 5 discusses the results to answer the research questions. It also gives the implications, limitations, and future directions of this research. Lastly, chapter 6 will conclude the thesis by summarizing the work.

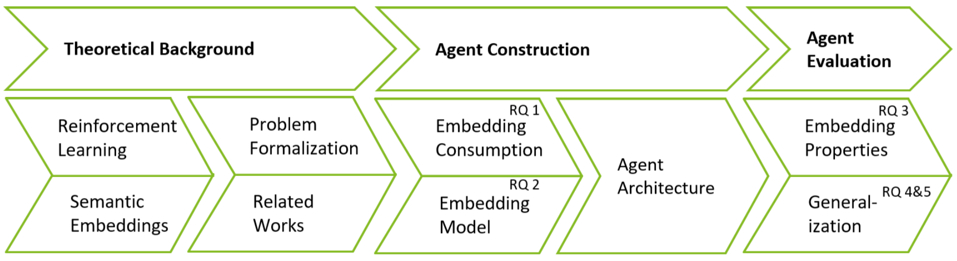


Figure 1.2: Research framework.

Chapter 2

Background and Related Works

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a field that aims to find ways to make agents in some environment learn how to accomplish a goal without explicitly being programmed to do so. Or, more precisely, RL is a field that proposes reasoning about sequential decision-making as an optimization process [139]. In RL, an agent interacts with an environment that might not be entirely familiar to the agent, where the results of its actions might not always have the same outcome (stochasticity), and where the state of the environment might be dynamic and change over time. The RL agent aims to find a *policy*: an action-selection strategy that optimizes some performance metric [50].

A notable difference between RL and supervised machine learning is that with supervised learning, the dataset to train on is independent of the model, while in RL, the dataset is generated by the model interacting with the environment. The RL agent learns from interacting with its environment, so the set of interactions is the dataset it should learn from. This means that with RL, the dataset is not independently and identically distributed (i.i.d.) as is the case with supervised learning but is instead dependent on the RL agent's policy that is being trained.

When an RL agent executes an action in a state, ends up in a (possibly new) state, and receives a reward, it executes a *transition*. RL agents in this research learn in *episodes*, meaning that an agent starts somewhere and transitions until it has reached its goal or a maximum time has passed. The environment is reset now, the agent saves the episode in its memory, and the RL agent is moved to its starting position and can try again.

This section will elaborate on the RL theory, notation, and algorithms used in this research.

2.1.1 Markov Decision Processes

The environment and goals of an RL agent can be modeled as a Markov Decision Process (MDP), which consists of the tuple $\langle \mathcal{S}, \mathcal{A}, T, s_0, R, \gamma \rangle$ [12, 120]. $\mathcal{S}$ is the set of all states s an agent can be in, $\mathcal{A}$ is the set of available actions a to the agent, $T(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the transition or dynamics function: a probability density function indicating the probability of ending up in state s' when executing action a in state s. This research will use the concepts of dynamics and transition function interchangeably.

Next, $s_0 \sim P(\mathcal{S})$ indicates the state the agent will start in. R is the reward function $R(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and provides the agent with feedback on how well it is doing: if it is doing well it gets positive feedback in the form of a positive reward, and vice versa. Lastly, $\gamma \in [0,1)$ is the discount factor and impacts how much the agent favors rewards that are far in the future.

The agent will then aim to find a behavior that executes the right actions to collect as much reward as possible, also known as a policy $\pi(a|s) = P(\mathcal{A} = a|\mathcal{S} = s)$ that maximizes the expected cumulative discounted reward $J^{\pi} = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}]$, or expected return. This policy has the *Markov* property as it decides which action to pick only on the most recently observed state. When the agent is in a state s, the expected return that is achieved by following a policy is called the *value function*: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^k r_{k+1}|s_0 = s]$. Lastly, when the agent is in a state s, the expected return of following policy π after executing action a is called the *Q-function*: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^k r_{k+1}|s_0 = s, a_0 = a]$. If the policy is optimal, meaning there is no other policy that receives a higher return, it is called π^*, and the functions are called J^*, $V^*(s)$, and $Q^*(s,a)$, respectively.

When all parameters of the MDP are known, the problem of finding the optimal policy that maximizes the cumulative future reward can be considered as *planning* [57, 120, 13, 50]. Conversely, when the MDP parameters are unknown, finding the optimal policy can be done with RL [57, 120, 13].

The performance of an RL agent can be measured in different ways. For example, how quickly it learns, how well it remembers (stable learning), or how well it approaches the optimal policy. The latter is used in this research as it is mainly focused on investigating how well an agent can respond to textual information about its environment, not how fast it learns. How close a policy approaches the optimal policy is called *regret* [50, 84], and can be defined as the difference between the expected value of the optimal policy versus the expected value of the current policy as shown by eq. (2.1) [84]. In this equation, μ is the MDP, π is the current policy, a^* is the action resulting from the optimal policy, and a is the action from the current policy. Lastly, $U(\mu,\pi) = \mathbb{E}_{\mu}^{\pi}[\sum_{t=1}^{T} r_t]$ is the expected utility, where utility is the sum of the rewards in an episode.

$$\begin{aligned} Regret(T) &= U^*(\mu) - U(\mu,\pi) \\ &= \mathbb{E}_{\mu}^{\pi^*}\left[\sum_{t=1}^{T^*} r(s_t, a^*, s_{t+1})\right] - \mathbb{E}_{\mu}^{\pi}\left[\sum_{t=1}^{T} r(s_t, a, s_{t+1})\right] \\ a^* &= \arg\max_{a' \in \mathcal{A}} \pi^*(a'|s_t) \\ a &= \arg\max_{a' \in \mathcal{A}} \pi(a'|s_t) \end{aligned} \tag{2.1}$$

As further explained in chapter 3, the research will focus on minimizing the *expected regret*: $\mathbb{E}[Regret(T)]$, where the regret is defined as the difference in the number of steps to reach the goal, between the policy and optimal policy. As eq. (2.1) shows, the policy is evaluated with greedy action sampling, where the action with the highest probability is sampled. This research also evaluates models based on the *cumulative regret*, defined as $R_c = \sum_{n=0}^{N} U_n^*(\mu) - U_n(\mu,\pi)$, where N is the number of episodes.

Agent Algorithms

To solve an MDP and learn how to behave in an environment, there are two main branches of algorithms: *model-based* and *model-free* [161, 50]. A model is a collection of acquired knowledge about the behavior of the environment. Training this model means learning where the agent ends up after taking an action, in other words, learning the MDP transition function $T(s,a,s')$. Model-free algorithms have no such model and instead try to find the optimal policy directly [161, 72], and can be more memory efficient and agile to changes [23]. This research used the Soft Actor-Critic (SAC) model-free algorithm [56]. More information supporting this choice will be given in section 3.4. This section provides the background to SAC by explaining the two categories of model-free algorithms: value- and policy-based[1] [50, 161, 72].

Value-based Value-based methods try to find the value function, either $V^\pi(s)$ or $Q^\pi(s,a)$. These functions can then be used to choose the action that leads to the highest value. In other words, whether it is a good idea to take action a in state, s is measured by the expected return of taking action a in state s. A value function is learned by optimizing the Bellman equation derived in eq. (2.2) [12].

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t r_t \mid s_0 = s\right] = \mathbb{E}_\pi\left[r_0 + \sum_{t=1}^{T}\gamma^t r_t \,\Big|_{a_0=a}^{s_0=s}\right] \\
&= r(s,a) + \gamma\mathbb{E}_\pi\left[\sum_{t=0}^{T-1}\gamma^t r_t \mid s_0 \sim P(\cdot|s,a)\right] \\
&= r(s,a) + \gamma\int_S T(s,a,s')\int_{\mathcal{A}}\pi(a'|s')Q^\pi(s',a')\,da'\,ds'
\end{aligned}
\tag{2.2}
$$

Equation (2.2) shows that the Q-value of one state depends on the states the agents visit after it. This means that a value function can be learned by computing the difference with the Q-values of successive states and actions. In deep RL, a value function is created using a neural network with the parameters θ. An optimal θ can be found by minimizing the loss function of eq. (2.3). Mnih et al. [105] created Deep Q-Networks (DQN), an algorithm that uses a neural network to find $Q(s,a)$, trained by sampling from a replay buffer. Extensions to this are Double DQN [146], Dueling DQN [151], Distributed DQN [10], Noisy DQN [44], and Rainbow DQN [63].

$$
\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}\left[(r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1},a_{t+1}) - Q(s,a)\right]
\tag{2.3}
$$

[1]Credit is due for most of the equations to the Deep RL course at the TUDelft by dr. Wendelin Böhmer

Policy-based Policy-based algorithms directly learn the policy $\pi(a|s)$. The goal is to increase the probability of taking good actions in a state and decrease the probability of taking bad ones. With θ being the parameters of π, optimal values for θ can be found by solving eq. (2.4) [141]. In this case, $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$ is the discounted return starting from time t.

$$\begin{aligned} \max_\theta J[\pi_\theta] &= \max_\theta \mathbb{E}\left[R_0 \middle| \begin{matrix} s_0 \sim P(\cdot),\, s_{t+1} \sim P(\cdot|s_t, a_t) \\ a_t \sim \pi_\theta(\cdot|s_t),\, r_t = r(s_t, a_t) \end{matrix} \right] \\ \nabla_\theta J[\pi_\theta] &= \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t \nabla_\theta \ln \pi_\theta(a_t|s_t) \middle| \begin{matrix} s_0 \sim P(\cdot),\, s_{t+1} \sim P(\cdot|s_t, a_t) \\ a_t \sim \pi_\theta(\cdot|s_t),\, r_t = r(s_t, a_t) \end{matrix} \right] \end{aligned} \tag{2.4}$$

The REINFORCE algorithm approaches the optimization of eq. (2.4) by optimizing for eq. (2.5) by gathering m episodes of length n under policy π_θ [155].

$$\nabla_\theta \mathcal{L}_\pi(\theta) \approx \nabla_\theta - \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{n-1} \gamma^t R_t^i \ln \pi_\theta(a_t^i | s_t^i) \tag{2.5}$$

Policy-based algorithms become more powerful when the difference between the policy before the update (called μ) and after the update (called π_θ) is kept within a bound. TRPO does so by adding a hard constraint using the Kullback-Leibler divergence:
$\mathbb{E}\left[\sum_{t=0}^{n-1} D_{KL}\left[\mu(\cdot|s_t) \| \pi_\theta(\cdot|s_t)\right]\right] \leq \delta$ [131], while PPO achieves this by clipping the gradients ∇_θ [132].

Actor-Critic A merger between value-based and policy-based algorithms can be found in *actor-critic* algorithms [161]. They originate from the issue in policy-gradient algorithms where R_t^i of eq. (2.5) can have high variance, which is bad for training. This variance can be lowered by subtracting a value function: $R_t^i - v_\phi(s_t^i)$.

Actor-critic algorithms consist of two parts: the actor, which is policy-based, and the critic, which is value-based. The actor takes the actual actions, while the critic models a value function for the actor to optimize its policy by. Notable examples are AC [140], A3C [106], DDPG [89], TD3 [46], and Soft Actor-Critic (SAC) [56]. The latter algorithms employ the improvements from the policy-based and value-based algorithms for their actor- and critic architecture and loss functions, respectively.

Soft Actor-Critic This research uses SAC, an off-policy algorithm that optimizes for the expected reward and for taking random actions. In other words, it aims to succeed at its task while acting as randomly as possible [56]. It does so by training to increase the *entropy* of its policy using a loss regularization component, as shown in eq. (2.6) [55].

$$\begin{aligned} \bar{\mathcal{L}}_{SAC}(\theta) &= \mathcal{L}[\theta] - \alpha \mathcal{H}[\pi_\theta] \\ \mathcal{H}[\pi_\theta] &= -\frac{1}{n} \sum_{t=1}^{n-1} \int_{\mathcal{A}} \pi_\theta(a|s_t) \ln \pi_\theta(a|s_t) da \end{aligned} \tag{2.6}$$

SAC extends upon DDPG, which is sensitive to hyperparameter tuning [38, 61, 56]. SAC solves this by making it possible to tune parameter α automatically during training. It employs both value functions $Q_\phi(s,a)$ and $V_\psi(v)$, and uses two Q-functions as proposed by Double-DQN [146] but in a pessimistic way: $\bar{Q}(s,a) = \min_{i\in\{1,2\}} Q_{\phi_i}(s,a)$, which is also done by TD3 [46]. Furthermore, a stochastic policy is created using the reparameterization trick, and for continuous action spaces, actions can be squashed using the hyperbolic tangent. As this research has discrete actions, one action logit is generated for each action (without hyperbolic tangent). In stochastic action sampling, an action is sampled from the Boltzmann distribution [116] of eq. (2.7) based on the action logits [111]. In eq. (2.7), l_a is the logit value of selecting action a. The action with the highest action logit is executed in deterministic action sampling.

$$\pi(a_t^i|s_t) = \frac{e^{l_a}}{\sum_{l_{a'}} e^{l_{a'}}} \tag{2.7}$$

2.1.2 Partially Observable MDP

Initially, the agent of this research should not be able to see the object described by the language assistance. This is because this study is interested in whether the agent can learn about the object based on language assistance. As the agent cannot see the object initially, the agent should be limited to seeing only some parts of the state. Agents not viewing their full environment is more common, for example, in a robot's camera only filming its front [137], an advertising company not being aware of a user's location, or in predictive maintenance where the exact state of the object under maintenance is unknown [137, 41, 134].

Such environments can be modeled with a Partially Observable Markov Decision Process (POMDP), an extension of an MDP where the agent receives observations with partial information about the current state of the environment [3, 134, 79, 50]. A POMDP consists of the tuple $\langle \mathcal{S}, \mathcal{A}, \Omega, T, O, s_0, R, \gamma \rangle$ [137]. In this case Ω is the set of observations available to the agent, $O(s_t, a_t, o_{t+1}) : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow [0,1]$ is a probability density function indicating the probability of receiving observation $o_{t+1} \in \Omega$ when executing action a_t in state s_t. The other variables in the POMDP tuple equal the MDP definitions introduced in section 2.1.1.

In such a setting, the Markov property does not necessarily hold anymore, which states that all information required to select an action can be found in the current observation. As the agent does not directly observe the state, the agent must interpret the history of observations and actions and infer which states it might be in. This is called a *belief*. A *belief MDP* is a representation of a POMDP, where the belief is treated as part of the state space. It is a simplification over the POMDP, as now actions are selected based on a belief of where the agent is instead of a distribution of states the agent might be in. While this requires methods to approximate a belief state based on observations and actions, it also allows the use of standard MDP tools to solve the MDP resulting from inserting the belief in the state space [50, 22].

Bayes Adaptive MDP

In this research, the hidden state[2] is held constant throughout an episode. This allows the environment to be formulated as a Bayes Adaptive Markov Decision Process (BA-MDP) [164, 39, 79]. In BA-MDPs, the hidden state is usually the transition or reward function that is active in the episode. Multiple transition functions or reward functions may exist, but which one is active is not observed. Examples of BA-MDPs are navigation in a static world (e.g., a maze), the game of patience, and medical diagnosis [3].

A BA-MDP is an extension of the MDP introduced in section 2.1.1, and modeled as the tuple $\langle \mathcal{S}', \mathcal{A}, T', s'_0, R', \gamma \rangle$ [50]. In this case, $\mathcal{S}' = \mathcal{S} \times \Phi$ is a new variable indicating the set of *hyper-states*. Each hyper-state consists of the earlier introduced state $s \in \mathcal{S}$ and a new value $\phi \in \Phi$ called the posterior of the transition function: the likelihood that some specific transition function is active in this episode. The MDP transition function $T'(s'_t, a, s'_{t+1})$: $\mathcal{S} \times \Phi \times \mathcal{A} \times \mathcal{S} \times \Phi \to [0,1]$ is a probability density function indicating the probability of resulting in hyper-state s'_{t+1} when executing action a_t in hyper-state s'_t with posterior $\phi \in s'_t$. Next, s'_0 is the starting hyper-state, and $R'(s', a) : \mathcal{S} \times \Phi \times \mathcal{A} \to \mathbb{R}$ represents the reward obtained when taking an action in a hyper-state.

Within reinforcement learning, a vital issue of any agent is the tradeoff between *exploration* and *exploitation* [139, 164]. Exploration involves discovering new ways to reach the goal, whereas exploitation aims to collect as much reward as possible. In BA-MDPs, the agent must explore to find out what the current transition and reward functions are. When it has done so, it can start exploitation to optimize its policy for receiving rewards. In Bayesian RL, policies balance selecting their actions to reduce the uncertainty of the expected reward from executing an action in a state. They choose their actions based on the state-action combination's expected reward (value). A *Bayes-optimal policy* balances exploration and exploitation optimally [50, 164, 11], and optimizes the value function shown in eq. (2.8). Any Bayesian-optimal policy is, therefore, mathematically lower than the optimal policy as defined in section 2.1.1, as the Bayesian-optimal policy needs to explore, which the optimal policy does not.

$$V_t^*(s'_t) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \int_{\mathcal{S} \times \Phi} T(s'_t, a, s'_{t+1}) V_{t-1}^*(s', \phi')\, ds'_{t+1} \right] \tag{2.8}$$

This new definition of an optimal policy introduces a new kind of regret: the *Bayesian-optimal regret*. This type of regret measures the difference between the return of a Bayesian-optimal policy and the current policy. It still uses the definition of eq. (2.1), but now a^* is the action that optimizes eq. (2.8). Bayes Optimal Regret tells us how much performance is lost when the agent starts with partial information about its environment [84] instead of complete information.

[2]part of the state that is not observed

[3]Given that the patient's wellness does not change during diagnosis.

Contextual MDP

This research does allow the agent to make decisions based on a sentence describing the dynamics in its environment. This means the environment can be specified using a Contextual Markov Decision Process (CMDP), which differs from a BA-MDP in that a *context* indicates which transition and reward functions are active. For this research, the assisting sentence contains only partial information, which means it serves as a partial proxy for the context.

A CMDP consists of a tuple $\langle C, S, A, \mathcal{M}(c)\rangle$, where C is the set of possible contexts, S and A are the sets of possible states and actions as in section 2.1.1. Lastly, M is a function mapping a context $c \in C$ to an MDP $\mathcal{M}(c) = \langle S, A, T^c, R^c, \gamma\rangle$ [57]. With the context providing information about the hidden state, there is less need for a well-trained policy to explore. This means that the performance of a policy in a CMDP can be higher than the Bayesian-optimal policy in a BA-MDP, and it can even approach the performance of the optimal policy.

Agent Algorithms

It is hard to solve a POMDP with an MDP algorithm such as DQN, PPO, or SAC directly, as the history cannot directly be fed into the algorithms because of the curse of dimensionality[115, 90]: possibly exponentially more data is needed when the input dimensionality is increased [12, 4]. When approaching solving POMDPs with FFNs, the input dimensionality increases linearly with the horizon length [111] [4], which means that to act on a longer horizon, exponentially more data is needed.

A solution is to use memory-based policies, which keep a belief in their memory and update it based on new observations. Memory-based policies are often implemented with *Recurrent Neural Networks* (RNN) [130, 8, 153, 111], where *Recurrent* implicates the memory. The output of this RNN is then fed into an MDP algorithm as if the output of the RNN represents a state. This change is both simple, as only a few lines of code have to be changed [111], and general, as RNNs are universal function approximators [129] and Turing-complete [133, 111].

Notable algorithms are DRQN [60], Recurrent A2C [102], DVRL [75], R2D2 [81], and LSTM-TD3 [99], and Agent57 [5]. [111] argue that recurrent methods achieve state-of-the art results on POMDP environments.

Long Short-Term Memory

A Long-Short-Term Memory (LSTM) network is a type of RNN used for efficiently remembering long- and short-term dependencies in sequential input [68]. RNNs are neural networks that work similarly to feed-forward networks, except that they save some internal state (also, *hidden* state, called *memory* in the previous section) to be used by the network's subsequent invocation. This means an RNN has two outputs: y and the hidden state h_t, and two inputs, x and the hidden state h_{t-1} of the previous invocation.

[4]In episodic RL, the horizon length is $\min(\frac{1}{1-\gamma}, T)$ where T is the maximum episode length.

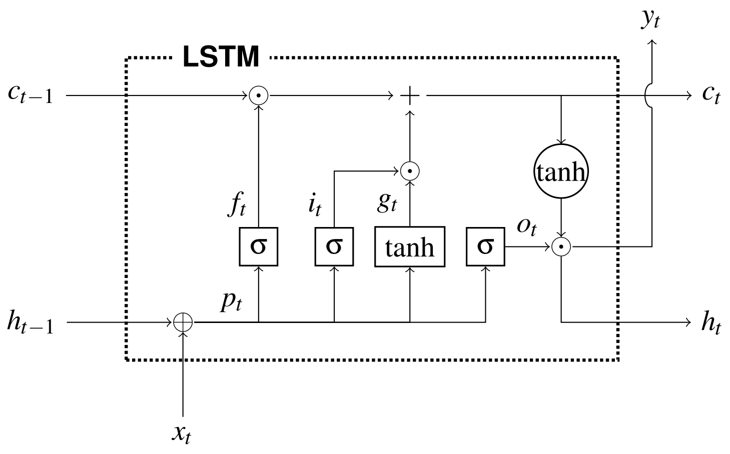


Figure 2.1: LSTM architecture used in this research. The square nodes include matrix multiplications and subsequent activation layers. The other nodes indicate element-wise operations. Note: $\oplus$ indicates vector concatenation, $+$ denotes vector addition, and diverging lines indicate a vector copy.

The LSTM was created to solve the vanishing-gradients problem, which poses the challenge that if backpropagation is applied to an RNN through time, the gradients approach zero or explode to infinity. The LSTM is one of the most commonly used RNNs [87]. It is used in this research because Ni et al. [111] evaluated it to work best for a recurrent RL agent in a POMDP meta-learning setting.

A visual representation is made in fig. 2.1, and the equations are shown in eq. (2.9). Equation (2.9) is adapted from the documentation of the PyTorch LSTM implementation used in this research [45]. In the equation, $x_t \in \mathbb{R}^m$ is the input of timestep t, $h_{t-1} \in \mathbb{R}^n$ is the output from the LSTM at timestep $t-1$, $W \in \mathbb{R}^{n\times(m+n)}$ are matrices of trainable weights, $b \in \mathbb{R}^n$ are bias vectors with trainable weights, $\odot$ is the Hadamard product [45, 128], and $\oplus$ denotes vector concatenation. As there are four matrices W and four bias vectors b, the total number of parameters in the LSTM used in this research is $4(mn+n^2+n)$.

$$\begin{aligned}
p_t &= x_t \oplus h_{t-1} \\
i_t &= \sigma(W_i p_t + b_i) \\
f_t &= \sigma(W_f p_t + b_f) \\
g_t &= \tanh(W_g p_t + b_g) \\
o_t &= \sigma(W_o p_t + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= y_t = o_t \odot \tanh(c_t)
\end{aligned} \tag{2.9}$$

The intuition behind the LSTM is that the cell state c_t contains information that can be retained over a long period of time. Every iteration of the LSTM, f_t provides a value between 0 and 1 for every dimension, indicating how much of the information from c_{t-1} should be retained. Next, g_t determines which information from the input x_t and the previous output h_{t-1} should be remembered in c_t for the next cycle, and i_t indicates how much of that information. Lastly, o_t specifies what information from the input and the new cell state is relevant for the output. In deep RL, the initial vectors c_0 and h_0 are often set to zero [60, 111, 99, 5], also known as a blank vector or blank initialization.

2.2 Semantic Sentence Embeddings

To make the agent capable of responding to text, the text needs to be transformed into a form the agent can understand well. The neural networks of an RL agent have a fixed input length, so the text should be translated into a *semantic embedding*: a vector that represents the meaning of the text. For an embedding to be valid, it should have the properties listed below.

1. Embeddings should have equal dimensionality.
2. Texts with similar semantics should get similar embeddings. Even texts that differ substantially in the selection of words and characters but mean the same thing (e.g., *'a friend is close by'*, and *'your buddy is near'*).
3. Texts with dissimilar semantics should get distinguishable embeddings. This is especially important for lexically similar texts (e.g., *'she left a note on the stationery'* and *'she left a vote on the stationary'*[5]).
4. Word order of sentences must be taken into account. For example, *Alice is to the right of Bob* should not generate the same embedding as *Bob is to the right of Alice*.
5. Idiomatic phrases such as *call it a day*, *break a leg* and *through thick and thin* should be interpreted in the right context.

2.2.1 Sentence Embedding Architectures

Incitti et al. [76] categorize existing sentence embedding architectures into six groups, of which five are relevant. It also categorizes a sixth group of *multi-modal* text embeddings that also takes non-text inputs, which is outside this research's scope. This research is mainly interested in how LLMs can improve how agents respond to language assistance, so the current state-of-the-art embeddings will be compared to the state-of-the-art techniques before the invention of LLMs.

Transformers The architecture of LLMs is derived from Transformers, which are encoder-decoder models that use *self-attention* to find both short- and long-range connections between input dimensions [148]. The encoder compresses the sentence into a latent vector, and the decoder generates text from this vector. The Transformer can take context into account as it sees all words at the same time and uses *self-attention* to pay attention to specific words and the relationships between them [6, 157]. A notable decoder model is

[5]*Stationery* refers to paper or envelopes, *stationary* refers to something that is not moving or in place.

GPT [122, 121, 18, 112], and an encoder model is BERT [34]. BERT can generate state-of-the-art word embeddings [58], and is trained unsupervisedly through *Masked Language Modeling* (MLM): predicting words based on the words around it, and *Next Sentence Prediction* (NSP): predicting whether two sentences are ensuing in a text. Often, transformer encoders are pre-trained in a self-supervised manner using MLM and NSP and finetuned using a smaller supervised dataset to become more domain-specific. The embedding is created from the values of the last layer of the neural network when provided with a sentence.

A derivative of BERT for sentence embeddings is Sentence-BERT (SBERT) [125], which is trained in a supervised way to predict sentiment. Another is SimCSE, which uses BERT with contrastive learning in an unsupervised way to create sentence embeddings [47]. Both SBERT and SimCSE have been trained on RoBERTa, a version of BERT trained with more data and compute resources [93]. Transformer-based embeddings adhere to all five properties mentioned above and are considered state-of-the-art [47]. Although the authors of SimCSE claim their model outperforms SBERT by 4.4% in semantic similarity detection [47], this research evaluates both models.

Supervised Supervised methods for creating semantic embeddings employ neural networks to predict a label linked to the text. The primary type of neural network used in supervised methods is the RNN [76, 31], also sometimes combined with attention [159, 142], and encoder-decoder structures [27]. Furthermore, Convolutional Neural Networks (CNNs) have been used to capture both short and long-range relations [80], as well as Feed Forward Networks (FFNs) [71]. Usually, one of the last layers before the prediction layer is taken as the embedding [76]. RNN-based approaches adhere to all six properties. This research evaluates InferSent [31], which employs bidirectional LSTMs and was trained on the supervised task of checking whether sentences contradict. InferSent was chosen as it was state-of-the-art before LLM-based embeddings and source code was provided.

Compositional These methods use rules to combine word embeddings to create sentence embeddings [76, 104]. Semantic word embeddings can be made by looking at the context around them. This follows Firth's distributional hypothesis: *"You shall know a word by the company it keeps"* [43]. Methods like Word2Vec [100], fastText [14], GloVe [117], and ELMo [118] try to predict a masked word given the surrounding words, similar to MLM in BERT, but with a smaller context window and with an FFN. The embedding is created by compressing the weights of the neural network. This research uses Word2Vec to generate word embeddings as it is open-source and was a state-of-the-art word embedding method before LLMs. Mitchell and Lapata [104] propose that taking the mean of the word embeddings in a sentence is an effective way, and while often used, this method does not adhere to property 4. Word2Vec is used in this research as it is one of the most popular word embeddings available. Sentence embeddings are created using the formula shown in eq. (2.10), where S is a sentence denoted as an array of words w. The function $\text{Word2Vec}(w)$ maps a word to its embedding.

$$E(S) = \frac{1}{|S|} \sum_{w \in S} \text{Word2Vec}(w) \tag{2.10}$$

To create compositional embeddings that adhere to property 4, a second embedding is created by multiplying each word embedding with a positional encoding, as shown in eq. (2.11). The positional encoding is inspired by the Transformer [148] and multiplication by the Roformer [138]. In eq. (2.11) $S[pos]$ retrieves the word in the sentence at position pos, $\odot$ the Hadamard product, d_{model} is the embedding dimensionality (300 for Word2Vec), and $i \in \{1, \ldots, d_{model}\}$. The sentence embedding created with eq. (2.11) is referred to as *Word2Vec + Pos* in this research.

$$\begin{aligned} E(S) &= \frac{1}{|S|} \sum_{pos \in \{1,\ldots,|S|\}} \text{Word2Vec}(S[pos]) \odot PE(pos) \\ PE_{2i}(pos) &= sin(pos/1000^{2i/d_{model}}) \\ PE_{2i+1}(pos) &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.11}$$

Others Another method is *count-based*, in which case the frequency of each word in a sentence is counted, and the array of frequencies is the embedding. The most notable example is *Bag of Words (BoW)*[59]. However, this method does not adhere to properties 1 and 4 as the vector dimensionality grows with the text size, and the embedding is independent of word order.

Lastly, *unsupervised* methods refer to the machine learning field where models are trained on unlabeled datasets. One approach is *Paragraph Vector Based* such as doc2vec [86] that tries to predict the next word using a neural network, the last layer of which is interpreted as the embedding. Another unsupervised approach consists of *Encoder-Decoder* models such as Skip-Thought [83] and the Denoising Autoencoder [64] where a sentence is mapped to an embedding space by the encoder and mapped back to the original sentence by the decoder. It should then learn to create similar embeddings for similar sentences. The standard encoder-decoder model does not allow for arbitrary-length sentences, which means they have been combined with RNNs. In that case, both these fields adhere to all six properties.

2.3 Related Works

This research aims to study whether RL agents can improve learning by utilizing language assistance and how LLMs can help them. It investigates the use of language to help the agent with its decisions by providing information about its environment. This means the agent uses language as an additional input, not as an output. Specifically, in this research, the agent uses language as an aid to reach its goal instead of the language specifying what the agent should do and how. The former is called *language-assisted* RL, and the latter *language-conditional* RL [95, 98].

In language-assisted RL, interaction with language is used to facilitate learning and handle the environment, for example when an agent should solve a maze, and the language informs the agent of the shape of the maze. The agent does not need to listen to the language, but it may do so to reach its goal faster. In language-conditional RL, interaction with language is necessary to reach a goal. An example of this is giving an agent a box of

LEGOs. If you don't tell it what to build, the agent has no idea how to reach its goal. This research focuses on language-assisted RL, as it is a more challenging situation of showing language understanding: can we make an agent listen to the language assistance, even though it does not need to, to reach its goal?

Earlier research in understanding language in RL has focused chiefly on language-conditional RL because of its use of embodied agents [156, 135, 149, 78, 42, 108] in the form of hierarchical RL. In hierarchical RL, a high-level controller breaks a complex task down into simpler sub-tasks, which a low-level controller executes. As LLMs are trained to replicate or understand language [34, 121, 122], and language reflects the structure of the world [2, 52], some say LLMs do not only understand the language but may also contain a view of the world [54, 2, 73]. This worldview can help to create a high-level controller that breaks a complex task down into descriptions of subtasks for the low-level RL agent to execute. Using language in this latent subtask space helps the generalization of low-level controllers [162, 2, 65, 150].

This section will go into the related works of both language-assisted and -conditional RL, as they both deal with grounding language in an RL setting [53, 7, 26], which means the low-level controller of embodied agents have similar challenges to the agent in this research.

2.3.1 Language-Conditional RL

Liu et al. [91] propose InstructRL, where the low-level controller consists of a multi-modal M3AE [49] encoder that encodes an image state and a textual goal into an embedding, which is passed to a Transformer-based [148] policy network to choose an action. Lynch et al. [97] train a robotic arm that achieves a 93.5% success rate on 87,000 unique instructions. They use imitation learning and a complex architecture with a transformer for state-language fusion, another for history representation, and finally, a residual neural network for action selection. Jin et al. [78] introduce CogLoop to also train a robotic arm and use a complex high-level controller of Vision Transformers (ViT) [35] and GPT-4 [112] but do not elaborate on their low-level controller. Mu et al. [108] introduce EmbodiedGPT with an extensive low-level controller: an embedding from the observation from the ViT is concatenated with the textual subgoal before being input into their Embodied-Former. This creates a state-goal embedding that is given to a feed-forward policy network. This means that every subtask embedding is concatenated with every observation. While all these approaches are powerful, these Transformer-based low-level controllers need a lot of data to train, which is not always available.

Wang et al. [150] use two LLMs in their high-level controller to explain and plan subtasks and evaluate the MC-Controller [21] and Steve-1 [88] low-level controllers but do not indicate any preference towards either. The MC-Controller embeds the subtask description using MineCLIP [42] and concatenates the subtask embedding to every observation before they enter the RNN. This approach exhibits zero-shot generalization across tasks and language [21]. MineCLIP is a contrastive video-language model that computes a correlation between a language string and a video snippet through text and video embeddings, similar to CLIP [123], and is especially useful if a comparison between text and images or videos should be made. Steve-1 also embeds the subtask goals using MineCLIP, but then uses a

Conditional Variational AutoEncoder (CVAE) [136] to create a representation of the goal state described by the subtask. This goal state is concatenated with each observation before they are entered into a transformer-based policy. This approach is also interesting for this research but has not been further explored due to limited computational resources.

Driess et al. [37] present PaLM-E, a multi-modal high-level controller based on a PaLM [28] LLM and a large ViT [33]. As low-level controllers, they use the work of Lynch and Sermanet [96] and Brohan et al. [17]. Lynch and Sermanet [96] propse a similar agent to this work and create subtask embeddings using the Multilingual Universal Sentence Encoder (MUSE) [158] LLM before concatenating them to each observation. They claim that pre-trained LLM-based embedders make policies robust to out-of-distribution synonyms but do not support this statement with experiments. They do show that using the LLM-based MUSE embeddings outperforms using RNN-based sentence embeddings trained on random word embeddings. Brohan et al. [17] present RT-1 created for task and language generalization with a 97% success rate over 700 training instructions. They use the LLM-based Universal Sentence Encoder [24] to create sentence embeddings, which are used to condition their state encoder. This means the embedding is concatenated to each intermediate step in the EfficientNet [143] image encoder. This approach is, in a way, comparable to enabling configurations 1 and 4 of section 3.4.1.

Prakash et al. [119] use hierarchical RL in a multi-task 2D grid world. Their high-level controller does not create human-like sentences but chooses sentences from a list. The low-level controller creates a sentence embedding by mapping the words of the subtask description to word embeddings and passing them through a self-trained LSTM. These embeddings are concatenated to each state before entering the policy network. Chen et al. [26] take a similar approach but let their high-level controller create simple sentences as subtasks, creating more natural descriptions. Their agent displays generalization over tasks, and as tasks are executed based on language descriptions, it also generalizes over language.

Andreas et al. [2] recognize that language reflects the structure of the world and can help generalization by using it as a latent parameter space [73], similar to the hierarchical RL settings. They specifically evaluate its use for few-shot learning for policy search, classification, and transduction. In the policy search, they use language to describe a subtask in a multi-task, fully observable setting and use a self-trained RNN to embed the sentence before it is concatenated to the observation. Zhang and Lu [162] take a similar approach in their search for an adapter between LLM and the RL agent but embed their sentences with an SBERT model [125] without stating why.

Ahn et al. [1] and Hu and Sadigh [70] also recognize that language provides a structure of the world. They define specific policies for specific sentences in their search for a low-level controller that can work with humans and computer-based high-level controllers. Ahn et al. [1] extend the work of Prakash et al. [119] and introduce SayCan. Instead of choosing subtasks from a list, they rank the available subtasks by the probability that the subtask completes the complex task. The latter is calculated by multiplying the likelihood given by the LLM[6] of the available subtask to complete the complex task, by the probability that the

[6]Computed using the softmax of the final layer of the LLM decoder, which indicates $\Pi_{i \leq j \leq k} P(w_j | w_{<j})$ for complex task description $w_{<i}$ and subtask description $\{w_j | i \leq j < k\}$.

subtask can be completed. Hu and Sadigh [70] take another approach to find policies for specific sentences and regularize the training of each policy's loss by a prior taken from the LLM-based high-level controller when it generates a sentence for a subtask. They assume the high-level controller contains a human-like worldview, which means that the low-level controller's policy can be steered towards human-preferred action strategies by regularizing its loss by the high-level controller's embedding from its last neural network layer. The sentence-policy mapping of Hu and Sadigh [70] and Ahn et al. [1] greatly lowers the number of possible sentences to pass to the low-level controller.

Wang et al. [149] extend upon the approach of Ahn et al. [1] and introduce Voyager, which enables the agent to improve the skills of the low-level controller (and thereby increasing the probability of completing a subtask), and to learn new skills. However, it does not contain an RL-based low-level controller, as the LLM generates code for each subtask.

Yu et al. [160] notice that grounding language during training is important [15, 110, 1, 62, 37]. They define the grounding of a word as a process where an agent can link a word to the word's representation in a state by passing the word to the agent in a sentence, together with the word's representation in a past, current, or future observation. Yu et al. [160] train the low-level controller to navigate to objects indicated by a sentence and to answer questions about the state. However, they use random meaningless word embeddings and define their own RNN embedder to create a sentence embedding. This limits the possibility of generalization beyond their environment.

Co-Reyes et al. [30] worked on meta-learning of low-level controllers that can be given corrections by humans when it is not doing what they are supposed to do. Because of these corrections, new tasks can be learned quickly. Sentences are embedded by passing word embeddings through a self-learned 1D CNN. As the agent acts in a fully observable environment, each embedding is concatenated to each observation.

Williams et al. [154] and Tambwekar et al. [142] take a different approach and decompose sentences into formal grammar before executing a policy based on this grammar. Williams et al. [154] use sentences that describe a desired state, transforms them into lambda calculus, and uses a planner to create a path from the current state to the desired state described by the lambda calculus. This only works for tabular domains. Tambwekar et al. [142] decompose the sentences in lexical decision trees, which are further mapped to differential decision trees that constitute the policy of the low-level controller. This has the advantage that the low-level policy is interpretable by humans, but it cannot leverage the knowledge of an LLM.

2.3.2 Language-Assisted RL

In language-assisted RL, research has gathered information about the environment from books, manuals, wikis, or the web. The challenge in doing this is 1) finding helpful information for a given context and 2) knowing which information to assist the agent with, in which state [95]. Branavan et al. [16] tackled this by learning sentence relevance for observations using Q-Learning. They used this approach to improve a Monte-Carlo tree search planning in Civilization II. Eisenstein et al. [40] used texts from manuals to extract logical predicates that inform the agent about how the environment behaves. It then uses these predicates in its decision-making to generate the optimal policy.

Narasimhan et al. [110] approach language assistance very similar to this work. In their environment, they annotate entities in a game. Descriptive annotations such as *'enemy chasing you'* and *'randomly moving enemy'* helped the agent learn a mapping between the annotations and the dynamics. These sentences are embedded using a mean BoW approach (what this research calls Word2Vec embeddings) and a self-trained LSTM approach. Every embedding is concatenated with the observations. They see that using an LSTM provides significantly better embeddings.

2.3.3 Research Gap

This research differs from earlier noted research in five different ways. Firstly, this research focuses on approaches to create language-assisted RL agents in partially observable and possibly continuous environments, whereas previous work often focuses on fully observable environments. Secondly, previous work only appends the embedding to the observation before they enter the RNN in the low-level controller. In contrast, this work investigates whether that is the best method to enter an embedding into an RL agent. Thirdly, this research uses generic LLM-based embeddings, whereas most approaches use self-trained RNNs on word embeddings that limit generalization or use text-image embeddings such as CLIP [123] or MineCLIP [42]. This research also aims to study whether LLM-based embeddings support scaling to more complex environments. Lastly, it investigates if the choice of formulation or words in a sentence impacts the agent's ability to generalize to unseen sentences, which has not been seen in previous research.

Chapter 3

Study Design

This chapter proposes a formalization of the problem and introduces an environment that allows the RL agent to show its understanding of the language. The environment is used to determine how to assist the agent and how well the agent performs. The experiments will be evaluated on regret and cumulative regret to capture both the learning speed and stability, where regret is defined as the difference in the number of steps taken in an episode between the optimal policy and the effective policy. Regret is not defined as a difference in return, as the penalty of -1 for bumping into a wall disproportionally impacts the return. Every experiment evaluates the agent after every 200 environment steps and is run until most of the experiment's configurations are converged to a stable policy.

Furthermore, this chapter explains the data collection and the experimental setup. The hypotheses of this research will be stated along with their substantiation throughout this chapter.

3.1 Problem Formalization

This research aims to study whether RL agents can improve learning by utilizing language assistance and how LLMs can help them. Natural language assistance indicates the behavior of the transition function. Therefore, this research will use a merger between the CMDP, BA-MDP, and POMDP frameworks as defined in section 2.1. It is crucial that the agent cannot see in its observation space which transition function is active, which means the MDP should only be partially observable and thus be an instance of a POMDP. Furthermore, the sentence introduced to the agent gives possibly incomplete information about the transition function, so it cannot be regarded as the *context* from the CMDP, which means the agent acts somewhere between a BA-MDP and a CMDP.

This research introduces a form of the CPOMDP, which is closely related to the BA-POMDP [50, 126] and the CMDP [57]. This CPOMDP is defined as a tuple of $\langle \mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}(c) \rangle$. This tuple uses the context value $c \in \mathcal{C}$, where c is the value of the context of the current episode. Similar to a CMDP, $\mathcal{M}(c)$ is a POMDP generating function $\mathcal{M}(c) = \langle \mathcal{S}, \mathcal{A}, \Omega^c, T^c, O^c, s_0, R^c, \gamma \rangle$. In this case, $\Omega^c = \Omega \times \mathcal{W}^c$ where $\mathcal{W}^c$ is the set of sentences in the English language describing the environment under context c. $T^c(s_{t+1}|s_t, a_t)$:

$\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ indicates the transition function in context c of moving from state s_t to state s_{t+1} when action a_t has been performed. Next, $O^c(\omega^c_{t+1}|s_t,a_t) : \mathcal{S} \times \mathcal{A} \times \Omega^c \rightarrow [0,1]$ is a probability density function indicating the probability of receiving observation $\omega^c_{t+1} \in \Omega^c$ when executing action a_t in state s_t. Lastly, $R^c(s_t,a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ indicates the reward received in context c when resulting in state s_{t+1} due to performing action a_t in state s_t. The agent aims to find a policy $\pi(a|\tau_t)$ that maximizes the same expected cumulative discounted return as in section 2.1.1. In this case $\tau_t = \{\omega^c_0, a_0, \ldots, \omega^c_t\}$ is the history of observations and actions.

The observations ω^c contain the POMDP observation ω and an English sentence w describing the environment. This sentence serves as an imperfect proxy for the context c. This research is interested in open-vocabulary language policies where the set of sentences $\mathcal{W}^c$ has no predefined grammar or template. As this set of sentences is possibly infinite, it is impossible to learn a different model for each sentence or even for each change in dynamics. Therefore, one model should be able to interpret all sentences and use them for decision-making. The language sentence will be generated in the form of text. This textual information will be highly task- and environment-specific.

3.2 Environment

To show that an agent can understand sentences and properties of objects, it should operate in an environment that requires it to exhibit different behaviors for different sentences. This means the dynamics of the environment (what actions the agent can take and where it ends up) must be changeable so that sentences can describe what has changed. Also, multiple solutions should exist to reach the goal of the environment. One of those must be optimal so that the agent can find another option to reach its goal if presented with anything blocking its optimal path to the goal.

Figure 3.1 shows the design of the environment. The agent (blue) should reach a goal (green) as fast as possible but is possibly obstructed by an object (yellow). The object's property determines whether the agent can walk over the item and reach the goal on the left. The property chosen is *weight*. The agent can walk across light objects without passing or moving heavy objects. This means that in the case of heavy objects, the agent needs to go to the goal on the right. The complete list of 120 objects can be found in table D.1.

To ensure the environment does not reduce the agent to a classifier but lets it retain RL features, the agent only sees a $3x3$ window of the squares around it. This means the agent cannot immediately know where the goals and the object are and what type of object there is unless the agent is positioned right next to it. The agent must learn to count its steps to find the optimal policy. The agent can only partially observe its environment and operates in a POMDP. The goal on the right is placed behind a corner to discourage the agent from only taking actions to the left or right.

The distances in the environment are chosen such that the Bayes-optimal policy always goes left immediately to check if the agent can pass the object. If it can pass the object, it goes to the goal on the left. If not, it goes to the goal on the right. The expected value

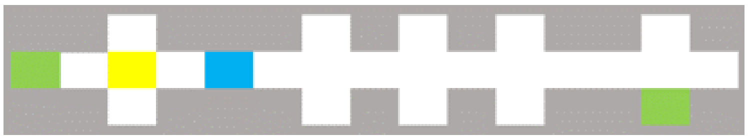


Figure 3.1: Environment used for training and evaluation. The blue square is the agent's start position, the yellow square is the object, and the green positions are the goals.

of immediately going to the right is $Q(s_0,R) = \gamma^{10} = 0.484$[1], while going to the left is $Q(s_0,L) = 100 \cdot \frac{1}{2}(\gamma^4 + \gamma^{3+10}) - \frac{1}{2}\gamma^2 = 56.4$. This means an agent without assistance should go left first, as it is a 16.6% better choice on average. Additionally, the environment of fig. 3.1 is created such that for most positions, the observation window does not inform the agent of its exact location, and the far-away goal on the right is placed behind a corner to discourage only taking actions left or right.

3.2.1 Multidirectional Variations

The environment of fig. 3.1 only allows for sentences informing the agent that the object is to its left, and as the far-away goal is always on the right, meaning the sentence always has a 1-1 relation to the action sequence to take. To make the environment more challenging, two alterations are made. Both alterations make for a more dynamic environment, meaning the environment is not necessarily equal between two episodes. However, all alterations keep the property that going to the close-by goal first is a better choice for an uninformed agent. The variations of this multidirectional environment are visualized in appendix G.

Firstly, the object is not only placed to the agent's left but also to its right, above the agent, and below it. This means that when looking at fig. 3.1, the object is always between the agent's start position and close-by goal. However, the object's position and close-by goal relative to the agent's start position can change. This alteration means that the description of the object's position becomes relevant in the sentence, which has the effect that the decision boundary stops being linear as with each added direction, two action sequences (*light* or *heavy*) are introduced that need to be identified.

Secondly, to remove the 1-on-1 relation to the action sequence to take, the far-away goal can also rotate. For example, if the close-by goal is to the agent's left, the far-away goal is to the agent's right, above it, or below it, with equal probability. This decouples the 1-on-1 relation between the sentence and the action sequence to take because if the object is heavy, the agent has to figure out where to go instead.

The environment with variations will be called the *multidirectional* environment. It will only be used for H_4, while the other hypotheses will use the *unidirectional* environment of fig. 3.1.

[1]Appendix B.4.1 explains a value of $\gamma = 0.93$ was selected.

3.3 Sentence Generation

GPT-4[2] was used to generate sentences describing the environment. GPT-4 is a Large Language Model (LLM) trained to replicate text that people created. Research agrees distinguishing GPT generated sentences from human sentences is hard [36, 103, 101], or impossible [29, 74, 48, 127]. The agent's performance on human-generated sentences was not verified due to resource constraints.

3.3.1 Unidirectional Sentences

Unidirectional sentences correspond to the unidirectional environment where the close-by goal and the object are always on the left, and the far-away goal is always on the right. These sentences were collected on September 15th 2023, using the `gpt-4` model of OpenAI[3], with a top-p and temperature value of 1. The prompt was: *"Create 30 sentences that tell someone else there is a <object> to his/her left"*. The part of *<object>* would be replaced with the supposed object. The code used is shown in appendix D. This approach enables the research to directly connect sentences to objects and their MDP.

Thirty sentences were generated for 120 objects, totaling 3600 sentences. Of these sentences, there was only one duplicate, as discovered by a similarity check that compared sentences after removing the object from the sentence. This duplicate was revised in such a way that all sentences are unique. However, it was detected that leakage occurred: Sentences that contain words that belong to other objects or even object types. The complete list is shown in appendix D. It was chosen not to change the dataset because of this leakage, as the sentences were human-like, and the agent should be robust against leakage sentences.

3.3.2 Multidirectional Sentences

Multidirectional sentences are created for the multidirectional environment where the close-by goal may be to the agent's east, west, north, or south. They may contain four possible indications where the object is relative to the agent's start position per direction, as per table D.3 (e.g., left, portside, west, 9 o'clock). The same 120 objects are used for the unidirectional environment, and negation is added: if the object to the left is *not* light, it's heavy. Five formulations for each case were made, leading to $4 \cdot 4 \cdot 120 \cdot 2 \cdot 5 = 19,200$ sentences.

The setup was equal to the setup of the unidirectional sentences, except that the sentences were collected on November 3rd 20. The prompt was: *"Give me 5 sentences that tell someone: there is <not?> a <object> placed <direction>"*. No duplicates were found.

3.4 Agent Design

The Soft Actor-Critic (SAC) algorithm was chosen because Ni et al. [111] provided working source code for it, it could be connected to the environments of section 3.2, and it performed

[2]`https://openai.com/research/gpt-4`
[3]`https://platform.openai.com/docs/models/gpt-4`

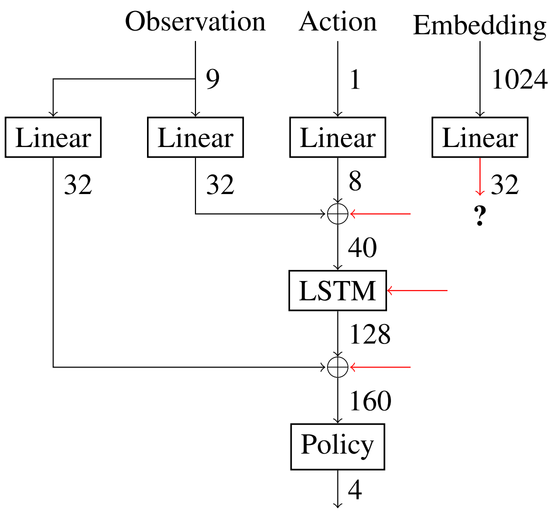


Figure 3.2: Design of the actor. The numbers indicate the dimensionality of the vector, and $\oplus$ indicates the concatenation of vectors. The policy is a feedforward network. The linear layers also contain a ReLU activation layer. Red arrows indicate where the embedding can enter the actor. See appendix E for the implementations.

well enough. The algorithm is used with a separate RNN for the actor and critic as advised by Ni et al. [111]; it is trained using a stochastic policy with Boltzmann exploration and evaluated using a deterministic policy as Ni et al. [111] and Haarnoja et al. [56] have evaluated to work best. The source code was edited to input the embedding of the language assistance into the agent, and regret metrics were created. It executed 64 episodes of random exploration before learning and exploring by itself. The list of hyperparameters of the agent can be found in table C.1.

3.4.1 Embedding Consumption

The architecture of a recurrent SAC agent shown in fig. 3.2 allows the embedding to be entered at multiple positions. This experiment will research using which position(s) the agent can best consume the embedding to answer RQ_1. The critic has a similar architecture as the actor shown in fig. 3.2 except for the policy network and accepts the embedding in the same places.

Sixteen experiments will be executed on embeddings with perfect information. This *perfect embedding* is of a single dimension and contains a 0 if the object is light and a 1 if the object is heavy. The sixteen experiments will consist of the following options.

1. Concatenate the embedding with the observation and action.
2. Initialize the LSTM hidden state with the embedding.
3. Initialize the LSTM cell state with the embedding.
4. Concatenate the embedding with the output of the LSTM.

A fifth option exists where the embedding is passed as action a_0, usually a blank vector [111, 60]. However, this research uses embeddings with a dimensionality of up to 4096 and an action dimensionality of 1; this dimensionality reduction is expected to be a source of information loss. It is not considered a suitable option for more complex environments.

The hypothesis is that agents with at least option 4 enabled will work best, as this means only the policy weights need to be trained on the embedding, meaning a smaller search space and faster convergence. As the embedding does not change over time within one episode, it should not need to be passed in the history to the LSTM. Options 1, 2, or 3 mean that the LSTM and the policy should learn to make decisions based on the embedding, meaning more parameters must be trained.

Every option above can be turned on and off, creating $2^4 = 16$ experiments. The experiment with all options turned off has an agent that receives no embedding information and can be seen as the baseline the other agents should outperform. The experiment runs for 50,000 environment steps in the unidirectional environment to determine when the learning stabilizes and is trained and evaluated in both situations: the object is heavy or light.

Hypothesis

H_1: Concatenating the embedding to the output of the LSTM leads to the lowest cumulative regret compared to other proposed methods. (RQ_1)

3.4.2 Embedding Model

RQ_2 researches which existing semantic sentence embedder leads to the best performance. In this ablation study, the non-LLM-based embedders of Word2Vec, Word2Vec with positional encodings, and InferSent are compared with the LLM-based embedders of SBERT and SimCSE to find out which embedder leads to the fastest convergence to the optimal policy.

H_2 states that as LLM-based methods can attend over the complete sentence simultaneously, they can create more informative embeddings and account for word order and context. The non-LLM-based techniques either cannot do this in the case of the Word2Vec models as it cannot attend over the complete sentence at all, or are expected to do so less performantly in the case of the InferSent model as it cannot attend over the complete sentence simultaneously. An agent assisted by LLM-based embeddings is expected to learn faster and more stably than non-LLM-based embeddings.

Two experiments will be executed: firstly, all five embedders are compared on unidirectional sentences, and secondly, Word2Vec is compared with SimCSE on the embeddings of only objects to gauge how the sentence complexity impacts the embedder performance and if attention has a significant impact. The agent is trained and evaluated on the complete set of objects/sentences for 50,000 steps.

Hypothesis

H_2: Agents assisted by LLM-based embeddings learn faster and more stably than non-LLM-based embeddings. (RQ_2)

3.5 Information Dilution & Compression

This part aims to answer RQ_3 and determine how information and noise in an embedding affect the agent's performance. Inserting the same information in a higher-dimensional embedding and introducing noise is introduced as *dilution*, while including more relevant information in an embedding is called *compression*.

3.5.1 Dilution

Sentences in natural language contain words that are not always necessary to bring the meaning across. For example, the sentences *The robot is turned on* and *The robot is turned off* are more complex and contain more noise than a binary 1 or 0 to indicate if the robot is turned on or off. This means the agent must learn what part of the sentence to focus on and what it can ignore. With perfect information (binary 1-0), there is less of a learning curve because of the absence of noise and the concentration of the information in one dimension.

To evaluate the type of information dilution the agent is sensitive to, it will be evaluated on the following four types of embeddings. Table 3.1 shows how pre-trained, perfect, object type, object, and sentence embeddings are related. A linear decision boundary exists to distinguish between *light* and *heavy* for all embedding datasets in the unidirectional environment.

- **Pre-trained**: This contains a network that can classify each embedding into a 0 or 1. Contrary to the case of the perfect embedding, no trainable linear layer is involved. More information about pre-training is given in section 3.6.2.
- **Perfect**: These embeddings contain a 1 for a blocking object and a 0 for non-blocking objects.
- **Object Type**: These are the embeddings of the words *light* and *heavy*. The resulting embeddings have a dimensionality of 1024, so this will test the impact of diluting the information over multiple dimensions.
- **Object**: These are the embeddings of the words for the objects, such as *cotton* and *feather* for light objects, and *anvil* and *boulder* for heavy objects. The embeddings also have 1024 dimensions. This tests how the performance changes when noise from redundant object properties is introduced.
- **Sentence**: These embeddings are created from sentences, introducing redundant words and formulations an object is near. These embeddings also have 1024 dimensions. This tests the impact of adding the noise of redundant words and sentence formulations to the embedding.

These experiments investigate to what extent the amount of information dilution impacts the learning performance of the agent. H_3 expects that the performance correlates negatively with the level of information dilution. The expected performance in decreasing order is providing perfect and pre-trained embeddings, type embeddings, object embeddings, sentence embeddings, and no embeddings. These experiments will use the unidirectional environment for 50,000 steps, using the embedding consumption method from RQ_1

Object Type	Object	Sentence
Light (0)	Cotton	Peek to your left and you'll spot cotton
		Don't forget to look to your left where you'll find cotton
	Feather	To your left-hand side, you'll notice there's a feather
		There appears to be a feather to the left of you
Heavy (1)	Anvil	Excuse me, but there is an anvil to your left
		By your left, there is an anvil you may not have noticed
	Boulder	Heads up, there's a boulder to your left
		Mind your left, there's a big boulder you might stumble upon

Table 3.1: Example of the structure of object types, objects, and sentences used for the experiments. Perfect embeddings are based on the number after the object type.

and the embedding model from RQ_2. This means it can use the results of the previous two experiments for the perfect, object, and sentence embeddings but needs to do an experiment for the type embeddings. The results are evaluated on cumulative reward.

Hypothesis

H_3: The level of information dilution and noise in the embedding correlates negatively with the agent's learning speed. (RQ_3)

3.5.2 Compression

More complex environments may require the agent to extract information from more informative sentences describing these environments. Therefore, the agent should be evaluated on if it can extract this information and how well it can do so. The experiment uses the multidirectional environment to make the direction of the object also relevant in the sentence. The following four levels of information are experimented on.

1. **Left**: The object and the close-by goal are always to the agent's left.
2. **Left & Right**: The object and the close-by goal may be to the agent's left or right. This makes the decision boundary non-linear.
3. **All**: The object and the close-by goal may be above or below the agent or to its left or right. This means the agent has to recognize more values for the directions.
4. **All & Negation**: The same as **All**, however, negation is also introduced. That means that if there is no light object, there is a heavy object, and vice versa.

The experiments are run for 200,000 environment steps. They are evaluated on the *cumulative regret gap*: the gap in cumulative regret between an agent with perfect embeddings and an agent with sentence embeddings. As an environment with four directions is more challenging than an environment with two directions, this metric accounts for the difference in difficulty. The resulting graphs show how well the agent learns to interpret the sentences.

Hypothesis

H_4: The amount of actionable information in an embedding correlates negatively with the agent's learning speed. (RQ_3)

3.6 Generalization

A sentence can have many forms, and similar words can be used to point something out in an agent's environment. Therefore, it is essential to assess to what extent the agent can understand objects or sentence structures it has not seen before.

3.6.1 Train-Test Splits

In this experiment, a train-test split is made: the agent is trained on the training set and evaluated on the test set not seen during training to answer RQ_4 and RQ_5. Two train-test splits will be made: 80%-20% and 30%-70%. This way, an indication can be made on how much language the agent needs to be trained to generalize and for this research to have practical implications. The performance will be compared with an agent trained on all sentences and the baseline agent with no information. The experiment will be run for 100,000 environment steps. The train-test split is made in the following two ways.

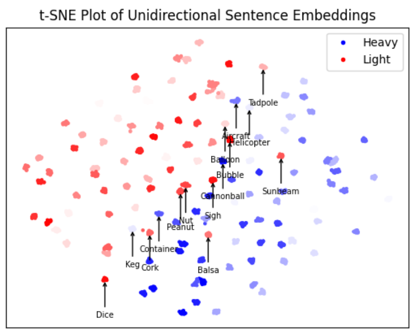
Firstly, to assess whether the agent understands object *properties*, the agent must be evaluated on sentences about objects it has never seen before. This controlled experiment will assess whether the agent learns to pay attention to the right parts of an embedding that indicate whether an object in the sentence is light or heavy. This experiment will be called the *unknown objects* experiment.

Secondly, sentences can be formed in many ways while still having similar meanings. The agent should understand this. This controlled experiment with *unknown formulations* creates a train-test split within the sentences of each object. This way, the agent has seen each object before but not each sentence structure.

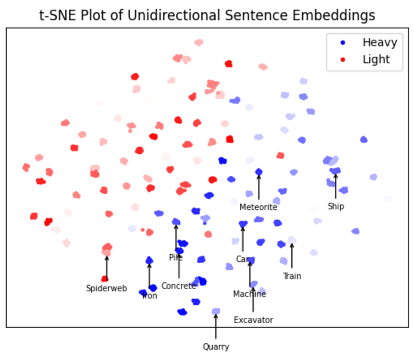
Also, to indicate how the agent would perform in a real setting, an *uncontrolled* experiment could be done where the agent is trained on a random set of sentences and evaluated on the rest, disregarding to which object a sentence belongs. This experiment was not included, as it was deemed too similar to the *unknown formulations* experiment. Refer to appendix F for a more thorough explanation.

3.6.2 Generalization Hypotheses

H_5 states that distinguishing between unknown objects will be more challenging than between unknown formulations. This is because the embedder should extract the semantics independent of the formulation. With unknown objects, the agent may learn to focus on multiple properties of an object, whereas its weight is the main property it should focus on.

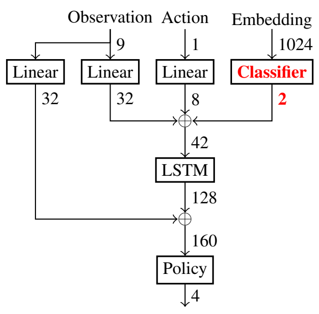


(a) Objects on t-SNE decision boundary.

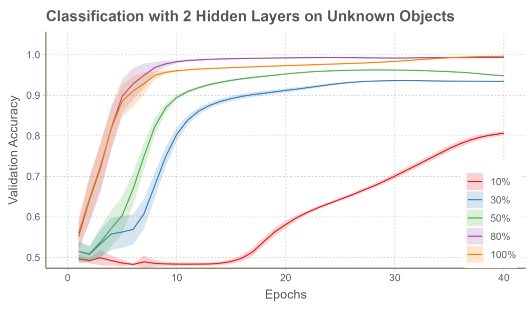


(b) Most often misclassified.

Figure 3.3: t-SNE plot of unidirectional sentence embeddings. The opacity of a dot indicates the object it belongs to. For fig. 3.3b, an 80%-20% train-test split was used (n=48). Distances used in the t-SNE analysis are based on cosine similarity.



(a) Agent Architecture



(b) Classifier Performance

Figure 3.4: Agent architecture with classifier, and classification performance of an FFN on sentences with unknown objects. The percentages are the sizes of the training sets relative to all data. $n = 38$, the shaded area indicates the 95% confidence interval.

Hypothesis

H_5: It is harder to distinguish between sentences containing unknown objects than sentences containing unknown formulations. (RQ_4 & RQ_5)

H_5 is supported by the t-SNE plot of fig. 3.3. The figure shows a projection of the 1024-dimensional embeddings to a 2-dimensional space using the t-SNE method, which focuses on retaining cluster information in the projection [66]. Figure 3.3 indicates that the embeddings form clusters of sentences with similar meanings. It shows that while some clusters overlap, all embeddings within a cluster generally belong to the same object. It must be said that the decision boundary as presented by the t-SNE plot is not necessarily the one found by the classifier, as the plot is just one possible simplification of the 1024-dimensional vectors to a 2-dimensional space.

To improve the performance on unknown objects, transfer learning is attempted. In transfer learning, a model is pre-trained in one domain and executed in another. In the case of this research, before the agent starts learning, a neural network is trained to classify the embeddings into *light* or *heavy*. After 40 epochs, the classifier is frozen and plugged into the RL agent. H_6 states that this can lower the regret of the agent, as the classifier is not hindered by noise from interacting with the environment and may find a better decision boundary. Figure 3.4 shows the agent architecture and that a classifier that has seen 80% of objects can approach 96% accuracy. Architecture considerations of this pre-trained classification network are discussed in appendix B.4.3.

Hypothesis

H_6: Pre-training on the classification of embeddings can lower the regret of an RL agent on sentences with unknown objects. (RQ_4)

Chapter 4

Results

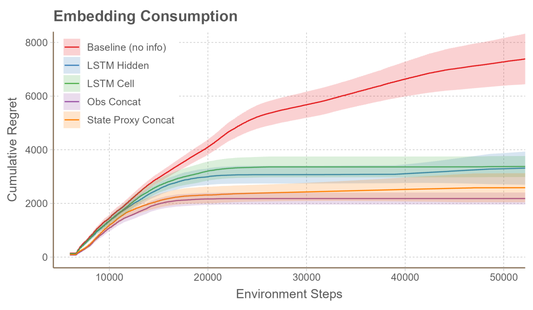
This chapter reports the results to indicate how an agent can use language embeddings to respond to language assistance and how well it can do so and accepts or rejects the hypotheses stated in chapter 3. All plots contain a 95% confidence interval over 48 experiments.

4.1 RQ_1 : Embedding Consumption

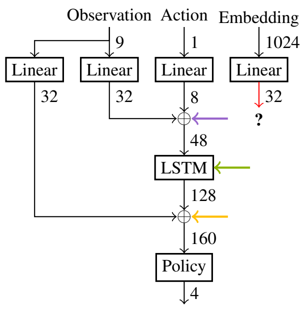
Figure 4.1 shows the cumulative regret of five agents learning how to solve the environment. Of the sixteen experiments fig. 4.1 shows five interesting cases, as the performance of the other cases follows from these: in all cases where multiple options are combined, the results are more or less similar to the best option in use. The full results are in appendix E.2.

Hypothesis H_1 of section 3.4.1 cannot be confirmed nor denied. The confidence intervals of the agents with observation concatenation and state proxy concatenation overlap significantly. However, the method of observation concatenation learns the optimal policy on average the quickest and stablest. Coincidentally, this agent requires learning the most

‘

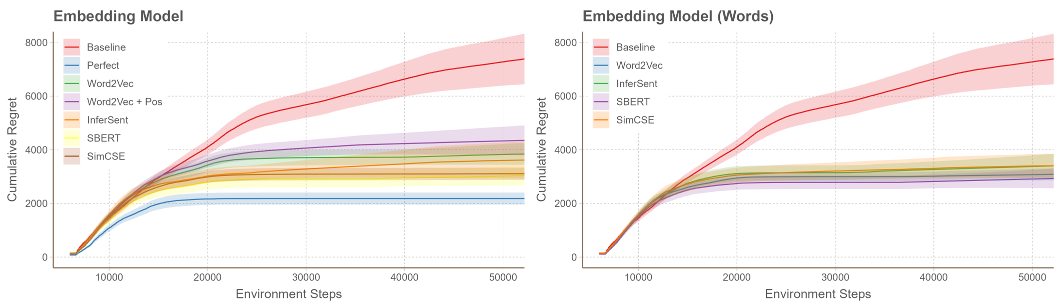


(a) Cumulative Regret



(b) Agent Architecture

Figure 4.1: Regret of different ways of entering the embedding into the agent. Lower is better. The *LSTM Hidden* configuration enters the architecture at the position of the green arrow as well.



(a) Sentences (b) Objects

Figure 4.2: Cumulative regret on embeddings of sentences and objects.

parameters to distinguish between embeddings. The other agents perform slightly worse as they take longer to find the optimal policy and seem to unlearn it in some experiments.

The experiments hereafter are executed using the agent that has the embedding concatenated with the observation and action, as this configuration outperformed the others in terms of learning speed and learning stability measured by the agent's cumulative regret.

4.2 RQ_2 : Embedding Model

Figure 4.2a shows that LLM-based embeddings outperform non-LLM-based embeddings for sentences, as agents using the former embeddings learn the optimal policy faster. This confirms hypothesis H_2 that agents using LLM-based embeddings learn quicker and more stably than non-LLM-based embeddings. This shows that large language models generate more interpretable sentence embeddings than their counterparts. This experiment shows that LLM-based embedders have learned to pay attention to specific parts of the sentence and their relations, which Word2Vec, Word2Vec+Pos cannot do, and InferSent cannot do as well. This indicates that LLM-based embeddings outperform non-LLM-based embeddings for decision-making and classification in this setting.

The decreasing order in performance is SBERT, SimCSE, InferSent, Word2Vec, and Word2Vec + positional encodings. Exact metrics are shown in table D.4. The positional encodings added to the Word2Vec embeddings degrade performance in this case. It must be said this may be due to word position not being too important in this experiment, as the deciding factor in every sentence is the weight of the object in the sentence. Also, as one could expect, the LSTM-based InferSent model outperforms the Word2Vec embeddings as it can relate words to each other, but not as well as the LLM-based methods which can do so in $O(1)$ instead of $O(n)$.

LLM-based methods do not outperform non-LLM-based methods for embedding objects as fig. 4.2b shows. The mean performance is close, and the confidence intervals overlap significantly. This lack of comparative performance of the LLM-based embeddings may be due to SimCSE being trained on sentences and paragraphs instead of single words. The results from the word-based embeddings support the hypothesis that the LLM has learned to attend to specific parts of the sentence. This is because if one creates a composite LLM-based sentence embedding by taking the average of the embeddings of the words of the sentence, it will perform similarly to the Word2Vec sentence embeddings. This indicates that the addition of the capability of LLMs to attend to specific words and their relations is significant for this setting.

Because LLM-based embeddings outperform non-LLM-based embeddings for sentences, the experiments hereafter are executed with LLM-based embeddings. Specifically, SimCSE embeddings are used, as the literature agrees they are state-of-the-art [47, 76] and are not significantly outperformed by SBERT.

4.3 RQ_3 : Information Dilution & Compression

The following section aims to accept or reject hypotheses H_3 and H_4 to prepare for answering RQ_3 and indicate how the information and noise in an embedding impact the performance of an agent.

4.3.1 Dilution

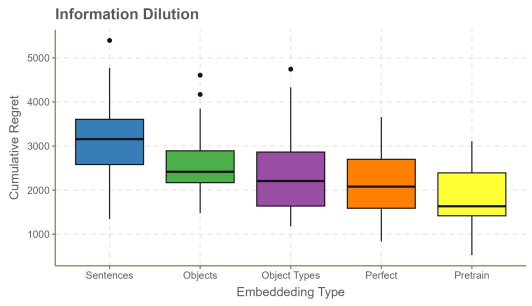
Figure 4.3 cannot reject nor confirm H_3 that the performance correlates positively with the amount of information dilution, as the results visibly overlap. The figure does show that the level of information dilution seems to correlate negatively with the RL performance. The performance in decreasing order seems to be: perfect embeddings, embeddings from object types (light versus heavy), embeddings from objects (anvil, boulder, cotton, feather), embeddings from sentences about those objects, and the baseline agent. All experiments find the optimal policy except for the baseline, and experiments with lower information dilution find it faster on average.

The added noise from redundant words in a sentence seems to have the largest effect on the agent's performance, whereas the pre-trained network provides the fastest learning. This is because of the lack of a trainable linear layer between the classifier and the LSTM, which the agent with perfect embeddings does have. It has been tested that it is not due to the one-hot encoding of the embedding output by the pre-trained classifier.

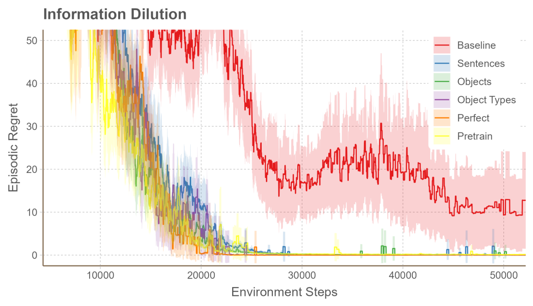
4.3.2 Compression

Figure 4.4 confirms H_4 and shows that there is a negative correlation between the amount of relevant information in an embedding and the agent's learning speed. When more directions or negations are added to the sentences, the agent takes longer to learn. See appendix G for designs of the *Left*, *Left & Right*, and *All* multidirectional environment instances.

The results for the *left* and *left & right* environments are interesting: the difference between the performance of an agent with sentence embeddings and one with perfect embeddings with only one direction is much smaller than the difference between the performance of those agents with two directions. This shows that the agent finds it more difficult to find the decision boundary when it should not only pay attention to the embedding of *weight* but also to *direction*. Close inspection shows that the cumulative regret gap of the *left* environment remains about 100 and does not go below 0.

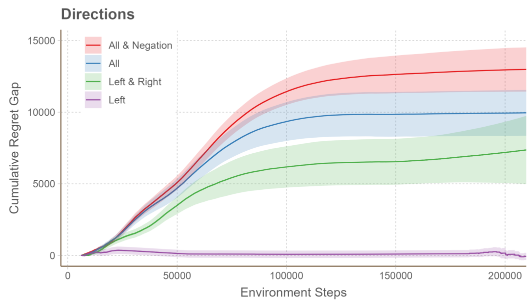


(a) Cumulative Regret

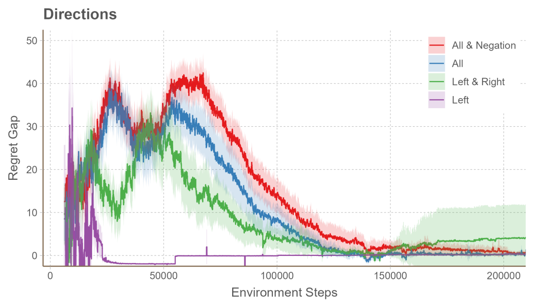


(b) Episodic Regret

Figure 4.3: Regret for different levels of information dilution. The confidence intervals of the cumulative regret significantly overlap, so a boxplot is shown for the cumulative regret at 50,000 environment steps.



(a) Cumulative Regret Gap



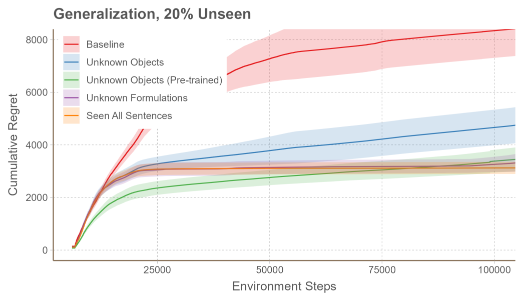
(b) Episodic Regret Gap

Figure 4.4: Regret for different levels of information compression. Note that the vertical axis is the regret *gap*, which means the difference in regret with an agent trained on perfect embeddings in the same environment to correct for environment complexity.

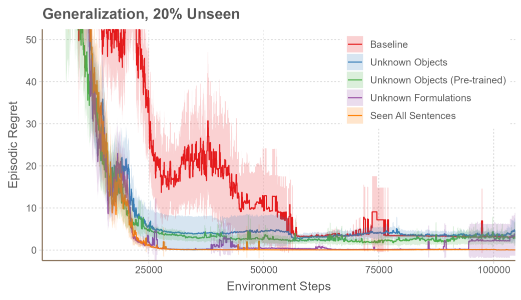
4.4 $RQ_{4,5}$: Generalization

Figure 4.5 shows that the agent can generalize well across formulations, only somewhat across objects, and that the pre-trained agent does not improve performance. This confirms H_5 of section 3.6 that the unknown formulations experiment would outperform the unknown objects experiment and rejects H_6 that a pre-trained embedding classifier can lower the regret on sentences with unknown objects.

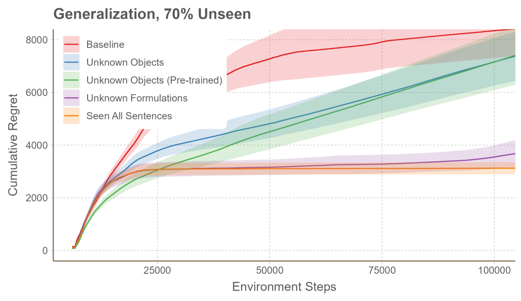
The unknown objects experiments perform worse than the unknown formulations experiment. In the case of 20% unknown objects, the success rate stabilizes around 91%, which means it fails to perform on 65 of the 720 sentences it is evaluated on. This is likely because the agent does not learn to focus on the weight property of the objects but on some other (combination of) properties that can distinguish the embeddings in the training set. The agent does not show signs of overfitting as indicated by a stable regret. The agents with and without the pre-trained classifier converge to a similar regret and possibly a similar decision boundary.



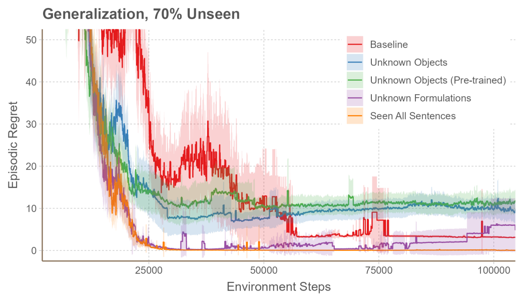
(a) Cumulative regret of 20% unseen sentences.



(b) Episodic regret of 20% unseen sentences.



(c) Cumulative regret of 70% unseen sentences.



(d) Episodic regret of 70% unseen sentences.

Figure 4.5: Regret for agents generalizing to 20% and 70% unknown objects and sentence formulations.

The unknown formulations experiment performs almost as well as the experiment where 100% of the sentences were seen during training. This indicates that the embedder generates similar embeddings for different formulations of the same meaning and that the agent recognizes them as such. This is supported by the t-SNE visualization of fig. 3.3 and by the results in fig. B.5, where the classifier is trained in a supervised manner to distinguish the embeddings. It also performs better on the unknown formulations than the unknown objects split.

Chapter 5

Discussion

5.1 Agent Architecture ($RQ_{1,2}$)

Contrary to H_1, concatenating the embedding to the observation and the action works best. Such an agent has the fastest learning curve regarding environment steps, the most stable performance, and the lowest cumulative regret. This research hypothesizes the reason for this to be the power of the LSTM. The more the LSTM sees of the embedding, the lower the cumulative regret. Appendix B.4.2 supports this, as it evaluates a more extreme case of concatenating the embedding directly to the observation instead of doing so via a mapping to a lower dimensional space. This direct concatenation leads to the lowest cumulative regret found in this research.

If the embedding is input directly into the LSTM, the LSTM becomes larger. Section 2.1.2 derives that the number of parameters of the LSTM scales linearly with its input dimensionality. The drawback of an LSTM with a lot of parameters is computation time. Figure E.4 shows the computation time for different sizes of the LSTM and shows that smaller LSTMs lead to faster training times. This research prefers consuming the embedding via observation concatenation via a linear layer because it compromises performance and computation time and is easier to implement than state proxy concatenation or LSTM initialization.

If there is a dataset available of sentences and corresponding informative labels, fig. 4.3 shows it works well to split the training into two stages: first, a supervised phase where a classifier is trained in a supervised way on the embeddings and labels, and next an RL phase where the output of the classifier for each embedding is concatenated with the observation and action.

Answer to RQ_1

How can semantic sentence embeddings best be consumed by the RL agent?
A recurrent RL agent can best consume the semantic embedding by mapping it to a lower dimensional space and subsequently concatenating it with every observation and action before they enter the LSTM. If a dataset exists of sentences and an informative label for the task, a classifier should be pre-trained in a supervised manner, and its output should be concatenated to every observation and action.

Both literature and the experiments agree that LLM-based embeddings outperform non-LLM-based embeddings. Specifically for sequential decision making, section 4.2 shows that agents based on LLM-based sentence embeddings lead to a faster and more stable learning curve than their counterparts. Specifically, adding attention to large language embedders compared with LSTM or FFN-based approaches seems to impact the embedding quality. In the case of word embeddings, no embedder appears to have a significant edge over another.

Answer to RQ_2

Which existing approach can best generate semantic sentence embeddings?
LLM-based embeddings outperform non-LLM-based embeddings, and specifically, SimCSE should be used. Non-LLM-based embeddings are not as easily interpretable as their counterparts, indicated by a slower ability of the RL agent to distinguish between them. Although the experiments did not show a significant difference between performance on SBERT or SimCSE, the literature agrees that SimCSE is the state-of-the-art sentence embedding method [47].

The answers to RQ_1 and RQ_2 lead to the architecture shown in fig. 5.1. An embedding should be created from the sentence using SimCSE, which should be concatenated to every observation and action.

5.2 Embedding Characteristics (RQ_3)

The results of the information dilution experiment could not fully confirm nor reject hypothesis H_3 that the amounts of information dilution and noise correlate negatively with the agent's learning speed. However, the results show that the dilution of the same information over more dimensions does not pose a challenge to the agent, while noise introduced from redundant words in a sentence negatively affects the learning speed.

Although this research shows that an RL agent can perform well with embeddings from embedders that were not specifically created for the RL domain, an effort to create better semantic sentence embeddings for the RL domain should focus on reducing noise and not reducing embedding dimensionality. Better embeddings for sequential decision-making can be made by pre-training embedders with domain-specific knowledge or by fine-tuning existing embedders.

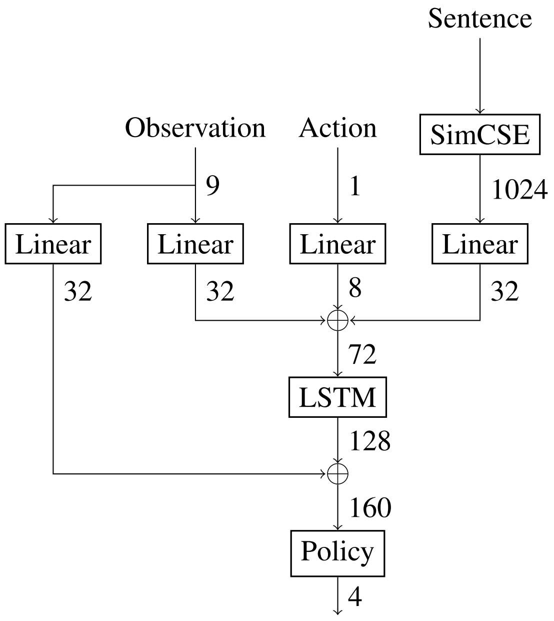


Figure 5.1: Proposed actor architecture for language-assisted RL agents.

In the case of information compression, H_4 was accepted as a higher level of information compression leads to a slower learning curve. Especially the introduction of a non-linear decision boundary seemed to have a big impact. However, the optimal policy was closely approached in all cases, which gives a first indication that scaling up to more complex environments is possible using the approach suggested in this research. An example could be environments with image inputs and using image embedders like CLIP [123].

Answer to RQ_3

How do changes in the amount of information and noise in an embedding affect the agent's performance?
The language-assisted RL agent's performance is mainly impacted by the level of noise and information and the linearity of the decision boundary in the embeddings, not by their dimensionality. The agent performs well on general embeddings from the NP domain. However, improvements could be made by creating embedders that use domain knowledge to lower the noise in embeddings. Furthermore, the agent has a slower learning curve when the embedding contains more relevant information. Still, it does find the correct decision boundaries, indicating this result can be scaled to more complex environments and sentences.

5.3 Agent Performance ($RQ_{4,5}$)

The agent can recognize sentences with unknown formulations very well but has trouble recognizing sentences with unknown objects in line with H_5. The performance on unknown formulations is nearly similar to the performance without any unknown sentences, which fig. 3.3 shows is primarily due to the embedder creating similar embeddings for sentences that should have a similar meaning. The figure also shows the embeddings for the sentences in this research differ a lot based on the sentence subject.

An effort was made to improve the performance on unknown objects using a pre-trained embedding classifier. While this pre-trained agent had a quicker learning curve, the regret for both methods converged to a similar value, indicating they found comparable decision boundaries and focused on different (combinations of) object properties instead of just weight. This rejects H_6. Although most sentences are still correctly interpreted (92% for 70% unknown objects, and 96% for 20% unknown objects, a similar success rate as Lynch et al. [97]), this research suggests including sentences with as many different subject synonyms as possible and not focusing on including different formulations of sentences. Words in the sentence need to be grounded during training; formulations do not.

Answer to RQ_4

How well can the agent generalize over unseen objects with similar properties?
The agent is prone to finding decision boundaries based on different (combinations of) object properties and misinterprets 4-8% of the sentences. Words in a sentence need to be grounded during training. This research advises creating a training set with a vocabulary as large as possible for creating language-assisted RL agents.

Answer to RQ_5

How well can the agent generalize over unseen linguistic formulations?
The agent can very well generalize over sentences with similar meanings but different linguistic formulations. All sentences with unknown formulations but known objects are interpreted correctly. This is mainly due to the embedder creating similar embeddings for semantically similar sentences. In training language-assisted RL agents through embeddings, the training set of sentences should not focus on including sentences in different formulations of a similar meaning.

5.4 Implications

This section discusses the implications of the findings of this research for RL- and semantic embedding research.

Implications for Human-Robot Interaction Humans can inform robots of their environment using a multitude of sentence formulations and usage of words. This research shows that the proposed architecture can recognize differently formulated sentences with similar meanings perfectly (RQ_5) in this setting and recognizes 92%-96% of sentences with words for unseen objects with similar properties to objects seen in the training set (RQ_4). This indicates that agents can more readily be applied in the real world if the agent's training phase contains a language-assistance dataset with a large enough vocabulary for the task at hand.

Implications for Embodied Agents Sample efficient and stable RL agents are essential for the upcoming field of Embodied Agents, which enable LLMs to interact with an environment through RL-based low-level controllers [37, 150, 1]. This research indicates how low-level controllers for partially observable environments can best be architected and how well they can generalize across similar language descriptions. Suppose language is not used as the latent embedding space between the LLM and the controller. In that case, this research also proposes reducing the noise and paying attention to linearity in decision boundaries in the embedding.

Ethical and Social Implications Efforts to create an agent that can react to human language better may lead to robots that can better understand and collaborate with people. It can help the agents learn faster, generalize better, and generalize to new tasks and environments more efficiently.

On the other hand, empowering a robot to harness human language to improve its decision-making can also introduce various forms of bias, manipulation, or misuse, which can lead to unfair or harmful outcomes. For example, a dataset can be biased, which causes the agent to react to stereotypes or reinforce power imbalances. As this research may be an indirect aid towards this, practitioners are advised to carefully design and evaluate the use of language assistance in RL, taking into account its potential benefits and risks and its ethical and social implications.

5.5 Limitations

Transfer to Human Sentences While text generated by GPT-4 can generally not be distinguished from human-generated text, this depends heavily on the prompt used to create the sentences. While humans could have uttered the sentences used in this research, people's emotional state might impact their sentence formulation and choice of words, possibly leading to an out-of-distribution embedding. This research did not have the time to test the agent's performance on human-generated sentences and proposes it as a direction for future research.

LSTM Dependence This research proposes concatenating the sentence embedding to every observation before they enter the LSTM in a recurrent agent for a partially observable environment. The LSTM was chosen because Ni et al. [111] evaluated it to work best for meta-RL settings. If, in the future, more performant agents are architected that do not depend on the LSTM (e.g., using Fast and Forgetful Memory [107], a GRU [27], or transformer-based methods) that are at least equally sample efficient, the most performant method of consuming the embedding should again be executed.

If the language assistance or subtask description stays constant throughout an episode, future researchers can also opt to concatenate the subtask to the input to the state proxy just before it is passed through the policy network. Figure 4.1 shows this has a similar performance.

Environment Simplicity The available computational resources limited this research to an environment as small as possible while enabling a language-assisted agent to exhibit language understanding. This made the agent very specific to this environment and unsuitable to test for task generalization. The one-directional environment of fig. 3.1 contains only 40 states[1], which is small compared to for example, the 10^{27} possible states of `XWORLD` [160]. The information compression experiment of RQ_3 with a total of 480 states indicates that this research's approach supports scaling up to more complex environments. Evaluating the architecture proposed by this research in more complex environments is left to future research.

5.5.1 Threats to Reproducibility

The sentences used in this research were generated using GPT-4. As GPT-4 is a commercially available system, its maintainer OpenAI might decide to change its internal weights, parameters, or other parts out of the control of this research that impact the sentences being generated. Additionally, GPT-4 is stochastic, meaning that two invocations of the same prompt do not need to yield the same output. This may impede the reproducibility of this research. To mitigate this, this study's agent code, sentences, and visualization scripts are packaged [147].

The training and evaluation stages are tested to be reproducible when executed on the abovementioned dataset, and the parameters and seeds of appendix C.

5.6 Future Work

Scale of Environment As explained in section 5.5, the environment used in this research is small and mostly fully deterministic. Scaling this to a stochastic environment with a continuous state space may introduce noise in the observation signal, possibly making it harder to receive language assistance-related rewards. This can have the effect that the agent stops listening to the language assistance because it cannot relate it anymore to obtaining a higher reward.

A first setup already has been made towards this by creating a MuJoCo [145] variant of fig. 3.1 based on the HalfCheetah environment [152], and is included in the reproducibility package [147].

Practicality For this research to become practical, it needs to be scaled up and evaluated on sentences humans would say. Future research should also include the emotional state of humans, as this can impact the phrasing of their message.

Also, any robot that is used alongside humans should not respond to *every* sentence that is uttered. For example, you would not want Alexa or Google Home to respond to everything you say at home. Therefore, the agent should also be trained on *irrelevant sentences* and learn that it should not try to relate the sentence's meaning to the environment state. This research can be done with the current setup, provided a set of sentences that do not

[1]Sentence providing language assistance is not counted.

give any information about the direction or weight of a nearby object. The agent is expected to learn slower, as it needs to learn a third policy[2] of having to find out the object's weight on its own. That is, it should execute the Bayes-optimal policy.

Multi-Modal Assistance This research focuses on *textual* assistance, but assistance could also come in the form of visual, auditory, or gestural content. Future research could use embedders such as CLIP [123] for images, Whisper [124] for audio, or MediaPipe [51] for gestures, and create more versatile and well-informed agents that can take advantage of more than text.

Actor/Critic Embedding Consumption A bug in implementing the state-proxy concatenation embedding method led to the insight that providing the language assistance only to the actor yields performance comparable to the baseline agent. This provides a first indication that the critic's value function must see the language assistance. Future research can investigate whether it holds true. Providing language assistance to only the actor could optimize the training runtime. This research considers providing language assistance to only the critic as an unviable option, as the actor cannot take actions based on it.

Domain-Specific Embedders This research shows that general NLP sentence embedders can inform an RL agent about its environment. However, the performance on unseen objects was not yet perfect. This research proposes two ways to solve this. Firstly, embedders could be fine-tuned during the training phase of the RL agent, possibly creating more generalizable language embeddings within a domain. This was not tried in this research as SimCSE is based on RoBERTa, which contains 335 million parameters [93, 47], leading to a slow training time.

Secondly, domain-specific embedders could be created bottom-up, for example, using formal grammar or self-traint RNNs. In specific domains, this could help map sentences to their effect. However, future researchers must note that this approach would lose the LLM knowledge and possibly increase synonym and sentence formulation sensitivity.

Few-Shot Generalization The generalization tests in this research were all zero-shot: the agent had never seen the sentences before. Future research could go into few-shot generalization, where after training the agent, it is trained on new sentences just a few times. This may stipulate how expensive retraining the agent is in case of a language corpus distribution shift.

[2]The first and second policies are executed when the agent is informed of a light or heavy object

Chapter 6

Conclusion

This research explored whether RL agents can improve learning by utilizing language assistance and how Large Language Models (LLMs) can help them. The primary goal was to design, implement, and evaluate an agent architecture for efficient and stable learning and generalization capabilities.

The research proposed using LLMs to create semantic embeddings of natural language sentences and feed them into a recurrent soft actor-critic agent. The results showed that the agent benefited from language assistance and achieved faster and more stable learning than an uninformed baseline agent. It finds it best to concatenate a mapping of the embedding to every observation before they enter the LSTM. Also, RL agents using LLM-based embeddings outperform those based on non-LLM-based embeddings.

When evaluating the agent on embedding characteristics like its dimensionality, the agent is sensitive to noise in the embedding but not its dimensionality. Also, this research indicates the proposed architecture supports scaling up to more complex environments.

Lastly, this research shows that the agent could generalize well across sentences with similar meanings but different formulations and could recognize 92%-96% of sentences with unknown objects with common properties to the objects in the training set. An improvement based on transfer learning was suggested to improve the generalization among sentences with unknown objects by pre-training a network to classify embeddings. However, this did not lead to a notable improvement. This research advises practitioners to focus on training an agent on a language corpus with an extensive vocabulary, not on unique sentence formulations.

Bibliography

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. *arXiv preprint arXiv:1711.00482*, 2017.

[3] Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205, 1965.

[4] Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

[5] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, pages 507–517. PMLR, 2020.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*, 2018.

[8] Bram Bakker. Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, 14, 2001.

[9] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[10] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[11] Richard Bellman. A problem in the sequential design of experiments. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 16(3/4):221–229, 1956.

[12] Richard Ernest Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[13] Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[14] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[15] Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, 2009.

[16] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43: 661–704, 2012.

[17] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[19] Nina G Buchina, Paula Sterkenburg, Tino Lourens, and Emilia I Barakova. Natural language interface for programming sensory-enabled scenarios for human-robot interaction. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8. IEEE, 2019.

[20] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[21] Shaofei Cai, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13734–13744, 2023.

[22] Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *Aaai*, volume 94, pages 1023–1028, 1994.

[23] Shicong Cen and Yuejie Chi. Global convergence of policy gradient methods in reinforcement learning, games and control. *arXiv preprint arXiv:2310.05230*, 2023.

[24] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[25] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[26] Valerie Chen, Abhinav Gupta, and Kenneth Marino. Ask your humans: Using human instructions to improve generalization in reinforcement learning. *arXiv preprint arXiv:2011.00517*, 2020.

[27] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[28] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[29] Elizabeth Clark, Tal August, Sofia Serrano, Nikita Haduong, Suchin Gururangan, and Noah A Smith. All that's' human'is not gold: Evaluating human evaluation of generated text. *arXiv preprint arXiv:2107.00061*, 2021.

[30] John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. *arXiv preprint arXiv:1811.07882*, 2018.

[31] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[32] Brett Daley and Christopher Amato. Human-level control without server-grade hardware. *arXiv preprint arXiv:2111.01264*, 2021.

[33] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[36] Yao Dou, Maxwell Forbes, Rik Koncel-Kedziorski, Noah A Smith, and Yejin Choi. Is gpt-3 text indistinguishable from human text? scarecrow: A framework for scrutinizing machine text. *arXiv preprint arXiv:2107.01294*, 2021.

[37] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model, 2023.

[38] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[39] Michael O'Gordon Duff and Andrew Barto. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

[40] Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. Reading to learn: Constructing features from semantic abstracts. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 958–967, 2009.

[41] Hugh Ellis, Mingxiang Jiang, and Ross B Corotis. Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems*, 1(2):92–99, 1995.

[42] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

[43] John Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, pages 10–32, 1957.

[44] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[45] The PyTorch Foundation. Lstm. PyTorch 2.1 Documentation, 2023. URL https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html. URL:https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html (version: 2023-11-29).

[46] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[47] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*, 2021.

[48] Sebastian Gehrmann, Hendrik Strobelt, and Alexander M Rush. Gltr: Statistical detection and visualization of generated text. *arXiv preprint arXiv:1906.04043*, 2019.

[49] Xinyang Geng, Hao Liu, Lisa Lee, Dale Schuurmans, Sergey Levine, and Pieter Abbeel. Multimodal masked autoencoders learn transferable representations. *arXiv preprint arXiv:2205.14204*, 2022.

[50] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[51] Google. *Gesture recognition task guide*, 2023. https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer [Accessed: 10-12-2023].

[52] Alison Gopnik and Andrew Meltzoff. The development of categorization in the second year and its relation to other cognitive and linguistic developments. *Child development*, pages 1523–1531, 1987.

[53] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

[54] Wes Gurnee and Max Tegmark. Language models represent space and time. *arXiv preprint arXiv:2310.02207*, 2023.

[55] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[56] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[57] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[58] Mengting Han, Xuan Zhang, Xin Yuan, Jiahao Jiang, Wei Yun, and Chen Gao. A survey on the techniques, applications, and performance of short text semantic similarity. *Concurrency and Computation: Practice and Experience*, 33(5):e5971, 2021.

[59] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[60] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[61] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[62] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

[63] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[64] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. *arXiv preprint arXiv:1602.03483*, 2016.

[65] Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020.

[66] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

[67] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[68] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[69] Gabriel Romon (https://math.stackexchange.com/users/66096/gabriel romon). Probability that an item from a group is not in a random sample (sampling without replacement). Mathematics Stack Exchange, 2023. URL `https://math.stackexchange.com/q/4812287`. URL:https://math.stackexchange.com/q/4812287 (version: 2023-11-22).

[70] Hengyuan Hu and Dorsa Sadigh. Language instructed reinforcement learning for human-ai coordination. *arXiv preprint arXiv:2304.07297*, 2023.

[71] Chaochao Huang, Xipeng Qiu, and Xuanjing Huang. Text classification with document embeddings. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data: 13th China National Conference, CCL 2014, and Second International Symposium, NLP-NABD 2014, Wuhan, China, October 18-19, 2014. Proceedings*, pages 131–140. Springer, 2014.

[72] Qingyan Huang. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221. IEEE, 2020.

[73] Evan Hubinger, Adam Jermyn, Johannes Treutlein, Rubi Hudson, and Kate Woolverton. Conditioning predictive models: Risks and strategies. *arXiv preprint arXiv:2302.00805*, 2023.

[74] Matthew Hutson. Robo-writers: the rise and risks of language-generating ai. *Nature*, 591(7848):22–25, 2021.

[75] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.

[76] Francesca Incitti, Federico Urli, and Lauro Snidaro. Beyond word embeddings: A survey. *Information Fusion*, 89:418–436, 2023.

[77] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[78] Chuhao Jin, Wenhui Tan, Jiange Yang, Bei Liu, Ruihua Song, Limin Wang, and Jianlong Fu. Alphablock: Embodied finetuning for vision-language reasoning in robot manipulation. *arXiv preprint arXiv:2305.18898*, 2023.

[79] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2): 99–134, 1998.

[80] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[81] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[82] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[83] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. *Advances in neural information processing systems*, 28, 2015.

[84] Thomas Kleine Buening, Christos Dimitrakakis, Hannes Eriksson, Divya Grover, and Emilio Jorge. Minimax-bayes reinforcement learning. *arXiv e-prints*, pages arXiv–2302, 2023.

[85] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, 2022.

[86] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[87] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[88] Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023.

[89] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[90] Michael Lederman Littman. *Algorithms for sequential decision-making*. Brown University, 1996.

[91] Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-following agents with multimodal transformer. *arXiv preprint arXiv:2210.13431*, 2022.

[92] Rui Liu and Xiaoli Zhang. A review of methodologies for natural-language-facilitated human–robot cooperation. *International Journal of Advanced Robotic Systems*, 16(3):1729881419851402, 2019.

[93] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[94] Owen Lockwood and Mei Si. A review of uncertainty for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 155–162, 2022.

[95] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.

[96] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[97] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023.

[98] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.

[99] Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.

[100] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[101] Fatemehsadat Mireshghallah, Justus Mattern, Sicun Gao, Reza Shokri, and Taylor Berg-Kirkpatrick. Smaller language models are better black-box machine-generated text detectors. *arXiv preprint arXiv:2305.09859*, 2023.

[102] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[103] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023.

[104] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *proceedings of ACL-08: HLT*, pages 236–244, 2008.

[105] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[106] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[107] Steven Morad, Ryan Kortvelesy, Stephan Liwicki, and Amanda Prorok. Reinforcement learning with fast and forgetful memory. *arXiv preprint arXiv:2310.04128*, 2023.

[108] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pretraining via embodied chain of thought, 2023.

[109] H Albert Napier, Richard R Batsell, Norman S Guadango, and David M Lane. Impact of a restricted natural language interface on ease of learning and productivity. *Communications of the ACM*, 32(10):1190–1198, 1989.

[110] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874, 2018.

[111] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

[112] OpenAI. Gpt-4 technical report, 2023.

[113] Philip Osborne, Heido Nõmm, and André Freitas. A survey of text games for reinforcement learning informed by natural language. *Transactions of the Association for Computational Linguistics*, 10:873–887, 2022.

[114] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[115] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[116] Raj Kumar Pathria. *Statistical mechanics*. Elsevier, 2016.

[117] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[118] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[119] Bharat Prakash, Nicholas Waytowich, Tim Oates, and Tinoosh Mohsenin. Interactive hierarchical guidance using language. *arXiv preprint arXiv:2110.04649*, 2021.

[120] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[121] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *No journal found.*, 2018.

[122] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1 (8):9, 2019.

[123] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[124] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[125] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[126] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptivemacmahon2006walk pomdps. *Advances in neural information processing systems*, 20, 2007.

[127] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[128] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[129] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16*, pages 632–640. Springer, 2006.

[130] Jürgen Schmidhuber. Reinforcement learning in markovian and non-markovian environments. *Advances in neural information processing systems*, 3, 1990.

[131] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[133] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

[134] Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088, 1973.

[135] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artificial life*, 11(1-2):13–29, 2005.

[136] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[137] Matthijs TJ Spaan. Partially observable markov decision processes. *Reinforcement learning: State-of-the-art*, pages 387–414, 2012.

[138] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, page 127063, 2023.

[139] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[140] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[141] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[142] Pradyumna Tambwekar, Andrew Silva, Nakul Gopalan, and Matthew Gombolay. Natural language specification of reinforcement learning policies through differentiable decision trees. *IEEE Robotics and Automation Letters*, 2023.

[143] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[144] Yingjie Tian and Yuqi Zhang. A comprehensive survey on regularization strategies in machine learning. *Information Fusion*, 80:146–166, 2022.

[145] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[146] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[147] Sander van Leeuwen. Language Assistance in Reinforcement Learning in Dynamic Environments - Reproducibility Package, December 2023. URL `https://doi.org/10.5281/zenodo.10322510`.

[148] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[149] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[150] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

[151] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[152] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *Adaptive and Natural Computing Algorithms: 9th International Conference, ICANNGA 2009, Kuopio, Finland, April 23-25, 2009, Revised Selected Papers 9*, pages 380–390. Springer, 2009.

[153] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *Artificial Neural Networks–ICANN 2007: 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part I 17*, pages 697–706. Springer, 2007.

[154] Edward C Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *2018 ieee international conference on robotics and automation (icra)*, pages 4430–4436. IEEE, 2018.

[155] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[156] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

[157] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[158] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, et al. Multilingual universal sentence encoder for semantic retrieval. *arXiv preprint arXiv:1907.04307*, 2019.

[159] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[160] Haonan Yu, Haichao Zhang, and Wei Xu. Interactive grounded language acquisition and generalization in a 2d world. *arXiv preprint arXiv:1802.01433*, 2018.

[161] Hongming Zhang and Tianyang Yu. Taxonomy of reinforcement learning algorithms. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 125–133, 2020.

[162] Wanpeng Zhang and Zongqing Lu. Rladapter: Bridging large language models to reinforcement learning in open worlds. *arXiv preprint arXiv:2309.17176*, 2023.

[163] Ziqiang Zhang, Long Zhou, Junyi Ao, Shujie Liu, Lirong Dai, Jinyu Li, and Furu Wei. Speechut: Bridging speech and text with hidden-unit for encoder-decoder based speech-text pre-training. *arXiv preprint arXiv:2210.03730*, 2022.

[164] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.

Appendix A

Glossary

- **Activation Layer**: Non-linear function. Used to enable neural networks to find non-linear decision boundaries.
- **BA-MDP**: Bayes-Adaptive Markov Decision Process. A version of a POMDP where the hidden state remains constant throughout an episode. Further explained in section 2.1.2.
- **BERT**: Bidirectional Encoder Representations from Transformers [34]. An encoder-only Transformer model used for text understanding.
- **CMDP**: Contextual Markov Decision Process. A version of a BA-MDP where the *context* informs the agent about the hidden state. Further explained in section 2.1.2.
- **CPOMDP**: Contextual Partially Observable Markov Decision Process. Merger of the a CMDP and a POMDP where the context only informs the agent about a part of the hidden state. Further explained in section 3.1.
- **Cumulative Regret**: Sum of regret obtained since starting training. Can be interpreted as the sum of the number of wrong actions taken since starting training.
- **DRQN**: Deep Recurrent Q-Networks [60]. A reinforcement learning algorithm for solving POMDP environments. DRQN extends on DQN by making decisions based on *histories* instead of single states. The histories are encoded using an RNN. Further explained in section 2.1.2 and appendix B.2.
- **DQN**: Deep Q-Networks [105]. An off-policy reinforcement learning algorithm for solving MDP environments utilizing a neural network to learn the $Q(s,a)$ value function.
- **FFN**: Feed-Forward Network. A network mapping an input to an output by alternating linear layers and activation functions without residual or recurrent connections.
- **GPT**: Generative Pretrained Transformer [121, 122]. A decoder-only Transformer model created for text generation.
- **H**: Hypothesis
- **InferSent**: Sentence Embedder based on bidirectional LSTM networks [31]. Further explained in section 2.2.

- **Linear Layer**: Trainable mapping of $y = \sigma(Wx + b)$ from input space $x_t \in \mathbb{R}^i$ to output space $y_t \in \mathbb{R}^o$ using a matrix $W \in \mathbb{R}^{o \times i}$ and bias $b \in \mathbb{R}^o$ containing trainable weights, and non-linear activation function σ.
- **LLM**: Large Language Model. A stack of encoder-, decoder-, or encoder-decoder models of the Transformer architecture capable of understanding and/or generating text, trained on a very large language corpus.
- **LSTM**: Long Short-Term Memory [68]. An RNN architecture that solves the vanishing gradients problem. Further explained in section 2.1.2.
- **MDP**: Markov Decision Process. A way of modeling a sequential decision-making process using states, actions, and rewards. Further explained in section 2.1.1.
- **MLM**: Masked Language Modeling. One of the ways to train a BERT model, by making the model predict a word in a sentence using the words around it.
- **NSP**: Next Sentence Prediction. One of the ways to train a BERT model, by making it predict if two sentences are subsequent.
- **NLP**: Natural Language Processing. The field of researching how computers can understand and generate text.
- **Policy Network**: Neural network to select an action. In this research a FFN was used with an architecture of (I $\times$ 128 $\times$ 128 $\times|\mathcal{A}|$).
- **POMDP**: Partially Observable Markov Decision Process. An extension to the MDP where the agent does not receive the environment's full state in its observations. Further explained in section 2.1.2.
- **Regret**: Difference in number of steps taken and number of steps required by the optimal policy.
- **RL**: Reinforcement Learning. The field that aims to make agents learn how to accomplish a goal without being explicitly programmed to do so. Further explained in section 2.1.
- **RNN**: Recurrent Neural Network. A neural network that outputs an output and a hidden state. The hidden state is taken as input by the next invocation. This hidden state makes for a recurrence.
- **RQ**: Research Question.
- **SAC**: Soft Actor-Critic [56]. A recurrent version of this algorithm was used in this research. Its goal is achieving the goal as optimally as possible, while taking actions as randomly as possible. Further explained in section 2.1.2.
- **SBERT**: Sentence-BERT [125]. An approach of creating models that can create sentence embeddings using siamese BERT networks. Further explained in section 2.2.
- **SimCSE**: Simple Contrastive Learning of Sentence Embeddings [47]. A model that can create sentence embeddings using contrastive BERT networks.
- **t-SNE**: Machine learning method to visualize in a high-dimensional clusters in a lower-dimensional plot. Used for fig. 3.3.

- **Word2Vec**: Model that maps words to semantic embeddings [100]. In an experimental context, it refers to a sentence embedding created by averaging over the Word2Vec embedding of each word in a sentence. Further explained in section 2.2.
- **Word2Vec + Pos**: Sentence embedding created by averaging over the positional Word2Vec embedding of each word. The positional embedding is created by multiplying each embedding with a sinusoid dependent on the word's position in the sentence. Further explained in section 2.2.

Appendix B

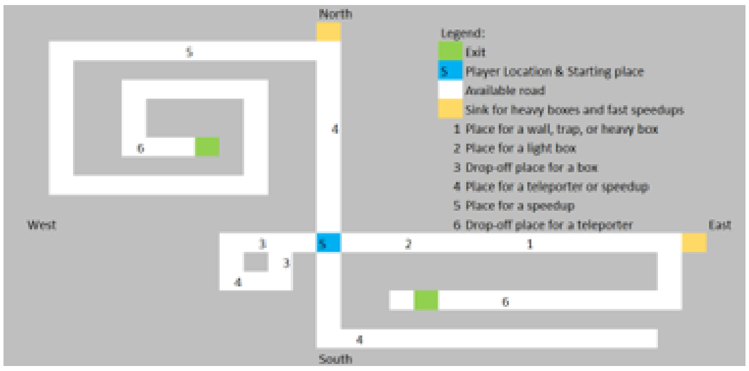
Engineering Decisions

The following chapter gives a background on choices made to make the research in the main chapters possible.

B.1 Initial Environment Design

The first design of an environment to train and test a language-assisted RL agent is shown in fig. B.1. It was built such that multiple objects could be introduced: walls blocking a path, traps catching the agent, boxes that could be moved, or speedups that made the agent walk faster. The items and goals were placed such that each introduction of an object resulted in a unique optimal path, encouraging the agent to make decisions based on the sentence it was provided with. However, this environment was rejected because it supports a low number of objects, and a lot of exploration is necessary to find the goal.

Often, people do not say what happens exactly but instead, communicate the essence of their message and may use synonyms or words that are similar to the subject. For example, if there is water on the floor where someone might slip over, people may warn each other about a *puddle*, a *pool of water*, a *splash*, or a *spill*, with the intended meaning that it might be slippery. Therefore, the real challenge for the agent is to identify the *property* of an object



North
West
East
South
S
1
2
3
4
5
6
Legend:
Exit
Player Location & Starting place
Available road
Sink for heavy boxes and fast speedups
1 Place for a wall, trap, or heavy box
2 Place for a light box
3 Drop-off place for a box
4 Place for a teleporter or speedup
5 Place for a speedup
6 Drop-off place for a teleporter

Figure B.1: Initially proposed environment.

and respond to it. The environment of fig. B.1 is not made with this in mind, but rather to find the differences between objects themselves. Also, the goals of the environment in fig. B.1 are far away: 27 steps and 56 steps need to be taken to reach them, without obstruction from any item. This means that if the agent would explore randomly for 1000 environment steps, the probability of finding the goal closest by is approximately 12.6%, while finding the solution that is furthest away in the case of an obstruction on position 1 is only 0.6%. Restricting the action space to only valid actions would improve this to 24.2% and 3.3%, respectively. Solving this environment would require a lot of computation power and effort in explorative strategies. Section 3.4 explains the efforts that have been made in this regard. However as exploration is not the primary focus of this research, the decision was made to create a smaller environment.

B.2 Deep Recurrent Q-Networks

Deep Q-Networks (DQN) is an algorithm that uses neural networks to approximate the Q-function of an MDP [105] (see section 2.1.1). The algorithm Deep Recurrent Q-Networks (DRQN) extends DQN to the POMDP domain by making decisions on a *history* of observations and actions [60]. This history is encoded into a vector with a recurrent neural network such as an LSTM or a GRU, and this vector is passed to the DQN neural network.

DRQN was already not state-of-the-art at the time this thesis started, as DQN had already been improved upon numerous times [63, 146, 151, 10, 44] and deep policy gradients as well as deep actor-critic methods were outperforming DQN too [106, 132, 89, 56, 46]. However, DRQN's learning capacity was considered good enough to learn the environment introduced in appendix B.1 while being relatively straightforward to implement.

A custom DRQN implementation was made, built upon the DQN implementation of dr. Wendelin Böhmer[1], adhering to the software architecture proposed by Daley and Amato [32]. However, the implementation of the algorithm required a few weeks of debugging efforts, as the source of unwanted behavior could be attributed to the agent, the new environment, as well as the replay buffer, which all had changed. A lot was learned about the implementation of DRQN and satisfactory results were achieved on the `CartPole-v1`[2] environment. However, remaining uncertainty as to whether the implementation of DRQN was correct led to the search for proven implementations of POMDP-solving agents.

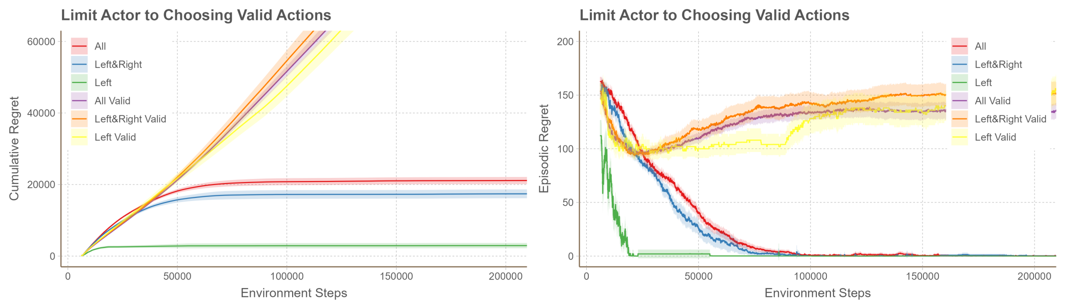
B.3 Exploration

Exploration refers to how much an agent scouts its environment before it starts to optimize for collecting reward. An agent nearly always needs to explore to reduce its epistemic uncertainty and find its goal. However, as the agent does not know when it has reached the optimal way to its goal, it might stop exploring too soon and never find its goal. To make the agent explore, three methods have been tried in this research: *random* exploration, and exploration due to adding an *intrinsic reward* signal. This part gives an overview of the

[1]Committee member of this thesis. Code was for the course CS4400 Deep Reinforcement Learning

[2]`https://www.gymlibrary.dev/environments/classic_control/cart_pole/`

‘



(a) Cumulative regret

(b) Episodic regret

Figure B.2: Limiting the actor to only valid actions. The lines with *Valid* limit the actions to choose only valid actions. The others allow all actions, but incur a -1 penalty for choosing an invalid action.

approaches used in this research and is by no means a complete overview of RL exploration methods.

B.3.1 Random Exploration

A way to explore an environment is to take random actions in every state with a uniform probability. This is called random exploration [85]. Random exploration is a way to gather data about the environment when the neural network of the agent's policy is still close to its random initialization, however, it can not cover a lot of space as its reach decays exponentially. That is because it is just as likely that the agent goes back to where it has been as to go anywhere new. For example: Let's say there is an agent that is in a hall and has the actions to go left, right, up or down. Except up or down make it stay in the same place. Then, after 100 random steps, the probability of reaching further than 12 steps is only 3.8%. An improvement could be to limit the action set to only left or right, which improves the probability to 9.6%.

B.3.2 Limiting to Valid Actions

The environment can also help with exploration by limiting the action space to only the actions available in the state the agent is in. In limiting the valid actions, the SAC action logits for invalid actions were set to $-\infty$, so the probability of sampling these actions became zero. This led to the environment rewards becoming sparse, as the reward would always be 0, except for reaching the goal. This did not improve the agent's behavior: in environments where the goal was further away than 6 blocks, the agent would not learn to reach the goal and would start oscillating between a state where it could only go back and a state with four choices. I assume it presented this behavior because in the environment of fig. 3.1 the Up and Down was only available 50% of the time and thus did not get much feedback compared to going Left or Right. Figure B.2b shows that agents in an environment where invalid actions cannot be executed, the agents converge to oscillation.

B.3.3 Intrinsic Reward

Whereas *extrinsic* reward $r(s,a)$ is given by the MDP, *intrinsic* reward comes from the agent itself. It can incentivize the agent to explore more by giving the agent a reward for ending up in a state it has not often been in. The simplest solution is *count uncertainty* [94]. Count uncertainty keeps count of how often a state was visited, and upon a state gives an intrinsic reward inversely proportional to the count, often using the inverse square of the count [94]. This can work well in environments where states can be mapped well [114, 9]. Figure B.3 shows the impact of different scales of intrinsic reward when tested on the environment of fig. B.3e. It was chosen not to include intrinsic reward, as it did not have a positive impact in the small environment of fig. 3.1.

If states cannot be mapped to a discrete count, *Random Network Distillation* (RND) can be a solution [20]. It proposes to find the novelty of a state by comparing the output of two neural networks, f_s and f_d. f_s is a static network and does not change. f_d is a dynamic network and is being trained to mimic the output of f_s on the states the agent visits. The weights of both networks are initialized randomly. If the agent has not visited a state often, there will be a bigger difference between the outputs of f_s and f_d compared to an often visited state. The agent receives an intrinsic reward proportional to the L1 norm of the difference between the output vectors. RND was not used in this research because it is a more challenging intrinsic reward source than count uncertainty. Given that count uncertainty did not work for my environment where discrete counts are possible, RND would also not work.

B.4 Hyperparameter Search

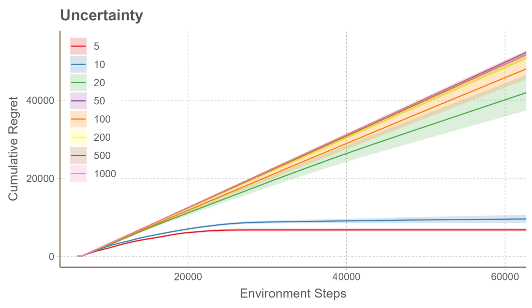
The following section explains the search for suitable hyperparameters for the agent and environment.

B.4.1 Discount Factor

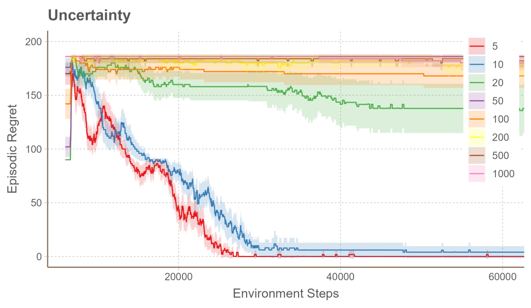
The discount factor $\gamma \in [0,1]$ is part of the MDP as explained in section 2.1.1 and influences the horizon length. It is used to discount future rewards, and as such can be used to set how much the agent prioritizes short-term over long-term rewards. As the expected future rewards are discounted with γ for step $t+1$, γ^2 for step $t+2$, etc, the impact of future rewards decays exponentially with regards to γ. As this is a geometric series, it means that the horizon length of an agent is specified by $\frac{1}{1-\gamma}$.

A value of $\gamma = 0.93$ was found experimentally for this research. Values in $\gamma \in \{0.9, 0.91, \dots, 0.99\}$ were evaluated. The graphs did not exhibit any significant results and the value of $\gamma = 0.93$ performed only marginally better than the others. Theoretically, a value of 0.93 makes sense as its horizon length is $\frac{1}{1-0.93} = 14.3$ which means that it includes the longest optimal path for an uninformed agent of 13 steps.

‘

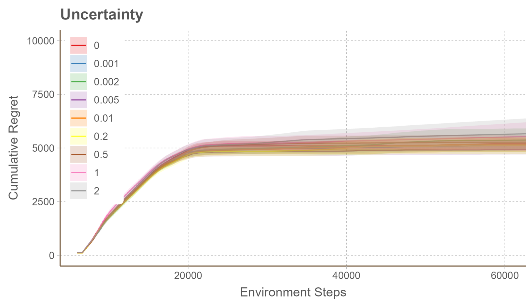


(a) Cumulative regret with high intrinsic reward

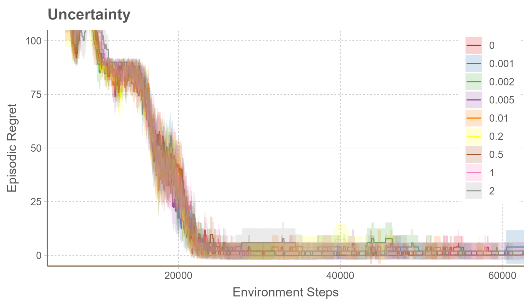


(b) Episodic regret with high intrinsic reward

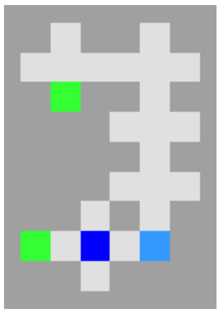
‘



(c) Cumulative regret with low intrinsic reward

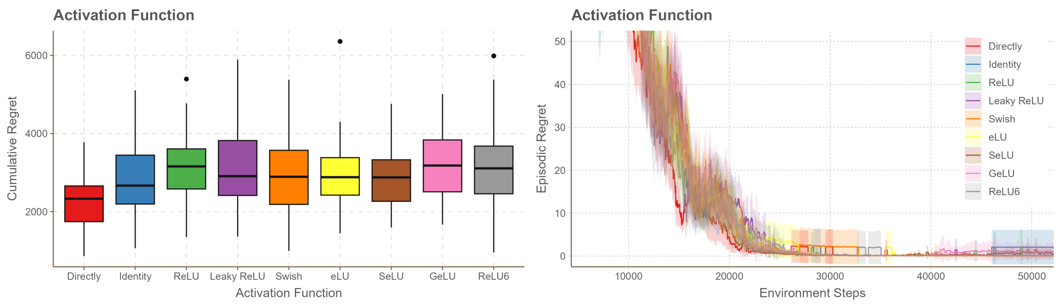


(d) Episodic regret with low intrinsic reward



(e) Environment. The object is dark blue.

Figure B.3: The agent’s performance over various scales for intrinsic reward, based on count uncertainty. A value of 0 was chosen. The environment of fig. B.3e was used.



(a) Cumulative Regret at 50k environment steps. (b) Episodic Regret

Figure B.4: The agent's performance after 50.000 environment steps over different activation functions used after the embedding linear layer. *Directly* means no linear layer and no activation function. *Identity* means no activation function. ReLU was chosen.

B.4.2 Embedding Activation Functions

Neural networks consist of alternating matrix multiplications and non-linear functions. These non-linear functions are called activation functions and allow the neural networks to learn non-linear relations. The architectures of fig. E.3 contain a linear layer that maps the sentence embedding to a lower dimensional space and applies an activation function. This experiment researches for *only the embedding linear layer* which activation function works best.

Figure B.4 visualizes various agents' regret on the unidirectional environment with SimCSE sentence embeddings, and shows that not including a linear layer or activation function at all and instead inserting the embedding *Directly* into the LSTM outperforms all other methods. However, fig. E.4 shows that this approach has a large impact on the runtime of the algorithm because the LSTM contains about six times as much parameters, leading to a similar increased factor in computation time.

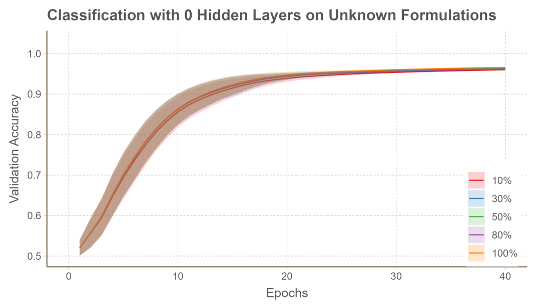
Because the cumulative regret from the activation functions overlaps significantly, ReLU was chosen as it is commonly used and simple to implement.

B.4.3 Classification Network Architecture

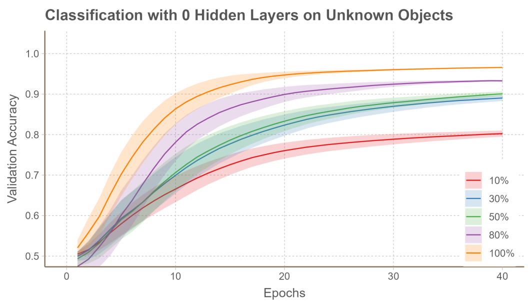
The classifier network is created in an attempt to get better generalization on the unknown objects experiment, as explained in section 3.6.2. Following the idea of transfer learning, it is pre-trained on classifying embeddings into whether their sentences indicate the object to be *light* or *heavy*. After pre-training, the network is frozen and plugged into the agent.

During this research, a mistake was made in the implementation of the state-proxy concatenation embedding consumption method. This caused this method to perform comparably to the baseline, which led to the hypothesis that the policy neural network would not be powerful enough to find the right decision boundary. Because of this, the policy network of $(1024 \times 128 \times 128 \times 2)$ was trained on classifying the embeddings, as shown in fig. B.5c

‘

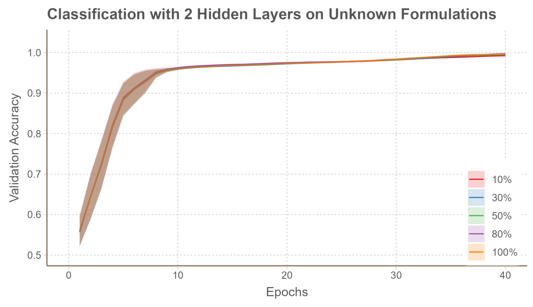


(a) Small Classifier on Unknown Formulations

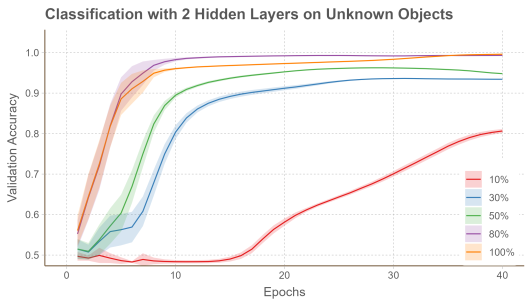


(b) Small Classifier on Unknown Objects

‘

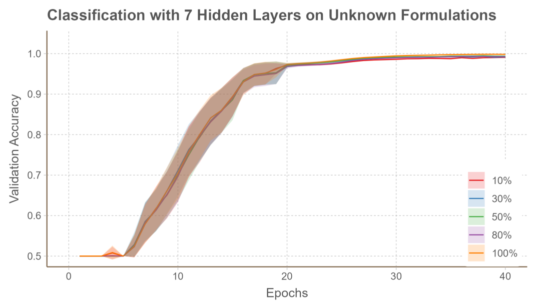


(c) Policy Classifier on Unknown Formulations

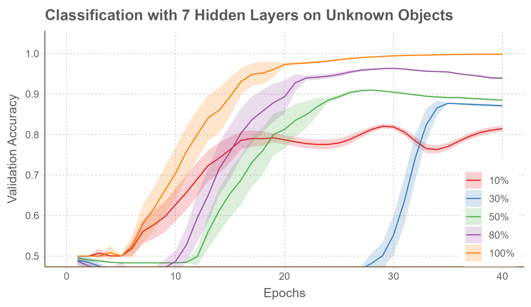


(d) Policy Classifier on Unknown Objects

‘



(e) Large Classifier on Unknown Formulations



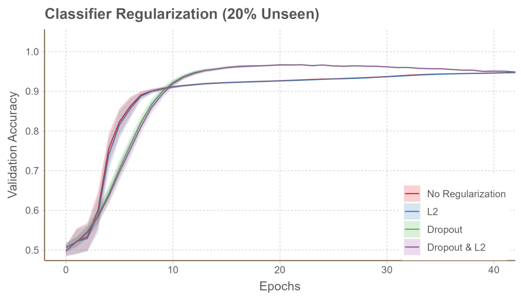
(f) Large Classifier on Unknown Objects

Figure B.5: Impact of classifier architecture on generalization tasks.

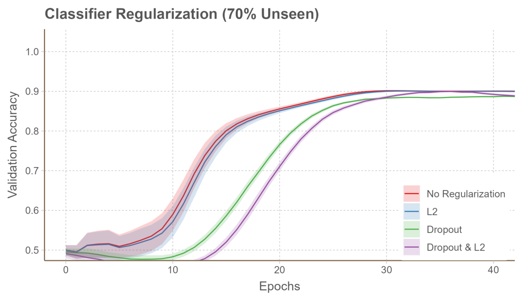
and fig. B.5d. It showed the policy network does contain enough weights to find a good decision boundary, indicating a bug in the implementation of the state proxy concatenation.

Later, this classifier network was re-used for generalization purposes. A small architecture search was executed to determine if having a shallower network (fig. B.5a and fig. B.5b) or a deeper network (fig. B.5e and fig. B.5f) would perform better. The original policy network was chosen in the end instead, as it provided a balance between the underfitting of the small network and the overfitting of the large network. No regularization was applied in the experiments. The Adam optimizer was used with a learning rate of 0.001 [82].

A neural network can also improve generalization beyond the training set by using regularization. Regularization lowers the probability a neural network overfits on its training



(a) 20% Unknown Objects



(b) 70% Unknown Objects

Figure B.6: Results of applying dropout with $p = 0.5$ after every layer and using L2 regularization with a weight of 10^{-4}.

set. L2 regularization does so by introducing a loss component that penalizes the neural network for large weights and pushes those weights to zero [144]. Dropout does so by resetting individual neurons with a probability p [67]. Figure B.6 shows the impact of dropout and l2 regularization on the accuracy on a withheld test set is not significant, therefore it was chosen not to regularize the classifier.

B.4.4 Gradient Update Frequency

The frequency of gradient updates refers to how often an agent learns from its experiences. For each gradient update, the agent samples a number of trajectories equal to the batch size from its replay buffer and calculates its critic and actor using the SAC losses. The derivative over these losses indicates the direction in which to update the gradients of the neural networks used.

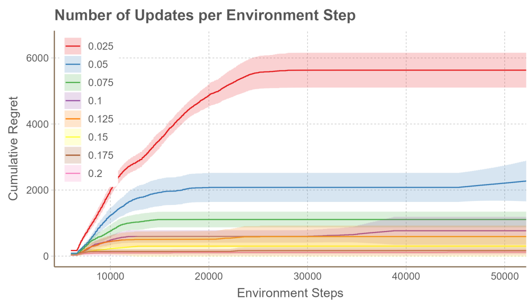
During the first evaluations, the frequency was set to one update every five rollouts. This works fine when an agent has not approximated the optimal policy but starts being a computational challenge once the agent does learn because its rollouts become 10-25 times shorter. This means that the agent starts learning much more often once it has learned a good policy, which leads to a slow progression in the number of environment steps.

A choice was made to set the gradient update frequency relative to the number of environment steps as all results in chapter 4 are also visualized relative to the number of environment steps. Figure B.7 indicates the impact of the gradient update frequency per environment step on the agent's learning speed, tested in the unidirectional environment with perfect embeddings. A value of 0.05 was chosen as it meant the agent would not learn too quickly that differences between other hyperparameters (e.g., embedding consumption, embedding model, generalization) could not be distinguished.

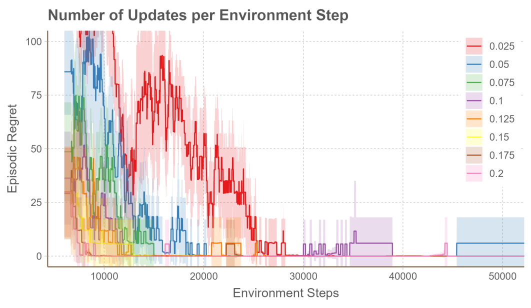
B.4.5 Bends in Multidirectional Environment

Like section 3.2.1 describes, one of the goals of the multidirectional environment is decoupling the 1-on-1 relation between a sentence and the actions it should take. To decouple this as much as possible, this experiment tried to gauge the learning speed of the agent when 0,

‘



(a) Cumulative Regret



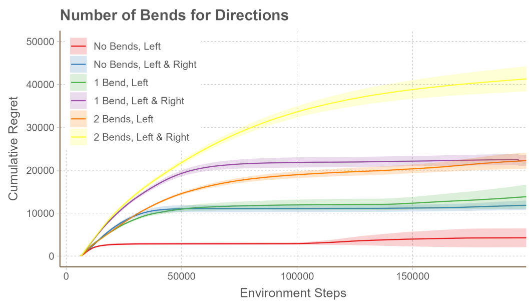
(b) Episodic Regret

Figure B.7: The performance of the agent over a range of different numbers of updates per environment step. Perfect embeddings were used. A value of 0.05 was chosen.

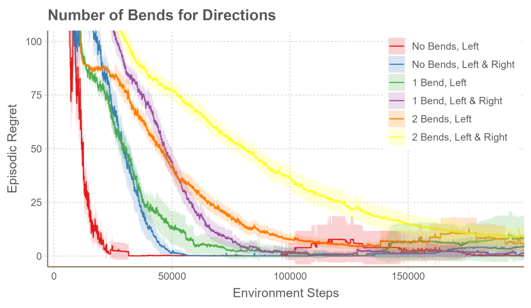
1, and 2 bends were introduced to the environment. Every bend that is introduced creates three times as many possible mazes, as each bend can have a hook in three ways.

At the time of executing this experiment, the algorithm was not yet fully optimized. Because of the 24-hour time limit set by the cluster on which this experiment was executed[3], not more than 150.000 environment steps could be executed. Figure B.8 shows that not all experiments with two bends were able to converge to the optimal policy, so it was chosen to only bend the environment at the agent's starting position.

‘



(a) Cumulative Regret



(b) Episodic Regret

Figure B.8: The impact of the number of bends in a multidirectional environment on the agent's performance. One bend was chosen.

[3]DelftBlue

Appendix C

Hyperparameters

Parameter	**Value**
Limit to only valid actions	No
Number of iterations (after random exploration)	2500 (1000 for generalization tests)
Random exploration rollouts	64
Rollouts per iteration	1
Updates per iteration	0.05 (one update per 20 environment steps)
Buffer size	10^6
Batch size	256
Separate RNN for actor & critic	Yes
RNN model	LSTM
RNN hidden size	128
RL Algorithm	SAC Discrete
Action mapping dimensionality	8
Observation mapping dimensionality	32
Embedding mapping dimensionality	32
Reward mapping dimensionality	0
Policy network	(LSTM output) $\times 128 \times 128 \times 4$
Classification network (if used)	(Embedding Size) $\times 128 \times 128 \times 4$
Learning rate	0.001
γ	0.93
τ	0.005
Embedding consumption	Trained mapping of embedding concatenated to each observation
SACD target entropy	0.7
Learning rate entropy Tuning	0.0003
Intrinsic reward	None used
Seeds	$\{42,\ldots,89\}$

Table C.1: Hyperparameters of the agent. One rollout is one episode. One iteration is some rollouts, followed up with several gradient updates.

Appendix D

Sentence Generation

```python
def generate_direction_prompt(
    word: str, num_sentences: int, direction: str, negation: bool
):
    negation_str = "not" if negation else ""
    return f"Give me {num_sentences} sentences that tell someone: there is {negation_str} a {word[0].lower()} placed {direction}"


def generate_prompt(word: str, num_sentences: int):
    return f"Create {num_sentences} sentences that tell someone else there is a {word} to his/her left"


def get_sentence(prompt: str) -> list[str]:
    for i in range(3):
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": prompt},
            ],
        )
        break
    else:
        raise Exception("Could not get response from OpenAI")
    return [
        sentence
        for sentence in response.choices[0].message.content.split("\n")
        if len(sentence) > 0
    ]
```

Listing D.1: Python code used for generating sentences using GPT-4. The code was executed on September 15th, 2023 for the unidirectional sentences, and on November 3rd, 2023 for multidirectional sentences.

Heavy		**Light**	
Boulder	Battleship	Feather	Breeze
Anvil	Bulldozer	Cotton	Bubble
Barbell	Rocket	Leaf	Powder
Cannonball	Cruise	Paperclip	Marshmallow
Cement	Dam	Pencil	Balsa
Concrete	Elephant	Seed	Chiffon
Dumbbell	Excavator	Foam	Silk
Granite	Factory	Straw	Cork
Iron	Lighthouse	Chalk	Pen
Lead	Mammoth	Dust	Dice
Log	Lumber	Balloon	Rice
Meteorite	Obelisk	Smoke	Cobweb
Brick	Quarry	Air	Gauze
Car	Ferry	Hair	Sunbeam
Engine	Tanker	Bandage	Froth
Freight	Steamroller	Mist	Wrap
Locomotive	Tugboat	Cream	Peanut
Machine	Warship	Cloud	Nut
Mountain	Windmill	Thread	Confetti
Pile	Hippopotamus	Tadpole	Scent
Rock	Iceberg	Swab	Twig
Container	Moose	Meringue	Quill
Ship	Rhinoceros	Spiderweb	Sugar
Submarine	Statue	Lace	Filament
Tank	Turbine	Breath	Residue
Tractor	Walrus	Ribbon	Velvet
Truck	Tram	Pollen	Eyelash
Whale	Train	Floating	Glitter
Yacht	Keg	Fog	Sigh
Aircraft	Helicopter	Tissue	Zephyr

Table D.1: List of objects used to denote heavy and light objects in the environment.

Class	Object	Leaked Word	Sentence
	Breath	Air	Look to your left-hand side, there's a breath of air
	Breath	Air	A gust of air is floating from your left direction
	Breath	Air	I sense fresh air is coming from your left
	Breath	Air	There's an air draft coming from the area on your left
	Breath	Air	There is some movement of air to your left
	Breath	Air	A breath of air is seeping out on your left side
	Breath	Air	You're having an air movement on your left-hand side
	Breath	Air	There seems to be a breath of fresh air flitting to your left direction
	Breath	Air	A fresh waft of air is springing up on your left
	Breath	Breeze	There's a breeze originating from your left side
	Breath	Breeze	If you turn to your left, you might feel a gentle breeze
	Breath	Breeze	You may sense a breeze from the left-hand side
	Breath	Breeze	You will feel a breeze if you lean slightly to your left
	Breath	Breeze	Have you noticed a cool breeze from your left?
	Breath	Floating	A gust of air is floating from your left direction
	Breeze	Air	You can feel the air motion to your left, that's a breeze
	Breeze	Air	A soft current of air is coming your way from the left
	Breeze	Air	The breeze is softly stirring the air to your left
	Breeze	Floating	A gentle zephyr is floating in from your left
	Breeze	Hair	The hair on your left side is slightly moving due to the breeze
	Breeze	Zephyr	You should be experiencing a cool zephyr on your left
	Breeze	Zephyr	A gentle zephyr is floating in from your left
	Bubble	Floating	Just to let you know, a bubble is floating to your left
	Bubble	Floating	Did you notice the bubble floating to your left?
	Bubble	Floating	Kindly take note, there is a bubble floating on your left
	Bubble	Floating	Please don't overlook the little bubble that's floating to your left
	Cement	Pile	A pile of cement is waiting for you on your left
	Cloud	Floating	Above and to your left, there's a cloud floating
	Cruise	Ship	To your left is a magnificent cruise ship
	Cruise	Ship	Observe carefully, there's a gorgeous cruise ship to your left
	Cruise	Ship	Glance to your left, there's an outstanding cruise ship you would appreciate
	Cruise	Ship	If you glance to your left, you will see a charming cruise ship
Different	Foam	Pile	Hey, did you notice that pile of foam to your left?
Different	Helicopter	Floating	You'll find a helicopter floating to the left of you
	Smoke	Cloud	There is a cloud of smoke to your left hand side
	Smoke	Cloud	Quick heads up, there is a cloud of smoke to your left
	Smoke	Mist	I noticed a slight smoky mist to your left

Table D.2: Leakage of object words in unidirectional sentences.

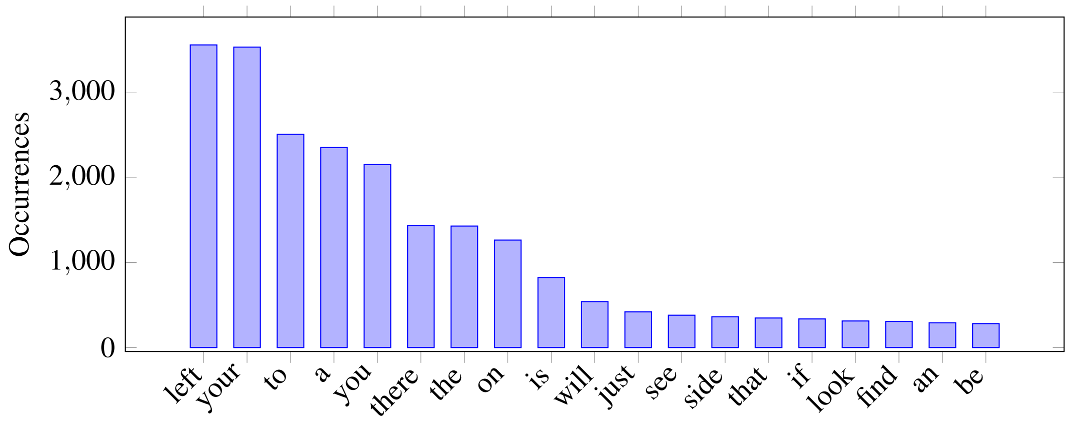


Figure D.1: The twenty most frequent words in the onedirectional sentence dataset.

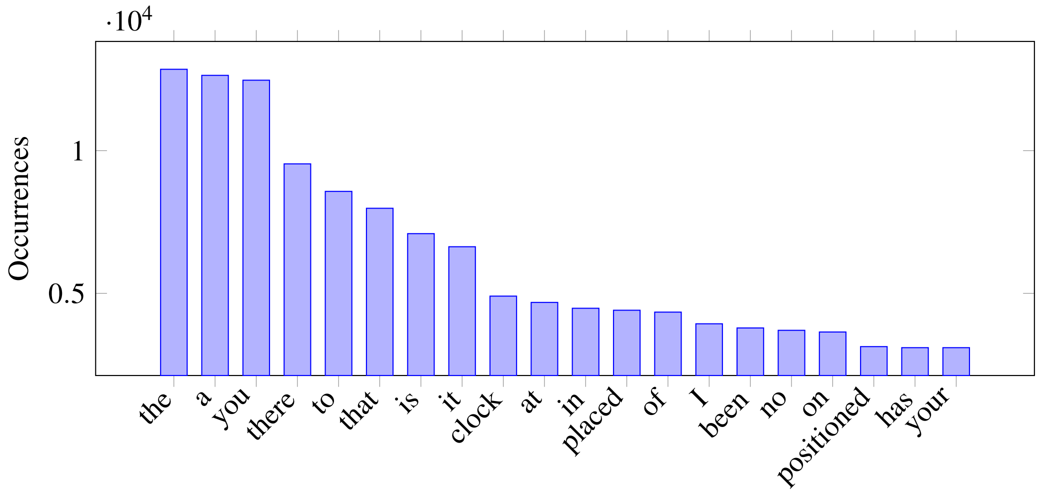


Figure D.2: The twenty most frequent words in the the multidirectional sentence dataset.

Left	**Right**	**Up**	**Down**
on its left	on its right	in front of it	behind it
in western direction	in eastern direction	in northern direction	in southern direction
at nine o’clock	at three o’clock	at twelve o’clock	at six o’clock
on portside	on starboard	above it	below it

Table D.3: Indications of to which direction the object is placed relative to the agent’s start position for the multidirectional dataset.

D.1 Embedding Model Performance

Embedder	**Sentences**	**Objects**
Word2Vec	3826 ± 1349	3073 ± 1039
Word2Vec + Pos	4329 ± 1823	NA
InferSent	3600 ± 2304	3377 ± 1431
SBERT	$\mathbf{3020} \pm 1228$	$\mathbf{2910} \pm 1164$
SimCSE	3107 ± 788	3395 ± 1525

Table D.4: Mean and standard deviation of the cumulative regret of the agent on different embedders after 50.000 environment steps. SimCSE was chosen for sentences as it yields the most stable performance and literature agrees it to be the most performant.

Appendix E

Embedding Consumption

E.1 Agent Architecture

The following section presents the architecture of the actor of the agent, with the positions the embedding can be consumed by the actor. Figure E.3 shows the results of entering the embedding in the positions shown in fig. E.2. The positions are identical for the critic. The different lines of fig. E.3 indicate which of the LSTM initialization options is turned on. The LSTM initialization configuration is shown in fig. E.1. All results are plotted with a 95% confidence interval over 48 runs.

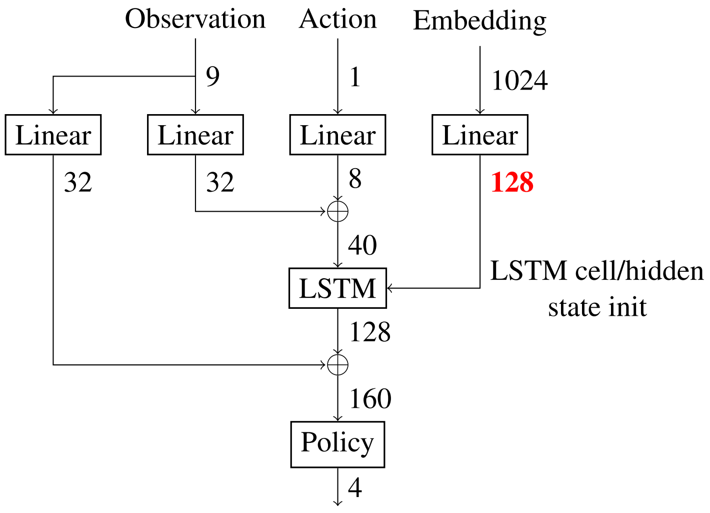
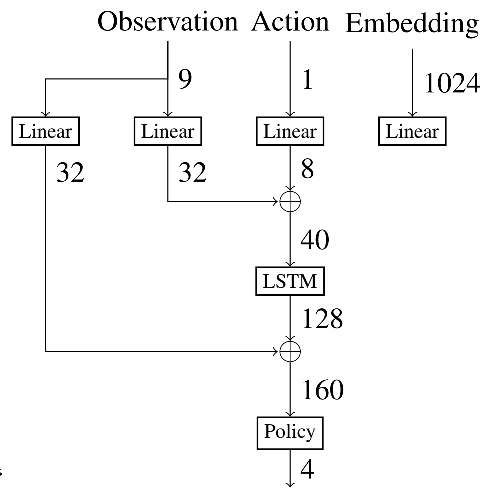
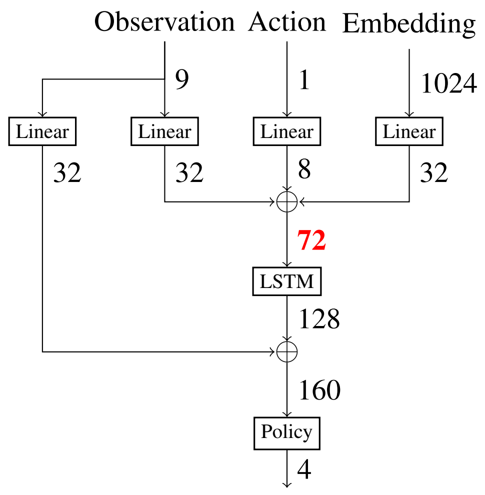


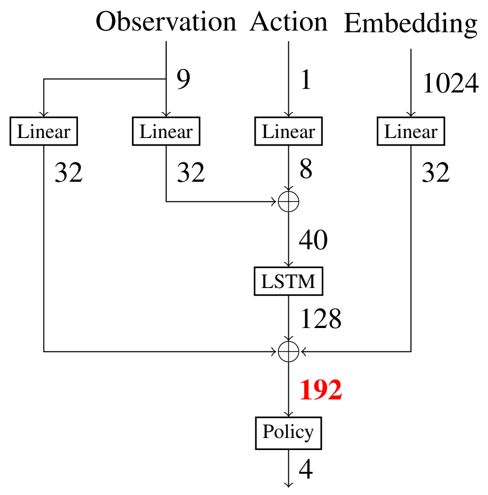
Figure E.1: Design of the actor with the linear embedding initializing the LSTM hidden- and cell-state.



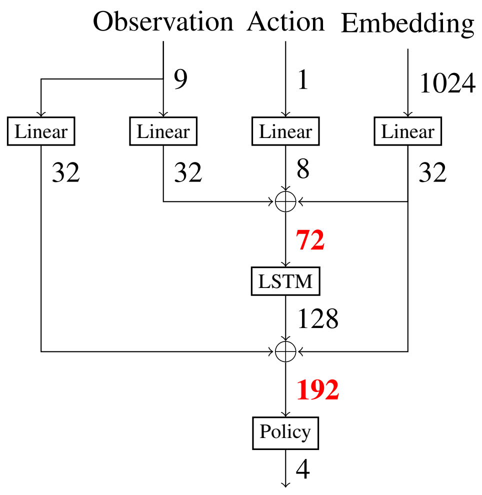
(a) No embedding concatenation.



(b) Observation embedding concatenation.



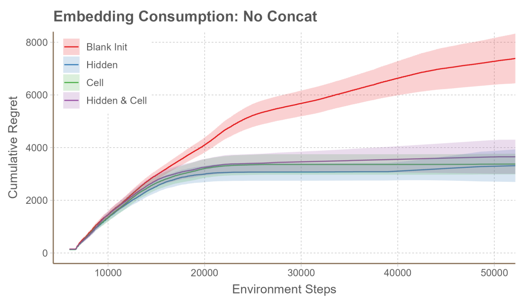
(c) State proxy embedding concatenation.



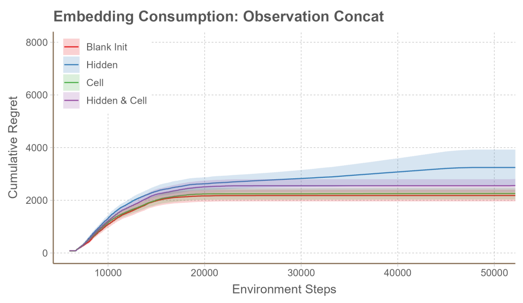
(d) Both embedding concatenations.

Figure E.2: Actor architectures corresponding to the results of fig. E.3

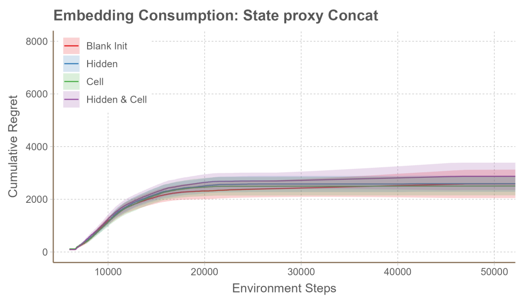
E.2 Results



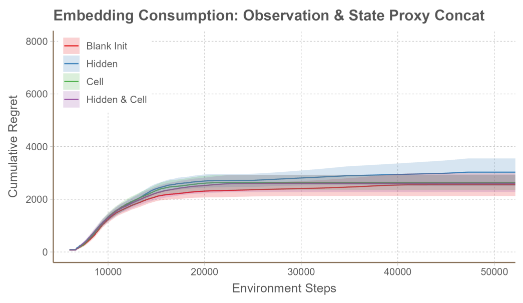
(a) No embedding concatenation.



(b) Observation embedding concatenation.



(c) State proxy embedding concatenation.



(d) Both embedding concatenations.

Figure E.3: Full results for the embedding consumption results. The colors of the lines indicate which LSTM vector was initialized with the embedding (see fig. E.1). Corresponding architectures can be seen in fig. E.3.

		LSTM Init			
		No	Hidden	Cell	Both
Concatenation	No	7275 ± 3051*	3373 ± 1343	3373 ± 1343	3648 ± 2206
	Observation	**2182** ± 770	3245 ± 2342	2257 ± 771	2553 ± 853
	State Proxy	2585 ± 1854	2592 ± 1059	2504 ± 1215	2872 ± 1787
	Both	2553 ± 1483	3030 ± 1796	2638 ± 996	2605 ± 1131

Table E.1: Mean and standard deviation of the cumulative regret of the embedding consumption agents after 50.000 environment steps. * Uninformed baseline agent.

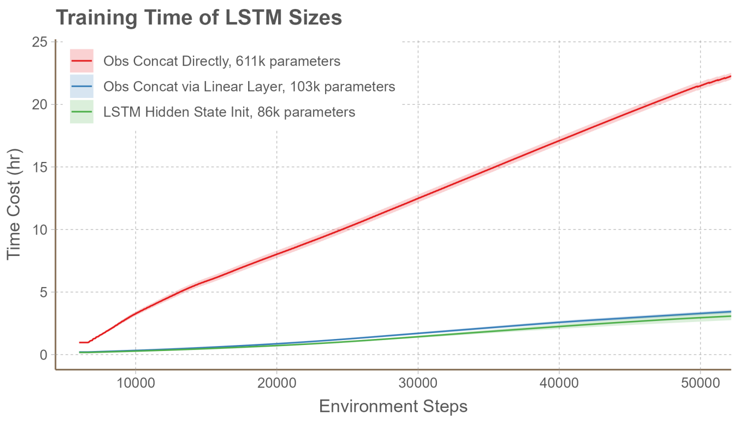
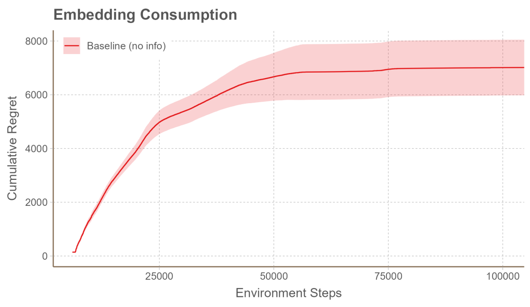


Figure E.4: Training time of different relative sizes of the LSTM on the unidirectional environment.

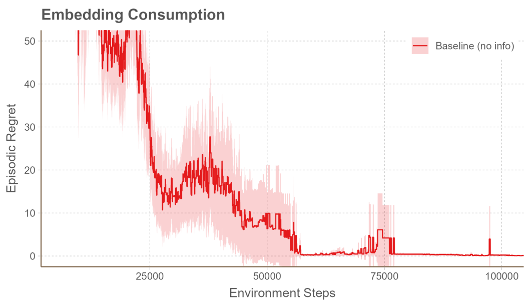
E.3 Baseline Agent

Figure E.5 shows that the uninformed baseline agent learns the Bayes-optimal policy after about 80.000 environment steps.

‘



(a) Cumulative regret



(b) Episodic regret

Figure E.5: Performance of the baseline uninformed agent. Regret is based on the Bayes-optimal policy.

Appendix F

Generalization Experiment

For the generalization tests, it was chosen not to include an *uncontrolled* experiment in the results. Where an *unknown objects* experiment deliberately excludes all sentences belonging to an object, and an *unknown formulations* make a train-test split in the sentences belonging to each object, an uncontrolled experiment randomly sample from the total set of sentences, disregarding which sentences belong to which objects. An uncontrolled experiment would have been an addition to the results if the content of its training set and/or test set were sufficiently different from the other experiments.

The reason the uncontrolled experiment was not included is that the probability of the training set to contain sentences for all available objects is $7.882 \cdot 10^{-20}$ for an 80%-20% train-test split, and $2.564 \cdot 10^{-3}$ for a 30%-70% split. This is for the unidirectional dataset of in total 3600 sentences, each belonging to one of 120 objects. This means that the probability that at least one of the 48 runs in an experiment contains a training set that does not have at least one sentence for each object is $3.783 \cdot 10^{-18}$ [1] and 0.116, respectively.

[1]About half the probability of finding one specific grain of sand on Earth.

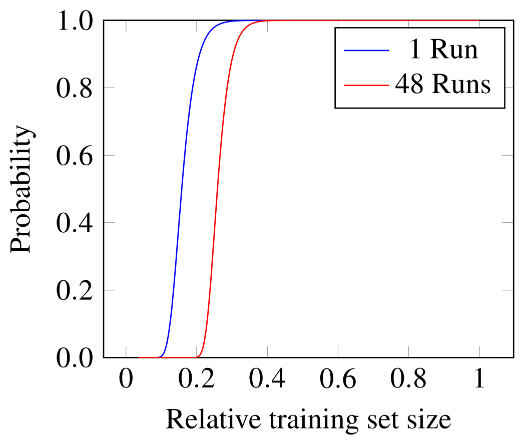


Figure F.1: Probability that the training set of an uncontrolled experiment contains at least one sentence for every object. One experiment consists of 48 runs.

Figure F.1 indicates the probability that one run and one experiment do not have a sentence for at least one object in its training set. This research advises to re-run the generalization experiment with a 20%-80% train-test split.

F.1 Derivation

The derivation of fig. F.1 was found with the help of the internet [69]. For completeness, the derivation is given here as well.

Each of the 3600 sentences belongs to exactly one object. Lets call every group of sentences belonging to one object O and label them $O_1,\ldots,O_{120}$. The training set is called $\mathcal{X}$ and the number of sentences in the training set is $|\mathcal{X}|$. The training set is sampled from the total dataset without replacement. We are interested in the probability that there is at least one sentence for every object in the training set, or

$$P(\bigcap_{i=1}^{120}[|\mathcal{X}\cup O_i|>0]) = 1-P(\bigcup_{i=1}^{120}[|\mathcal{X}\cap O_i|=0])$$

.

This union can be solved by the inclusion-exclusion principle, which states that for random events A and B, the probability of at least one happening can be calculated by $P(A\cup B)=P(A)+P(B)-P(A\cap B)$. The general form of the principle is:

$$P(A_j\cup\ldots\cup A_{j+k}) = P(\bigcup_{i=j}^{j+k}A_i) = \sum_{i=j}^{j+k}(-1)^{i+1}\sum_{I\subset\{j,\ldots,j+k\},|I|=i}P(\bigcap_{g\in I}A_g)$$

This means the following:

$$\begin{aligned}P(\bigcup_{i=1}^{120}[|\mathcal{X}\cap O_i|=0]) &= \sum_{i=1}^{120}(-1)^{i+1}\sum_{I\subset\{1,\ldots,120\},|I|=i}P(\bigcap_{g\in I}[|\mathcal{X}\cap O_g|=0]\\ &= \sum_{i=1}^{120}(-1)^{i+1}\binom{120}{i}P(\bigcap_{g\in\{1,\ldots,i\}}[|\mathcal{X}\cap O_g|=0])\end{aligned}$$

The last step can be made because every sentence has an equal probability of being included, so for every I with equal size, the probability of their intersection is equal. Also, are $\binom{120}{i}$ possible sets of I with size $|I|=i$ to be made from the total set $\{1,\ldots,120\}$.

As all sets O_i are fully disjoint and each contains 30 sentences, the probability of the intersection is found as follows:

$$\begin{aligned}P(\bigcap_{g\in\{1,\ldots,m\}}[|\mathcal{X}\cap O_g|=0]) &= P(\mathcal{X}\subset(O_{m+1}\cup\ldots\cup O_{120}))\\ &= \frac{\binom{30(120-m)}{|\mathcal{X}|}}{\binom{3600}{|\mathcal{X}|}}\end{aligned}$$

Thus, the probability that the training set does not contain any sentence for at least one object can be found with:

$$P(\bigcap_{i=1}^{120}[|\mathcal{X} \cap O_i| > 0]) = 1 - \binom{3600}{|\mathcal{X}|}^{-1} \sum_{i=1}^{120}(-1)^{i+1}\binom{120}{i}\binom{30(120-i)}{|\mathcal{X}|}$$

.

This formula is visualized in the blue line in fig. F.1.

Appendix G

Environments

G.1 Unidirectional Environment

The unidirectional environment consists solely of the environment visualized in fig. G.1.

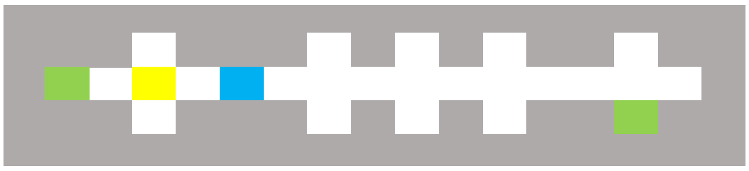
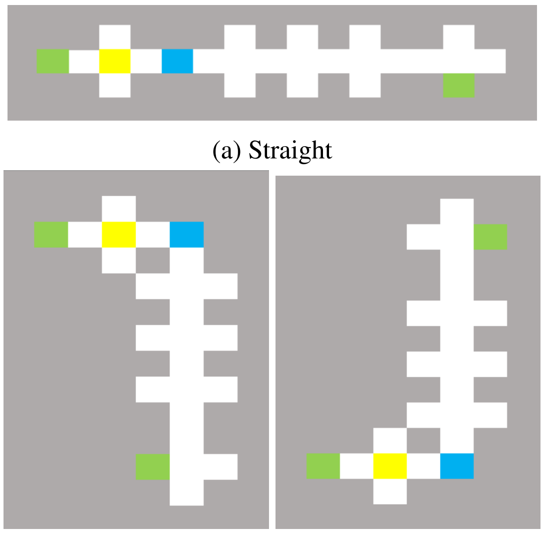


Figure G.1: The left unidirectional environment.

G.2 Left Multidirectional Environment

The multidirectional environment where the object is always to the *left* of the agent's starting position, is visualized in fig. G.2. All three instances of the environment are equally likely to occur. In this case, the agent does not have to pay attention to the directionality indication in the sentence.



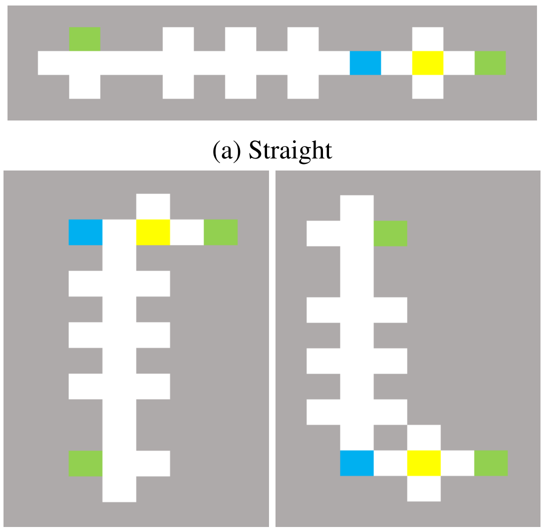
(a) Straight

(b) Downward (c) Upward

Figure G.2: Instances of the multidirectional environment with the object to the left of the agent's starting position.

G.3 Left & Right Multidirectional Environment

This environment includes instances of fig. G.2 and fig. G.3, and as such the object can be to the left or to the right of the agent's starting position.



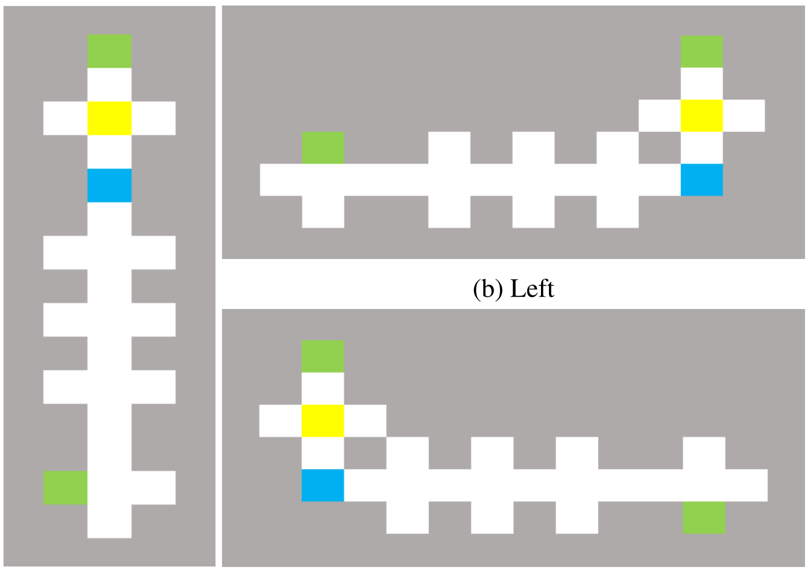
(a) Straight

(b) Downward (c) Upward

Figure G.3: Instances of the multidirectional environment with the object to the right of the agent's starting position.

G.4 All Multidirectional Environment

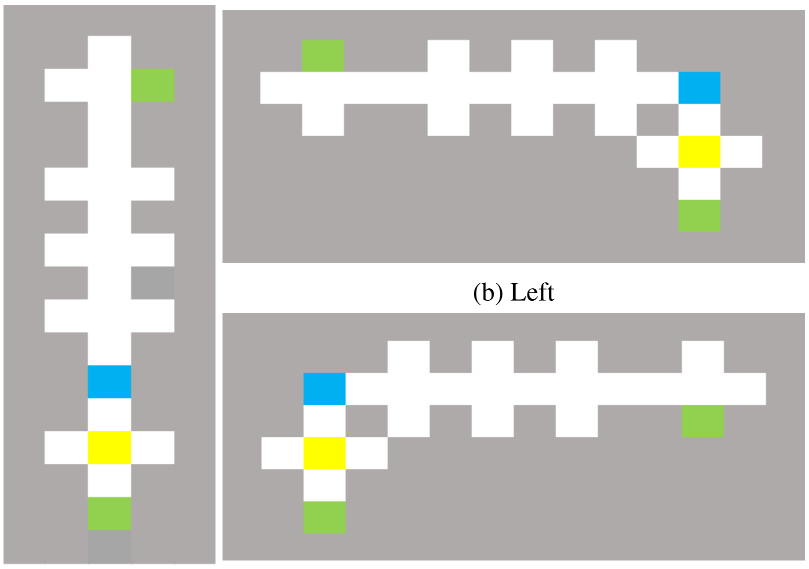
The *All* multidirectional environment consists of the situations where the object might be to the agent's starting position's left, right, up, or down. That means it consists of all the instances of fig. G.2, fig. G.3, fig. G.4, and fig. G.5 with equal probability.



(b) Left

(a) Straight (c) Right

Figure G.4: Instances of the multidirectional environment with the object in the upward direction of the agent's starting position.



(b) Left

(a) Straight

(c) Right

Figure G.5: Instances of the multidirectional environment with the object in the downward direction of the agent's starting position.