Answering Causal Questions with Reinforcement Learning

Lukas Blübaum

July 1, 2024
Version: Final

Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
DICE Research Group

Master's Thesis

Answering Causal Questions with Reinforcement Learning

Lukas Blübaum

1. Reviewer Prof. Dr. Axel-Cyrille Ngonga Ngomo
Department of Computer Science
Paderborn University

2. Reviewer Jun.-Prof. Dr. Sebastian Peitz
Department of Computer Science
Paderborn University

Supervisor Dr. Stefan Heindorf

July 1, 2024

Lukas Blübaum
Answering Causal Questions with Reinforcement Learning
Master's Thesis, July 1, 2024
Reviewers: Prof. Dr. Axel-Cyrille Ngonga Ngomo and Jun.-Prof. Dr. Sebastian Peitz
Supervisor: Dr. Stefan Heindorf

Paderborn University
DICE Research Group
Department of Computer Science
Faculty for Computer Science, Electrical Engineering and Mathematics
Warburger Straße 100
33098 Paderborn

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Abstract

Causal questions seek to determine whether there exists a causal relationship between different events or phenomena. Specifically, they often aim to understand the underlying causes of an effect or what kind of effects a particular cause might have in the future. Causal questions are important for various use cases, for example, in virtual assistants or search engines. However, many current approaches to causal question answering cannot provide explanations or verifications for their answers. Additionally, the approaches were often missing large-scale datasets of causal relations. In this thesis, we explore the application of reinforcement learning on CauseNet, a large-scale causal knowledge graph, to answer binary causal questions inspired by the successful application of reinforcement learning in knowledge graph tasks, such as link prediction and fact-checking. We introduce an Actor-Critic based agent which learns to search through the graph to answer binary causal questions. Additionally, we bootstrap the agent with a supervised learning procedure and adapt an existing reward shaping strategy to deal with large action spaces and sparse rewards. Our evaluation shows that our agent successfully prunes the search to answer binary causal questions efficiently. Moreover, our ablation study indicates that the supervised learning strategy provides a strong foundation upon which the agent can improve. Finally, we demonstrate how the paths learned by the agent can be used to explain causal relations, to indicate the specific mechanisms by which a cause produces an effect.

Contents

1 Introduction

Causal question answering addresses the problem of determining the causal relations between given causes and effects [Kay+20; DAB21]. This could involve examining whether a causal relation exists or how a causal relation can be explained. For example, causal questions like *"Does pneumonia cause anemia?"* ask for the validity of a causal relation. Instead, more complicated questions like *"How does pneumonia cause death?"* might ask for an explanation of a causal relation. Nowadays, the necessity to answer causal questions arises in various domains. For example, users often seek answers to causal questions from virtual assistants like Alexa or from search engines [Ngu+16b; Hei+20]. Similarly, causal questions and explanations for causal relations are important for argumentation [Wal07; Hab+18] and automated decision-making [Has+19; Kay+20; Hei+20]. When possible, tracing the causality chain that led to an answer can provide further insights and a deeper understanding of the causal relationship in question.

Therefore, the literature started to introduce approaches for causal question answering [Sha+16; Has+19; Kay+20; Dal21]. Some approaches focus on binary questions [Has+19; Kay+20], while others extend the setting to multiple-choice questions [Sha+16; Dal21]. However, a limitation of most of these approaches is the lack of explanations or verifiability of their answers. Additionally, they identified the problem of a lack of large, high-quality datasets for causal relations [Sha+16; Has+19; Kay+20]. Consequently, the authors resorted to creating their own datasets from sources like news articles or Wikipedia to support the identification of causal relations to answer causal questions. Thus, the introduction of CauseNet [Hei+20], a large-scale knowledge graph consisting of causal relations with context information, provides new opportunities to build effective causal question answering systems. Accordingly, we can apply various approaches from the knowledge graph literature that utilize the structure and rich semantics.

In particular, reinforcement learning was successfully applied to knowledge graphs on different tasks such as link prediction [XHW17], fact-checking [Das+18], or conversational question answering [KSW21]. In this thesis, we explore whether we can model the causal question answering task as a sequential decision problem similar to the aforementioned approaches. Our initial focus is on binary causal

questions, with the potential for extension to open-ended questions in future works. We train a reinforcement learning agent that learns to walk over CauseNet to find good inference paths to answer binary causal questions. The resulting reasoning capabilities are beneficial for questions whose answers require a chain of multiple hops [XHW17; Das+18; WD21].

We implemented the agent via the Synchronous Advantage Actor-Critic (A2C) algorithm [Mni+16] and used generalized advantage estimation (GAE) [Sch+16] to compute the advantage. To address the challenge of a large action space in CauseNet [Hei+20], we bootstrap the agent with a supervised learning procedure at the start [XHW17]. During the supervised learning, the agent receives expert demonstrations to understand what good paths look like. To handle sparse rewards during training, we enhance the agent with a reward shaping technique based on prior works [Yas+21]. Our evaluation demonstrates that the agent can effectively prune the search space, considering only a small number of nodes for each question. Additionally, our experiments confirm that the supervised learning phase establishes a strong foundation for the agent, decreasing the uncertainty in identifying good paths and accelerating the learning process. Moreover, we demonstrate how paths found by our agent can be used to explain the relations between a given cause and effect, including the option to take the original source into account [Hei+20]. Furthermore, we discuss straightforward options to extend our approach to different causal question types.

To summarize, our contributions are:

1. We introduce the first reinforcement learning approach for causal question answering on knowledge graphs.
2. We implemented an agent with the Synchronous Advantage Actor-Critic (A2C) algorithm, including a supervised learning procedure and a reward shaping technique to handle the challenges of the large action space and sparse rewards.
3. We extend an existing approach for extracting binary causal questions and introduce a new binary causal question dataset.

Overall, this thesis is structured as follows. First, we introduce related work in Chapter 2. Next, we present preliminaries discussing key methods and ideas our approach relies upon in Chapter 3. In Chapter 4, we introduce our approach in detail, and in Chapter 5, we discuss our implementation. Chapter 6 evaluates our approach, including an ablation study. Finally, we discuss some findings and future work in Chapter 7, followed by the conclusion in Chapter 8.

2 Related Work

In the following, we summarize related work. First, we present several knowledge graphs that focus on causal and commonsense knowledge. Afterward, we provide an overview of approaches for causal question answering. These approaches mainly employ pattern-based searches, embeddings, and language models. To the best of our knowledge, we are the first to use reinforcement learning for causal question answering. Lastly, we discuss approaches that apply reinforcement learning to reasoning and question answering tasks on knowledge graphs.

Causal Knowledge Graphs ConceptNet [SCH17] is a general knowledge graph consisting of 36 relations between natural language terms, including a *Causes* relation. CauseNet [Hei+20] and Cause Effect Graph [Li+20] specifically focus on causal relations extracted via linguistic patterns from web sources like Wikipedia and ClueWeb12.[1]

In contrast, ATOMIC [Sap+19] focuses on inferential knowledge of commonsense reasoning in everyday life. Specifically, ATOMIC consists of *"If-Event-Then-X"* relations based on social interactions or real-world events. ATOMIC$^{20}_{20}$ [Hwa+21] selects relations from ATOMIC and ConceptNet to create an improved graph while adding more relations via crowdsourcing. Instead, West et al. [Wes+21] automate the curation of causal relations by clever prompting of a language model. Finally, CSKG [ISZ21] builds a consolidated graph combining seven knowledge graphs, including ConceptNet and ATOMIC. For this thesis, we use CauseNet because most of the other knowledge graphs either focus on different relations types or are not limited to causal relations. However, future work may involve combining multiple knowledge graphs, which we discuss further in Chapter 7.

Causal Question Answering Currently, only a few approaches directly focus on the causal question answering task. Most of these approaches focus on binary questions, i.e., questions such as "*Does X cause Y?*" which expect a "yes" or "no" answer. Kayesh et al. [Kay+20] model the task as a transfer learning approach. Therefore, they extract cause-effect pairs from news articles via causal cue words.

[1] `https://lemurproject.org/clueweb12/`

Subsequently, they transform the pairs into sentences of the form "*X may cause Y*" and use them to finetune BERT [Dev+19]. Similarly, Hassanzadesh et al. [Has+19] employ large-scale text mining to answer binary causal questions. They introduce several unsupervised approaches ranging from simple string matching to embeddings computed via BERT. Each approach uses the text corpora to find evidence for a potential causal relation. Afterward, the evidence is used to compute a score that results in a "yes" or "no" answer depending on a threshold value.

Instead, Sharp et al. [Sha+16] consider multiple-choice questions of the form "*What causes X?*". First, they mine cause-effect pairs from Wikipedia via syntactic patterns and train an embedding model to capture the semantics between them. At inference time, they compute the embedding similarity between the question and each answer candidate. The approach by Dalal et al. [DAB21; Dal21] combines a language model with CauseNet [Hei+20]. Given a question, they apply string matching to extract relevant causal relations from CauseNet. Subsequently, they provide the question with the causal relations as additional context to a language model and let the language model answer the question.

Another approach that combines knowledge graphs and language models is QA-GNN [Yas+21]. Specifically, QA-GNN creates an augmented graph by scoring entities and relations with a pre-trained language model according to their relevance to the question. Then, QA-GNN applies a graph attention network (GAT) [Vel+18] on the augmented graph to answer the question. Likewise, we use their scoring function to apply the reinforcement learning agent on an augmented graph to provide a better reward signal, similar to the reward shaping strategies by prior reinforcement learning approaches on knowledge graphs [LSX18; Qiu+20].

Knowledge Graph Reasoning with Reinforcement Learning In recent years, reinforcement learning on knowledge graphs has been successfully applied in link prediction [Das+18], fact-checking [XHW17], or question answering [Qiu+20]. Given a source and a target entity, DeepPath [XHW17] learns to find paths between them. In the first step, DeepPath is trained in a supervised manner on paths found by BFS. Afterward, DeepPath applies REINFORCE [Wil92] policy gradients to improve the policy further. During inference time, the paths are used to predict links between entities or check the validity of triples. Subsequently, MINERVA [Das+18] improves on DeepPath by introducing an LSTM [HS97] into the policy network to account for the path history. Moreover, MINERVA does not require knowledge of the target entity and is trained end-to-end without supervision at the start. Lin et al. [LSX18] propose two improvements for MINERVA. First, they apply reward

shaping by scoring the paths with a pre-trained KG embedding model [Det+18] to reduce the problem of sparse rewards. Second, they introduce a technique called action dropout, which randomly disables edges at each step. Action dropout serves as additional regularization and helps the agent learn diverse paths.

Contrary to these approaches, M-Walk [She+18] does not rely on REINFORCE, but instead looks at the problem from a model-based viewpoint. Like AlphaZero [Sil+18], M-Walk applies Monte Carlo Tree Search (MCTS) as a policy improvement operator. Thus, at each step, M-Walk applies MCTS to produce trajectories of an improved policy and subsequently trains the current policy to imitate the improved one. GaussianPath [WD21] takes a Bayesian view of the problem and represents each entity by a gaussian distribution to better model uncertainty.

While all these approaches focus on link prediction or fact-checking, there has been some work on natural language question answering. Qiu et al. [Qiu+20] introduce the Stepwise Reasoning Network (SRN). They observe that for questions requiring multiple hops, different parts of the question should be important at each time step. Therefore, they introduce an attention mechanism into the policy network to enable the agent to attend to different parts of the question at every step. In turn, this should help the agent to focus on the aspect of the question that is important for the current decision. Additionally, SRN utilizes extra reward signals by considering the semantic similarities between the question and the action history via a potential-based reward function [NHR99]. CONQUER [KSW21] employs REINFORCE policy gradients for conversational question answering. Multiple agents search through the graph in parallel and aggregate their answers at the end. Importantly, the reward depends on user feedback. A positive reward corresponds to a new question and a negative reward to a reformulation of the last question.

3 Preliminaries

To start with, we provide an overview of the key concepts and methods our approach relies on. First, we discuss the causal knowledge graph CauseNet [Hei+20] in more detail. Next, we examine different types of causal questions and discuss the process of curating our causal question dataset. Finally, we provide a brief introduction to reinforcement learning, including the REINFORCE [Wil92] and Synchronous Advantage Actor-Critic (A2C) [Mni+16] algorithms.

3.1 CauseNet

CauseNet [Hei+20] is a large-scale causal knowledge graph of causal relations. Specifically, CauseNet contains *claimed* causal relations because they were extracted from web sources like Wikipedia and ClueWeb12[1] that are not necessarily true. Formally, CauseNet is defined as $\mathcal{K} = (\mathcal{E}, \mathcal{R})$, where $\mathcal{E}$ represents the set of entities and $\mathcal{R}$ is the set of relations. The entities in CauseNet are single words or noun phrases, while the set of relations contains only the *mayCause* relation, i.e., $\mathcal{R} = \{mayCause\}$.[2] Figure 4.1 shows an excerpt from CauseNet. It shows the entity *pneumonia* together with some of its causes and effects. The causal relations are stored as triples, e.g., $(pneumonia, mayCause, sepsis) \in \mathcal{K}$ where $pneumonia, sepsis \in \mathcal{E}$. Moreover, CauseNet contains additional meta-information for each relation. Included are the original source, i.e., the URL of the web page and the original sentence, if applicable.

CauseNet has two configurations, a high-precision one (CauseNet-Precision) and a high-recall one (CauseNet-Full). The statistics of both graphs can be seen in Table 3.1. CauseNet-Precision contains around 2% of the original relations, increasing the precision from 83% to 96%. The precision was evaluated by randomly selecting a number of relations from the graphs and manually evaluating them for correctness.

[1] `https://lemurproject.org/clueweb12/`
[2] In this thesis we will use *cause* when talking about the relations.

Tab. 3.1: The statistics of CauseNet-Full and CauseNet-Precision, including their precision, the number of entities, and the number of relations. The numbers were taken from the paper [Hei+20].

Graph	Precision	|Concepts/Entities|	|Relations|
CauseNet-Full	83%	12,186,31	11,609,89
CauseNet-Precision	96%	80,223	197,806

3.2 Causal Questions

Causal questions seek to determine whether there exists a causal relationship between different events or phenomena. Often, their goal is to understand why something happened or to predict what kind of effects a certain cause might have in the future.

Generally, we can differentiate between different types of causal questions. Starting with simple binary questions that ask for the validity of a causal relation, such as *"Does smoking cause cancer?"*. Another type are Cause-Effect questions which seek to identify the cause of a given effect or vice versa. An example of this type of question is *"What causes the rise in global temperatures?"*. Additionally, there are more complex questions that investigate the relationship between multiple entities. For example, a question like *"How does smoking cause cancer?"* examines the specific mechanisms by which the cause produces the effect. For the first version of our approach, we only consider binary causal questions like many prior approaches [Has+19; Kay+20]. However, we discuss a simple extension to Cause-Effect questions in Chapter 7 for future work.

To construct a dataset of binary causal questions, we extracted them from the Webis-CausalQA-22 corpus [Bon+22].[3] CausalQA is a corpus for causal question answering containing around 1.1 million causal questions. The corpus was constructed by extracting causal questions from ten well-known question answering datasets, such as SQuAD v2.0 [RJL18] and MS MARCO [Ngu+16a]. For a full list of the datasets, including their references, see Table 3.3. The table also shows the number of total causal questions and the number of binary causal questions that we extracted for each dataset. After the extraction, only MS MARCO contains a sufficient number of binary causal questions for use in our experiments. The scarcity of binary causal questions in the datasets may be due to a number of reasons. One factor is that many datasets, such as NewsQA [Tri+17] and HotpotQA [Yan+18], have a limited

[3] `https://zenodo.org/record/7476615`

Tab. 3.2: A few binary causal questions that we extracted from the datasets. The cause of each question is colored in green, and the effect in blue.

Question	Answer
Can alcohol cause diabetes?	Yes
Will hot weather affect blood sugar?	Yes
Is malaria caused by a fungus?	No
Does depo provera cause cancer?	No

number of causal questions to begin with. Additionally, some larger datasets like ELI5 [Fan+19] are focused on open-ended question answering.

For the extraction of binary causal questions, we built on work by Heindorf et al. [Hei+20] from CauseNet. For their evaluation, they implemented an extraction mechanism for binary causal questions. Specifically, the questions were extracted via patterns of the following form:

```
[question word] [cause/effect] [causal cue word] [cause/effect] ?
```

where the `[question word]` placeholder either represents one of the question words from Table 3.4 or is empty. The `[causal cue word]` placeholder represents words that are good indicators for causal relations together with their appropriate prepositions. Therefore, they should identify whether a question is causal or not. The original approach only considered *cause* in different verb forms, e.g., infinitive, past, or progressive. We extended this to consider a greater number of causal cue words. Specifically, we used the collection from Girju et al. [GM02]. Girju et al. curated a collection of causal cue words and ranked them by their frequency and ambiguity, i.e., how often they appear in text and how often they refer to a causal relation. Among these, we selected the ones that were ranked with high frequency and low ambiguity. Table 3.5 shows the full list of 23 words.

Moreover, the `[cause/effect]` placeholder represents causal concepts, where one takes the role of the cause and the other the role of the effect. The order depends on the question word and the causal cue word. Like Heindorf et al. [Hei+20], we place a few restrictions on the questions to keep the causal concepts simpler. This should increase the probability that the concepts can be found in CauseNet because we can only use questions where both the cause and effect can be found. The restrictions are enforced by filtering questions on a number of POS-Tags from the Stanford CoreNLP [Man+14]. The full list is shown in Table 3.4 on the right. Among others, we disallow coordinating conjunctions or subordinating conjunctions. Finally, we check whether the questions are answered with “yes” or “no” and remove

Tab. 3.3: The ten datasets that are part of CausalQA [Bon+22]. The *Total* column shows the total number of causal questions for each dataset, while the *Binary* column shows the number of binary causal questions. Note that the test sets are either not publically available or are missing the answers.

Dataset	Total		Binary		Reference
	|Train|	|Valid|	|Train|	|Valid|	
PAQ	692,645	76,961	1	–	[Lew+21]
GooAQ	146,253	33	167	–	[Kha+21]
MS MARCO	41,764	2,558	2,410	263	[Ngu+16a]
Natural Questions	2,796	71	–	–	[Kwi+19]
ELI5	131,035	4,834	2	–	[Fan+19]
SearchQA	663	117	–	–	[Dun+17]
SQuAD v2.0	4,342	483	–	–	[RJL18]
NewsQA	1,303	62	–	–	[Tri+17]
HotpotQA	355	35	–	–	[Yan+18]
TriviaQA	637	66	–	–	[Jos+17]

Tab. 3.4: The left table shows the question words we used for the binary question extraction. The right table shows the forbidden POS tags for the binary question extraction.

Question Words		POS-Tag	Description
is	do	CC	Coordinating conjunction
can	does	IN	Preposition or subordinating conjunction
might	did	TO	To-prepositions
would	will	WDT	Wh-determiner
could	are	WP	Wh-pronoun
may		WRB	Wh-adverb

any further explanation. Table 3.2 shows a few questions which we extracted from the datasets.

To summarize, the extraction focuses on binary causal questions that contain exactly one cause and one effect identified by one of 23 causal cue words.

3.3 Reinforcement Learning

In the following, we introduce some preliminaries regarding reinforcement learning, including the REINFORCE [Wil92] and Synchronous Advantage Actor-Critic (A2C) [Mni+16] algorithms. As done by related work [XHW17; Das+18; KSW21], we focused on policy gradient methods [SB18] because they can better deal with

Tab. 3.5: The causal cue words we used to detect binary causal questions. The list was curated by Girju et al. [GM02] to include words that are good indicators for causal relations, meaning they should have low ambiguity and high frequency.

Causal Cue Words			
induce	provoke	relate (to)	trigger off
give rise (to)	arouse	link (to)	bring on
produce	elicit	stem (from)	result (from)
generate	lead (to)	originate	trigger
effect	derive (from)	bring forth	cause
bring about	associate (with)	lead up	

the large action space in knowledge graphs compared to value function-based methods [XHW17].

In the reinforcement learning setting, an agent interacts with an environment to maximize a reward depending on some specified goal, e.g., moving a robot [Pen+18]. Formally, we define a Markov Decision Process (MDP) [SB18] as a 4-tuple $(\mathcal{S}, \mathcal{A}, \delta, \mathcal{R})$. The MDP consists of the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition function $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and the reward function $\mathcal{R} : \mathcal{S} \to \mathbb{R}$. When selecting an action a_t in state s_t, the environment changes from state s_t to the next state s_{t+1} via the transition function, i.e., $\delta(s_t, a_t) = s_{t+1}$. Generally, the transition function can be probabilistic or deterministic. In this thesis, we consider deterministic transition functions because the graph is static and completely defines the transition function [She+18]. Therefore, for each action a_t in state s_t, the next state s_{t+1} is known. Additionally, the environment gives a reward $\mathcal{R}(s_{t+1}) = r_t$ at each time step t.

The agent is represented by a policy network $\pi_\theta(a_t|s_t)$ parametrized by θ. The policy network computes a probability distribution over actions a_t at state s_t at each time step t.

Figure 3.1 demonstrates the interaction between the agent and the environment:

1. In state s_t, the agent selects an action a_t via the policy network and applies the action to the environment.

2. The environment evolves via $\delta(s_t, a_t)$ to s_{t+1} and gives a scalar reward r_{t+1}.

3. The agent receives state s_{t+1} and reward r_{t+1}, and the next iteration starts.

The process is continued until the maximum number of steps T is reached, or the agent reaches a terminal state, e.g., the end of a game.

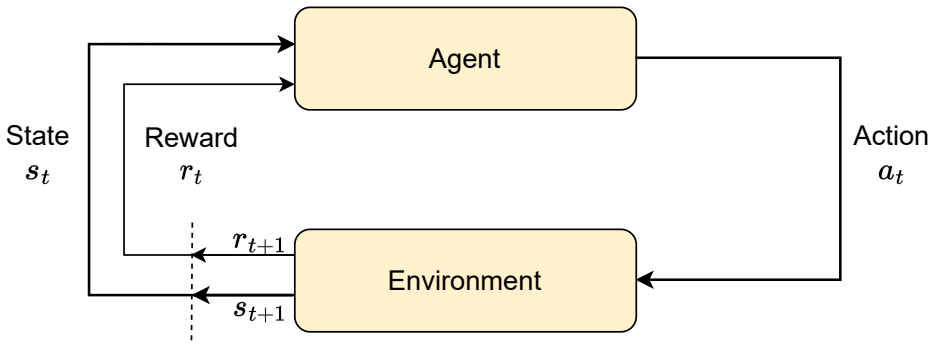

Fig. 3.1: The figure shows the general reinforcement learning setting, where an agent interacts with an environment. The agent applies an action on the environment at each time step and receives the next state and a reward. The figure was taken from Sutton et al. [SB18].

Overall, the agent is supposed to maximize the expected return [Sch+16; Pen+18]:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t \, r_t\right] \tag{3.1}$$

where r_t are the rewards from the environment for each time step t and the expectation is taken over episodes sampled from the current policy $\pi_\theta(a_t|s_t)$. The parameter γ is the discount factor between 0 and 1, which is used to weigh rewards from later time steps less. In the general setting, the episode length T can be infinite. However, we only consider finite episodes up to a maximum length of T as described in Section 4.4.[4]

For many different policy gradient methods, the expected return is optimized via the following gradient [Sch+16]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \, \log(\pi_\theta(a_t|s_t))\Psi_t\right] \tag{3.2}$$

where Ψ_t is a value estimating the returns or advantages as described below, and the expectation is again taken over episodes sampled from the current policy $\pi_\theta(a_t|s_t)$. As usual, the gradient steps are done in batches of episodes via gradient descent, as shown in Section 4.4.

[4] In the infinite setting, the discount factor γ is also used to keep the expected return finite [Pen+18].

The value Ψ_t can take several forms [Sch+16]. In the following, we introduce three of them, the Monte-Carlo return, the advantage, and GAE [Sch+16].

Monte-Carlo Return The first option is the Monte-Carlo return [Sch+16; Pen+18], also called the reward to go. The Monte-Carlo return computes the sum of rewards from step t onwards and is defined as:

$$R_t = \sum_{i=0}^{T-t} \gamma^i \, r_{t+i} \tag{3.3}$$

where r_t are the rewards from the environment at each time step t and γ is the discount factor. Setting $\Psi_t = R_t$ yields the REINFORCE [Wil92] algorithm. In this setup, R_t determines the direction of the gradient update in Equation 3.2. It should increase the probability of actions a_t that lead to positive rewards and decrease the probability of actions leading to negative rewards.

However, while the Monte-Carlo return is an unbiased estimate of the expected return, it has a very high variance because each reward r_t is a random variable that depends on randomness coming from the policy network $\pi_\theta(a_t|s_t)$ [Pen+18].[5] One possibility to reduce the variance while keeping the estimate unbiased is the substraction of a baseline b_t, such that $\Psi_t = R_t - b_t$ [Mni+16]. For example, b_t could be a moving average of the Monte-Carlo return [Das+18].

Advantage Another option to decrease the variance of the estimate is the introduction of a critic [SB18]. In the resulting Actor-Critic setting, we have our policy network $\pi_\theta(a_t|s_t)$ representing the actor and a value network $V_\psi(s_t)$ parametrized by ψ representing the critic. The value network $V_\psi(s_t)$ should predict the value of the state s_t. In the Synchronous Advantage Actor-Critic (A2C) algorithm [Mni+16], the value network is trained to predict the Monte-Carlo return R_t from Equation 3.3. Subsequently, the advantage is defined as:

$$\mathcal{A}_t^\psi = R_t - V_\psi(s_t) \tag{3.4}$$

A2C then sets $\Psi_t = \mathcal{A}_t^\psi$ and uses Equation 3.2 to update the policy network. In this case, the advantage $\mathcal{A}_t$ determines the direction of the gradient update. It increases or decreases the probability of actions a_t depending on whether they are better or worse than average [Sch+16]. Simultaneously, the value network is updated via the

[5] In case of probabilistic transition functions, there is also additional randomness coming from the environment.

mean-squared error between the predictions of the value network $V_\psi(s_t)$ and the Monte-Carlo return R_t.

Generalized Advantage Estimation (GAE) Alternatively, we can replace the advantage in the policy network update with the generalized advantage estimate (GAE) [Sch+16] to further reduce the variance and achieve more control over the trade-off between bias and variance. Similarly, we can replace the Monte-Carlo return in the value network update with the λ-return [SB18]. In the following, we introduce the GAE and λ-return following the explanation from Peng et al. [Pen+18].

Until now, the expected return is estimated via the unbiased Monte-Carlo return (Equation 3.3). As discussed above, this estimator has a very high variance. To address this problem, we can introduce n-step returns to decrease the variance. The n-step return does not compute the complete sum until time step T like the Monte-Carlo return. Instead, the sum is computed for n steps, and after that point, the estimate is bootstrapped via the value network $V_\psi(s_{t+n})$ [Sil15; Pen+18]:

$$R_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i \, r_{t+i} + \gamma^n V_\psi(s_{t+n}) \tag{3.5}$$

The disadvantage is that the bootstrap estimate via the value network introduces some bias. Hence, the parameter n can be used to control the trade-off between bias and variance. Specifically, high n yield a lower bias but higher variance, whereas small n yield a higher bias but lower variance. For example, setting $n = \infty$ gives us the Monte-Carlo return R_t from Equation 3.3, assuming that all rewards after the maximum episode length T are 0 [Pen+18].

However, now arises the question of which n we should choose. We could treat it as a hyperparameter and optimize the setting on some validation set. Instead, the literature introduced the λ-return [SB18] to deal with this problem. The λ-return does not take a specific n but rather computes the exponentially weighted average over all of them [Pen+18]:

$$R_t(\lambda) \overset{(1)}{=} (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \overset{(2)}{=} (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} \, R_t^{(n)} + \lambda^{T-t-1} R_t^{(T-t)} \tag{3.6}$$

where λ is a hyperparameter between 0 and 1, which can be used to control the trade-off between bias and variance. Equality (1) shows the general form of the λ-return up to infinity. Equality (2) holds if all rewards after time step T are 0 [Pen+18]. For more details, see the appendix of Peng et al. [Pen+18].

Finally, we set the advantage in Equation 3.4 to $\mathcal{A}_t^{\psi} = R_t(\lambda) - V_{\psi}(s_t)$, which corresponds to the GAE [Sch+16]. Moreover, we optimize the value network via the mean-squared error between the predictions of the value network $V_{\psi}(s_t)$ and the λ-returns $R_t(\lambda)$ [Pen+18], as shown in Section 4.4.

4 Approach

In the following, we describe our approach in detail. First, we provide the concrete problem definition. Second, we introduce the environment details and formulate the question-answering task as a sequential decision problem on a causal knowledge graph. Afterward, we present our reinforcement learning agent for causal question answering. This includes the network architecture, the training procedure, and the search strategy at inference time. Finally, we discuss extensions like bootstrapping with supervised learning and reward-shaping techniques.

4.1 Problem Definition

Given a causal question q in natural language and a causal knowledge graph $\mathcal{K} = (\mathcal{E}, \mathcal{R})$, the reinforcement learning agent walks over the graph to answer the question. Note that, the given causal knowledge graph only contains the *cause* relation such that $\mathcal{R} = \{cause\}$. In particular, the search should be as efficient as possible. In the ideal case, it should run in under half a second to give users an answer in a reasonable time frame [ABC14]. We consider binary causal questions q, where the agent must determine the validity of a causal relation. The curation of our binary causal question dataset was described in detail in Section 3.2. To reiterate, we consider binary causal questions like *"Can X cause Y?"* which contain exactly one cause and one effect.

In the following, we elucidate the binary causal question answering task on the knowledge graph $\mathcal{K}$ on the basis of the example in Figure 4.1. The example shows an excerpt of a causal knowledge graph (CauseNet) and the binary causal question *"Does pneumonia cause anemia?"*. In this question, *pneumonia* takes the role of the cause, and *anemia* the role of the effect.

First, the cause and effect are linked to the graph. Specifically, we find entities $e_c, e_e \in \mathcal{E}$ such that *pneumonia* maps to e_c and *anemia* to e_e. Currently, we link them via exact string matching. However, more sophisticated strategies can be considered in future work [KSW21]. Consequently, starting from e_c the agent has to find a path

Tab. 4.1: Eight cause-effect pairs randomly sampled from CauseNet. The table shows the number of paths of length two starting at the cause (|*Paths*|) compared to the number of paths between the cause and effect (|*Solutions*|), **not including** inverse edges.

Cause	**Effect**	**|Paths|**	**|Solutions|**
accident	death	18,967	173
heart failure	angina	3,533	5
pneumonia	dehydration	7,662	18
cancer	abnormalities	24,168	20
illness	suicidal behavior	25,226	3
stroke	pain	10,700	55
complications	confusion	12,839	19
infection	abdominal pain	28,955	53

Tab. 4.2: Eight cause-effect pairs randomly sampled from CauseNet. The table shows the number of paths of length two starting at the cause (|*Paths*|) compared to the number of paths between the cause and effect (|*Solutions*|), **including** inverse edges.

Cause	**Effect**	**|Paths|**	**|Solutions|**
accident	death	117,952	466
heart failure	angina	71,892	27
pneumonia	dehydration	66,715	84
cancer	abnormalities	160,000	82
illness	suicidal behavior	192,224	13
stroke	pain	109,305	255
complications	confusion	152,593	138
infection	abdominal pain	163,487	97

$(e_c, e_1, e_2, \ldots, e_e)$ with $e_i \in \mathcal{E}$, where the agent arrives at the effect e_e.[1] If the agent finds such a path, the question is answered with "yes" and with "no" otherwise. For the example, a possible path is *(pneumonia, sepsis, kidney failure, anemia)*. Afterward, we can inspect the path to get further insights into the relationship between cause and effect.

Challenges Contrary to previous approaches for reinforcement learning on knowledge graphs [XHW17; Das+18; Qiu+20], our causal knowledge graph only contains one relation type, i.e., $\mathcal{R} = \{cause\}$. Thus, the relations do not provide any learning signal, which makes the question answering task particularly challenging. Prior approaches, such as *DeepPath* [XHW17] or *MINERVA* [Das+18], used knowledge graphs with multiple relation types. This enabled them to use the relations types as

[1] The relations on the path were omitted, because the graph contains only one relation type.

actions at each time step. In our case, the action space consists of all entities $\mathcal{E}$ in the graph.

For example, we use CauseNet [Hei+20] to illustrate these challenges in more detail. CauseNet-Precision contains 80,223 entities (Table 3.1), where each entity corresponds to an action for our agent. In comparison, prior works considered graphs with only up to 237 relation types [Das+18] which they used as actions. Table 4.1 shows eight cause-effect pairs sampled from CauseNet. The $|\textit{Paths}|$ column shows the total number of paths of length two starting from the *cause*, while the $|\textit{Solution}|$ column shows the number of paths of length two between the *cause* and *effect*. Table 4.2 shows the same statistics with inverse edges included. These numbers further demonstrate the huge action space, with some examples having over 100,000 paths of length two, where each unique entity on these paths corresponds to a different action.

To address these challenges, we experiment with different techniques to improve the reinforcement learning agent. For example, via supervised learning to bootstrap the agent at the beginning of learning [XHW17], which we discuss in Section 4.6. This way, the agent receives correct paths at the start to help navigate the large action space.

4.2 Environment

As done by related work [XHW17; Das+18; Qiu+20], we formulate the causal question answering task as a sequential decision problem on the knowledge graph $\mathcal{K}$. The agent walks over the graph and decides which relation to take at each entity. Therefore, we define a Markov Decision Process (MDP) as a 4-tuple $(\mathcal{S}, \mathcal{A}, \delta, \mathcal{R})$. The MDP consists of the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition function $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and the reward function $\mathcal{R} : \mathcal{S} \to \mathbb{R}$. Hence, at each state $s_t \in \mathcal{S}$ the agent selects an action $a_t \in \mathcal{A}$, which changes the current state via $\delta(s_t, a_t)$ to s_{t+1}. Additionally, the agent receives a reward $\mathcal{R}(s_{t+1}) = r_t$. Note that the transition function δ is known and deterministic because the graph entirely defines δ. So, for each action a_t in state s_t, the next state s_{t+1} is known.

Agent Our agent consists of a policy network $\pi_\theta(a_t|s_t)$ (Actor) parametrized with θ and a value network $V_\psi(s_t)$ (Critic) parametrized with ψ. The policy network

$\pi_\theta(a_t|s_t)$ generates a distribution over actions a_t at the current state s_t. The value network $V_\psi(s_t)$ generates a scalar to estimate the value of the state s_t. More specifically, the value network should predict the future reward from state s_t onwards.

States At each time step t, we define the state $s_t = (\mathbf{q}, e_t, \mathbf{e_t}, \mathbf{h_t}, e_e) \in \mathcal{S}$, where $\mathbf{q}$ represents the embedding of the question q, $e_t \in \mathcal{E}$ the current entity, and $\mathbf{e_t}$ its embedding. The entity e_t is needed to define the action space, and its embedding $\mathbf{e_t}$ is used as input to the agent's networks. Additionally, $\mathbf{h_t}$ represents the path history of the agent and e_e the entity corresponding to the effect found in the question q. Moreover, $e_0 = e_c$ and $\mathbf{h_0} = \mathbf{0}$, where e_c is the entity corresponding to the cause of the question (e.g., *pneumonia* in the example in Figure 4.1). The path history is represented by the hidden states of an LSTM.

Actions The action space at each time step t consists of all neighboring entities of the current entity in state s_t. Therefore, the set of possible actions in state $s_t = (\mathbf{q}, e_t, \mathbf{e_t}, \mathbf{h_t}, e_e)$ is defined as $A(s_t) = \{e|(e_t, r, e) \in \mathcal{K}\}$ where $r = cause$. So only the current entity e_t is needed to define the action space $A(s_t)$. Note that while the additional components inside the state are not needed to define the action space, they are needed for other parts of the learning algorithm, as described below in Sections 4.4 and 4.3.

As described in Section 3.1, CauseNet [Hei+20] contains additional meta-information for each relation in the form of the original sentence s. Therefore, we include the original sentence s when computing the embedding $\mathbf{a_t}$ for an action a_t. This is done by concatenating the sentence embedding $\mathbf{s}$ with the embedding of the chosen entity $e = a_t$. Thus, the action embedding becomes $\mathbf{a_t} = [\mathbf{s}; \mathbf{e}]$.

As done by prior works [Das+18; LSX18; Qiu+20], we add a special *STAY* action at each step, so the action space becomes $A(s_t) = A(s_t) \cup \{STAY\}$. When selecting this action, the agent stays at the current entity. This way, we can keep all episodes to the same length, even though different questions might require a different number of hops. Another option would be to add a stop action. However, in that case, we would have episodes of different lengths.[2] Moreover, we add inverse edges to the graph because our experiments showed that their addition increases the performance. In general, inverse edges allow the agent to undo wrong decisions and to reach nodes

[2] In principle, episodes of different lengths are not a problem. However, keeping them to the same length simplifies the implementation. We chose this simplification because it worked well in prior works [Das+18; Qiu+20].

that could otherwise not be reached under a given episode length. We discuss some implications and tradeoffs regarding inverse edges in Chapter 7.

Transitions As described above, the transition function is deterministic, meaning the next state is fixed after the agent selects an action. Let $s_t = (\mathbf{q}, e_t, \mathbf{e_t}, \mathbf{h_t}, e_e)$ be the current state and $a_t \in A(s_t)$ be the action the agent selected in state s_t. Subsequently, the environment evolves via $\delta(s_t, a_t)$ to $s_{t+1} = (\mathbf{q}, e_{t+1}, \mathbf{e_{t+1}}, \mathbf{h_{t+1}}, e_e)$, where $e_{t+1} = a_t$.

Rewards In the default setup, the agent only receives a terminal reward at the final time step $T-1$. Specifically, the agent receives a reward of $\mathcal{R}(s_{T-1}) = 1$ if $s_{T-1} = (\mathbf{q}, e_{T-1}, \mathbf{e_{T-1}}, \mathbf{h_{T-1}}, e_e)$ with $e_{T-1} = e_e$. Conversely, the agent receives a reward of $\mathcal{R}(s_{T-1}) = 0$ if $e_{T-1} \neq e_e$. Similarly, for all other time steps, with $t < T-1$, the reward is 0 as well. This setup inhibits the typical sparse reward problem found in many reinforcement learning applications. Thus, we experimented with reward-shaping techniques similar to related work [LSX18; Yas+21]. For more details, see Section 4.7.

Path Rollouts — Episodes We define a path rollout or episode as a sequence of three tuples containing a state, action, and reward. Assuming a path rollout length of T, an example for a path rollout is: $((s_0, a_0, r_0), \ldots, (s_{T-2}, a_{T-2}, r_{T-2}))$. For path rollout length T, a path rollout contains $T-1$ tuples. This is because the last state s_{T-1} is only needed for the calculation of reward r_{T-2}, and no further action is taken.[3] For brevity, the rewards r_t can be omitted.

4.3 Network Architecture

We use a Long Short-Term Memory (LSTM) [HS97] to parametrize our agent. LSTMs were introduced to improve standard RNNs and can better handle the vanishing gradient problem [HS97]. Additionally, we experimented with a simple feedforward architecture but found that incorporating the path history is crucial for our needs. This aligns with previous research, where approaches such as MINERVA [Das+18] and SRN [Qiu+20] also used LSTMs and GRUs. Just CONQUER [KSW21] used a feedforward architecture, but they only considered paths of length one.

[3] See Algorithm 1 Lines 17-22 for more details.

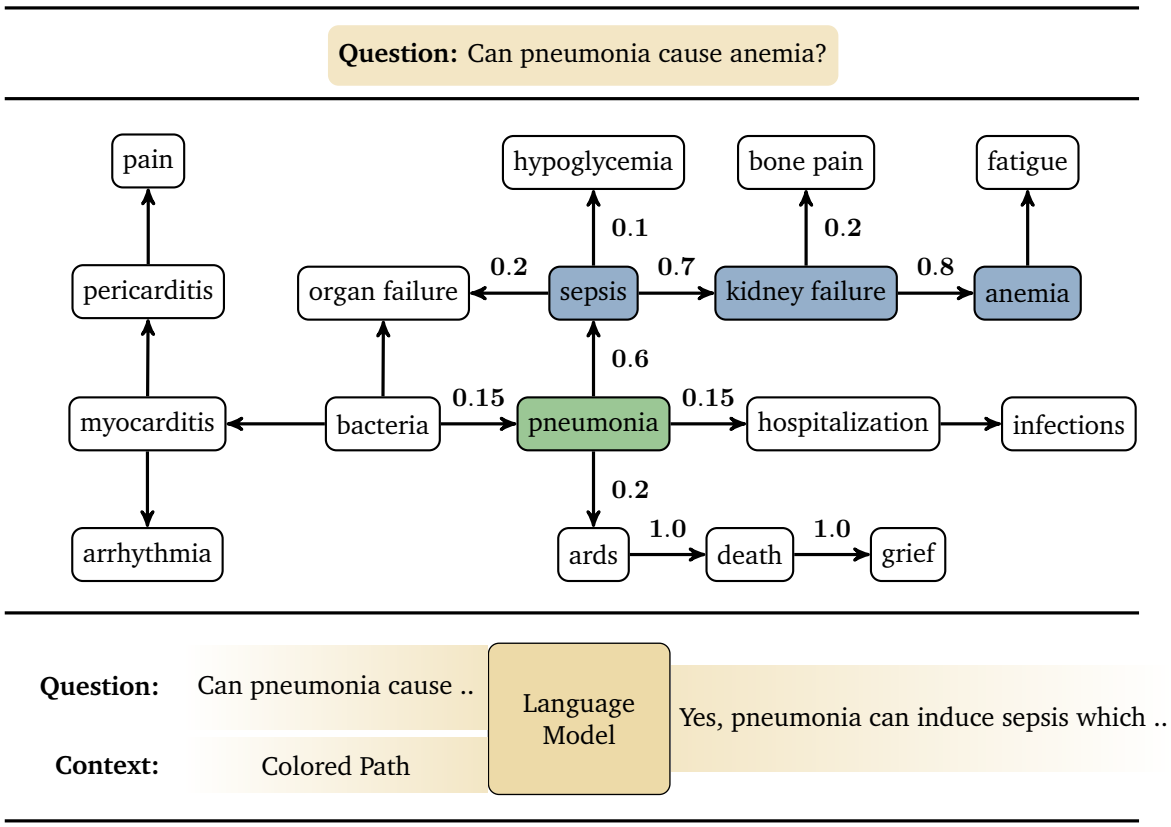

Fig. 4.1: An excerpt from CauseNet [Hei+20] showing the entity *pneumonia* together with its neighborhood containing causes and effects. Each edge depicts a *cause* relation. Given the question *"Can pneumonia cause anemia?"*, the search starts at the entity *pneumonia*. Following the path *(pneumonia, sepsis, kidney failure, anemia)*, the question can be answered with "yes". The numbers on the edges show the probability of taking this edge under the current policy $\pi_\theta(a_t|s_t)$. For brevity, we only show the relevant probabilities for the given path. The lower part of the figure shows the possibility to combine our agent with a language model. In that setup, we provide the paths the agent learned for a question as additional context to the language model. We briefly explore this setup in Section 6.5.

Let $\mathbf{q} \in \mathcal{R}^d$ be the embedding of the question q and $\mathbf{E} \in \mathcal{R}^{|\mathcal{E}| \times d}$ the embedding matrix containing the embeddings for each entity $e \in \mathcal{E}$ of the knowledge graph. The parameter d specifies the dimension of the embeddings. $\mathcal{K}$. The LSTM formula is then applied as follows:

$$\mathbf{h_t} = \begin{cases} LSTM(\mathbf{0}; [\mathbf{q}, \mathbf{e_c}]), & \text{if } t = 0 \\ LSTM(\mathbf{h_{t-1}}, [\mathbf{q}; \mathbf{e_t}]), & \text{otherwise} \end{cases} \tag{4.1}$$

where $\mathbf{h_t} \in \mathcal{R}^{2d}$ represents the hidden state vector (history) of the LSTM, and $[;]$ is the vector concatenation operator. At each time step, the LSTM takes the previous history $\mathbf{h_{t-1}}$ and the concatenation of the question embedding $\mathbf{q}$ and the current node embedding $\mathbf{e_t} \in \mathcal{R}^d$ to produce $\mathbf{h_t}$. In the first time step, $\mathbf{h_0}$ is initialized with the zero vector and $\mathbf{e_0} = \mathbf{e_c}$, where $\mathbf{e_c}$ is the embedding of the entity corresponding to the cause found in the current question.

On top of the LSTM, we stack two feedforward networks: one for the policy network $\pi_\theta(a_t|s_t)$ and one for the value network $V_\psi(s_t)$. In Section 4.2, we defined the action space $\mathcal{A}(s_t)$ at time step t and state $s_t = (\mathbf{q}, e_t, \mathbf{e_t}, \mathbf{h_t}, e_e)$ to contain all neighbors of the entity e_t. Therefore, we introduce an embedding matrix $\mathbf{A_t} \in \mathcal{R}^{|A(s_t)| \times 2d}$, where the rows contain the embeddings of the actions $a_t \in \mathcal{A}(s_t)$. Consequently, the output of the policy network $\pi_\theta(a_t|s_t)$ is computed as follows:

$$\begin{aligned} \pi_\theta(a_t|s_t) &= \sigma(\mathbf{A_t} \times W_2 \times ReLU(W_1 \times \mathbf{h_t})) \\ a_t &\sim Categorical(\pi_\theta(a_t|s_t)) \end{aligned} \tag{4.2}$$

where $W_1 \in \mathcal{R}^{h \times 2d}$ and $W_2 \in \mathcal{R}^{2d \times h}$ are weight matrices with hidden dimension h and σ is the softmax operator. The final output of the policy network is a categorical probability distribution over all actions $a_t \in \mathcal{A}(s_t)$. Similarly, the output of the value network $V_\psi(s_t)$ is computed with the following feedforward network:

$$V_\psi(s_t) = W_4 \times ReLU(W_3 \times \mathbf{h_t}) \tag{4.3}$$

where $W_3 \in \mathcal{R}^{h \times 2d}$ and $W_4 \in \mathcal{R}^{1 \times h}$ are weight matrices with hidden dimension h, and the output is a scalar that estimates the future reward from state s_t onwards. Overall, the weights of the LSTM are shared between the policy and value network, while each network has its own weights in the form of its feedforward head.

4.4 Training of the Reinforcement Learning Agent

In the following, we describe the training procedure of our reinforcement learning agent. This includes the pre-processing of the questions, the sampling of path rollouts, and the update rules for the weights of the agent. We start with the observation that CauseNet [Hei+20] does not contain negative information. Thus, we only train the agent on positive causal questions, i.e., questions whose answer is "yes". Similarly, we must remove all questions where the cause, effect, or both cannot be found in CauseNet. In that case, we are restricted by CauseNet, so there is nothing to learn.

Algorithm 1 displays the pseudocode of the whole training phase. The pseudocode assumes that only positive causal questions, where cause and effect can be found in CauseNet, remain in the given $questions$. First, we pre-process the questions by linking the cause and effect to the corresponding entities e_c and e_e in CauseNet. This is followed by the computation of embeddings for the question and entities.[4] Next, the weights θ and ψ of the agent are initialized. In the default setup, both are initialized randomly. In the case of a preceding supervised learning phase (Section 4.6), the weights of the policy network θ are initialized with the resulting weights from the supervised learning.

Afterward, we start sampling path rollouts from the environment via the current policy network $\pi_\theta(a_t|s_t)$. Each path rollout has the same length T. Hence, the agent should learn to use the *STAY* action in case it arrives at the target entity before a length of T is reached. Given a pre-processed question q, we construct the first state s_0. Subsequently, the agent interacts with the environment for T time steps. At each time step, the agent applies an action a_t and receives a reward r_t while the environment evolves via the transition function $\delta(s_t, a_t)$ to the next state s_{t+1}. This procedure is continued until a full batch of path rollouts is accumulated.

The training of the agent is facilitated via the Synchronous Advantage Actor-Critic (A2C) [Mni+16] algorithm, as described in Section 3.3. The policy network $\pi_\theta(a_t|s_t)$ takes the role of the actor while the value network $V_\theta(s_t)$ takes the role of the critic. We briefly experimented with Proximal Policy Optimization (PPO) [Sch+17] but found no significant performance improvements.

[4] We use GloVe [PSM14] embeddings to embed the questions and entities. For more details, see Section 6.4

Thus, the update rule for the policy network $\pi_\theta(a_t|s_t)$ looks as follows:

$$\nabla_\theta J(\theta) = -\frac{1}{B}\sum_{i}^{B}\sum_{t=0}^{T-2}\nabla_\theta \log(\pi_\theta(a_t|s_t))\,\mathcal{A}_t^\psi \tag{4.4}$$

where B is the batch size, T the path rollout length, and $\mathcal{A}_t^\psi$ the generalized advantage estimate (GAE) as described in Section 3.3.

As commonly done, we add an entropy regularization term to the objective [Das+18; KSW21]. The entropy regularization should help the agent with the exploitation vs. exploration tradeoff. Specifically, it should encourage exploration during training and stop the agent from getting stuck in local minima. Therefore, the resulting policy should be more robust and have a higher diversity of explored actions. We compute the average entropy of the action distribution of $\pi_\theta(a_t|s_t)$ over all actions $a_t \in \mathcal{A}(s_t)$ at each time step t and take the average over the whole batch:

$$H_{\pi_\theta} = \frac{1}{B(T-1)}\sum_{i}^{B}\sum_{t=0}^{T-2}(-\sum_{a_t\in\mathcal{A}(s_t)}\pi_\theta(a_t|s_t)\log\pi_\theta(a_t|s_t)) \tag{4.5}$$

The final update for the policy network becomes:

$$\theta = \theta - lr\cdot(\nabla_\theta J(\theta) + \beta H_{\pi_\theta}) \tag{4.6}$$

where lr is the learning rate and β is a hyperparameter that determines the weight of the entropy regularization term.

Simultaneously, we update the value network $V_\psi(s_t)$ via the mean-squared error between the λ-return and the predictions of the value network:

$$\nabla_\psi J(\psi) = \frac{1}{B(T-1)}\sum_{i}^{B}\sum_{t=0}^{T-2}\nabla_\psi(R_t(\lambda) - V_\psi(s_t))^2 \tag{4.7}$$

where $R_t(\lambda)$ is the λ-return as described in Section 3.3. Therefore, the final update for the value network becomes:

$$\psi = \psi - lr\cdot\nabla_\psi J(\psi) \tag{4.8}$$

where lr is the learning rate. In our experiments, we use the same learning rate lr for both networks. This is not compulsory and they could also use different learning rates. Overall, the training loop of selecting path rollouts and using them to update the policy and value networks is repeated for a number of optimization $steps$.

Input: Knowledge graph $\mathcal{K}$, questions $questions$, optimization steps $steps$, batch size B, path rollout length T, learning rate lr, entropy weight β, discount factor γ, GAE lambda λ
Output: Trained Agent θ, ψ
Function `TrainAgent`($\mathcal{K}$, $questions$, $steps$, B, T, lr, β, γ, λ)**:**
    $processed\_questions = [\,]$
    for ***each*** q ***in*** $questions$ **do**
        Link the cause and effect of q to entities e_c and e_e in $\mathcal{K}$
        Compute embeddings $\mathbf{q}$, $\mathbf{e_c}$ for q, e_c
        $processed\_questions.append((\mathbf{q}, e_c, \mathbf{e_c}, e_e))$
    end

    Initialize agent weights θ, ψ
    $path\_rollouts = [\,]$
    $step = 0$
    while *step* $<$ *steps* **do**
        $path\_rollout = [\,]$
        $(\mathbf{q}, e_c, \mathbf{e_c}, e_e) = SampleUniform(processed\_questions)$
        $s_0 = (\mathbf{q}, e_c, \mathbf{e_c}, \mathbf{0}, e_e)$
        for $t = 0$ ***to*** $T - 1$ **do**
            $a_t = SampleCategorical(\pi_\theta(a_t|s_t))$
            Receive state $s_{t+1} = \delta(s_t, a_t)$ and reward $r_t = \mathcal{R}(s_{t+1})$
            $path\_rollout.append((s_t, a_t, r_t))$
        end
        $path\_rollouts.append(path\_rollout)$

        if $|path\_rollouts| = B$ **then**
            Compute the GAE $\mathcal{A}_t^\psi$ and λ-returns $R_t(\lambda)$ for each episode in $path\_rollouts$ using λ, γ
            $policy\_update,\ value\_update = 0$
            for ***each*** $((s_0, a_0, r_0, \mathcal{A}_0^\psi, R_0(\lambda)), \ldots, (s_{T-2}, a_{T-2}, r_{T-2}, \mathcal{A}_{T-2}^\psi, R_{T-2}(\lambda)))$ ***in*** $path\_rollouts$ **do**
                $policy\_update \mathrel{+}= \sum_{t=0}^{T-2} \nabla_\theta \log \pi_\theta(a_t|s_t)\ \mathcal{A}_t^\psi$
                $value\_update \mathrel{+}= \sum_{t=0}^{T-2} \nabla_\psi (R_t(\lambda) - V_\psi(s_t))^2$
            end
            Compute the entropy regularization term H_{π_θ}
            $policy\_update = -\frac{policy\_update}{|path\_rollouts|},\ value\_update = \frac{value\_update}{|path\_rollouts| \cdot (T-1)}$
            $\theta = \theta - lr \cdot (policy\_update + \beta H_{\pi_\theta})$
            $\psi = \psi - lr \cdot value\_update$
            $path\_rollouts = [\,]$
            $step = step + 1$
        end
    end
return θ

Algorithm 1: Pseudocode of the training procedure for the policy network $\pi_\theta(a_t|s_t)$ and value network $V_\psi(s_t)$ as described in Section 4.4.

4.5 Search Strategy

At inference time, the agent receives both positive and negative questions. To answer a given question, we sample multiple paths p of length T from the agent. If any path contains the entity e_e, the agents answers the question with "yes" and "no" otherwise. In case the cause, effect, or both cannot be found in CauseNet, the question is answered with "no" per default.

For each path rollout $((s_0, a_0), (s_1, a_1), \ldots, (s_{T-2}, a_{T-2}))$, the path that was taken on the graph consists of the entity e_0 in s_0 and the actions taken at each time step t, i.e., $p = (e_0, e_1, \ldots, e_{T-1})$ where $a_{t-1} = e_t \in \mathcal{E}$ for $t > 0$.

The probability of path p can be formulated as follows:

$$\mathbb{P}(p) = \prod_{t=0}^{T-2} \pi_\theta(a_t|s_t) \tag{4.9}$$

where the probability of p is computed as the product of the probabilities of taking action a_t at state s_t under the current policy $\pi_\theta(a_t|s_t)$ for $t \in \{0, \ldots, T-2\}$. Note that, this formulation has a similar autoregressive nature as a language model objective [Ben+03]. The next action a_t depends on the current state s_t and implicitly, through the history $\mathbf{h_t}$, on all previous states. Thus, we use decoding methods from the language model literature to sample paths from the agent. Specifically, we consider greedy decoding and beam search, as done by prior works [Das+18; Qiu+20].

In the following, we illustrate the decoding methods on the basis of the example in Figure 4.1. The figure shows an excerpt from CauseNet, and each edge is annotated with the probability of taking this edge under the current policy. Given the question *"Can pneumonia cause anemia?"* the search starts at the entity *pneumonia*.

Greedy decoding takes the action with the highest probability at each time step, i.e., $arg\,max_{a_t \in \mathcal{A}(s_t)}\, \pi_\theta(a_t|s_t)$. So in the example, the agent would select *sepsis* in the first time step, *kidney failure* in the second time step, and *anemia* in the third time step. However, one disadvantage of greedy decoding is its myopic behavior. Therefore, greedy decoding might miss high-probability actions in later time steps. Beam search tries to alleviate this problem by always keeping a set of the best partial solutions up to the current timestep. The size of this set is a hyperparameter called *beam width*. In our case, partial solutions are paths of length t, where t is the current timestep. Furthermore, the paths are ranked by their probability, as defined in Equation 4.9.

Assuming a beam width of two, we demonstrate the decoding for the example in Figure 4.1. At each time step, we show the two top paths together with their probability:

- **Time Step 1**
  - **Path 1:** *(pneumonia, sepsis)* $\rightarrow 0.6$
  - **Path 2:** *(pneumonia, ards)* $\rightarrow 0.2$

- **Time Step 2**
  - **Path 1:** *(pneumonia, sepsis, kidney failure)* $\rightarrow 0.6 \cdot 0.7 = 0.42$
  - **Path 2:** *(pneumonia, ards, death)* $\rightarrow 0.2 \cdot 1.0 = 0.2$

- **Time Step 3**
  - **Path 1:** *(pneumonia, sepsis, kidney failure, anemia)* $\rightarrow 0.42 \cdot 0.8 = 0.336$
  - **Path 2:** *(pneumonia, ards, death, grief)* $\rightarrow 0.2 \cdot 1.0 = 0.2$

Finally, the agent checks whether one of the two paths found by beam search contains the effect. If that is the case, the agent answers the question with “yes” and “no” otherwise. In this example, *anemia* is found, so the agent answers with “yes”. In practice, the computations are done in log space to avoid numerical problems.

In our case, beam search can be viewed as an interpolation between greedy decoding and breadth-first search (BFS). As the beam width increases, the search strategy of the agent gets closer to an exhaustive search. This should increase performance but might make the task too easy if the beam width is set too high. However, at the same time, there are also runtime considerations. While a higher beam width increases performance, it will also increase the runtime. We further analyze these trade-offs in Sections 6.8 and 6.9.

4.6 Bootstrapping via Supervised Learning

Reinforcement learning algorithms often take a long time to converge due to their trial-and-error nature combined with large action spaces and sparse rewards [XHW17; LSX18]. Thus, the reinforcement learning agent can be bootstrapped, by first training it on a series of expert demonstrations. For example, AlphaGo [Sil+16] trained the agent on demonstrations from expert Go players before continuing with their reinforcement learning algorithm.

In our case, the expert demonstrations come from a breadth-first search (BFS) on CauseNet. This setup was first explored by DeepPath [XHW17]. Subsequent approaches like MINERVA [Das+18] and SRN [Qiu+20] discarded supervised learning in favor of several improvements. These improvements included the introduction of an LSTM in the policy network [Das+18] or the usage of attention mechanisms [Qiu+20]. Despite that, we decided to reintroduce supervised learning due to the lack of relation types and the resulting large action space in CauseNet. The original DeepPath [XHW17] implementation used a two-sided randomized BFS. Given entities e_1 and e_2, DeepPath selects a random node e_r. Next, DeepPath runs a BFS between $e_1 \rightarrow e_r$ and $e_r \rightarrow e_2$ and concatenates the found paths. This setup is supposed to help the agent to learn longer paths and not be biased toward shorter ones. However, in a pilot study, we found it unnecessary and used a standard BFS directly between e_1 and e_2.

Algorithm 2 shows the supervised training procedure. First, we randomly select a subset $\overline{Q}$ of size $\alpha \cdot |questions|$ of the training questions, where α is a hyperparameter. Subsequently, we run a BFS on the cause e_c and effect e_e of each question in $\overline{Q}$ and build a path rollout for each found path. If a path rollout is shorter than the path rollout length T, it is padded with the *STAY* action.[5] Next, we update the policy network $\pi_\theta(a_t|s_t)$ via the standard REINFORCE update as described in Section 3.3:

$$\nabla_\theta J(\theta) = -\frac{1}{B} \sum_{i}^{B} \sum_{t=0}^{T-2} \nabla_\theta \log(\pi_\theta(a_t|s_t))\, r_t + \beta H_{\pi_\theta} \tag{4.10}$$

where B is the batch size, T the path rollout length, and H_{π_θ} the entropy regularization from Section 4.4. During supervised training, the reward r_t is set to 1 at each step. Note that we only train the policy network $\pi_\theta(a_t|s_t)$ during supervised learning. We excluded the value network because we do not have negative examples in the current setup and found the current setup to work well enough.

[5] For simplicity, the pseudocode assumes that a path of length less than or equal to T can be found between e_c and e_e for each question q. In practice, the question q would have to be discarded if no path is found.

Input: Knowledge graph $\mathcal{K}$, questions $questions$, optimization steps $steps$, batch size B, path rollout length T, learning rate lr, supervised ratio α, entropy weight β
Output: Trained Agent θ
Function `SupervisedTrainAgent(`$\mathcal{K}$, $questions$, $steps$, B, T, lr, α, β`):`
    Sample $\lfloor \alpha \cdot |questions| \rfloor$ $\overline{Q}$ from $questions$
    $path\_rollouts = [\,]$
    for *each* q ***in*** $\overline{Q}$ **do**
        Link the cause and effect of q to entities e_c and e_e in $\mathcal{K}$
        `// Run BFS, note that` $e_0 = e_c$ `and` $e_n = e_e$
        $(e_0, e_1, \ldots, e_n) =$ `BFS(`$\mathcal{K}, e_c, e_e, T$`)`
        Compute embeddings $\mathbf{q}$ and $\mathbf{e_t}$ for question q and $0 \leq t \leq n-1$
        Build $path\_rollout$ $((s_0, a_0, r_0), \ldots, (s_{n-1}, a_{n-1}, r_{n-1}))$
            where $s_t = (\mathbf{q}, e_t, \mathbf{e_t}, \mathbf{0}, e_e)$, $a_t = e_{t+1}$, and $r_t = 1$ with $0 \leq t \leq n-1$
        if $n - 1 < T - 2$ **then**
            Pad the $path\_rollout$ with (s_t, a_t, r_t) where $s_t = (\mathbf{q}, e_{n-1}, \mathbf{e_{n-1}}, \mathbf{0}, e_e)$, $a_t = \mathit{STAY}$, and $r_t = 1$ for $n - 1 < t \leq T - 2$
        end
        $path\_rollouts.append(((s_0, a_0, r_0), \ldots, (s_{T-2}, a_{T-2}, r_{T-2})))$
    end
    $batches =$ `BuildBatches(`$path\_rollouts, B$`)`

    Initialize agent weights θ
    for $step = 0$ ***to*** $steps - 1$ **do**
        $batch = batches[step \,\%\, |batches|]$
        $policy\_update = 0$
        for *each* $((s_0, a_0, r_0), \ldots, (s_{T-2}, a_{T-2}, r_{T-2}))$ ***in*** $batch$ **do**
            $policy\_update \mathrel{+}= \sum_{t=0}^{T-2} \nabla_\theta \log \pi_\theta(a_t|s_t)\, r_t$
        end
        Compute the entropy regularization term H_{π_θ}
        $policy\_update = -\frac{policy\_update}{|batch|}$
        $\theta = \theta - lr \cdot (policy\_update + \beta H_{\pi_\theta})$
    end
return θ

Algorithm 2: The supervised learning algorithm that is used to bootstrap our agent. In the first step, we use a BFS to create expert demonstrations from the training questions. Afterward, we train the policy network $\pi_\theta(a_t|s_t)$ via the standard REINFORCE update using the expert demonstrations.

Afterward, we further train the policy value networks via Algorithm 1, as explained in Section 4.4.

4.7 Reward Shaping

Under the current setup, the agent only receives a reward at the final time step $T-1$ if it finds the target entity, such that $e_{T-1} = e_e$. Thus, in large action spaces, the agent might not receive any learning signal for a long time if it only rarely finds correct paths. To mitigate this problem, we introduced a supervised learning procedure in the previous section that provides a series of expert demonstrations to the agent. In the following, we explore a different direction by introducing auxiliary reward signals through a reward shaping technique.

Prior works for reinforcement learning on knowledge graphs already experimented with reward shaping techniques: (1) Lin et al. [LSX18] scored the last node with a knowledge graph embedding model if the path was not correct, (2) Qiu et al. [Qiu+20] scored each path by its cosine similarity with the question. Conversely, we experiment with a different technique introduced by Yasunaga et al. [Yas+21] for their QA-GNN model. Yasunaga et al. use a language model to score the entities on a knowledge graph according to their relevance to the question. Specifically, the score should represent how semantically close an entity is to the question. This is particularly suitable in our case because the entities in CauseNet also have labels in natural language.

Consequently, we adapt this approach to our task by scoring the last entity e_{T-1} on paths that were not successful. The idea is that the agent should still receive some indication of how "good" a path is if it does not lead to the target entity. Therefore, given a question q and entity e_{T-1} the score function is defined as follows [Yas+21]:

$$Score(q, e_{T-1}) = LM_{head}(LM_{enc}([q; e_{T-1}])) \tag{4.11}$$

where LM_{enc} is the encoder of the language model, LM_{head} is a two-layer feedforward network, and $[;]$ the concatenation operator.

The new reward function $\mathcal{R}'(s_{T-1})$ returns the default reward $\mathcal{R}(s_{T-1})$ if $e_{T-1} = e_e$ and $Score(q, e_{T-1})$ otherwise. Just like the default reward it is 0 at all other time

steps $t < T - 1$. We experimented with scoring each entity on a path with little success. Hence, given state s_{T-1} the new reward function is defined as [LSX18]:

$$\mathcal{R}'(s_{T-1}) = \mathcal{R}(s_{T-1}) + (1 - \mathcal{R}(s_{T-1})) \cdot Score(q, e_{T-1}) \cdot \omega \tag{4.12}$$

where $\mathcal{R}(s_t)$ is the default reward defined in Section 4.2 and ω is a hyperparameter. Generally, the scores of the language model are not on the same scale as the reward $\mathcal{R}(s_t)$. Thus, ω is used to determine the weight of the score and scale it appropriately so that the auxiliary reward does not dominate over the default reward.

The addition of reward shaping only requires minor changes in the environment setup. The states s_t currently only contain the question embeddings $\mathbf{q}$. For the new reward function to receive the necessary information, we must add the question q itself to each state, i.e., $s_t = (q, \mathbf{q}, e_t, \mathbf{e_t}, \mathbf{h_t}, e_e)$. Additionally, we need to replace $\mathcal{R}(s_t)$ with $\mathcal{R}'(s_t)$ in Algorithm 1 Line 20. Reward shaping is not used during supervised learning because supervised learning only uses correct paths.

5 Implementation

The following chapter provides information about our implementation decisions. First, we briefly review implementations of related approaches. This is followed by a detailed description of our implementation, including an overview of the architecture used for training the agent and the interactions between the various components. The open-source implementation of our approach can be found at `https://github.com/LukasBluebaum/Master_Thesis`.

5.1 Frameworks

To begin with, we looked into the implementations of prior works for reinforcement learning on knowledge graphs. Most implementations were in Python [XHW17; Das+18; KSW21], with some exceptions in C++ [Zha+18] or even C# [She+18]. Comparing the usage of deep learning frameworks in the Python implementations, it was a mix between PyTorch [Pas+19] and TensorFlow [Mar+15]. Interestingly, most approaches did not use a specialized reinforcement learning framework but wrote their implementation from scratch.

Following CONQUER [KSW21], we first implemented a minimal working version using TF-Agents [Gua+18], a reinforcement learning framework that is natively integrated into TensorFlow. However, we realized the framework is a bit outdated and hard to extend. Therefore, we looked into other reinforcement learning frameworks like RLlib [Lia+18] and Stable-Baselines3 [Raf+21]. In fact, we encountered similar problems, as it would mean a lot of work to extend the frameworks to the appropriate knowledge graph setting for our needs. Thus, we also decided to write our implementation from scratch using Python and PyTorch, similar to most other approaches.

For the extraction and pre-processing of the binary causal questions, we used the NLP frameworks NLTK [BKL09] and Stanford CoreNLP [Man+14]. For logging, experiment tracking, and model archival, we used Weights & Biases [Bie20]. Additionally, we used Huggingface Transformers [Wol+20] for one of the baselines in our experiments.

5.2 Architecture

As described in Section 5.1, we wrote our implementation from scratch without any specialized reinforcement learning framework. Still, our implementation is influenced by the general structure of frameworks like Stable-Baselines3 [Raf+21] or TF-Agents [Gua+18].

Figure 5.1 shows the architecture of our implementation in the form of a UML Class Diagram. The *EmbeddingProvider* classes are simple wrappers around the embeddings and provide methods to retrieve embeddings for entities and relations.[1] These methods are called once by the *KnowledgeGraph* class with lists of entities and relations of the current graph. Note that we kept the implementations of the *EmbeddingProvider* and *KnowledgeGraph* classes more general to allow for the integration of graphs with multiple relation types. In the case of CauseNet, all methods dealing with *relations* will return the sentence from the metadata for the given entity pair. Moreover, for CauseNet, the *KnowledgeGraph* class stores the graph as an adjacency list together with an associative array that maps each entity pair to its sentence. Moreover, each entity is assigned an ID. Additionally, it provides methods to retrieve the embedding for a given entity, the embedding of the sentence of an entity pair, and the neighbors of an entity.

The *Environment* class handles the current state of the agent at each time step. Whenever the *reset* method is called, the environment selects the next question and positions the agent on the cause $e_c = e_0$ contained in the question. Afterward, it returns the embeddings for the state $[\mathbf{q}; \mathbf{e_0}]$ and the action embedding matrix $\mathbf{A_t}$ as described in Section 4.3.[2] Similarly, the *step* method receives an action at each time step t and evolves the environment accordingly. Next, it returns the embeddings $[\mathbf{q}; \mathbf{e_t}]$ and the embedding matrix $\mathbf{A_t}$ together with the reward r_t and a boolean value indicating whether the path rollout length T is reached. As commonly done, we use two *Environment* instances, one for training and one for evaluation [Gua+18; Lia+18; Raf+21]. The reason is that the behavior in reinforcement learning problems often differs between training and inference time. In our case, the behavior changes w.r.t. positive and negative causal questions (Section 4.4) and different decoding methods at inference time (Section 4.5).

[1] In our experiments, we used GloVe [PSM14] embeddings since they performed better than BERT embeddings [Liu+19]. Therefore, the *BERTEmbeddings* class is just provided for the sake of completeness.

[2] We assume that the embeddings for the question q were precomputed and are part of the question objects.

The *Agent* class consists of the policy network and value network. The value network returns a scalar predicting the value of the current state s_t, and the policy network returns a probability distribution over all actions $a_t \in A(s_t)$. The *Trainer* class does not use the value network if we only use the REINFORCE algorithm. The *Trainer* class optimizes the *Agent* according to the given parameters. During each training step, the *Trainer* applies the *Agent* on the *Environment* through the *reset* and *step* functions to sample path rollouts. Specifically, the *Trainer* provides the output of the *Environment* to the *Agent*, receives an action from the *Agent*, and applies it on the Environment via the *step* function. The path rollouts are stored in the *PathRolloutStorage* until a full batch is accumulated. Consequently, the *Trainer* performs a gradient step on the *Agent* as described in Section 4.4, and the next training iteration starts. The *Trainer* is an abstract class implemented by the *A2CTrainer* and the *REINFORCETrainer*, which run the A2C and REINFORCE algorithms described in Section 3.3 and Section 4.4.

Generally, the diagram should only provide a high-level overview. In particular, not all parameters or components are depicted. For example, the computation of rewards and advantages, including the reward shaping and the decoding methods at inference time, are hidden in the *Environment* and *Trainer* classes. Moreover, parameters like the supervised ratio α or the supervised batch size are also missing. The exact details can be seen in the code on our repository.

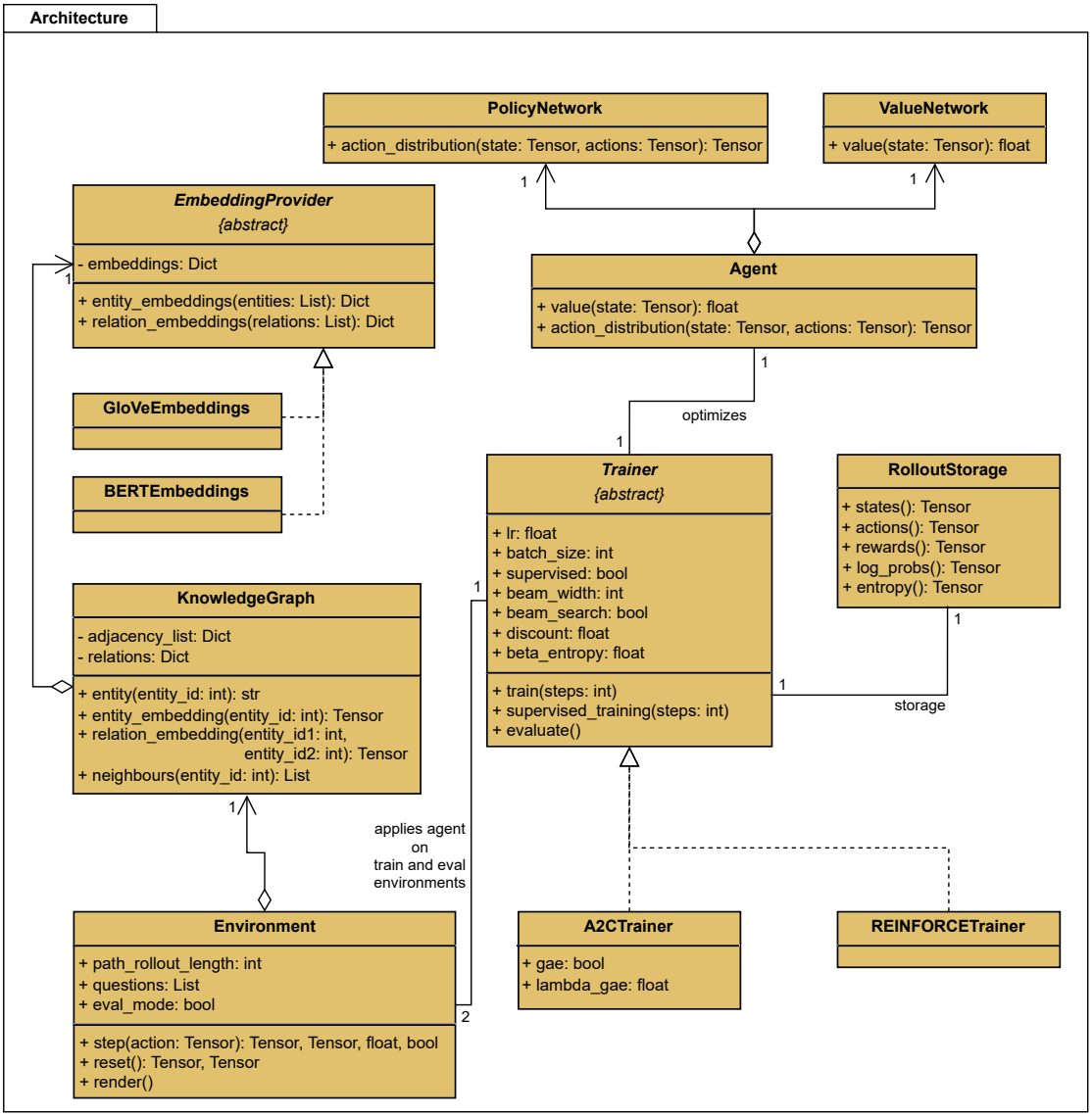

Fig. 5.1: UML-Diagram depicting the architecture of our implementation. It shows the most important classes, including the Trainer, Agent, and Environment, and how they interact with each other. For brevity, not all parameters or methods are shown. Therefore, this figure should only provide a high-level overview. For the complete picture, we provide an open-source implementation at:
`https://github.com/LukasBluebaum/Master_Thesis`

6 Evaluation

In this chapter, we present the evaluation of our reinforcement learning agent for causal question answering. To start, we provide an overview of the used datasets, discuss the baselines that we used for comparison, and introduce the applied evaluation measures. Following this, we discuss parameter settings and further implementation details like framework versions and the hardware used for training. Afterward, we compare our agent to two baselines on the binary causal question answering task. Next, we conduct an ablation analysis to evaluate the effectiveness of the different parts of our approach. Similarly, we showcase several experiments to analyze the impact of supervised learning and different decoding techniques. Finally, we conclude this section by providing a few example paths found by our agent.

6.1 Datasets

As shown in Section 3.2, only one of the ten datasets from CausalQA [Bon+22] contained a larger number of binary causal questions. Thus, we used the extracted questions from MS MARCO [Ngu+16b] as one dataset and added SemEval [Hen+10; Sha+16] as a second dataset. SemEval was curated by Sharp et al. [Sha+16] by selecting a subset of 1730 word pairs from the semantic relation classification benchmark SemEval 2010 Task 8 [Hen+10]. Among the 1730 word pairs, there are 865 causal pairs and 865 non-causal pairs.

Table 6.1 shows the number of training, validation, and test questions for both datasets. For SemEval, we randomly selected 10% for validation and 10% for testing. For MS MARCO, we used the original validation set for testing and randomly selected

Tab. 6.1: Number of questions for training, validation, and testing for the MS MARCO [Ngu+16b] and SemEval [Sha+16] datasets. The "|Effective Train|" column shows the number of questions the agent has available for learning.

Dataset	|Train|	|Validation|	|Test|	|Effective Train|
MS MARCO	2169	241	263	1350
SemEval	1384	173	173	812

10% from the training set for validation. We optimized hyperparameters on the validation sets and then retrained using the combined training and validation sets.

The "|Effective Train|" column shows the number of questions available for learning when combining the training and validation sets. As discussed in Section 4.4, we only train on positive causal questions, so we remove the negative causal questions during training. Additionally, we have to remove the questions from the training set where cause or effect cannot be found in CauseNet. Finally, that leaves us with 1350 questions for MS MARCO and 812 questions for SemEval.

6.2 Baselines

We compare our agent with two baselines: a breadth-first search (BFS) on CauseNet and the question answering system UnifiedQA-v2 [Kha+20; KKH22]. The CauseNet paper [Hei+20] already used a BFS to estimate the recall of the graph. While they only considered paths up to length two, we extended their implementation and added support for paths of arbitrary length. As done for our agent, we link the cause and effect of each question to the graph using exact string matching. Afterward, BFS tries to find a path between cause and effect to answer the question.

BFS serves as a strong baseline and an upper limit on performance in certain circumstances. However, it should be noted that BFS can only be applied to binary causal questions. Our agent only supports binary questions as proof of concept, but there are straightforward extensions to open-ended questions, as discussed in Chapter 7. For a given path length constrained, the BFS performs an exhaustive search and can represent an upper limit on performance, with one exception. The exception is the potential introduction of false positives through inverse edges and errors in CauseNet. For example, assume we have a binary causal question *"Does X cause Y?"* for which a causal relation only holds in the opposite direction, such that Y causes X. Through the introduction of inverse edges, the BFS might find a path between the two and erroneously answer "yes". Similarly, if CauseNet contains an error and indicates that X causes Y, the BFS will also provide an incorrect answer. While the BFS will always make these mistakes,[1] the agent prunes the search space and can learn to avoid them. We summarize the implications of inverse edges in Chapter 7.

[1] Assuming the false positives exist in CauseNet and can be found under the given path length constraint.

As a second baseline, we use UnifiedQA-v2 [Kha+20; KKH22]. UnifiedQA-v2 is a text-to-text language model based on the T5 architecture [Raf+20] and achieved state-of-the-art performance on multiple datasets. It was pre-trained on 20 question answering datasets of different formats, including extractive and abstractive question answering, multiple choice, and yes/no questions. We chose UnifiedQA-v2 because it was used by CausalQA [Bon+22] for their evaluation, from which we extracted the binary causal questions.

We could not include the causal question answering approaches from related work because they are not open source, except for the approach by Kayesh et al. [Kay+20]. However, Kayesh et al. only released the code for training without the datasets and pre-trained models, so it was not possible for us to use their approach. Similarly, the reinforcement learning-based approaches we encountered were either developed for different tasks [KSW21], had no available code [Qiu+20], or were designed for graphs with multiple relation types rather than the single relation type found in CauseNet [Das+18; LSX18].

6.3 Evaluation Measures

We evaluated our agent using standard binary classification measures: accuracy, F_1-score, precision, and recall. Precision represents the fraction of correct positive predictions, and recall represents the fraction of positive examples that were correctly predicted.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{6.1}$$

Subsequently, the F_1-Score represents the harmonic mean of precision and recall, while accuracy represents the fraction of correct predictions among all predictions.

$$F_1\text{-}Score = 2\frac{precision \cdot recall}{precision + recall} \tag{6.2}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6.3}$$

Note that, at inference time, the accuracy is equal to the normalized reward of the agent. During inference, the reward is always 0 or 1 since there is no cumulative reward calculation or reward shaping. Specifically, the agent receives a reward of

Tab. 6.2: The parameter configurations and the ranges we tested during hyperparameter optimization. They are divided into general deep learning parameters, general reinforcement learning parameters, and the parameters that are specific to our approach. This should not be a definite categorization, e.g., there are many other approaches which also use beam search, it is just provided to give a better overview.

Parameter	Settings	Settings Sweeps
General		
Learning Rate	1e-4	{1e-3, 1e-4, 3e-4, 5e-5}
Batch Size	128	{16, 32, 64, 128}
Gradient Norm Clipping	0.5	{0.5}
Hidden Dimension h	2048	{2048}
Seed	42	{42}
Training Steps	2000	{2000}
General Reinforcement Learning		
Discount γ	0.99	{0.9, 0.99, 1.0}
GAE lambda λ	0.95	{0.95, 0.99, 1.0}
Entropy Weight β	0.01	{0.01, 0.05, 0.1}
Specific to Our Approach		
Beam Width	50	{1, 5, 10, 50}
Path Rollout Length Training	3	{2, 3}
Path Rollout Length Evaluation	3	{2, 3}
Supervised Ratio α	0.8	{0.2, 0.4, 0.6, 0.8, 1.0}
Supervised Steps	300	{100, 200, 300}
Supervised Batch Size	64	{32, 64}
Reward Shaping Weight ω	0.0	{0.0, 0.1, 0.5, 1.0}

1 if the answer is correct, which is defined as the agent finding a path when the causal relationship holds or not finding a path when the causal relationship does not hold. Conversely, the agent receives a reward of 0 if the answer is incorrect. Thus, summing the rewards over all questions and dividing by the number of questions is equal to the accuracy.

6.4 Experimental Setup

In the following, we discuss the experimental setup for our agent, including the setup of the baselines, hyperparameter optimizations, and further implementation details. The agent is configured with supervised learning at the start and uses the A2C algorithm together with beam search during inference time. Furthermore, inverse edges are added to CauseNet. However, we did not include the reward

shaping in the general configuration since it did not increase performance. We further discuss the reward shaping in the ablation study in Section 6.6.

Table 6.2 presents the configurations for the experiments' parameters, including the ranges for hyperparameter sweeps. For each dataset, we optimized the parameters on the validation set and retrained on the combined training and validation set afterward. The table shows the configurations for MS MARCO, whereas SemEval differs in two cases: 1.0 for the supervised ratio α and 32 for the supervised batch size. For the hyperparameter ranges, we selected common values from the literature and values successfully used in prior works [Das+18; LSX18; Pen+18]. In the experiments below, we denote the agent with path rollout length 2 as Agent 2-Hop and path rollout length 3 as Agent 3-Hop. Following related work [Das+18; Qiu+20], we consider paths up to length 3. However, this can be extended in future work. Similarly, the BFS baseline is called CauseNet 1-Hop, CauseNet 2-Hop, or CauseNet 3-Hop.

For optimization, we used the AdamW [LH19] optimizer, with the PyTorch default settings of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay 0.01. Additionally, we apply gradient norm clipping [PMB13] with a value of 0.5. We initialized the weights of the LSTM via orthogonal initialization [SMG14] and the weights of the feedforward networks with the kaiming initialization [He+15]. Moreover, we used Python 3.8.16 and PyTorch 1.13.0 for our experiments. For the UnifiedQA-v2 [Kha+20; KKH22] baseline, we chose the *allenai/unifiedqa-v2-t5-base-1363200* checkpoint, which is included in the HuggingFace Transformers framework [Wol+20]. Furthermore, we used CauseNet-Precision [Hei+20] for our experiments due to its smaller action space and higher precision compared to CauseNet-Full. In future work, we will experiment with CauseNet-Full, especially with the performance tradeoffs between the precision and recall orientations of the two graphs.

To embed the states and actions, we used GloVe embeddings [PSM14] with a dimensionality of 300. When an entity contained more than one word, we used the average of their embeddings. Similarly, for the questions and sentences, we used the average of the embeddings of their words. We experimented with embeddings computed from RoBERTa [Liu+19] and MPNet [Son+20] but did not see performance improvements. One disadvantage of using GloVe embeddings might be that not all occurring words have a corresponding embedding. However, this occurs only for around 1% of the entities in CauseNet-Precision, and manual checks showed that these do not pose problems for the questions in our datasets. For the reward shaping, we slightly adapted the implementation of Yasunaga et al. [Yas+21] for

our task. As the language model, we used the *roberta-large* [Liu+19] checkpoint from the Huggingface Transformers framework [Wol+20].

All experiments were run on Noctua 2[2] on the Paderborn Center for Parallel Computing (PC^2)[3] and Google Colab. Accordingly, we used a number of different GPUs for our experiments, mainly NVIDIA A100 40GB, NVIDIA Tesla T4 16GB, and NVIDIA Tesla V100 16GB.

6.5 Evaluation of our Reinforcement Learning Agent

In Table 6.3, we show the evaluation results of our agent on the MS MARCO and SemEval test sets. We compare our agent in the Agent 2-Hop and Agent 3-Hop configurations with the corresponding BFS configurations CauseNet 1-Hop, CauseNet 2-Hop, and CauseNet 3-Hop. Moreover, we compare the results to UnifiedQA-v2 and explore some combinations of approaches as described below. The table displays the accuracy, F_1-Score, recall, precision, and the number of visited nodes summed over all questions for each dataset. Note that the number of visited nodes is not applicable for the UnifiedQA-v2-based approaches and the majority baseline.

When comparing the results of our agent and BFS, we observe that the agent does not fully match the performance of BFS in most configurations. For instance, in the 2-Hop case, the agent is slightly behind by around 0.03 accuracy on MS MARCO and around 0.05 accuracy on SemEval. However, the agent achieves better precision for all configurations, particularly for the 3-Hop configuration on SemEval. In that case, the results of BFS are worse in the 3-Hop setting compared to the 2-Hop setting due to the introduction of many false positives through inverse edges.[4] As discussed in 6.2, inverse edges can lead to mistakes by potentially introducing false positives. When going from 2 hops to 3 hops on SemEval, it is possible to reach more false positives than before. This is especially indicated by the reduced precision of CauseNet 3-Hop, i.e., the precision is now 0.75, down from 0.875 of CauseNet 2-Hop.

In contrast, the agent mitigates this problem by pruning the search space and avoiding paths leading to wrong answers, still achieving a precision of 0.929. Therefore,

[2] `https://pc2.uni-paderborn.de/de/hpc-services/available-systems/noctua2`
[3] `https://pc2.uni-paderborn.de/`
[4] The results of CauseNet 3-Hop on SemEval are improved when not using inverse edges. However, using inverse edges improves the results for all other configurations. Thus, we included them in the graph as they also improve the performance of our agent, as shown in the ablation study in Section 6.6.

Tab. 6.3: Evaluation results of our agent on the MS MARCO and SemEval test sets compared to the BFS baseline and UnifiedQA-v2. The table reports the accuracy: **A**, F_1-Score: $\mathbf{F_1}$, recall: **R**, precision: **P**, and the number of nodes **|Nodes|** that were visited. The results for UnifiedQA-v2 using the paths of the agent as context are denoted as *UnifiedQA-v2 – Context*. The results when combining two approaches by answering "yes" if at least one of them answers "yes" are denoted as *UnifiedQA-v2 | CauseNet 3-Hop* and *UnifiedQA-v2 | Agent 3-Hop*.

MS MARCO					
	A	$\mathbf{F_1}$	**R**	**P**	**|Nodes|**
Agent 2-Hop	0.460	0.562	0.408	0.901	6,774
Agent 3-Hop	0.529	0.648	0.511	0.884	7,034
CauseNet 1-Hop	0.259	0.241	0.139	0.912	14,937
CauseNet 2-Hop	0.494	0.612	0.471	0.875	454,126
CauseNet 3-Hop	0.589	0.714	0.605	0.871	878,090
UnifiedQA-v2	0.722	0.828	0.789	0.871	–
UnifiedQA-v2 | CauseNet 3-Hop	0.787	0.877	0.897	0.858	–
UnifiedQA-v2 | Agent 3-Hop	0.779	0.872	0.883	0.860	–
UnifiedQA-v2 — Context	0.661	0.789	0.740	0.842	–
Majority Baseline (True)	0.848	0.9177	1.000	0.848	–
SemEval					
	A	$\mathbf{F_1}$	**R**	**P**	**|Nodes|**
Agent 2-Hop	0.769	0.714	0.575	0.943	4,641
Agent 3-Hop	0.775	0.727	0.598	0.929	4,947
CauseNet 1-Hop	0.665	0.508	0.345	0.968	6,080
CauseNet 2-Hop	0.815	0.787	0.678	0.937	270,779
CauseNet 3-Hop	0.751	0.754	0.759	0.750	637,821
UnifiedQA-v2	0.497	0.653	0.943	0.500	–
UnifiedQA-v2 | CauseNet 3-Hop	0.520	0.677	1.000	0.512	–
UnifiedQA-v2 | Agent 3-Hop	0.520	0.675	0.989	0.512	–
UnifiedQA-v2 — Context	0.566	0.651	0.805	0.547	–
Majority Baseline (True)	0.503	0.669	1.000	0.503	–

the agent slightly improves the results when comparing the 2 hop and 3 hop configurations. Also, the agent performs better than BFS in the 3-Hop configuration on SemEval. Overall, compared to BFS, our agent has several advantages: (1) it can be extended to open-ended causal questions as discussed in Chapter 7, (2) it can to avoid false positives introduced by inverse edges and errors in CauseNet as shown above, (3) it prunes the search space and decreases the number of visited nodes by around 99%, on average searching only 25-30 nodes per question. In comparison, BFS searches approximately 1500 nodes per question with 2 hops and 3500 nodes per question with 3 hops.

Furthermore, the results of UnifiedQA-v2 [Kha+20; KKH22] vary significantly between the two datasets. Specifically, UnifiedQA-v2 is better than our agent on MS MARCO and worse on SemEval. This can be attributed to UnifiedQA-v2's tendency to answer with "yes" most of the time. In addition, the MS MARCO test set is heavily skewed towards positive questions (80% positive, 20% negative), whereas the SemEval test set is more balanced (50% positive, 50% negative). Thus, the results of UnifiedQA-v2's are better on MS MARCO than on SemEval. UnifiedQA-v2's tendency to answer "yes" is also indicated by the high recall of 0.943 and low precision of 0.500 on SemEval.

Finally, we explored a few simple strategies to combine our agent or BFS with UnifiedQA-v2. *UnifiedQA-v2 | CauseNet 3-Hop* and *UnifiedQA-v2 | Agent 3-Hop* answer with "yes" when at least one of the two approaches answers with "yes". Both strategies increase accuracy on MS MARCO by around 0.02 and SemEval by around 0.06. Indicating, that even though UnifiedQA-v2 answers with "yes" in most cases, it still misses some of the questions where "yes" is the correct answer.

Moreover, we tested the effectiveness of using paths learned by our agent as context for UnifiedQA-v2. For a given question, we took the learned paths and extracted the original sentence of each relation on the path. Subsequently, we concatenated the question with the sentences and provided the resulting text as input to UnifiedQA-v2. The results are mixed, with a decrease in accuracy by 0.06 on MS MARCO and an increase in accuracy by 0.06 on SemEval. The issue may be that the context provided by the agent can be misleading. For instance, as the agent is not perfect, it does not always find correct paths for "yes" questions. Thus, on MS MARCO, where a majority of questions are positive, the agent might mislead UnifiedQA-v2 to change correct "yes" answers to "no" erroneously. In contrast, on SemEval, where many more questions are negative, the agent helps UnifiedQA-v2 to correctly change wrong "yes" answers to "no".

Tab. 6.4: Results of the ablation study, where we compare different configurations of our approach. For each configuration, we either remove (−) or add (+) some component. The evaluation measures are abbreviated as follows: accuracy: **A**, F_1-Score: $\mathbf{F_1}$, recall: **R**, precision: **P**.

	MS MARCO				SemEval			
	A	$\mathbf{F_1}$	**R**	**P**	**A**	$\mathbf{F_1}$	**R**	**P**
Agent 2-Hop	**0.460**	**0.562**	**0.408**	0.901	**0.769**	**0.714**	**0.575**	0.943
− Beam Search	0.293	0.306	0.184	**0.911**	0.613	0.374	0.230	**1.000**
− Supervised Learning	0.342	0.397	0.257	0.891	0.682	0.538	0.369	**1.000**
− Actor-Critic	0.441	0.539	0.386	0.896	0.740	0.657	0.494	0.977
− Inverse Edges	0.422	0.513	0.359	0.899	0.740	0.651	0.483	**1.000**
+ Reward Shaping (0.1)	0.449	0.548	0.395	0.898	0.757	0.691	0.540	0.959
+ Reward Shaping (1.0)	0.403	0.489	0.336	0.893	**0.769**	0.706	0.552	0.980

6.6 Ablation Study

In our ablation study in Table 6.4, we investigate the performance impact of the different components of our approach. Accordingly, we try out the following configurations: (1) without supervised learning, i.e., we run Algorithm 1 directly and train the policy and value network with policy gradients from scratch, (2) without Actor-Critic, we remove the critic and only run the REINFORCE algorithm using the Monte-Carlo return as described in Section 3.3, (3) we remove the beam search and use greedy decoding to only sample the most probable path, (4) without inverse edges in the graph, (5) with the extra reward signal through the reward shaping technique as explained in Section 4.7. Hence, a − indicates the removal of some component, while a + indicates the addition of a component. Each configuration uses Agent 2-Hop with the settings described in Section 6.4.

Overall, beam search has the biggest impact on performance. When beam search is exchanged for greedy decoding, the accuracy drops from 0.460 to 0.293 on MS MARCO and from 0.769 to 0.613 on SemEval. As further analysis of the decoding techniques shows (Section 6.8), the accuracy keeps increasing with a higher beam width. Notably, greedy decoding slightly increases the precision on MS MARCO and does reach a precision of 1.0 compared to the 0.943 of beam search on SemEval. Thus, the number of false positives decreases when only using the most probable path found by greedy decoding. Moreover, supervised learning has the second highest impact. We analyze the effectiveness of supervised learning in Section 6.7 in more detail.

The Actor-Critic algorithm only has a minor impact with a difference of around 0.02-0.03 points accuracy on both datasets. Note that the λ-returns [SB18] and GAE [Sch+16], which we use in our Actor-Critic implementation, rely on multi-step returns and a bootstrap estimate via the value function to reduce the variance. At the moment, we only consider shorter paths of lengths 2 and 3, which might limit their impact. For this reason, we hypothesize that the impact compared to REINFORCE could further increase with longer paths. Additionally, the removal of inverse edges from the graph results in a slight decrease in overall performance but an increase in precision on the SemEval dataset to 1.0. That is because the removal of inverse edges reduces the probability of finding false positives, as discussed in Section 6.5.

Next, we discuss the impact of reward shaping. Recall that the reward shaping introduces the ω hyperparameter used to weigh the auxiliary rewards when the search is unsuccessful. During hyperparameter optimization, we tried a range of settings between 0.0 and 1.0. We show two configurations in Table 6.4, one for a smaller weight (0.1) and one for a larger weight (1.0). As ω increases, the accuracy on the MS MARCO dataset decreases. Smaller ω either decrease the performance slightly or make no difference at all. In contrast, on SemEval, the configuration of ω is less impactful, e.g., with $\omega = 1.0$, the accuracy is identical to the standard configuration. Notably, on SemEval, increasing ω also increases the precision. Overall, introducing the language model score as an auxiliary reward signal is not beneficial in its current form. Hence, this reward shaping technique requires further investigation and improvements in future work.

6.7 Effects of Supervised Learning

Below, we investigate the effects of supervised learning in more detail. To recall, the objective of supervised learning was to bootstrap the agent with expert demonstrations to better deal with the large action space. Thus, the expert demonstrations should leave the agent with a strong foundation for further improvement.

Therefore, we compare the performance of the agent when using different numbers of supervised training steps. Figure 6.1 shows the accuracy of the agent on the SemEval [Sha+16] test set depending on the number of reinforcement learning training steps. Each of the three runs was bootstrapped with a different number of supervised training steps. Thus, at step 0, we can see the accuracy directly after supervised learning without any training via reinforcement learning. We observe that the run with 100 steps is significantly worse than the runs with 200 and 300 steps. It

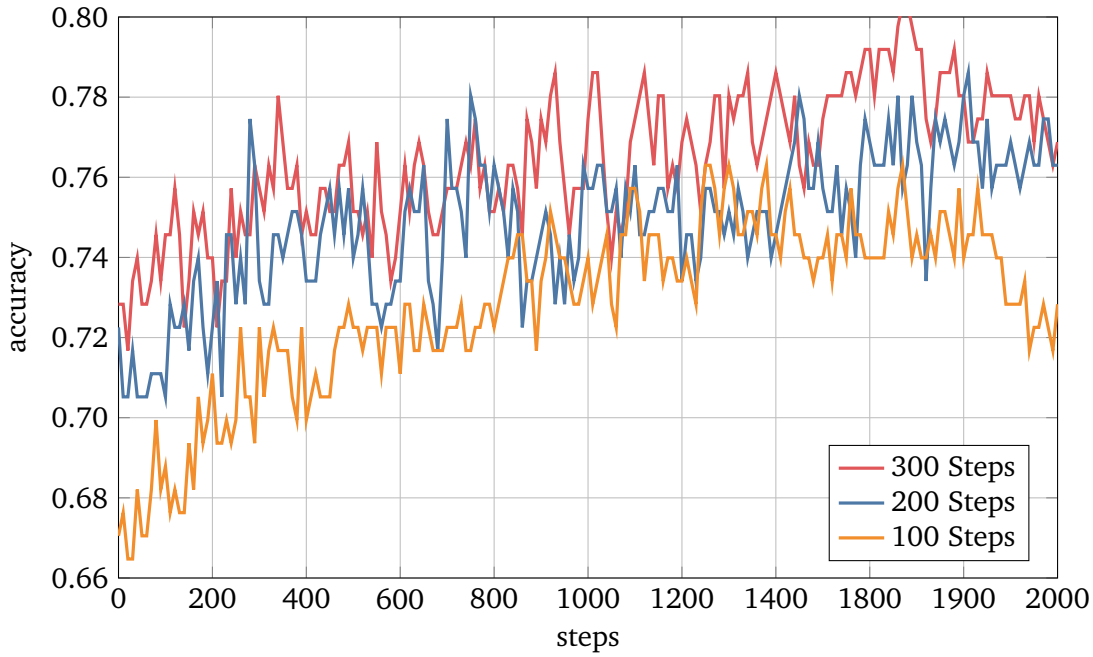

Fig. 6.1: The figure shows the accuracy of the agent on the SemEval test set depending on the number of reinforcement learning training steps. Each run was bootstrapped with a different number of supervised training steps. Step 0 shows the performance directly after supervised learning.

starts at around 0.67 directly after supervised learning and increases to around 0.72. Whereas the difference between 200 to 300 steps is already a lot smaller. Both start between 0.72 and 0.73 and follow similar trajectories afterward to reach an accuracy of around 0.76 after 2000 reinforcement learning steps. To maintain the clarity of the figures, we did not include a run with 400 steps, but the trend of diminishing returns on the number of supervised steps continues. This suggests that increasing the number of supervised steps beyond 300 does not improve performance. Likewise, we observed similar results on the MS MARCO [Ngu+16b] dataset.

Moreover, Figure 6.2 illustrates the number of unique paths explored during training on the left and the mean entropy of the action distribution of the policy network on the right. Notably, with an increasing number of supervised steps, the entropy of the policy network drops significantly. Hence, the number of explored paths during reinforcement learning also decreases as shown on the left in Figure 6.2. These findings indicate that supervised learning effectively establishes a strong foundation for the reinforcement learning agent. For example, an agent trained with 300 supervised steps only explores 21.6% of the paths of an agent without any supervised steps at the start. Accordingly, the agent trained with 300 supervised steps can exploit the knowledge acquired during supervised learning to follow better paths.

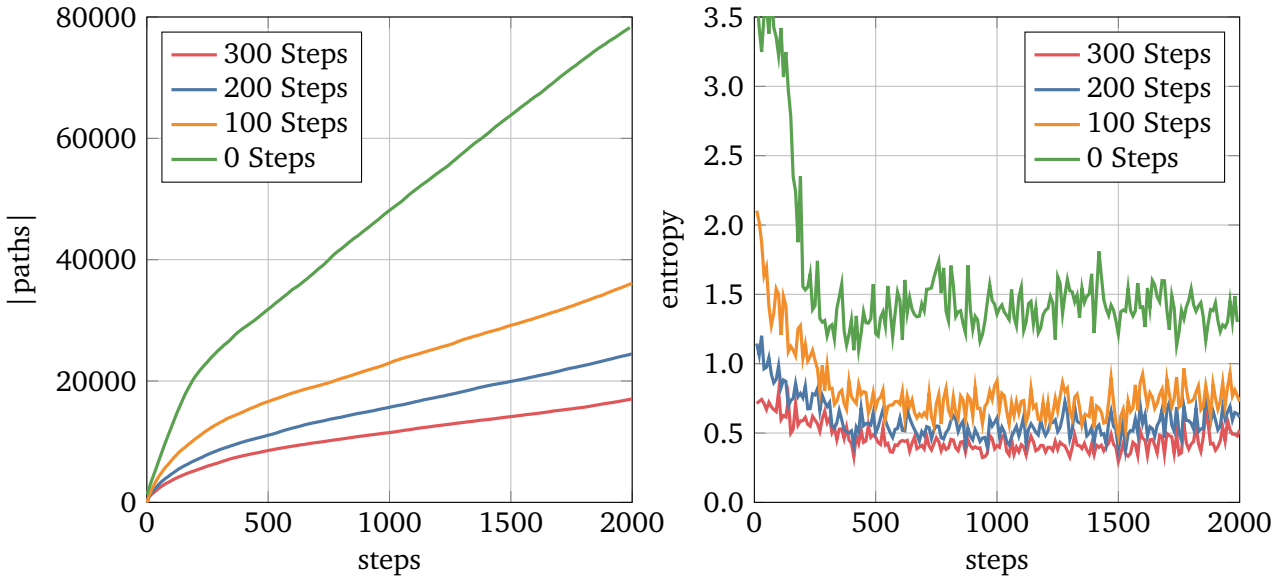

Fig. 6.2: The figure shows the number of unique paths explored during reinforcement learning training on the left and the mean entropy of the action distribution of the policy network on the right. Each run was bootstrapped with a different number of supervised training steps.

In contrast, the agent trained without any supervised learning steps requires more exploration and fails to achieve the same performance, as shown in Table 6.4.

6.8 Decoding Analysis

In the following, we investigate the effects of different beam widths on the MS MARCO test set. We experimented with beam widths of 1, 5, 10, and 50 and present the results in Figure 6.3. In these experiments, we did not apply supervised learning at the beginning to focus solely on the effects of beam width settings. Additionally, we increased the training steps from 2000 to 4000 because, without supervised learning, there is still some progress after 2000 steps.

In general, as the beam width increases, accuracy also increases. For example, the difference between a width of 1 and a width of 50 is 0.09 points accuracy after 2000 steps. The exception is between widths of 5 and 10, which reach the same accuracy after 2000 steps and have similar curves. In Section 4.5, we noted that beam search can be viewed as an interpolation between greedy decoding and BFS. Therefore, if the beam width is set too high, the task may become too easy as the agent performs an exhaustive search, disregarding runtime considerations for a moment. We can already observe this effect when looking at the accuracy

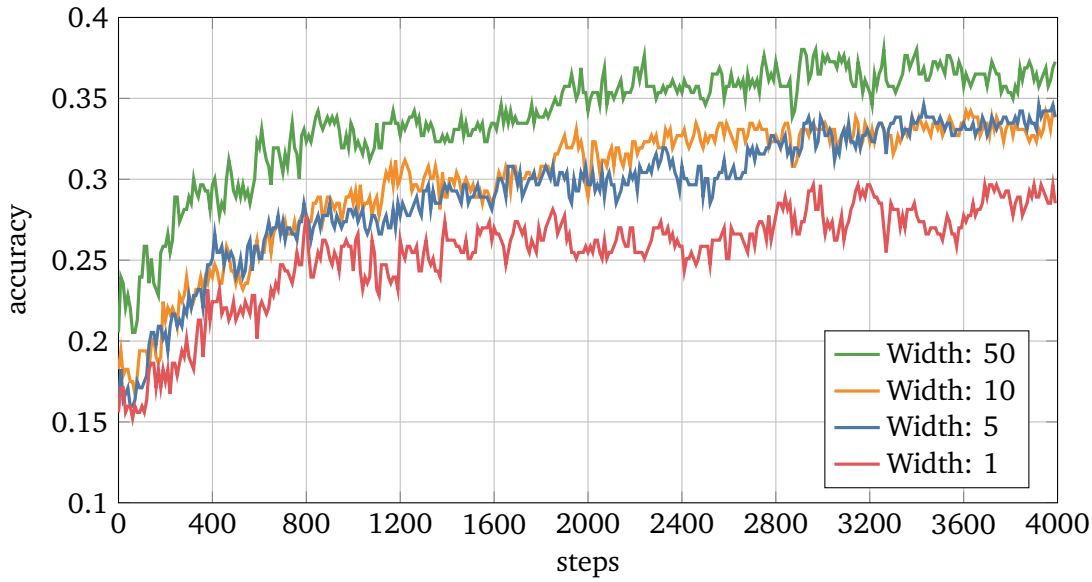

Fig. 6.3: Accuracy on the MS MARCO test set depending on the number of reinforcement learning training steps. Each run uses a different beam width and our standard setup, where we compare all entities on the paths to the target entity.

immediately after initialization at step 0. As the beam width increases, the accuracy without any learning also increases slightly, e.g., for widths of 10 and 50 from 0.18 to 0.21. However, the increases are minor, and even beam width 50 only starts at 0.21 and still has a lot of learning progress afterward. Additionally, the performance from width 50 does not reach the performance with supervised learning or the performance of the BFS. This suggests that a beam width of 50 is still reasonable, and we could have also used a higher beam width, like 100, for our experiments.

Our standard setup evaluates the paths by checking if any of their entities match the target entity. However, in Section 4.2, we introduced the *STAY* action and trained the agent on path rollouts of the same length. Hence, when the agent reaches the target entity, it should have learned to use the *STAY* action and stop advancing. Therefore, we investigate whether the agent behaves accordingly in Figure 6.4.

The runs in Figure 6.4 use the same setup as those in Figure 6.3, except that we only compare the last node on each path to the target entity. This setup allows us to determine whether the agent always stays at the target entity or sometimes walks over it. The figures illustrate that there is a slight difference between the two setups. In case we only check the last node, the accuracy after 4000 steps is always around 0.01-0.02 points accuracy lower when comparing the runs with the same beam width. While the difference is small, this suggests that the agent does not always stop when necessary. As an illustration, we provide an example in Section 6.10.

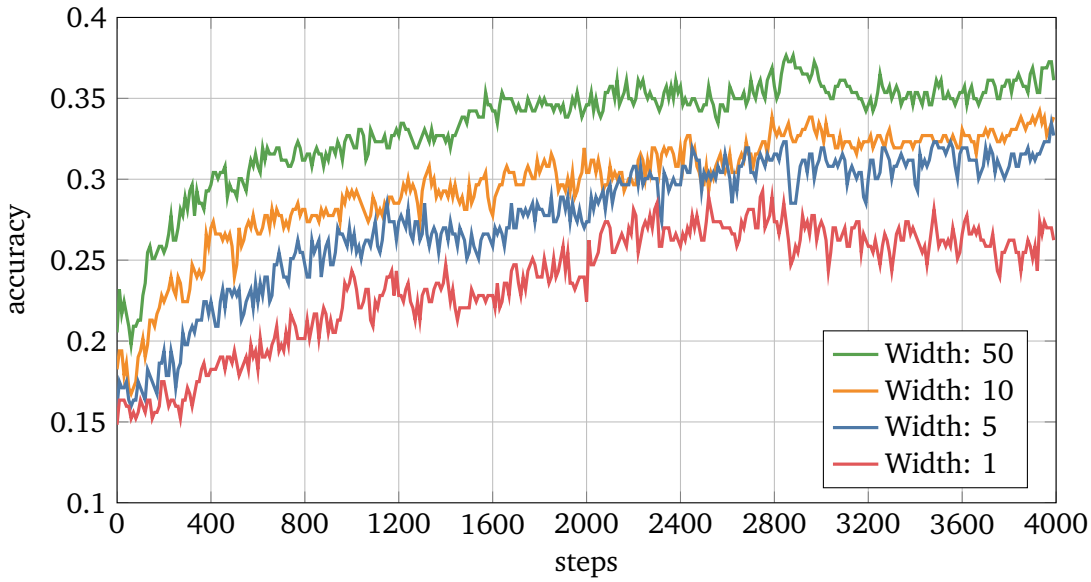

Fig. 6.4: Accuracy on the MS MARCO test set depending on the number of reinforcement learning training steps. Each run uses a different beam width. The figure illustrates a slightly different setup, where we only compare the last node on each path to the target entity.

6.9 Inference Time

Figure 6.5 illustrates the inference runtime for both test sets, depending on the beam width and path rollout length. The left figure presents the runtime on the MS MARCO test set, while the right figure shows the runtime for the SemEval test set. Note that the time measurements do not include the setup of the environment, such as the loading of the knowledge graph and embeddings.

The MS MARCO test set contains 263 questions, while the SemEval test set only contains 173 questions. As a result, the runtime for MS MARCO is longer in both cases. For a beam width of 50, the inference on MS MARCO takes 52 seconds for path rollout length 2 and 174 seconds for path rollout length 3, resulting in a runtime per question of around 0.20 seconds and 0.66 seconds, respectively. Also, on SemEval, the inference takes 36 (0.21) seconds and 97 (0.56) seconds, so the time per question is similar for both datasets. We observe that when increasing the path rollout length from 2 to 3, the runtime already increases by a factor of around 3. However, there is much room for improvement in the current beam search implementation, particularly regarding batching and memory movements between CPU and GPU. Generally, we can make a trade-off between performance and runtime. As the beam width decreases, runtime also decreases; for example, in the case of a

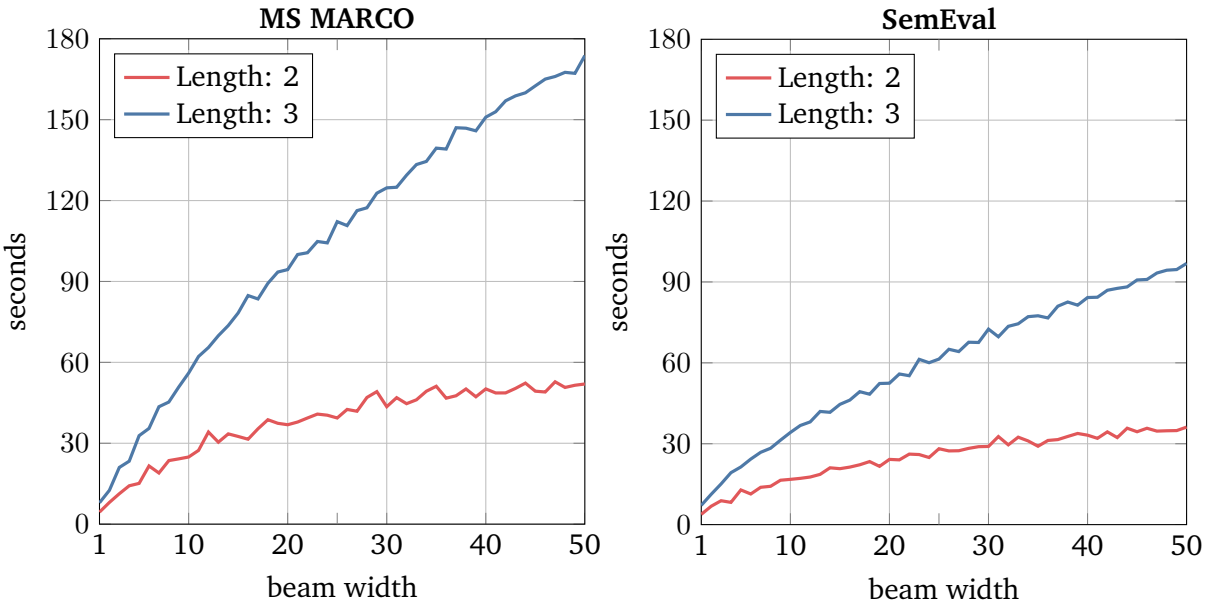

Fig. 6.5: Inference time in seconds for the test sets depending on the beam width and path rollout length. The figure on the left shows the time for the MS MARCO test set, and the figure on the right for the SemEval test set. The agent takes longer for MS MARCO in both cases because the MS MARCO test set contains 90 questions more compared to SemEval.

beam width of 1, the runtime would be around 0.05 seconds per question. However, this would also decrease the accuracy, as shown in Figure 6.3.

As illustrated in the plots, the beam width does not always increase the runtime. While the beam width is an important factor for the runtime, there are other factors to consider. For example, the runtime also depends on the degree of the nodes on the explored paths, as the beam search must consider each neighbor as a potential candidate.

6.10 Examples

Lastly, we provide some examples that our agent found. In Table 6.5, we illustrate two examples of Agent 2-Hop and four examples of Agent 3-Hop. Each example includes a cause and an effect together with a path between them. The arrows on the paths indicate whether the agent took the standard *cause* edge, the *STAY* action, or an inverse edge $cause^{-1}$.

The paths learned by the agent can be used to follow the complete reasoning chain to examine the mechanisms of how a cause produces an effect. For example, the

Tab. 6.5: Some examples that our agent found. Each example consists of a cause, an effect, and a path between them. For each path, we indicate whether the agent used a normal *cause* edge, the *STAY* action, or an inverse edge $cause^{-1}$.

Agent 2-Hop
Cause: stress **Effect:** skin rashes
Path: stress $\xrightarrow{\text{cause}}$ skin rashes $\xrightarrow{STAY}$ skin rashes
Cause: cigarettes **Effect:** nausea
Path: cigarettes $\xrightarrow{\text{cause}}$ cancer $\xrightarrow{\text{cause}}$ nausea
Agent 3-Hop
Cause: h. pylori **Effect:** vomiting
Path: h. pylori $\xrightarrow{\text{cause}}$ peptic ulcer disease $\xrightarrow{\text{cause}}$ vomiting $\xrightarrow{STAY}$ vomiting
Cause: Xanax **Effect:** hiccups
Path: xanax $\xrightarrow{\text{cause}}$ anxiety $\xrightarrow{\text{cause}}$ stress $\xrightarrow{\text{cause}}$ hiccups
Cause: chocolate **Effect:** constipation
Path: chocolate $\xrightarrow{\text{cause}}$ constipation $\xrightarrow{\text{cause}}$ depression $\xrightarrow{\text{cause}^{-1}}$ constipation
Cause: rainfall **Effect:** flooding
Path: rainfall $\xrightarrow{\text{cause}}$ flooding $\xrightarrow{\text{cause}}$ landslides $\xrightarrow{STAY}$ landslides

first path from Agent 3-Hop indicates that the bacterium *helicobacter pylori* can lead to the development of *peptic ulcer* disease, which can lead to *vomiting*. The first examples for Agent 2-Hop and Agent 3-Hop demonstrate the agent's ability to utilize the *STAY* action. In general, the usage of the *STAY* action works well in most cases. However, as shown in Section 6.8, there are some exceptions where the *STAY* should have been used but was not. An example can be seen in the last path for Agent 3-Hop in Table 6.5. The agent should have stayed at *flooding* but went ahead to *landslides* and stayed there.

Moreover, the agent learned to use inverse edges to recover from mistakes. In the third example for Agent 3-Hop, the agent went one step too far, from *constipation* to *depression*, and used an inverse edge afterward to return to the *constipation* entity.

7 Discussion & Future Work

Below, we discuss a few points, like the addition of inverse edges and how our approach can be used to explain the answers to causal questions. Additionally, we talk about possible directions for future work. These include the extension to more causal question types, the usage of different causal knowledge graphs, and model-based reinforcement learning techniques.

Inverse Edges For CauseNet [Hei+20], the addition of inverse edges implies that the agent can also walk from an effect to its cause. In general, their addition has a few benefits, like the possibility to undo wrong actions and to reach nodes that otherwise could not be reached under a given path length constraint. Conversely, they also introduce the possibility to make mistakes through false positives, as described in Section 6.2. While these mistakes will always happen for the BFS, our agent can minimize them by pruning the search space, as demonstrated in our experiments in Section 6.5.

Moreover, it might not always be clear that it makes sense to take an inverse edge in theory. For example, given a question like *"Does X cause Y?"*, we start the search at X. If we take an inverse edge from an entity Z to Y at some point during the search, it does not directly follow that X causes Y in theory. However, as our experiments show (Section 6.6), adding inverse edges improves performance in practice, so their benefits seem to outweigh the pitfalls. This is in line with prior works that also added inverse edges to improve performance [XHW17; Das+18].

Explainability of Causal Questions Our approach has the advantage of not only being able to answer binary causal questions but also to provide post-hoc explanations. Although, this only holds for questions that were answered with "yes" because CauseNet [Hei+20] does not contain negative information. If the agent found no path or the cause or effect could not be found in CauseNet, we can only answer with "no" without additional explanation why the causal relation does not hold. Additionally, errors in CauseNet may lead to incorrect answers and explanations. However, with a precision of 96% in CauseNet-Precision [Hei+20], such scenarios are rare.

As the examples in Section 6.10 show, the paths found during inference can be used to follow the complete reasoning chain. Thus, we can inspect the paths to examine the specific mechanisms by which the cause produces the effect, potentially through a chain of multiple entities. Moreover, since the agent might find multiple paths between a cause and effect pair, we can showcase multiple ways in which the cause and effect are related.

Finally, we can use the additional meta-information that is part of each relation [Hei+20]. For example, we can reference the original sentence from which the relation was extracted, which may provide additional insights. Furthermore, each relation contains the URL of the original web source, which we can check to verify the causal relations and receive further information. In general, this is not possible with a language model like UnifiedQA [Kha+20; KKH22]. For example, for binary causal questions, it provides a "yes" or "no" answer, possibly with an explanation, but with no direct way to verify this claim. Although, research is ongoing to enhance language models with the ability to cite their sources [Men+22].

Additional Causal Question Types At the moment, our approach only supports binary causal questions. In the following, we discuss a straightforward extension to Cause-Effect question. In Section 3.2, we defined Cause-Effect questions as questions like *"What causes pneumonia?"*, i.e., questions that ask for the cause of a given effect or vice versa. These questions only require changes at inference time and no changes at train time.[1]

Contrary to binary causal questions, we no longer have to verify a given causal relation but search for a suitable entity. Therefore, we need to change the decoding at inference time since we no longer know the entity we are looking for. Thus, we introduce a majority voting approach. Given a question like *"What causes pneumonia?"*, we still sample multiple paths from the agent. Next, we count the occurrence of the entities on these paths and select the one with the highest count as the answer. Alternatively, we can conduct the ranking by summing their probabilities on different paths. Moreover, we can also provide multiple answers by selecting the top n candidates. Overall, this approach only requires minor changes to our current code base. To answer more complex questions like *"How does smoking cause cancer?"*, we can run the agent to answer the question as if it were a binary question and then use one of the possibilities for explanations described above.

[1] During training, one minor difference is that we do not take the cause and effect from the question, but rather one from the question and the other from the answer.

Causal Knowledge Graphs Our approach is inherently limited by CauseNet. If a question requires information not present in CauseNet, our agent cannot provide an answer. A simple extension would be the introduction of additional graphs like Cause Effect Graph [Li+20]. Cause Effect Graph is another large-scale causal knowledge graph that is very similar to CauseNet's structure. Subsequently, we can apply the agent on each graph and answer with "yes" if one of them yields a result. Conversely, if we want to optimize for precision, we would require both to yield a result.

Another possible extension is the introduction of graphs like ATOMIC [Sap+19]. CauseNet contains causal relations between concepts like cancer, smoking, pneumonia, or anemia. In contrast, ATOMIC contains relations about commonsense reasoning in everyday life. These relations are presented in the form of "If-Event-Then-X" patterns. For example, social interactions like if "Person X makes Person Y a compliment", then "Person Y will feel good" [Sap+19]. These different knowledge types can complement each other and allow the agent to answer a greater variety of causal questions.

False Positives & Negative Causal Questions In future work, we plan to investigate different ways to handle false positives during training and incorporate negative causal questions. As discussed, inverse edges and errors in CauseNet can introduce false positives. Currently, during training, these errors are treated as any other error, resulting in a reward of 0. However, it could be argued that these types of errors should be treated differently, such as by assigning a negative reward. To incentivize the agent to ignore errors in CauseNet and only to take inverse edges when appropriate. Additionally, we will explore mechanisms for incorporating negative causal questions into training. In the current setup, we excluded them because CauseNet does not contain negative information [Hei+20].

Model-Based Reinforcement Learning As discussed above, the transition function of the MDP is known and deterministic because it is entirely defined by the static graph [She+18]. Thus, M-Walk [She+18] used a model-based reinforcement learning algorithm similar to AlphaGo Zero [Sil+17]. A possible improvement could be to use MuZero [Sch+20], a successor of AlphaGo Zero. Particularly interesting might be Sampled MuZero [Hub+21], an extension to MuZero designed to deal with large and complex action spaces.

Runtime Improvements In future works, we will add parallelization to our framework to decrease the training time. Typical A2C implementations use multiple workers in parallel to sample episodes. However, in our current implementation, we only use one worker. To improve the inference time, we can improve our beam search implementation regarding the usage of batches and memory movements between CPU and GPU. Furthermore, we can apply improvements like switching from eager mode to graph mode in PyTorch.

8 Conclusion

In this thesis, we introduced a reinforcement learning approach to answer binary causal questions on a causal knowledge graph. We extended the setup of many prior approaches for reasoning tasks via reinforcement learning on knowledge graphs [XHW17; Das+18; Qiu+20; KSW21] by transitioning from the REINFORCE [Wil92] algorithm to the A2C [Mni+16] algorithm with generalized advantage estimation (GAE) [Sch+16]. Furthermore, we bootstrapped the agent with a supervised training procedure [XHW17] to better handle the large action space in CauseNet [Hei+20]. Moreover, we adapted a reward shaping technique from the literature [Yas+21] to handle sparse rewards by introducing auxiliary reward signals. Additionally, we enhanced an existing approach from CauseNet for extracting binary causal questions.

Our evaluation showed that the agent's performance does not fully match the BFS baseline, indicating opportunities for improvement. However, while it did not fully match the performance in terms of accuracy or F_1-Score, the agent exhibited advantages in other areas. Specifically, the agent was able to mitigate the problem of false negatives. Hence, the agent learned to avoid paths leading to false positives introduced by inverse edges and errors in CauseNet. Besides, the agent effectively searches through the graph by pruning the search space by around 99%, as demonstrated in Table 6.3.

Regarding potential improvements and limitations, we observed that while the agent could mitigate the problem of false positives, it did not avoid them entirely. As shown in Table 6.3, the precision decreases when increasing the path length. As a result, we will investigate possible ways to handle false positives during training, as discussed in Chapter 7. Also, another area of improvement can be the increase of the beam width, as our experiments indicated that it is still possible to do so without simplifying the problem too much.

Besides, our evaluation suggested that bootstrapping via supervised learning provides a strong foundation for further improvement. The agent bootstrapped with supervised learning explored significantly fewer paths while achieving higher accuracy and F_1-Scores than the agent trained without supervised learning. In contrast, the reward shaping did not have the desired effect. Even when carefully tuned, it

had no impact on performance at best. In many cases, it decreased the performance by introducing conflicting reward signals. In future work, we will investigate and try to improve this reward shaping technique.

While there are areas for improvement, for example, w.r.t. the performance in terms of accuracy and F1-Score, we successfully introduced a new approach for binary causal question answering. Our agent can be especially valuable through its ability to provide explanations when answering a question. Although at the moment, explanations are only possible for questions the agent answered with "yes", as CauseNet does not contain negative information. For questions answered with "yes", end-users receive the learned paths and can follow the reasoning chain to better understand how cause and effect are related. Additionally, our agent provides the original sentence from which the cause-effect pair was extracted and the URL of the source web page, allowing users to verify the claims. In contrast, this is not possible for a system like UnifiedQA-v2 [Kha+20; KKH22].

As presented in this thesis, the first version of our agent is currently limited to binary causal questions. In the future, we will extend this to different causal question types, like Cause-Effect questions or multiple-choice questions, to cover a greater range of potential queries. In Chapter 7, we introduced a simple extension for Cause-Effect questions, which only requires minimal changes to our codebase. Note that our agent can already answer questions like *"How does pneumonia cause anemia?"* by finding a path between the cause and effect and afterward providing the original sentence and source as an explanation.

Further considerations for future work include using different causal knowledge graphs, model-based reinforcement learning techniques, and runtime improvements, as discussed in Chapter 7.

Bibliography

[ABC14] Ioannis Arapakis, Xiao Bai, and Berkant Barla Cambazoglu. “Impact of response latency on user behavior in web search”. In: *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’14, Gold Coast , QLD, Australia - July 06 - 11, 2014*. Ed. by Shlomo Geva, Andrew Trotman, Peter Bruza, Charles L. A. Clarke, and Kalervo Järvelin. ACM, 2014, pp. 103–112 (cit. on p. 17).

[Ben+03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3 (2003), pp. 1137–1155 (cit. on p. 27).

[Bie20] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020 (cit. on p. 33).

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. Beijing: O’Reilly, 2009 (cit. on p. 33).

[Bon+22] Alexander Bondarenko, Magdalena Wolska, Stefan Heindorf, et al. “CausalQA: A Benchmark for Causal Question Answering”. In: *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*. Ed. by Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, et al. International Committee on Computational Linguistics, 2022, pp. 3296–3308 (cit. on pp. 8, 10, 37, 39).

[Dal21] Dhairya Dalal. “Knowledge augmented language models for causal question answering”. In: *CEUR Workshop Proceedings* 3005 (2021), pp. 17–24 (cit. on pp. 1, 4).

[DAB21] Dhairya Dalal, Mihael Arcan, and Paul Buitelaar. “Enhancing Multiple-Choice Question Answering with Causal Knowledge”. In: *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Online: Association for Computational Linguistics, June 2021, pp. 70–80 (cit. on pp. 1, 4).

[Das+18] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, et al. “Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning”. In: *ICLR*. 2018 (cit. on pp. 1, 2, 4, 10, 13, 18–21, 25, 27, 29, 33, 39, 41, 53, 57).

[Det+18] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. "Convolutional 2D Knowledge Graph Embeddings". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, 2018 (cit. on p. 5).

[Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186 (cit. on p. 4).

[Dun+17] Matthew Dunn, Levent Sagun, Mike Higgins, et al. "SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine". In: *CoRR* abs/1704.05179 (2017). arXiv: 1704.05179 (cit. on p. 10).

[Fan+19] Angela Fan, Yacine Jernite, Ethan Perez, et al. "ELI5: Long Form Question Answering". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL 2019)*. 2019, pp. 3558–3567 (cit. on pp. 9, 10).

[GM02] Roxana Girju and Dan I. Moldovan. "Text Mining for Causal Relations". In: *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference, May 14-16, 2002, Pensacola Beach, Florida, USA*. Ed. by Susan M. Haller and Gene Simmons. AAAI Press, 2002, pp. 360–364 (cit. on pp. 9, 11).

[Gua+18] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, et al. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. https://github.com/tensorflow/agents. [Online; accessed 25-June-2019]. 2018 (cit. on pp. 33, 34).

[Hab+18] Ivan Habernal, Henning Wachsmuth, Iryna Gurevych, and Benno Stein. "The Argument Reasoning Comprehension Task: Identification and Reconstruction of Implicit Warrants". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. by Marilyn A. Walker, Heng Ji, and Amanda Stent. Association for Computational Linguistics, 2018, pp. 1930–1940 (cit. on p. 1).

[Has+19] Oktie Hassanzadeh, Debarun Bhattacharjya, Mark Feblowitz, et al. "Answering Binary Causal Questions Through Large-Scale Text Mining: An Evaluation Using Cause-Effect Pairs from Human Experts". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5003–5009 (cit. on pp. 1, 4, 8).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034 (cit. on p. 41).

[Hei+20] Stefan Heindorf, Yan Scholten, Henning Wachsmuth, Axel-Cyrille Ngonga Ngomo, and Martin Potthast. “CauseNet: Towards a Causality Graph Extracted from the Web”. In: *CIKM*. ACM, 2020 (cit. on pp. 1–4, 7–9, 19, 20, 22, 24, 38, 41, 53–55, 57).

[Hen+10] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, et al. “SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals”. In: *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*. Ed. by Katrin Erk and Carlo Strapparava. The Association for Computer Linguistics, 2010, pp. 33–38 (cit. on p. 37).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on pp. 4, 21).

[Hub+21] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, et al. “Learning and Planning in Complex Action Spaces”. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 4476–4486 (cit. on p. 55).

[Hwa+21] Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, et al. “(Comet-) Atomic 2020: On Symbolic and Neural Commonsense Knowledge Graphs”. In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 6384–6392 (cit. on p. 3).

[ISZ21] Filip Ilievski, Pedro A. Szekely, and Bin Zhang. “CSKG: The CommonSense Knowledge Graph”. In: *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*. Ed. by Ruben Verborgh, Katja Hose, Heiko Paulheim, et al. Vol. 12731. Lecture Notes in Computer Science. Springer, 2021, pp. 680–696 (cit. on p. 3).

[Jos+17] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. “TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*. 2017, pp. 1601–1611 (cit. on p. 10).

[KSW21] Magdalena Kaiser, Rishiraj Saha Roy, and Gerhard Weikum. “Reinforcement Learning from Reformulations in Conversational Question Answering over Knowledge Graphs”. In: *SIGIR*. 2021 (cit. on pp. 1, 5, 10, 17, 21, 25, 33, 39, 57).

[Kay+20] Humayun Kayesh, Md. Saiful Islam, Junhu Wang, et al. “Answering Binary Causal Questions: A Transfer Learning Based Approach”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pp. 1–9 (cit. on pp. 1, 3, 8, 39).

[KKH22] Daniel Khashabi, Yeganeh Kordi, and Hannaneh Hajishirzi. “UnifiedQA-v2: Stronger Generalization via Broader Cross-Format Training”. In: *CoRR* abs/2202.12359 (2022). arXiv: 2202.12359 (cit. on pp. 38, 39, 41, 44, 54, 58).

[Kha+20] Daniel Khashabi, Sewon Min, Tushar Khot, et al. “UnifiedQA: Crossing Format Boundaries With a Single QA System”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Vol. EMNLP 2020. Findings of ACL. Association for Computational Linguistics, 2020, pp. 1896–1907 (cit. on pp. 38, 39, 41, 44, 54, 58).

[Kha+21] Daniel Khashabi, Amos Ng, Tushar Khot, et al. “GooAQ: Open Question Answering with Diverse Answer Types”. In: *Findings of the Association for Computational Linguistics (EMNLP 2021)*. 2021, pp. 421–433 (cit. on p. 10).

[Kwi+19] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, et al. “Natural Questions: A Benchmark for Question Answering Research”. In: *Trans. Assoc. Comput. Linguistics* 7 (2019), pp. 452–466 (cit. on p. 10).

[Lew+21] Patrick S. H. Lewis, Yuxiang Wu, Linqing Liu, et al. “PAQ: 65 Million Probably-Asked Questions and What You Can Do With Them”. In: *Trans. Assoc. Comput. Linguistics* 9 (2021), pp. 1098–1115 (cit. on p. 10).

[Li+20] Zhongyang Li, Xiao Ding, Ting Liu, J. Edward Hu, and Benjamin Van Durme. “Guided Generation of Cause and Effect”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, 2020, pp. 3629–3636 (cit. on pp. 3, 55).

[Lia+18] Eric Liang, Richard Liaw, Robert Nishihara, et al. “RLlib: Abstractions for Distributed Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3059–3068 (cit. on pp. 33, 34).

[LSX18] Xi Victoria Lin, Richard Socher, and Caiming Xiong. “Multi-Hop Knowledge Graph Reasoning with Reward Shaping.” In: *EMNLP*. Ed. by Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii. Association for Computational Linguistics, 2018, pp. 3243–3253 (cit. on pp. 4, 20, 21, 29, 31, 32, 39, 41).

[Liu+19] Yinhan Liu, Myle Ott, Naman Goyal, et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692 (cit. on pp. 34, 41, 42).

[LH19] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on p. 41).

[Man+14] Christopher Manning, Mihai Surdeanu, John Bauer, et al. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *ACL (System Demonstrations)* (2014), pp. 55–60 (cit. on pp. 9, 33).

[Mar+15] Martín Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 33).

[Men+22] Jacob Menick, Maja Trebacz, Vladimir Mikulik, et al. “Teaching language models to support answers with verified quotes”. In: *CoRR* abs/2203.11147 (2022). arXiv: 2203.11147 (cit. on p. 54).

[Mni+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1928–1937 (cit. on pp. 2, 7, 10, 13, 24, 57).

[NHR99] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping”. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML ’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287 (cit. on p. 5).

[Ngu+16a] Tri Nguyen, Mir Rosenberg, Xia Song, et al. “MS MARCO: A Human Generated MAchine Reading COmprehension Dataset”. In: *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016)*. 2016 (cit. on pp. 8, 10).

[Ngu+16b] Tri Nguyen, Mir Rosenberg, Xia Song, et al. “MS MARCO: A Human Generated MAchine Reading COmprehension Dataset”. In: *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*. Ed. by Tarek Richard Besold, Antoine Bordes, Artur S. d’Avila Garcez, and Greg Wayne. Vol. 1773. CEUR Workshop Proceedings. CEUR-WS.org, 2016 (cit. on pp. 1, 37, 47).

[PMB13] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, 2013, pp. 1310–1318 (cit. on p. 41).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, et al. 2019, pp. 8024–8035 (cit. on p. 33).

[Pen+18] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. “DeepMimic: example-guided deep reinforcement learning of physics-based character skills”. In: *ACM Trans. Graph.* 37.4 (2018), p. 143 (cit. on pp. 11–15, 41).

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543 (cit. on pp. 24, 34, 41).

[Qiu+20] Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. “Stepwise Reasoning for Multi-Relation Question Answering over Knowledge Graph with Weak Supervision”. In: WSDM ’20. Houston, TX, USA: Association for Computing Machinery, 2020, pp. 474–482 (cit. on pp. 4, 5, 18–21, 27, 29, 31, 39, 41, 57).

[Raf+20] Colin Raffel, Noam Shazeer, Adam Roberts, et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67 (cit. on p. 39).

[Raf+21] Antonin Raffin, Ashley Hill, Adam Gleave, et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8 (cit. on pp. 33, 34).

[RJL18] Pranav Rajpurkar, Robin Jia, and Percy Liang. “Know What You Don’t Know: Unanswerable Questions for SQuAD”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*. 2018, pp. 784–789 (cit. on pp. 8, 10).

[Sap+19] Maarten Sap, Ronan Le Bras, Emily Allaway, et al. “ATOMIC: An Atlas of Machine Commonsense for If-Then Reasoning”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 3027–3035 (cit. on pp. 3, 55).

[SMG14] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014 (cit. on p. 41).

[Sch+20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, et al. “Mastering Atari, Go, chess and shogi by planning with a learned model”. In: *Nat.* 588.7839 (2020), pp. 604–609 (cit. on p. 55).

[Sch+16] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016 (cit. on pp. 2, 12–15, 46, 57).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347 (cit. on p. 24).

[Sha+16] Rebecca Sharp, Mihai Surdeanu, Peter Jansen, Peter Clark, and Michael Hammond. “Creating Causal Embeddings for Question Answering with Minimal Supervision”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 138–148 (cit. on pp. 1, 4, 37, 46).

[She+18] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. “M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, et al. Vol. 31. Curran Associates, Inc., 2018 (cit. on pp. 5, 11, 33, 55).

[Sil15] David Silver. *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/. 2015 (cit. on p. 14).

[Sil+16] David Silver, Aja Huang, Chris J. Maddison, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nat.* 529.7587 (2016), pp. 484–489 (cit. on p. 29).

[Sil+18] David Silver, Thomas Hubert, Julian Schrittwieser, et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404 (cit. on p. 5).

[Sil+17] David Silver, Julian Schrittwieser, Karen Simonyan, et al. “Mastering the game of Go without human knowledge”. In: *Nat.* 550.7676 (2017), pp. 354–359 (cit. on p. 55).

[Son+20] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. “MPNet: Masked and Permuted Pre-training for Language Understanding”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020 (cit. on p. 41).

[SCH17] Robyn Speer, Joshua Chin, and Catherine Havasi. “ConceptNet 5.5: An Open Multilingual Graph of General Knowledge”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by Satinder Singh and Shaul Markovitch. AAAI Press, 2017, pp. 4444–4451 (cit. on p. 3).

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018 (cit. on pp. 10–14, 46).

[Tri+17] Adam Trischler, Tong Wang, Xingdi Yuan, et al. "NewsQA: A Machine Comprehension Dataset". In: *Proceedings of the 2nd Workshop on Representation Learning for NLP (Rep4NLP@ACL 2017)*. 2017, pp. 191–200 (cit. on pp. 8, 10).

[Vel+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, et al. "Graph Attention Networks". In: *International Conference on Learning Representations* (2018) (cit. on p. 4).

[Wal07] Douglas N. Walton. *Dialog theory for critical argumentation*. Vol. 5. Controversis. Benjamin/Cummings, 2007 (cit. on p. 1).

[WD21] Guojia Wan and Bo Du. "GaussianPath:A Bayesian Multi-Hop Reasoning Framework for Knowledge Graph Reasoning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.5 (May 2021), pp. 4393–4401 (cit. on pp. 2, 5).

[Wes+21] Peter West, Chandra Bhagavatula, Jack Hessel, et al. "Symbolic Knowledge Distillation: from General Language Models to Commonsense Models". In: *CoRR* abs/2110.07178 (2021). arXiv: 2110.07178 (cit. on p. 3).

[Wil92] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256 (cit. on pp. 4, 7, 10, 13, 57).

[Wol+20] Thomas Wolf, Lysandre Debut, Victor Sanh, et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*. Ed. by Qun Liu and David Schlangen. Association for Computational Linguistics, 2020, pp. 38–45 (cit. on pp. 33, 41, 42).

[XHW17] Wenhan Xiong, Thien Hoang, and William Yang Wang. "DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 564–573 (cit. on pp. 1, 2, 4, 10, 11, 18, 19, 29, 33, 53, 57).

[Yan+18] Zhilin Yang, Peng Qi, Saizheng Zhang, et al. "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*. 2018, pp. 2369–2380 (cit. on pp. 8, 10).

[Yas+21] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. "QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 535–546 (cit. on pp. 2, 4, 21, 31, 41, 57).

[Zha+18] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. "Variational Reasoning for Question Answering With Knowledge Graph". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 6069–6076 (cit. on p. 33).