FLÁVIO NAKASATO CAÇÃO

A DEEP REINFORCEMENT LEARNING APPROACH TO COMPLEX OPEN-DOMAIN QUESTION ANSWERING

São Paulo
2023

FLÁVIO NAKASATO CAÇÃO

A DEEP REINFORCEMENT LEARNING APPROACH TO COMPLEX OPEN-DOMAIN QUESTION ANSWERING

Dissertation presented to Escola Politécnica of Universidade de São Paulo to obtain Master of Sciences degree in Electrical Engineering.

São Paulo
2023

FLÁVIO NAKASATO CAÇÃO

A DEEP REINFORCEMENT LEARNING APPROACH TO COMPLEX OPEN-DOMAIN QUESTION ANSWERING

Final Version

Dissertation presented to Escola Politécnica of Universidade de São Paulo to obtain Master of Sciences degree in Electrical Engineering.

Concentration Area:

Computer Engineering

Advisor:

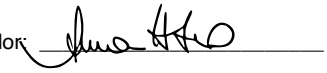
Anna Helena Reali Costa

São Paulo
2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 8 de Janeiro de 2023

Assinatura do autor: 

Assinatura do orientador: 

Catalogação-na-publicação

Cação, Flávio
Uma Abordagem de Aprendizado por Reforço Profundo para Respostas a Perguntas Complexas de Domínio Aberto / F. Cação -- versão corr. -- São Paulo, 2023.
70 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Perguntas e Respostas Complexas de Domínio Aberto 2.Aprendizado Profundo por Reforço 3.Agentes de Conversacionais I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

ACKNOWLEDGMENTS

This research is being carried out with the support of *Itaú Unibanco S.A.*, through the scholarship program of *Programa de Bolsas Itaú* (PBI), and it is also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001, Brazil.

Any opinions, findings, and conclusions expressed in this manuscript are those of the authors and do not necessarily reflect the views, official policy or position of the Itaú-Unibanco and CAPES.

"The good life is one inspired by love and guided by knowledge."

–Bertrand Russell–

RESUMO

Recentemente, modelos compostos por apenas módulos neurais de Recuperação de Informação e Compreensão de Leitura de Máquina/Gerador de Texto baseados em modelos de linguagem pré-treinados alcançaram o estado da arte em vários conjuntos de dados desafiadores de processamento de linguagem natural. No entanto, ainda há espaço significativo para melhorias na capacidade de raciocínio desses sistemas, especialmente no domínio de perguntas e respostas complexas de domínio aberto (CODQA - *Complex Open-Domain Question Answering*). Neste projeto, propomos uma arquitetura que combina as principais características desses modelos dentro de uma configuração de Aprendizado por Reforço, com a capacidade extra de realizar múltiplos "saltos" entre documentos para responder às perguntas dos usuários. Um sistema com esta capacidade é fundamental para construir agentes conversacionais capazes de responder a perguntas complexas que requerem múltiplas consultas em uma base de conhecimento não-estruturada. Nossos sistemas alcançaram um F1-score máximo de 0.13 ± 0.3 no conjunto de teste, usando em média apenas 47% das passagens de texto totais disponíveis.

Palavras-Chave – Perguntas e Respostas Complexas de Domínio Aberto, Aprendizado Profundo por Reforço, Agentes de Conversacionais.

ABSTRACT

Recently, models composed of only a neural Information Retrieval and a Machine Reading Comprehension/Text Generator modules based on pretrained language models have reached the state of the art in several challenging natural language processing datasets. However, there is still significant room for improvement in the reasoning capacity of these systems, especially in the realm of complex open-domain question answering (CODQA) datasets. In this project, we propose an architecture that combines the main features of these models within a Reinforcement Learning setting, with the extra ability to perform multiple "hops" among documents to answer to users' questions. A system with this capability is critical for building conversational agents able to answer difficult questions that require multiple queries on a non-structured database. Our systems achieved a maximum F1-score of 0.13 ± 0.3 on the test set, using on average only 47% of the total available text passages.

Keywords – Complex Open-Domain Question Answering, Deep Reinforcement Learning, Conversational Agents.

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATION SYMBOLS

$|\cdot|$ - number of elements in a set

$\mathbb{E}[\cdot]$ - expected value

$\hat{\mathbb{R}}^l$ - Set of real numbers of dimension l, where each element is normalized between -1 and 1

C - a corpora

k - number of most relevant documents retrieved

k_R - number of most relevant documents retrieved in a RETRIEVE action

k_C - number of most relevant documents retrieved in a CLEAN action

q - a question, in natural language

a_g - ground truth annotated answer

a_r - answer extracted or generated by a reading comprehension (reader) module

j - refers to an instance of the HotpotQA dataset

t - timestep

p_t - a textual passage at t

P_t - set of all passages available up to t to answer q

P_g - minimal set of passages the annotators considered relevant to answer q

P_t^{new} - closest set of new searched intermediate passages at t

d_t - a string resulting from the concatenation of the initial question q with its all subsequent passages up to t

D - set of all possible permutations of the elements from $P \cup \{q\}$

$b(\cdot)$ - transformation that maps a string into its normalized real-valued multidimensional embedding represention

$p_1, p_2, ..., p_{t-1}$ - string concatenation of passages $p_1, p_2, ..., p_{t-1}$

β_C - positive reward of 0.1 given to the agent in the case of a successful CLEAN action

$R_{ANSWER}, R_{RETRIEVE}, R_{CLEAN}$ - reward functions respective to each available action

S - set of all states through which the agent can pass

A - set of all actions that the agent can take

A_s - subset of A available to the agent from the state s

R - reward function

T - state transition function

γ - discount factor

a_t - an action at timestep t

s_t - a state at timestep t

π - policy function

$V^{\pi}(s_t)$ - value of a state at timestep t under a policy π

$Q^{\pi}(s_t, a_t)$ - value of a state-action pair at timestep t under a policy π

LIST OF ACRONYMS

Acc - Accuracy

CODQA - Complex Open-Domain Question Answering

CNN - Convolutional Neural Networks

C4 - Colossal Clean Crawled Corpus

DRL - Deep Reinforcement Learning

EM - Exact Match

F1 - F1-score

ft - Fine-tuned

GPU - Graphics Processing Unit

IR - Information Retrieval

KB - Knowledge Base

KG - Knowledge Graph

LSTM - Long Short-Term Memory

MCQA - Multiple-Choice Question Answering

MDP - Markov Decision Process

MRC - Machine Reading Comprehension

MSCQA - Multi-Step Coarse to Fine Question Answering

MTurk - Amazon Mechanical Turk

NLP - Natural Language Processing

QA - Question Answering

QA-pair - Question-Answer Pair

R^3 - Reinforced Ranker-Reader

RL - Reinforcement Learning

RNN - Recurrent Neural Networks

Seq2seq - Sequence-to-sequence

CONTENTS

1 INTRODUCTION

It is estimated that around eighty percent of the knowledge produced is in the form of unstructured data, such as news, conversation histories, minutes, etc. This landscape affects different corporations and represents an opportunity to gain new valuable knowledge and insights (SHILAKES; TYLMAN, 2002; PLEJIC; VUJNOVIC; PENCO, 2008). At the Academy, the huge amount of articles generated per day makes the work of compilation and organization of information, even in more restricted topics, arduous.

Furthermore, with the popularization of smartphones and their virtual assistants on board, such as the Apple's *Siri* and the *Google Assistant*, as well as the domestic ones, such as *Alexa* from Amazon, there is also a growing demand for direct and concise answers to questions asked by users in everyday language – a format for presenting results that is different from the usual ranked lists of web pages, as in the results of *Google* and *Bing* (GAO; GALLEY; LI, 2018). As a result, the number of people potentially impacted by unstructured data analytics systems rises into the billions globally [1].

A crucial part of those Natural Language Processing (NLP) systems is the Question Answering (QA) one, responsible for traversing databases and answer the user's questions. Figure 1 outlines a common use case of this kind, in which a virtual assistant is called to answer a simple user's question. In particular, after the advent of the Transformer architecture (VASWANI et al., 2017a) and the pretrained models derived from it, such as BERT (DEVLIN et al., 2018) and T5 (RAFFEL et al., 2020), there has been a dramatic improvement in these systems in machine reading comprehension (MRC) tasks, in which the answer to a question is simply an explicit passage of text in a paragraph. In datasets like SQuAD (RAJPURKAR et al., 2016), these neural models have already gone beyond the baseline of human performance, as can be seen in the SQuAD's leaderboard[2].

[1] Ericsson Mobility Report: ⟨https://www.ericsson.com/en/mobility-report/mobility-visualizer⟩.
[2] SQuAD's leaderboard: ⟨https://rajpurkar.github.io/SQuAD-explorer/⟩

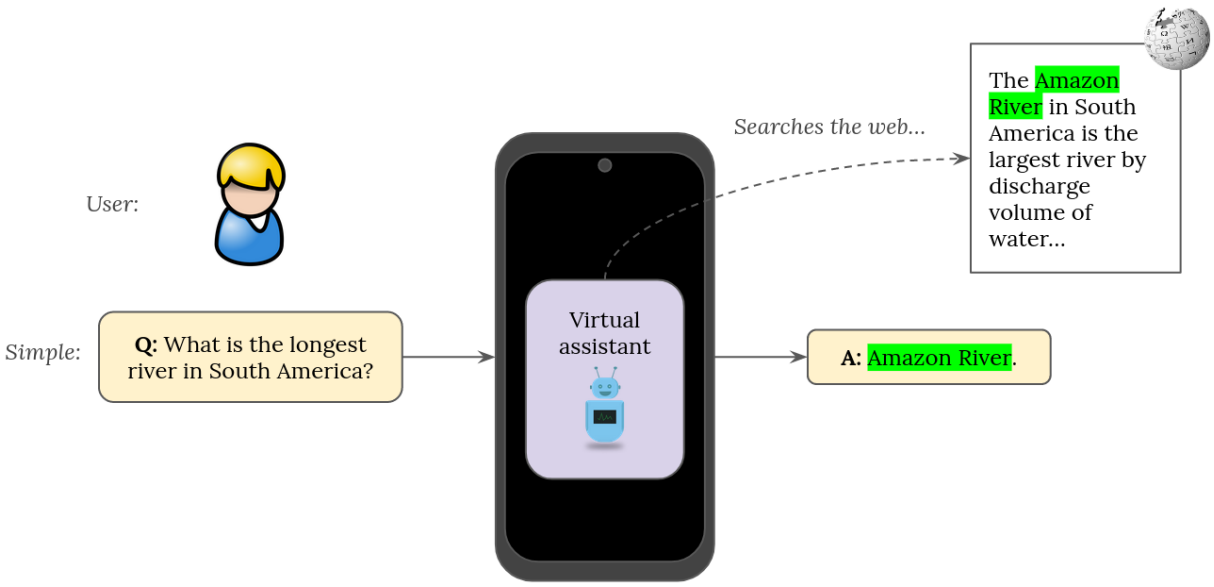


Figure 1: The core of the (simple) QA system, exemplified in a cellphone. Image created by the author.

1.1 The Role of Complex QA

However, these traditional MRC models are not suitable for dealing with so-called "complex QA". To answer questions that belong to this group, it is often necessary to *sequentially* collect and compose evidence spread across multiple documents, or develop some level of logical reasoning, such as the ability to compare two sentences (YANG et al., 2020). Figure 2 depicts a variation of Figure 1: now, the assistant is required to navigate ("hop") across two consecutive Wikipedia articles in order to provide an accurate response to the user's inquiry. We scrutinize these peculiarities in the subsection 2.1.2.

These represent a much more challenging QA task for current models than the first type of QA task ("simple QA") mentioned in the last section, as can be attested by comparing the top scores of the SQuAD (simple QA) leaderboad to those of HotpotQA (complex QA), one of the most popular multi-hop datasets available[3]. Yet, it is important to note that anyone routinely answers questions that require queries across multiple emails, books, or other notes. Thus, on the innovation side, developing better models capable of performing multi-hop reasoning in QA datasets represents an important advance on the NLP frontier and, on the practical side, opens up the possibility of massively scaling these capabilities to prohibitive amounts of data for humans (TALMOR; BERANT, 2018).

[3]HotpotQA's leaderboard: ⟨https://hotpotqa.github.io/⟩.

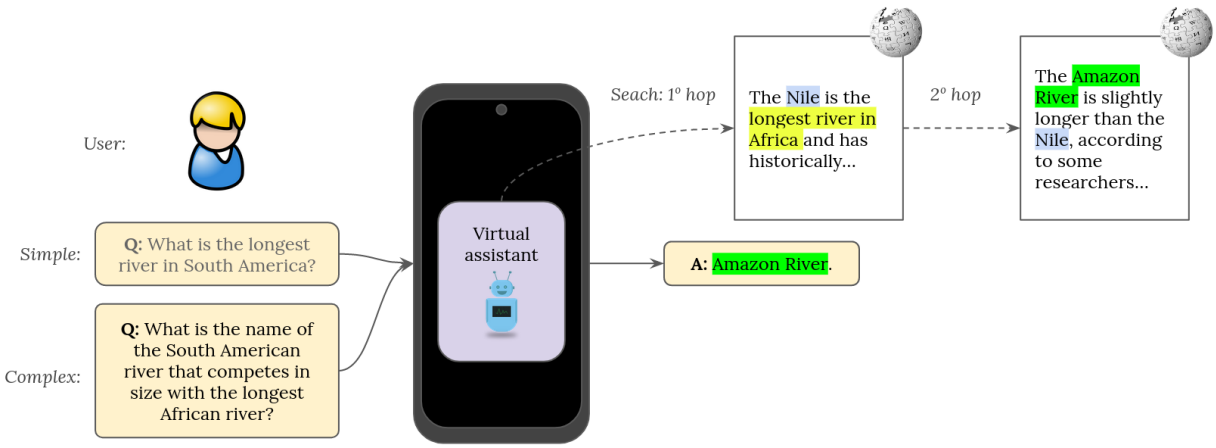


Figure 2: The core of the complex QA system, exemplified in a cellphone. Complex questions often require intermediate steps to accumulate information and successfully reach the answer. Image created by the author.

1.2 Leveraging Deep Reinforcement Learning on Complex QA

Among the most popular multi-hop QA datasets are: QAngaroo (WELBL; STENETORP; RIEDEL, 2018), QASC (KHOT et al., 2020), and HotpotQA (YANG et al., 2020). They provide the required intermediate annotated passages $\{p_0, p_1, ..., p_k\}$ between each complex question q and the corresponding ground truth answer a_g. This allows training under strong supervision. However, in real cases, such a set of passages is not known, even when the final answer is – numerous answers to complex questions are available on the internet without anyone explicitly presenting us the documents that fully support them. Therefore, reinforcement learning (RL) becomes a natural fit to the problem, since the intermediate passages are potentially unlabeled (WANG et al., 2018) and it is usually necessary to process them sequentially. In addition, the agent's exploration capabitility could lead to better or richer answers, and we can directly optimize it towards traditional NLP metrics, such as the F1-score or Exact Match, by injecting them into the reward definition (RAMAMURTHY; SIFA; BAUCKHAGE, 2020).

Despite the points raised, the literature on DRL models applied to QA problems such as that of complex questions, is still scarce. This could be due to the extra challenge of reframing the task as a sequential decision process, or because of the recognized difficulty in stabilizing DRL algorithms (HENDERSON et al., 2018). Hence, most current initiatives focus on supervised neural models, as can be seen in the HotpotQA's leaderboard, for example. Traditionally, these supervised QA systems rely on modules of neural readers or neural readers coupled with retrievers, when there is a need to gather more information

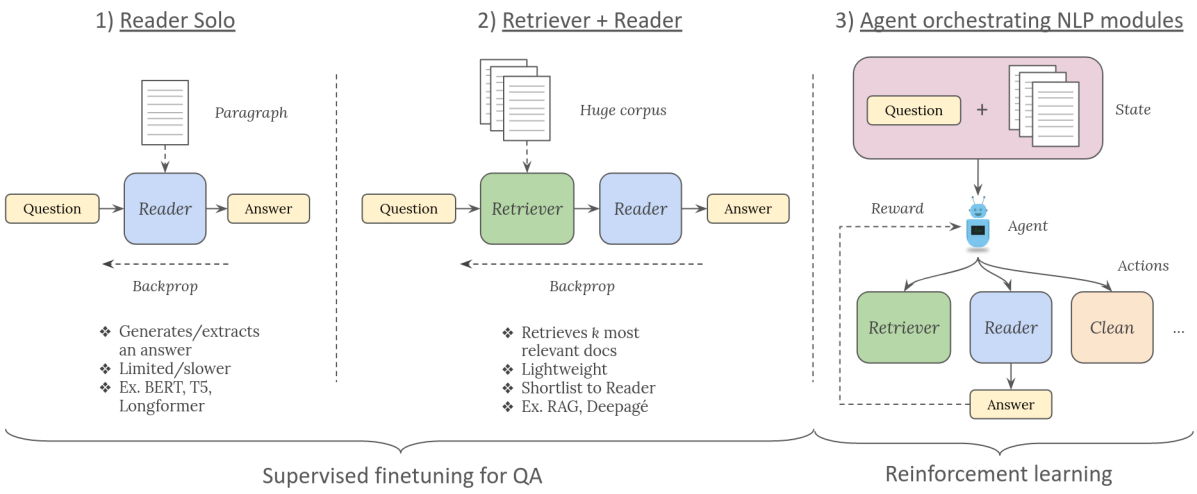


Figure 3: Comparison of different common architectures in QA systems. In the first two, a reader is responsible for generating the final answer – while in the second case, it is necessary to couple a retriever module to it to fetch additional information from a corpus. The figure on the right illustrates our solution, which integrates readers and retrievers as components orchestrated by an agent in a DRL setup. Image created by the author.

from a corpus. Figure 3 compares these traditional forms of QA systems to one based on a DRL setup – ours, in this case, which we will delve into further in the forthcoming chapters.

When DRL appears in the NLP literature, it is often related to text-based games, like TextWorld (CÔTÉ et al., 2019), or grounded language learning, as BabyAI (CHEVALIER-BOISVERT et al., 2019) and in (LUKETINA et al., 2019). Among the few examples of DRL applied to the multi-hop problem, in general they are dependent on a previously built knowledge graph (KG), as in the case of DeepPath (XIONG; HOANG; WANG, 2017), and do not work on simple plain text. Therefore, little attention and frameworks are devoted to standard NLP tasks, such as the presented ones, based solely on annotaded datasets (RAMAMURTHY; SIFA; BAUCKHAGE, 2020).

As a special exception, the NLPGym[4] recently released (RAMAMURTHY; SIFA; BAUCKHAGE, 2020) is very close to our approach, as it represents the first Reinforcement Learning (RL) framework analogous to the popular OpenAI Gym (BROCKMAN et al., 2016) with environments specifically designed to handle some standard NLP tasks such as sequence tagging, multi-label classification and multiple-choice question answering (MCQA). In order to model the agents and neural networks, the framework is integrated with the Stable Baselines library (HILL et al., 2018), which represent the state-of-the-art (SOTA) for DRL algorithm implementations.

[4]NLPGym's Github: ⟨https://github.com/rajcscw/nlp-gymlibrary1⟩

In the case of the MCQA task, Ramamurthy, Sifa and Bauckhage (2020) applied their framework to solve the QASC Dataset, an 8-way multi-hop multiple-choice QA dataset that comes with a large corpus on grade school science (the "8-way" expression means each question has 8 possible choices, from A to H, and only one is true). The two intermediate supporting passages (f_1, f_2) required for each question-answer pair (QA-pair) are annotated. Hence, a model typically needs to retrive important documents from the corpus and learn how to compose them in order to answer the questions, relying on (f_1, f_2) for supervision. When modeling, Ramamurthy, Sifa and Bauckhage (2020) abdicate the extra corpus and relies only on the concatenation of the question with the two annotated passages and, at each step, with the next key (A, B, ..., H), in order to compose the subsequent states of the environment. Although the MCQA task is the most challenging among the three studied in the paper, the work eventually achieves some promising results compared to the baseline of (which is an accuracy of $1/8 = 12.5\%$ in such case, once only one key is the correct one for each question), with its best model reaching 0.49 accuracy in the test set.

Whereas the work of Ramamurthy, Sifa and Bauckhage (2020) represents a clear advance in and a contribution to the interface between RL and NLP, which is still very little explored in the literature, there is plenty of room for exploration and improvement. First, datasets like QASC and ARC (CLARK et al., 2018) share unusual features with real-life practical cases, such as:

- Each question already comes with its possible keys; one of which is the true one – this allows the model to invariably hit the correct answer after a number of steps that is at most equal to the number of choices in the question. Instead, a more realistic scenario would require the answer to be extracted or generated merely from the supporting documents;
- "The topics are confined to scientific subjects that are typically only assessed in school exams, and do not involve questions about people, politics, economics, arts and other events that are common in practice.

Also, the Ramamurthy, Sifa and Bauckhage (2020)'s model intrinsically depends on previously annotated support passages to work, but this cannot be counted on in the more general case of the complex question problem. Another potential improvement already raised by the authors is that such system would certainly benefit from contextual embeddings of states, based on recent pretrained models, like BERT (DEVLIN et al., 2018).

Finally, NLPGym approach does not have a way to integrate information gathered on previous hops; it is limited to two passages and deals with them equally and simultaneously, rather than sequentially.

1.3 Contributions

This work reframes the more general problem of complex open-domain question answering (CODQA) as a deep reinforcement learning task (DRL). On the QA side, we address an open-domain explainable multi-hop QA dataset represented by the recently released HotpotQA dataset. The dataset is completely text-based and does not depends on a knowledge base (KB) and, unlike QASC and ARC, it does not provide choices to cheat the questions, but rather simple annotated answers, which is a more realistic and general scenario.

On the DRL side, our framework does not rely on a text-game-like structure and, instead, directly tackles the standard NLP QA task as a traditional dataset. RL is a natural fit for the problem, as complex questions often require sequential reasoning through the intermediate passages – which is different from the common Information Retrieval (IR) approach, that gathers all the relevant documents in a sigle turn. Furthermore, these intermediate passages have no labels in a real setting. Another bonus of this approach is that it allows us to directly optimize the system with common NLP scores such as F1-score or Exact Match (EM).

Hence, as our main contributions, we highlight:

- A DRL proposal for the more general problem of complex open-domain QA for which it is necessary to *extract or generate an answer* from the intermediate passages for each question and there are no predefined choices to guide the answer and to simply choose from. No text-game structure, as in (CÔTÉ et al., 2019), KB, as in (WELBL; STENETORP; RIEDEL, 2018; XIONG; HOANG; WANG, 2017), or golden intermediate passages, as in (RAMAMURTHY; SIFA; BAUCKHAGE, 2020), are required;

- Complete redefinitions of environment variables such as states, actions and rewards, when compared to Ramamurthy, Sifa and Bauckhage (2020)'s approach;

- Incorporation of recent advances in NLP into modeling, such as the use of contextual embeddings and new support document retrieval strategies, with the use o maximum

inner product search as a heuristic to select next passages and the aggregation of previous states into the current one, inspired by (XIONG et al., 2020).

1.4 Relevant Publications in the Context of the Present Study

As stated at the beginning of this manuscript, this work explores the interface between the fields of Natural Language Processing and Reinforcement Learning. So far we had the opportunity to accumulate practical experience with the various SOTA algorithms and techniques we mentioned in this document, which was fundamental for us to reach the maturation point where we are in this final stage.

To explicitly name them, at the *Brazilian Conference on Intelligent Systems* (BRACIS' 21), we published *DEEPAGÉ: Answering Questions in Portuguese about the Brazilian Environment* (CAÇÃO et al., 2021), in which we built a dataset on the Brazilian Environment by automatically filtering and translating a massive open-domain QA dataset and trained and compared multiple QA systems based on a IR and MRC setting composed by BM25 (ROBERTSON; ZARAGOZA, 2009) and a PTT5, a T5 model pre-trained in Portuguese language, (CARMO et al., 2020). In a paper called *DeepPolicyTracker: Tracking Changes In Environmental Policy In The Brazilian Federal Official Gazette With Deep Learning*, presented at the *38th International Conference on Machine Learning* (ICML'21) Workshop Tackling Climate Change With Machine Learning[5], we preprocessed government legal actions related to the Brazilian environment from the Federal Official Gazette and ft a BERTimbau model (SOUZA; NOGUEIRA; LOTUFO, 2020) to classify them in terms of potential environmental risk[6].

We also had the opportunity to colaborate to *Pirá: A Bilingual Portuguese-English Dataset for Question-Answering about the Ocean*, published at the *30th ACM International Conference on Information and Knowledge Management* (CIKM'21) (André F. A. Paschoal, Paulo Pirozelli, Valdinei Freire, Karina V. Delgado, Sarajane M. Peres, Marcos M. José, Flávio N. Cação, André S. Oliveira, Anarosa A. F. Brandão, and Anna H. R. Costa, 2021) and to *Interpretability of Attention Mechanisms in a Portuguese-Based Question Answering System about the Blue Amazon*, accepted at the *XVIII Encontro Nacional de Inteligência Artificial e Computacional* (ENIAC'21).

[5]This workshop did not publish proceedings, but the video of the work, as well as the associated paper, can be consulted at this link: ⟨https://www.climatechange.ai/papers/icml2021/35⟩

[6]This paper is now been extended to be submitted to the *International Conference on the Computational Processing of Portuguese* (PROPOR'22): ⟨https://sites.universidadedefortaleza.com/propor2022/⟩

After the practical knowledge accumulated in NLP, in special with QA that envolves IR in large databases, the next leading challenge was to precisely define the research gaps at the interface of the NLP and RL areas, as well as the most productive way to address them. The majority of research endeavors within this subfield remain in their formative stages, signifying a substantial room for exploration efforts. Simultaneously, the absence of robust standards entails a cautious approach. These twin factors serve as key driving forces behind the inception of this current investigation.

1.5 Organization of the Manuscript

This work is an endeavor on the frontier of NLP and DRL; thus, the document is structured accordingly. First, we established the required background of both fields in Chapter 2. Therefore, we start by presenting the problem we are addressing, the CODQA, positioning it in the broader literature of QA and conversational agents, and detailing the HotpotQA dataset that will serve to test our model. Next, we cover some important RL fundamentals. We close the chapter by introducing the Transformer framework and, in particular, some pretrained models that will be useful, in addition to examining the NLP metrics used in this work. Chapter 3 provides an overview of the related literature. In Chapter 4 we present the proposal of our systems and underscore the different influences from the works described in the previous chapter. Chapter 5 presents our results and provides a discussion on them. Finally, in Chapter 6 we conclude, highlighting our current contributions and limitations, as well as envisioning some promising future work.

2 BACKGROUND

In this chapter we present the more general scope of the CODQA challenge we are solving. We begin by describing the essential taxonomy of the QA area, a subarea of NLP, and the main historical approaches to the problem. In particular, we explain why they are not suitable for dealing with QA-pairs that require multiple "hops". Finally, we describe some of these most popular datasets in the literature, including HotpotQA, which will be our testbed.

2.1 Question Answering

Question Answering is the NLP research area dedicated to building systems (or "agents") designed to receive questions posed by humans and answer them in a concise and direct way. To do so, it can leverage one or more external data sources. Two common approaches to this problem are known as *KB-QA*, in which the agent relies on a KB as its previously structured database, and *text-QA*, in which only a collection of textual documents are available to the agent (GAO; GALLEY; LI, 2018; MANNING, 2021). In this work our focus will be entirely on the latter kind.

Once most of the first text-QA systems were designed to answer factoid questions, such as *"When was Nikola Tesla born?"*, whose answers can be extracted from text fragments, a typical QA system was composed of two sequential modules: an Information Retrieval (IR) and an MRC. The IR module is responsible for retrieving the k most relevant documents from the corpora C, where $k << |C|$ ($|\cdot|$ stands for the number of elements – e.g. number of documents here – in C), and passing them to the MRC module, which reads them and extract or generate an answer from the list of k documents (MANNING, 2021).

Most of the IR methods still in use are "sparse", like BM25 (ROBERTSON; ZARAGOZA, 2009). This means sentences are encoded as vectors where absent words are represented by 0, and present words by their term-frequency; consequently, the vectors are sparse since most of their components are null by construction. Nonetheless, recent neural retrievers,

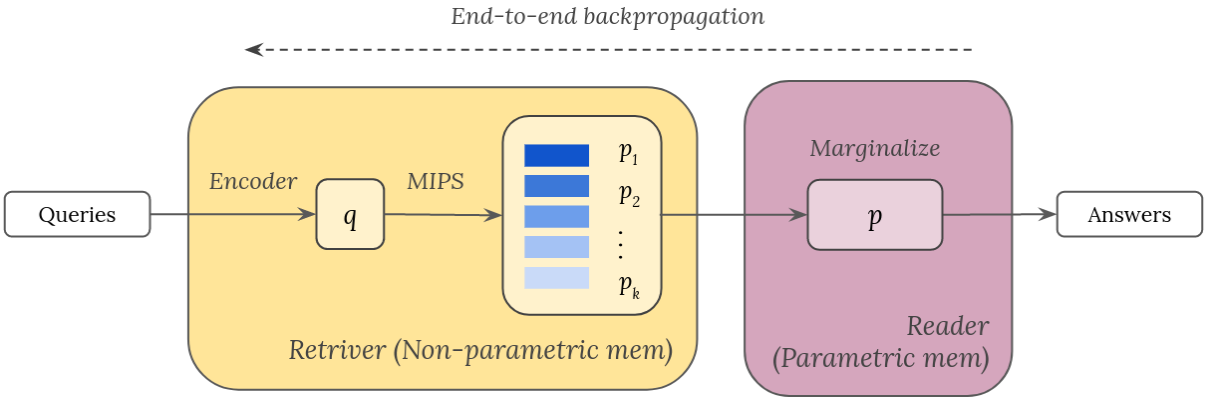


Figure 4: Example of "RAG", one of the most recent IR+MRC models, based on a neural Retriever and Reader ("Generator"). From a previously encoded corpus C, the retriever (DPR) receives and encodes a question q and ranks the k most similar passages to q via Maximum Inner Product Search (MIPS). These passages are sent to the Reader (BART) which generates a response as output. Image created by the author, adapted from Lewis et al. (2020b).

such as Dense Passage Retriever (DPR) (KARPUKHIN et al., 2020), have gained more space for being able to capture semantic features of sentences, by representing them as $\mathbb{R}^n$ vectors encoded by pretrained language models.

As for the neural MRC module, the most commonly adopted are based on recurrent neural networks (RNN), such as the LSTMs (HOCHREITER; SCHMIDHUBER, 1997), or also on pretrained language models like BERT (DEVLIN et al., 2018). The IR+MRC pair of modules is convenient because the MRC step is typically time-consuming due to its dependence on neural models with millions or billions of parameters.; thus, it would often be impractical to apply it directly to the entire corpus (WANG et al., 2018). Figure 4 shows an example of the Retrieval-Augmented Generation (RAG) model (LEWIS et al., 2020b). It is composed of a neural IR module based on DPR (KARPUKHIN et al., 2020) and a neural MRC one based on BART (LEWIS et al., 2020a), and their training is done jointly by backpropagation. To perform QA tasks, the corpus C is previously encoded and each question q is encoded in runtime by DPR. Then, DPR searches for the k most similar passages to q on C through Maximum Inner Product Search (MIPS), and sends these passages to the MRC component (BART). Once BART is a sequence-to-sequence (seq2seq) model, it finally generates an answer a_r from these passages.

The factoid question type previously described, as mention in the Introduction chapter, is also called "simple" and SQuAD is its canonical dataset (RAJPURKAR et al., 2016). In the case of SQuAD, paragraphs are so short that it is not even necessary to leverage an IR module; a standalone MRC model is enough. An example of this kind

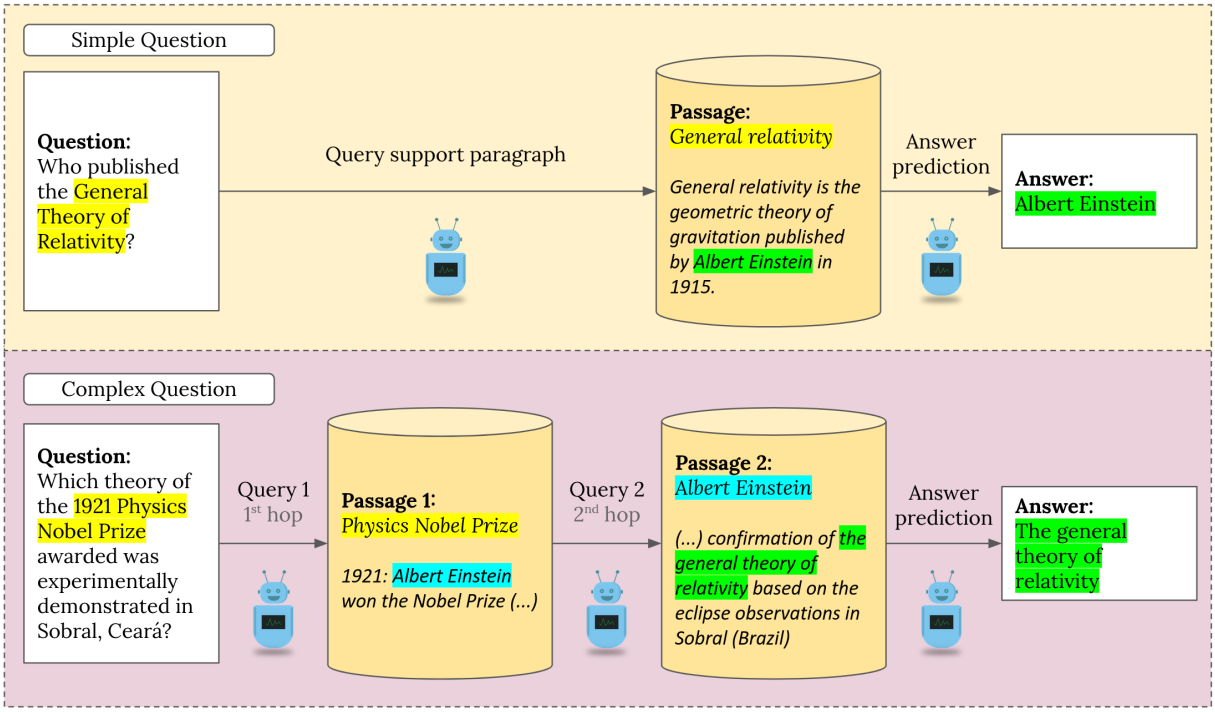


Figure 5: Comparison between an example of a "simple question", at the top, and a "complex question", at the bottom of the figure. In the first case, the answer is simply a span in a paragraph. In contrast, for a complex question, the agent must perform a sequential reasoning across multiple documents in order do gather enough information to answer it. Image created by the author.

of question is shown at the top of the Figure 5. It is noteworthy that human baseline of SQuAD has already been surpassed by current algorithms, as shown by the dataset's ranking.

In contrast, "complex" questions cannot be answered with a single text span from a paragraph, but require more elaborate forms of logical reasoning over the information presented and/or sequential processing of textual evidences spread across multiple documents. The reason is the question q itself often does not contain enough information for the IR module to rank the most relevant passages $p_0, ..., p_k$ in one single turn so that the MRC module can extract the correct answer from them. Instead, it is required a level of *compositional reasoning*, where the QA system must first leverage the knowledge posed by q to initially reach a supporting passage p_0, and then compose these latest two information (q, p_0) together to better select the third one p_1, and so on. This process of chaining subsequential facts stops when the tuple of gathered data $(q, p_0, ..., p_n)$ suffices to answer q. Moreover, the one-shot IR approach adds many potential and unnecessary distractors that burden the MRC module when drawing a coherent answer (XIONG et al., 2020). The previous sequential strategy is exemplified at the bottom of Figure 5 and Figure 6 presents two sequences of passages attempting to cover a_g.

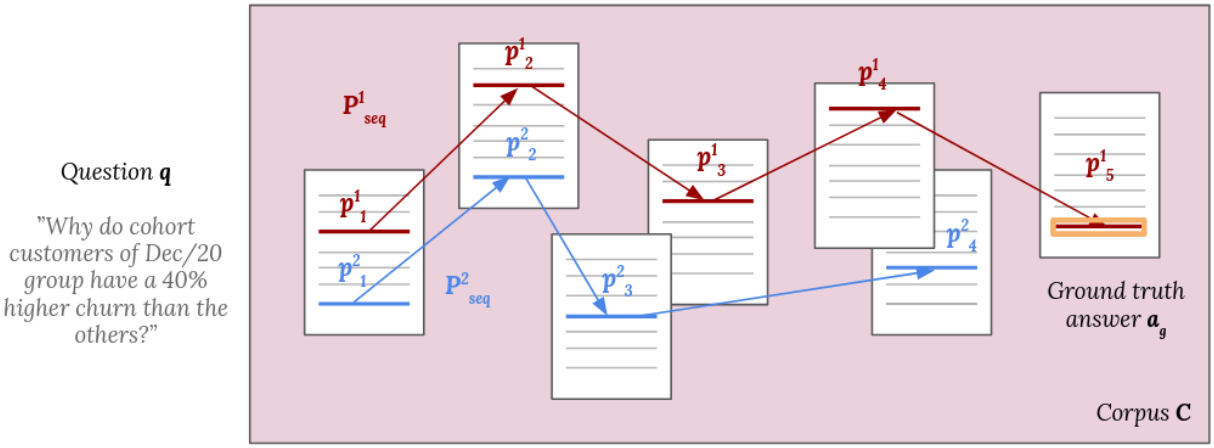


Figure 6: An illustration of two chains of reasoning via multi-hop over the documents in the corpus C. In this example, the red sequence P^1_{seq} had reached the ground-truth answer a_g for the question q after four hops, while the blue P^2_{seq}, after three hops, had not. Image created by the author.

Another important way to characterize a QA system is related to the breadth of the scope of subjects. QA systems built to answer queries only about specific topics, such as soccer or music, are called "closed-domain" QA and often run on a domain-specific ontology. Otherwise, when the system is not limited to a specific domain and aims to answer a question on virtually any subject, it is called "open-domain" QA. These systems are usually backed by universal ontologies or large encyclopedias, such as Wikipedia (ALLAM; HAGGAG, 2012).

Nowadays, end-to-end neural models like T5, pretrained in large databases like the Colossal Clean Crawled Corpus (C4) dominate as state-of-the-art in open-domain QA problems (RAFFEL et al., 2020). In the past, however, hybrid systems composed of multiple modules and parsers, such as IBM's Watson DeepQA (FERRUCCI et al., 2010), had already been able to defeat world champions in the trivia game Jeopardy!, in 2011 (ALLAM; HAGGAG, 2012).

2.1.1 QA System in the Broader Field of Conversational Agents

The QA systems described in the previous sections can also be thought of as components of a broader-purpose conversational agent, such as the one depicted in the Table 1, which illustrates an occasion in which the agent helps a user with a business decision. In addition to the QA agent, the other modules can be conceptually divided into two more types (GAO; GALLEY; LI, 2018):

- Social chats: where the agent is expected to have a social conversation with a human,

Table 1: Example of a dialog between a human user ("Person") and a conversational agent ("Agent") in which the performance of the three types of intelligent modules can be observed, with smooth transitions between each one of them. Table created by the author.

Turn	**Person/Agent**	**Conversa**
1	Person	Hello there
	Agent	Hello! How do you do?
2	Person	Actually, I'm a bit worried.
	Agent	What's worrying you?
3	Person	Revenues have been dropping since last month.
	Agent	Churn rates were up 10% in our main consumer market.
4	Person	Can you arrange a meeting with the local sales director?
	Agent	Sure, can I schedule it for next Wednesday at 2pm?
5	Person	Yes, please. Thanks!
	Agent	You are welcome!

without an explicit immediate demand. This agent generally needs to demonstrate more empathic behavior;

- Task completion: in which the agent receives a demand and needs to respond to it as efficiently as possible, like booking a movie at a virtual cinema box office.

In all of the above cases, there is the possibility for the agent to receive inputs (questions, demands, etc.) and return outputs (direct answers, empathetic continuation of the conversation and others) in the form of written or spoken dialogues, and a typical virtual assistant operates across the three domains, as in the Table 1.

2.1.2 Multi-hops QA Datasets

As outlined before, multi-hop QA (or, multi-step QA) datasets pose a much greater challenge for intelligent systems than traditional QA datasets dedicated to training MRC systems such as SQuAD. While in the latter answers to the questions are usually single and contiguous passages in short paragraphs, requiring no more than one hop, in the first case, the system needs to compose an answer from passages distributed in multiple documents or even perform logical reasoning through multiple sentences, like comparison (YANG et al., 2020).

We can frame current QA multi-hop datasets as KB-QA or Text-QA. Two important KB-QA datasets are QAngaroo (WELBL; STENETORP; RIEDEL, 2018) and ComplexWebQuestions (TALMOR; BERANT, 2018). QAngaroo was one of the first multi-step QA datasets. It consists of two datasets from distinct domains, both of which require

reasoning across documents. The first one, Wikihop, is built on general Wikipedia articles and the starting KB used is Wikidata[1]. The second, MedHop, is based on Medline abstracts[2] about proteins and drugs, and the objective is to detect potential interactions between drugs. ComplexWebQuestions is a large open-domain dataset focused on QA-pairs that require reasoning over multiple snippets crawled from the Internet. Each instance is accompanied by the SPARQL query originally used to assemble the answer from the Freebase database, a KB on general human knowledge (BOLLACKER et al., 2008).

One of the limitations of QAngaroo is that, due to the extractive construction process, the answers need to be explicitly present in the questions, as in the case of factoid questions, which makes them less realistic in practice (TALMOR; BERANT, 2018). Another relevant point is that both datasets are constrained by the extent of their own predefined schemas and incompleteness of entity relations in their KBs (YANG et al., 2020).

Among the Text-QA datasets, two pioneers are TriviaQA (JOSHI et al., 2017) and SearchQA (DUNN et al., 2017), in which multiple documents were retrieved from each QA-pair. However, because the document retrieval procedure is not sequential, but uses single-shot IR techniques instead in both cases, there was no guarantee that the questions required very elaborate multi-step reasoning procedures, as explained at the beginning of this section (YANG et al., 2020).

QASC is another Text-QA multi-step dataset, specially relevant to this work. Ramamurthy, Sifa and Bauckhage (2020), the closest work in the literature to ours, used it as a testbed for the NLPGym QA module. The dataset has 9,980 8-way QA-pairs in total about grade school science, and each one is present with two supporting facts, extracted from a 17M-sentence corpus. Each QA-pair and its related facts are assembled from an initial "combined fact" by crowdworkers (usually called "turkers" in such cases in the literature) from Amazon Mechanical Turk (MTurk)[3] in a way that it is not possible to resort to only one of the facts, but it is necessary to compose from both ones to properly answer the question (KHOT et al., 2020). Also, it is noteworthy that, as a multiple-choice dataset, the main metric of QASC is simply the accuracy; hence, its random baseline is $1/8 = 12.5\%$. Figure 7 presents an example from the dataset.

It is common for works that deals with multiple-choice QA datasets, such as QASC or ARC (CLARK et al., 2018), to use information present in the available choices themselves,

[1]Wikidata: ⟨https://www.wikidata.org/wiki/Wikidata:Main_Page⟩.
[2]Medline is searchable through PubMed: ⟨https://pubmed.ncbi.nlm.nih.gov/⟩.
[3]MTurk's website: ⟨https://www.mturk.com/⟩.

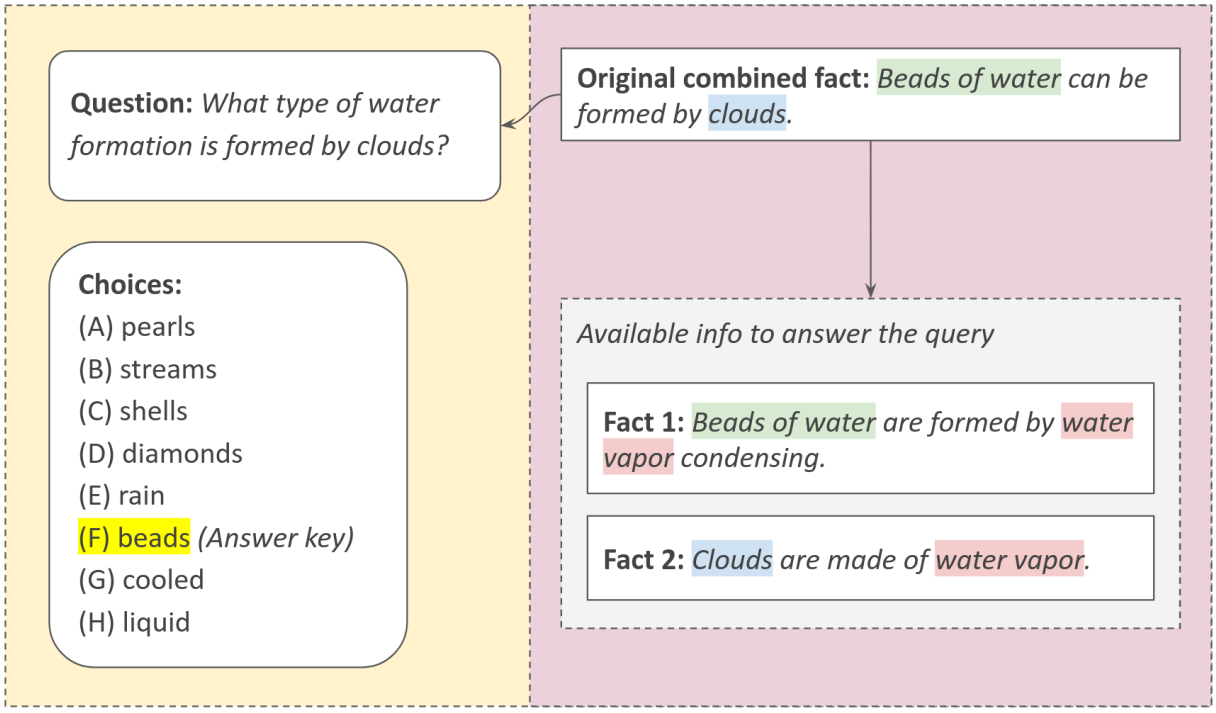


Figure 7: Example of a QA-pair with its available choices and associated supporting facts. The two chained facts and the QA-pair itself are assembled by crowdworkers from an original *combined fact*, present at the top right of the image. The key in yellow is the correct one, the "answer key". The words in green are the links between Fact 1 and the combined fact; the words in blue are the links between Fact 2 and the combined fact; and, finally, words in red are the links between the two facts themselves. As one can see, it is not possible to properly answer the question with only one of the two facts, but it is required to hop between them. Image created by the author.

either by composing each one with the question and using this "extended query" to retrieve relevant information in a corpus (PAULA et al., 2021), or by using them sequentially as states until reaching the correct answer (RAMAMURTHY; SIFA; BAUCKHAGE, 2020).

As mentioned previously, in this research we are interested in large-scale text-based CODQA datasets with very diverse questions and not restricted to high school science exam schemas. These requirements lead us to HotpotQA (YANG et al., 2020), a dataset with 112,779 QA-pairs, almost twelve times larger than QASC, also elaborated by turkers, focused on diverse, explainable multi-hop QA. Each QA-pair is accompanied by a list of supporting documents and the subset of those documents (the "golden passages") that turkers deemed necessary to respond to the question. This provides strong supervision so that the predicted passages can be compared to the annotated ones and provides a way to assess the explicability of the models that would be more difficult with only the correct answer available. Figure 8 shows an example of a QA-pair from the HotpotQA dataset.

As for diversity, most of the dataset's questions are about People (30%), Organiza-

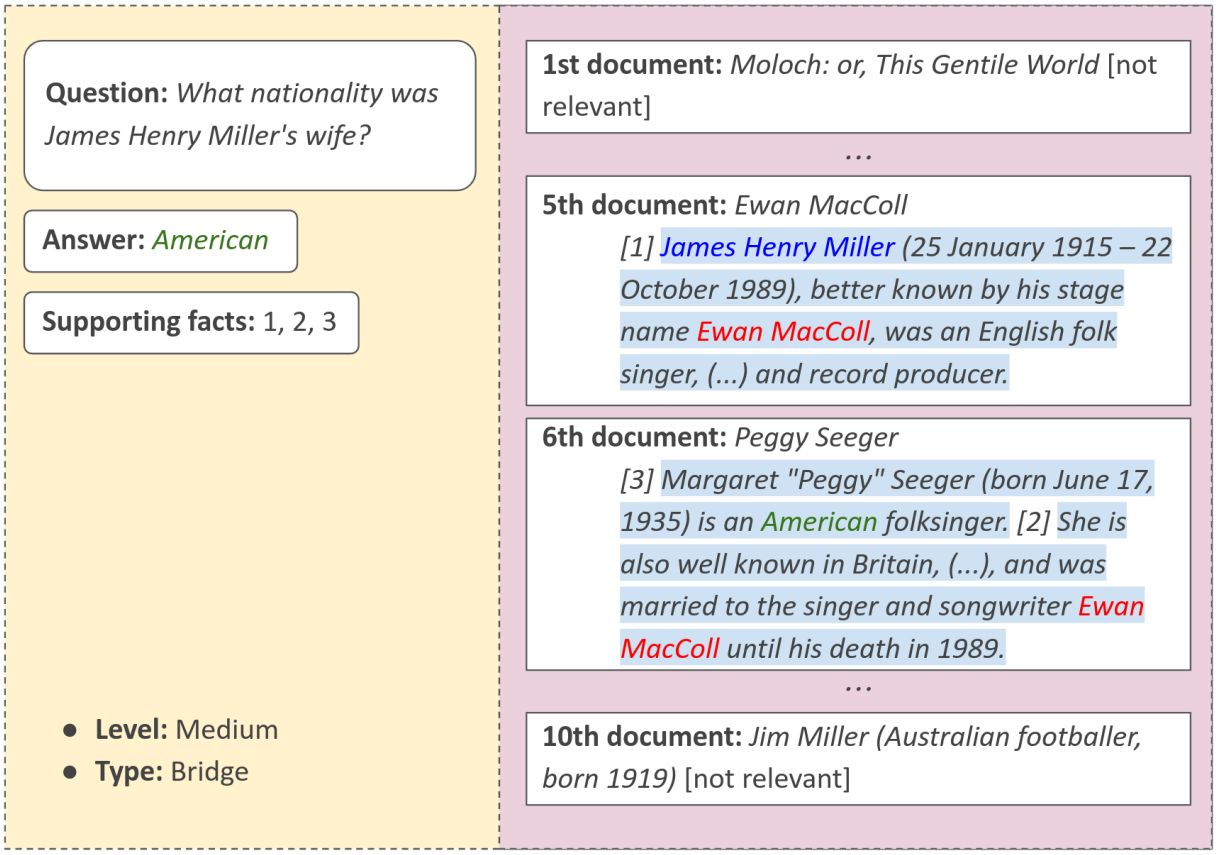


Figure 8: An example of medium-level QA-pair from the HotpotQA dataset. In this case, it is supplemented with ten documents, which are lists of snippets about a specific topic described in the document's title, as shown in the left panel. Here, only the documents that contain the annotated golden passages (1, 2 and 3) are shown. In blue, the expression most similar to the question. In red, the bridge terms between the 5^{th} and 6^{th} documents. The correct answer is written in green. Unlike comparison queries, this one's type is called "bridge", because it is necessary to find a term in each passage that allows the inference of the next one. Image created by the author.

tions (13%) and Locations (10%), followed by Date (9%), Numbers (8%) and Artwork (8%), as indicated in the Figure 9. There are also "Yes or No" questions (6%) and others about events and nouns to a lesser extent. Differently from other datasets, HotpotQA contains "Comparative questions" such as *"Which magazine was started first* ***Arthur's Magazine*** *or* ***First for Women****?"*. These kind of questions require at least two independent documents to be retrieved and compared in order to arrive to the answer. In this example, the model must somehow compare *"****Arthur's Magazine (1844–1846)*** *was an American literary periodical published in Philadelphia in the 19th century (...)"* to *"First for Women is a woman's magazine published by Bauer Media Group in the USA.* ***The magazine was started in 1989*** *(...)"* to finally reach the annotated answer, *"Arthur's Magazine"*. As for the type of questions in the dataset, there is a balanced distribution of WH-questions, such as "what" and "which", as expected, but also from

other constructs, such as those starting with “how”, “are” and “in”.

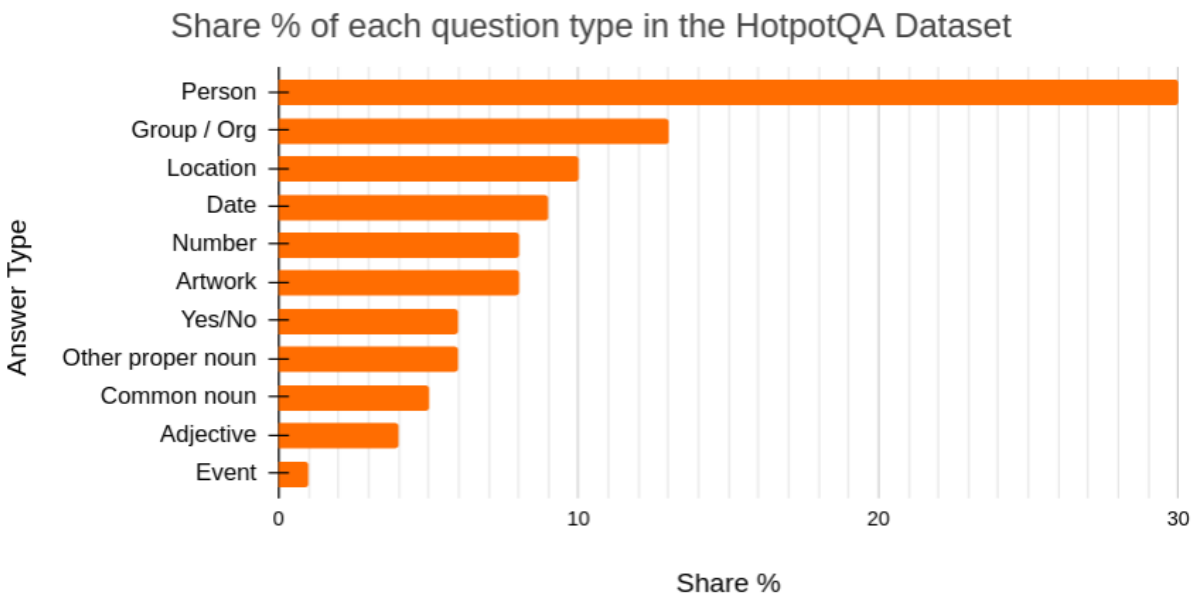


Figure 9: Distribution of question types in the HotpotQA dataset. Unlike the QASC dataset, HotpotQA covers a large range of subjects besides scientific subjects. Chart produced by the author.

The dataset was built upon the English Wikipedia articles as follows: the authors mapped all hyperlinks (to other Wikipedia articles) present in the first paragraph of each article. Thus, in the example shown at the bottom of Figure 5, to answer the question *“Which theory of the 1921 Physics Nobel Prize awarded was experimentally demonstrated in Sobral, Ceará?”*, it is first necessary to go to the article on “Physics Nobel Prize” to find out that “Albert Einstein” was the winner of this prize category in 1921. This term then serves as a *bridge* to the search for the next article, about Einstein himself, where it is stated that *“the general theory of relativity”* – the annotated answer – was experimentally confirmed in Sobral, Ceará.

A share of 16.0% (18,089) of the questions are answerable with only one of the available passes (they are single-hop) and are rated “train-easy”; 50.4% (56,814) are considered “train-medium” and require multiple hops to be answered; the remaining 33.6% (37,876) are, by exclusion, classified as “hard” and require a harder sequence of hops to arrive to the correct answer. This last set of hard QA-pairs is, in turn, divided into four groups: the “train-hard”, with 13.9% (15,661) of the total, belonging to the training set as well, and three other groups destined for validation and testing: “dev”, “test-distractor” and “test-fullwiki”, each with a share of 6.7% (7.405) of the total. In the “distrator” set, the 2 golden passages were mixed with 8 incorrect passages to add noise to the dataset. In the “fullwiki” set, golden passages were simply omitted to test the models’ ability to discover

the documents needed to answer without the extra supervision.

Considering the entire training dataset as a reference (90,447 instances), there is an average of 41 ± 11 supporting passages available per question, as shown in the left histogram at Figure 10. The string composed by the concatenation, for each instance, of the question with its respective supporting passages (an important concept we will explore in the Chapter 4) has an average of 1415 ± 385 tokens, with regard to the tokenizer based on the Sentence Transformers `all-MiniLM-L6-v2` model (REIMERS; GUREVYCH, 2019). The last distribuition is shown at the right side of the Figure 10. An observation that will be returned later from these distribuitions is that these questions could easily require a Reader able to handle more than $1,000$ input tokens – a common limitation in several state-of-the-art Transformers models, such as BERT (DEVLIN et al., 2018) and DistilBERT (SANH et al., 2019); but easily circumvented with the use of Longformers (BELTAGY; PETERS; COHAN, 2020).

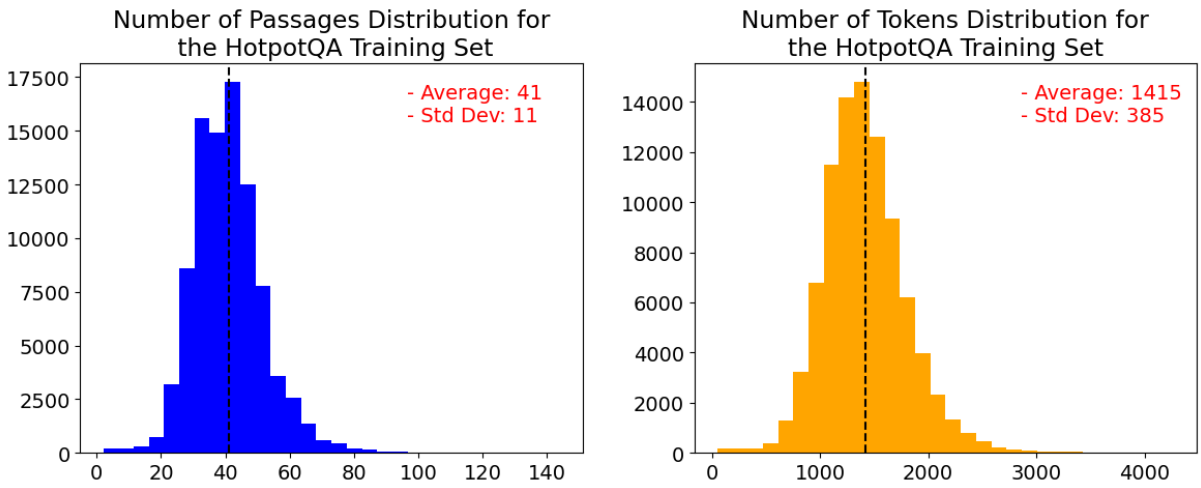


Figure 10: The left histogram shows the distribution of the number of the supporting passages available to answer each question of the HotpotQA training dataset. In the right side, the orange histogram presents the distribuition in the number of tokens for the concatenation of the question with all of its corresponding supporting passages. Both charts were drawn from 90k instances. Charts created by the author.

2.1.3 Evaluation Metrics

Once a model is trained, its performance is evaluated in the dev or test set. At this stage, three main evaluation metrics are common in most multi-hop datasets: the Accuracy (Acc), the Exact Match (EM) and the F1-score (F1). The last two ones are indeed the most dominant QA metrics in general.

Acc is suitable for multiple-choice QA datasets, such as ARC and QASC, where all

the possible choices are previously given; for instance, 4 for ARC and 8 for QASC. Then, a model must choose the right key, and it is evaluated by the ratio of its correct answers by the total number of QA-pairs (CLARK et al., 2018; KHOT et al., 2020).

EM has only two possible values for each QA-pair. It is 1 if the predicted answer is identical to the annotated one at the character level, and it is 0 otherwise. In contrast, F1 is a measure of the overlap between the bag-of-words versions of the predicted answer and of the true one.

A standard approach to preprocess the answer strings before calculate the aforementioned metrics was established by Rajpurkar et al. (2016). The following procedures apply to generated/extracted a_r and ground truth a_g answers. They normalize the answer by:

1. Changing all characters to lowercase;
2. Removing punctuations;
3. Removing articles, such as “a”, “an” and “the”;
4. Cleaning the remaining spare white spaces.

The EM score of the pair (a_g, a) is computed right from the resulting strings $norm(a_g)$ and $norm(a)$ after the 4^{th} procedure by a simple comparison:

$$EM = \begin{cases} 1, \text{if } norm(a_g) = norm(a) \\ 0, \text{otherwise.} \end{cases} \tag{2.1}$$

To compute F1, $norm(a_g)$ and $norm(a)$ are tokenized, producing the lists of tokens $t(norm(a_g))$ and $t(norm(a))$ respectively, and the number N_c of tokens common to both lists is counted. Thus, precision Prec and recall Rec are defined as:

$$\begin{cases} \text{Prec} := N_c/|t(norm(a))| \\ \text{Rec} := N_c/|t(norm(a_g))|. \end{cases} \tag{2.2}$$

Finally, F1 metric takes precision and recall into account as a harmonic average:

$$F_1 = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}. \tag{2.3}$$

The final metric values of both EM and F1 for the whole test dataset is simply the average over all individual instance scores – i.e., for each (a_g, a) pair (YANG et al., 2020).

2.2 Reinforcement Learning

The interaction between an agent and the environment, summarized in Figure 11, can be mathematically modeled as a Markov Decision Process (MDP), defined as the tuple (S, A, T, R, γ), where S is a set of states through which the agent can pass $\{s_1, ..., s_n\}$; A is a set of actions that the agent can take $\{a_1, ..., a_m\}$, where A_s is the subset of A available to the agent from the state s; T represents a state transition function such that $T(s_t, a_t, s_{t+1}) := p(s_{t+1}|s_t, a_t)$, where t demarcates the iteration; and R is the immediate reward received right after going to state s_{t+1}, starting from s_t for action a_t. That is, R is the function $R : S \times A \mapsto \mathbb{R}$. Finally, $\gamma \in [0, 1[$ is the "discount factor", the parameter responsible for calibrating the importance the agent gives to future rewards compared to the closest ones. (SUTTON; BARTO, 2018).

Figure 11: Classic configuration of an agent interacting with the environment in a reinforcement learning setting. Image created by the author.

Ideally, solving an MDP means finding a policy $\pi(s_t) \in A$ that maps each state $s_t \in S$ to an action $a_t \in A$, so that the decision-making agent transitions between the states of the *Markov chain* in such a way as to *maximize* the expected value of the accumulated sum of discounted rewards, i.e., $\mathbb{E}[\gamma^0 r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + ...]$ – we are considering here an infinite horizon situation, where we do not have a limited number of iterations t; in this scenario, conditioning $\gamma < 1$ guarantees the convergence of the geometric series (SUTTON; BARTO, 2018).

Under this *framework*, the *value* $V^\pi(s_t)$ of a state, given a policy π, can be understood as the accumulated value from s_t onwards (that is, taking $s_t = s_0$, the starting point), following π:

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | \pi, s_0 = s_t\right]. \tag{2.4}$$

Model-free RL algorithms can generally be organized into two categories: *Value-Based* and *Policy Gradient*, mainly distinguished by the action selection method; in the first case, the states s and actions a are mapped onto the functions $Q(s, a)$ (a variation of the value function defined in Equation 2.4) to then decide on a best action; in the second, the system's s states are mapped directly into the actions, directly optimizing the $\pi(s)$ policy (SUTTON; BARTO, 2018).

In the context of this work, we employed as the agent of our RL environment a Deep Q-Network (DQN), which is a method that combines deep neural networks and the Q-learning algorithm to solve the sequential decision-making problem we draw here. This agent takes the current state as input and outputs a Q-value for each possible action, balances the exploration-exploitation tradeoff by using an ϵ-greedy strategy and employs experience replay to enhance learning stability. By learning from the experiences found during training, DQN adjusts the network's parameters by minimizing the loss between predicted and the target Q-values, in order to improve its predictions over time. The algorithm 1 outlines these steps.

Algorithm 1 Deep Q-Network (DQN) algorithm

Initialize replay memory buffer D_B with capacity N
Initialize Q-network Q with random weights θ
Initialize target Q-network $\hat{Q}$ with weights $\theta' = \theta$
for episode = 1 to M **do**
    Initialize environment and state s
    for $t = 1$ to T **do**
        Choose action a using ϵ-greedy strategy
        Execute action a in environment, observe reward r and next state s'
        Store experience (s, a, r, s') in replay memory D_B
        Sample random minibatch of experiences (s_j, a_j, r_j, s'_j) from D_B
        Calculate target Q-values by setting $y_j = r_j$, for terminal s_{j+1}, or by setting $y_j = r_j + \gamma \max_{a'} \hat{Q}(s'_j, a'; \theta')$, otherwise
        Update Q-network by minimizing mean squared error:
$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_j \left(Q(s_j, a_j; \theta) - y_j\right)^2$, where B is the batch size for updating the Q-network and α is the learning rate
        Every C steps, reset $\hat{Q} = Q$
    end for
end for

2.3 Transformers

In comparison to the achievements in image recognition obtained through convolutional neural networks (CNN), the domain of natural language processing (NLP) has experienced a relatively slower progress. This discrepancy can be primarily attributed to the challenges associated with devising a neural network architecture capable of effectively harnessing the parallelization of operations within graphics processing units (GPUs), as well as ensuring the viability of transfer learning.

In the realm of language, two complicating factors persist: the significance of word order and the variability in the sizes of both inputs and outputs. Until a few years ago, some of the best neural architectures for problems of this type were RNN, in particular the LSTM (HOCHREITER; SCHMIDHUBER, 1997), which, in the case of encoder-decoder (also seq2seq) systems, such as translation, it is outlined in Figure 12, where x can be a string in English and y, one in French; the encoder and decoder modules correspond, in this case, to the recurrent application of a sequence of LSTMs:

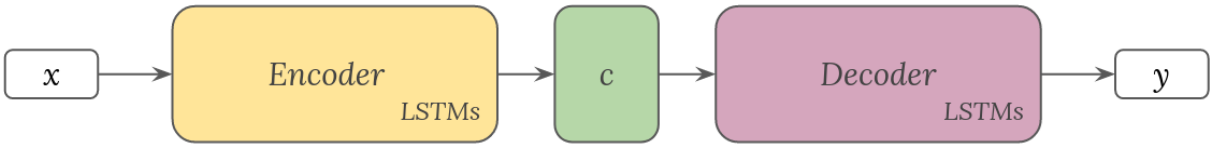


Figure 12: Illustration of a seq2seq network: x and y are the input and output strings, respectively, c is the context vector. On the left side, the encoder; on the right, the decoder. Image created by the author.

Although LSTMs handle long sentences much better than traditional RNNs, a bottleneck of this type of system is the fixed vector c, used by the encoder to represent the context of sentence x: the last words of x will weigh more than the first ones – and this is not due to any semantic aspect of the sentences, but the simple order of presentation of the words to the network (BAHDANAU; CHO; BENGIO, 2014).

The solution to the fixed vector c problem was to establish a linear relationship between the input and output words: instead of having just one vector c obscurely condensing the entire input context, each output word depends on a context that it will be a linear combination of the input words:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j, \tag{2.5}$$

where α_{ij} are the coefficients of the linear combination and h_j's are the hidden states of

the neural network (BAHDANAU; CHO; BENGIO, 2014).

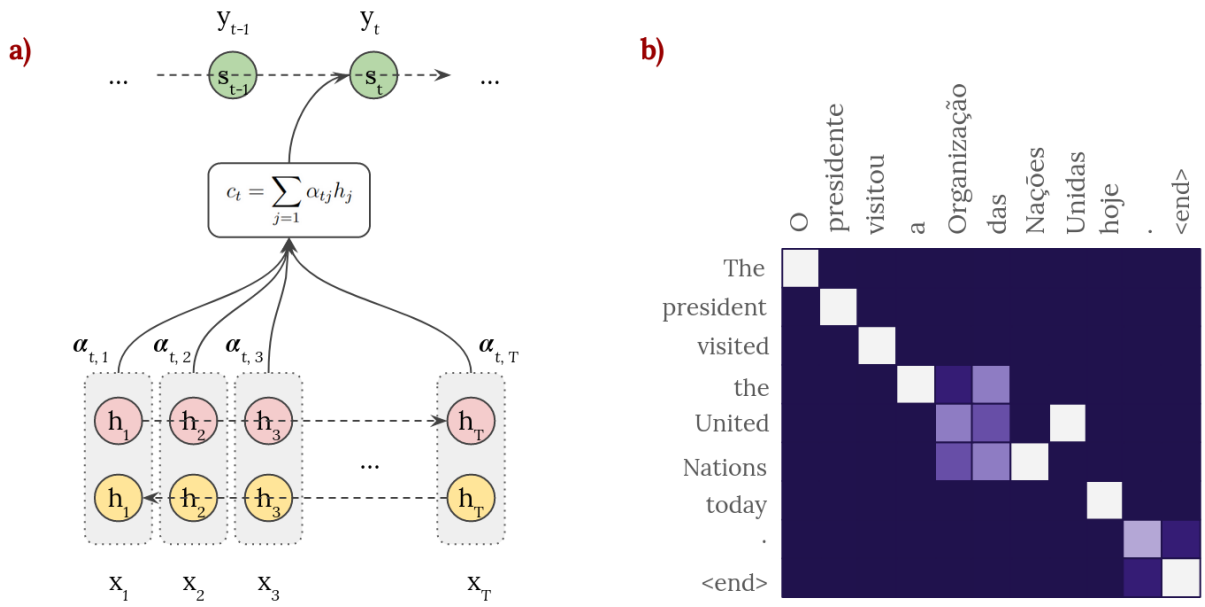


Figure 13: Attention with bidirectional RNNs in a machine translation task: a) Demonstration of an attention mechanism with a bidirectional RNN. Here x_t corresponds to Portuguese words and y_t to English ones; h_t and s_t are the hidden states and the α_{ij} are the linear combination weights of each context. b) Result of the alignment of the input sentence in Portuguese and its English translation: note that not only the main diagonal was highlighted, which indicates that the network captured non-obvious relationships between input and output. Image created by the author, adapted from Bahdanau, Cho and Bengio (2014).

This approach allows us to plot a sort of correlation graph between the input and output strings, as shown in Figure 13b: the clearer the square, the more associated is one word with another. It is to be expected, therefore, that the main diagonal will be the most intensely activated, as the translated phrase mimics the sequence of the original phrase: *visitou* is strongly linked (is the translation of) *visited*. But note that this does not happen with the *Organização das Nações Unidas* passage: while a naïve approach would align *Nações-United* and *Unidas-Nations*, the attention system correctly associated the two pairs *Nações-Nations* and *Unidas-United*.

The Transformer architecture (VASWANI et al., 2017b) completely eliminates the use of any type of RNN and only leverages attention mechanisms in place. This solves two bottlenecks of RNNs: forgetting "away" words, due to the attention mechanism, and sequential training, as they can benefit from parallel trainings on GPUs.

The architecture is shown in Figure 14 and consists of an encoder (the orange part, on the left, of the figure 14) and a decoder (the purple part, on the right, of the figure), both comprising multiple layers of self-attention, normalization layers and feed-forward neural

networks. The encoder processes the input string, capturing its contextual information, while the decoder generates the output string based on the previously encoded information and tokens generated.

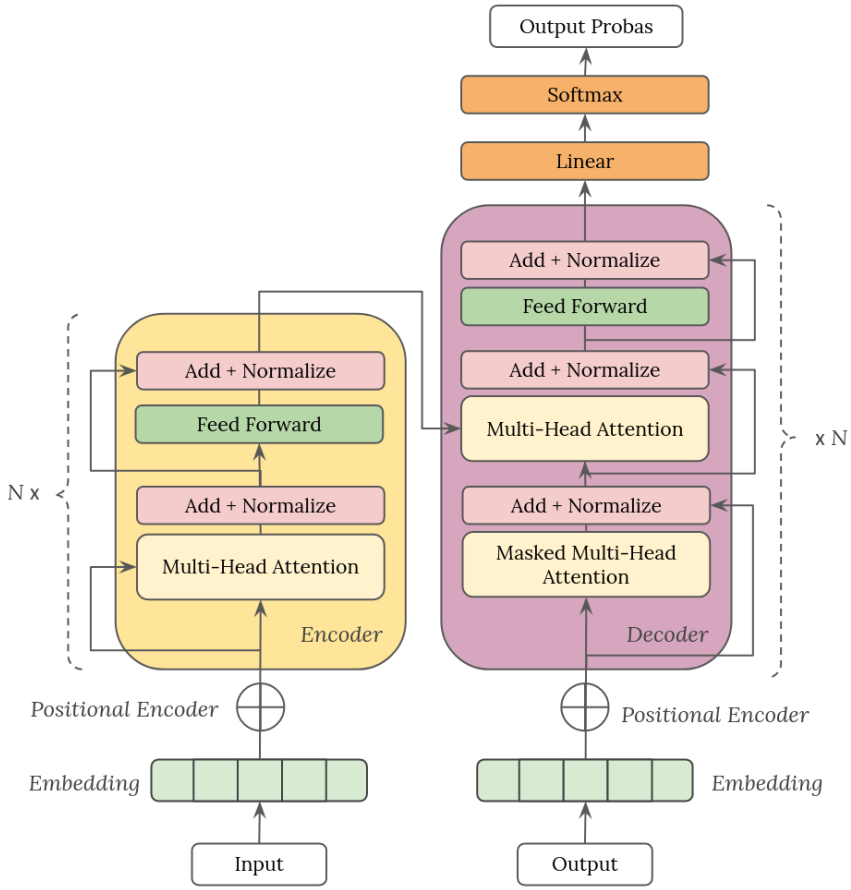


Figure 14: The Transformer architecture consists of an encoder (orange part, on the left) and a decoder (purple part, on the right), with multiple layers of self-attention, normalization, and feed-forward neural networks. By employing self-attention, the model captures relationships between sequence elements simultaneously, while positional encodings facilitate the understanding of token positions. The encoder processes contextual information from the input, and the decoder generates the output based on encoded information and generated tokens. Image created by the author, adapted from Vaswani et al. (2017a).

At the core of it is the concept of self-attention, illustrated by the light orange "Multi-head attention" blocks in both the encoder and the decoder modules, which allows the model to capture relationships between different elements within a sequence, simultaneously. Unlike traditional sequential models, where information is propagated sequentially through recurrent connections, the Transformer model computes attention scores across all pairs of positions in the input sequence. This attention mechanism allows each position to attend to all other positions, resulting in a comprehensive representation of the global context.

Furthermore, the Transformer framework incorporates positional encodings to imbue

the model with essential positional information. This addition is crucial since the inherent position-invariance of self-attention necessitates a mechanism to establish the relative order of tokens within the input sequence. By integrating positional encodings, the model gains a nuanced understanding of token positioning, enabling it to discern the distinctions between tokens based on their respective positions.

Finally, famous real-world functional models, such BERT, T5 or GPT, usually is made of a large stack of these basic structures, with multiple encoder and/or decoder blocks, so these models end up with millions or billions of parameters.

In recent years, there has been a surge in the popularity of pretrained language models utilizing the this Transformer architecture. These models have revolutionized every NLP task, such as classification, translation, and question answering by setting new benchmarks in terms of quality.

The training process of these models involves self-supervised learning using extensive databases, including comprehensive sources such as the Wikipedia. This pretrained stage equips the models with a strong initial understanding of language, enabling them to effectively address diverse language problems. Additionally, fine-tuning on smaller, domain-specific datasets further enhances their performance. When compared to models trained solely on these smaller datasets, the general outcome achieved by pretrained Transformer models is remarkably superior (RAFFEL et al., 2020).

3 RELATED WORK

Most recent approaches to NLP problems have mainly been end-to-end neural models (GAO; GALLEY; LI, 2018). Especially after the advent of massive pretrained Transformers models described in the previous chapter, such as BERT (DEVLIN et al., 2018), which established new standards for state-of-the-art in all classic NLP tasks (RAFFEL et al., 2020). In QA a similar process occurred, with intense exploitation of neural techniques, which can be attested by the leaderboards of the main QA datasets, such as the SQuAD. Other approaches to the problem, such as via RL, have not received much attention: a simple search in Scopus' indexed database with the keywords "reinforcement learning" and "question answering" returns only 126 results for the entire historical period until 2021[1]. However, an intense growth trend can be noticed in particular in the last 5 years, as shown in Figure 15.

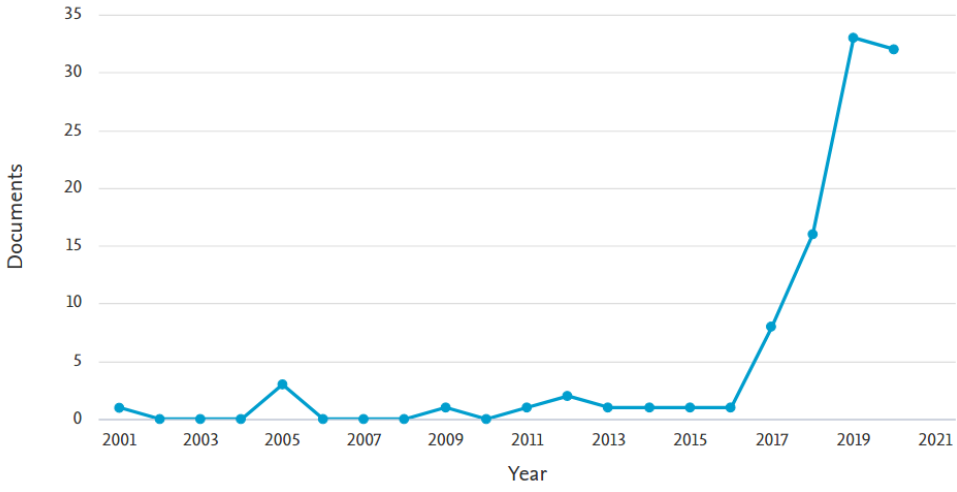


Figure 15: Articles with the keywords "reinforcement learning" and "question answering" indexed per year in the Scopus platform. Altogether, there are only 126 documents listed for the entire historical period, but an aggressive growth trend can be observed over the last 5 years. Figure extracted from the Scopus website.

In general, most of the works on the interface between RL and NLP are about text-

[1]Query used: "TITLE-ABS-KEY("reinforcement learning") AND TITLE-ABS-KEY("question answering")". Result processed in October 2021 at ⟨https://www.scopus.com/⟩.

based games (RAMAMURTHY; SIFA; BAUCKHAGE, 2020), such as TextWorld (CÔTÉ et al., 2019), in which the player advances through the game phases solely via textual instructions exchanged with the command prompt, or about grounded language learning, such as BabyAI (CHEVALIER-BOISVERT et al., 2019), in which the agent is trained to learn how to manipulate objects in a virtual environment. Traditional NLP tasks, such as summarization and QA, are mostly dominated by neural models.

3.1 Reinforcement Learning on KB-QA

Among the successful applications of RL in QA, there is an important set of them made for KB-QAs. In these systems, the KB is typically given by a graph $G(E, R)$, where E represents the set of nodes connected by the relations R, as in the case of M-Walk (SHEN et al., 2018). An MDP is defined in it so that the state s_t condenses all the history of previous nodes; the actions a_t can either answer with the information accumulated up to t or continue the search through new nodes; rewards are given only if the node associated with the true answer e_T is reached; finally, the transitions are deterministic. Thus, a neural agent with parameters θ is trained to learn a policy $\pi_\theta(a|s)$. Models analogous to this one are developed in the DeepPath (XIONG; HOANG; WANG, 2017) and MINERVA (DAS et al., 2017) models. Figure 16 illustrates the interaction between the agent and the defined KB environment.

Our work has similarities with the RL models of KB-QA, as will be demonstrated in the next chapter. However, we are interested in the more general problem of text-QA, where there is no ready-made KB available as the environment for the agent, but only QA-pairs and their respective documents. The next section presents the main works.

3.2 Reinforcement Learning on Text-QA

The Reinforced Ranker-Reader (R^3) (WANG et al., 2018) is an end-to-end framework to tackle open-domain QA datasets whose QA-pairs were not already accompanied by golden annotated passages, like in the SQuAD dataset (RAJPURKAR et al., 2016); instead, it requires searches in a large corpus C. Until then, the main approaches to this problem used an IR module (here, called *Ranker*) to initially list the first k passages from C most similar to the question q. This set $P = \{p_0, p_1..., p_k\}$ of passages were then sent to an MRC component (or, *Reader*), responsible for generating an answer based on P. A bottleneck in this procedure is that the overall quality of the answers depended directly

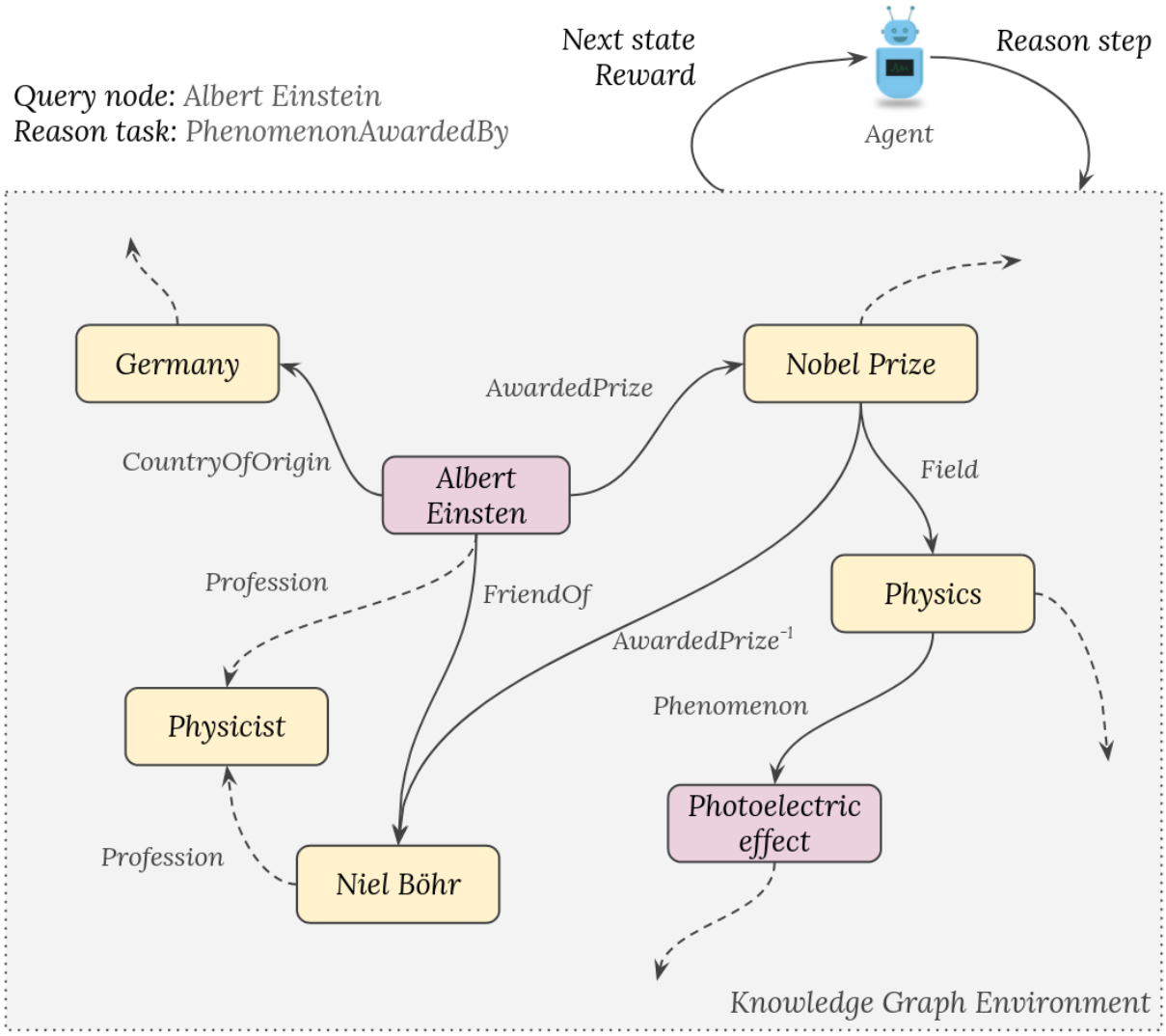


Figure 16: DeepPath's RL setting. A neural agent, represented on the right side of the picture, learns to reason through the KG by learning to map states (agent's position on the KG) to actions (paths to other nodes). The Figure above represents all the paths available as dotted arrows and paths explored by the agent as the bold ones. Image created by the author.

on the quality of the IR on retrieving congruent and useful passages from C (MANNING, 2021). In the case of sparse IR, for example, such as BM25 (ROBERTSON; ZARAGOZA, 2009), based on the relative frequency of the terms, passages with keywords common to the question are privileged, but not necessarily the most semantically aligned to q. The result of this is the inclusion of distracting passages in P that hinder the work of the MRC module. Table 2 illustrates an example of this phenomenon. The passages p_1, p_2 and p_3 are sorted in descending order by relevance, but p_1 clearly does not contain the correct answer and p_3 implies an incorrect answer as well.

To circumvent this issue, Wang et al. (2018) subjected both Ranker and Reader to joint training via REINFORCE (WILLIAMS, 1992), an RL policy gradient algorithm, so

Table 2: Example of situation where retrieved passages by the IR module degrade the MRC module. For the question *"What is the largest country in the world by land area?"*, whose answer is *"Russia"*, the highest-scoring passage, p_1, does not contain the correct answer, and p_3, although it does, refers to another entity. Table created by the author.

k	**Passage**
p_3	Canada is the second largest country in the world
p_3	Russia has 17 million square kilometers, making it the largest country in the world.
p_3	The European Union, together with Russia, is the largest producer of wheat in the world.

that the Ranker is trained to learn a policy $\pi_\theta(\tau|q)$, where τ is a passage, through rewards received according to the final quality of the answer generated a_r by the Reader when compared to the ground truth annotated answer a_g. In the Ranker, R^3 uses LSTMs to create the word embeddings H_1^{Rank} of the passages p_i and the question q and then maps them into fixed-dimensional vectors u_i with a max pooling operation. These vectors are finally concatenated and undergo a nonlinear transformation and used to compute the probabilities of each passage via a softmax operation, γ, such that $\pi_\theta(\tau|q) = \gamma\ _{tau}$, as illustrated on the left side of the Figure 17. Next, the selected passage τ is added by other negative passages (passages in which the annotated answer is not included), again vectorized as H^{Read} and processed by the Reader, responsible for extracting the answer a_r. Finally, the reward is computed by comparing a_r to the ground truth answer a_g by the following formula – the "else if" clause only means the remaining statements will not be executed if the first "if" clause is true:

$$R(a_g, a_r|\tau) = \begin{cases} 2, \text{if } a_g = a_r, \\ F_1(a_g, a_r), \text{ else if } a_g \cap a_r \neq \emptyset, \\ -1, \text{else.} \end{cases} \tag{3.1}$$

Thus, the reward computed by the Equation 3.1 is used to refine both the Reader, via standard backpropagation, and the Ranker, through REINFORCE. The system is shown in the Figure 17. At its time, R^3 reached state-of-the-art results in several datasets such as SQuAD and Quasar (DHINGRA; MAZAITIS; COHEN, 2017), even without the use of pretrained models. However, it was not intended to work on multi-hop questions.

Furthermore, the Multi-Step Coarse to Fine Question Answering System (MSCQA) (WANG; JIN, 2019) was designed to tackle QA on documents of variable sizes with a DRL setting in a multi-step manner. The system trains an actor-critic agent (KONDA; TSITSIKLIS, 2000) to learn how to efficiently handle documents of any size. As a testbed, they

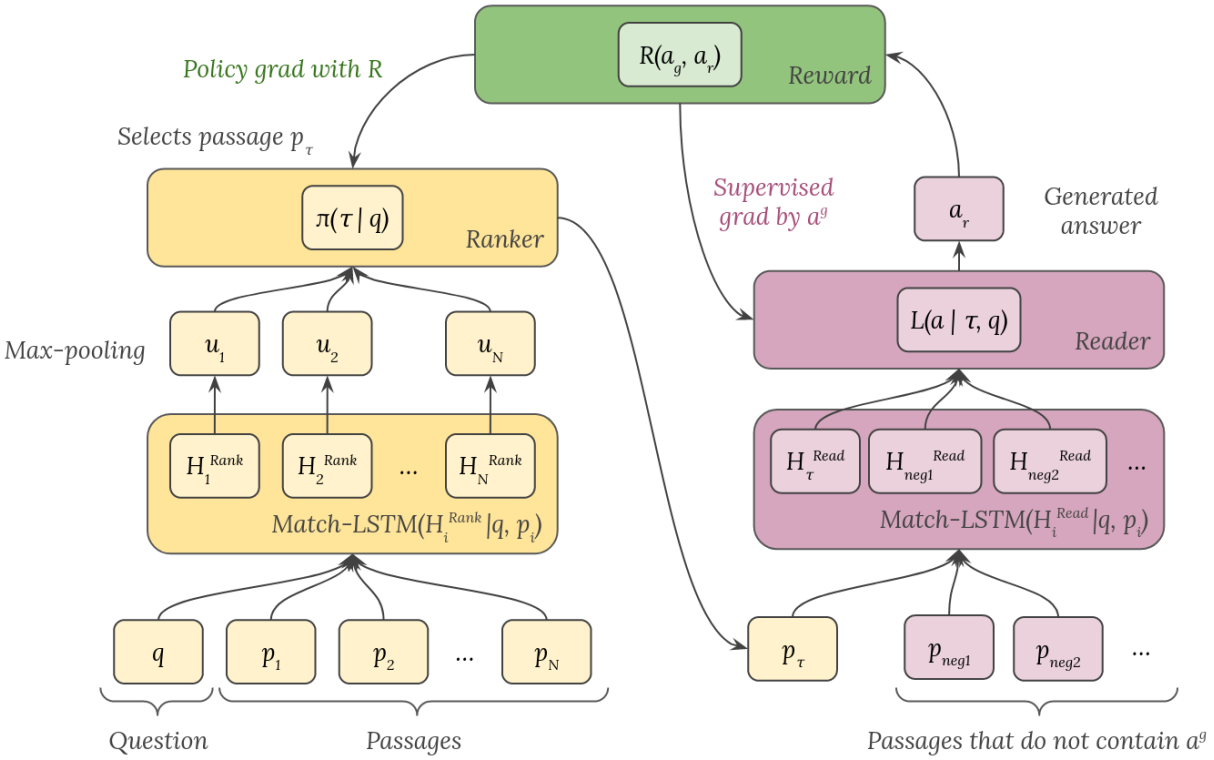


Figure 17: R^3 architecture. The Ranker (IR) and the Reader (MRC) undergo a joint training process. The question q and passages are encoded with an off-the-shelf encoder, and their embeddings are mapped to a fixed-size vector through a max-poling layer. Thus, the Ranker learns to map q to a best-scoring passage τ, which is paired with a set of negative passages to improve the learning process. Again, these passages are encoded and sent to the Reader. The answer a_r generated by the Reader is compared with the annotated one and a reward is calculated. Finally, this reward is used to adjust both the Ranker and the Reader. Image created by the author, adapted from Wang et al. (2018).

employed four datasets with supporting documents of different average sizes. For short documents, the SQuAD (RAJPURKAR et al., 2016) was used (122 tokens on average); for documents with intermediate sizes, CNN/Daily Mail (HERMANN et al., 2015) (763 tokens) and Wikireading (HEWLETT et al., 2016) (490 tokens); and for long documents, the Wikireading-Long (CHOI et al., 2017) (1.2k tokens) was used, which is basically a selection of QA-pairs with the longest supporting documents from Wikireading.

As Figure 18 demonstrates, the states are defined as the concatenation of the encoded question q with the encoded document D_t in step t. At each step, the agent can choose one of three actions – beside them, their respective rewards:

1. Simply generate the final answer a_r (terminal state). Reward: F_1-score(a_r, a_g);
2. Fetch new sentences from D_t. Reward: 1 if the new document $\hat{D}_t := D_s$ contains a_g, and 0 otherwise;

3. Remove the predicted answer a_r from the current state D_t, to avoid false positives. Reward: 1 if the remaining documents after deleting a_r contain the true answer a_g; 0 otherwise.

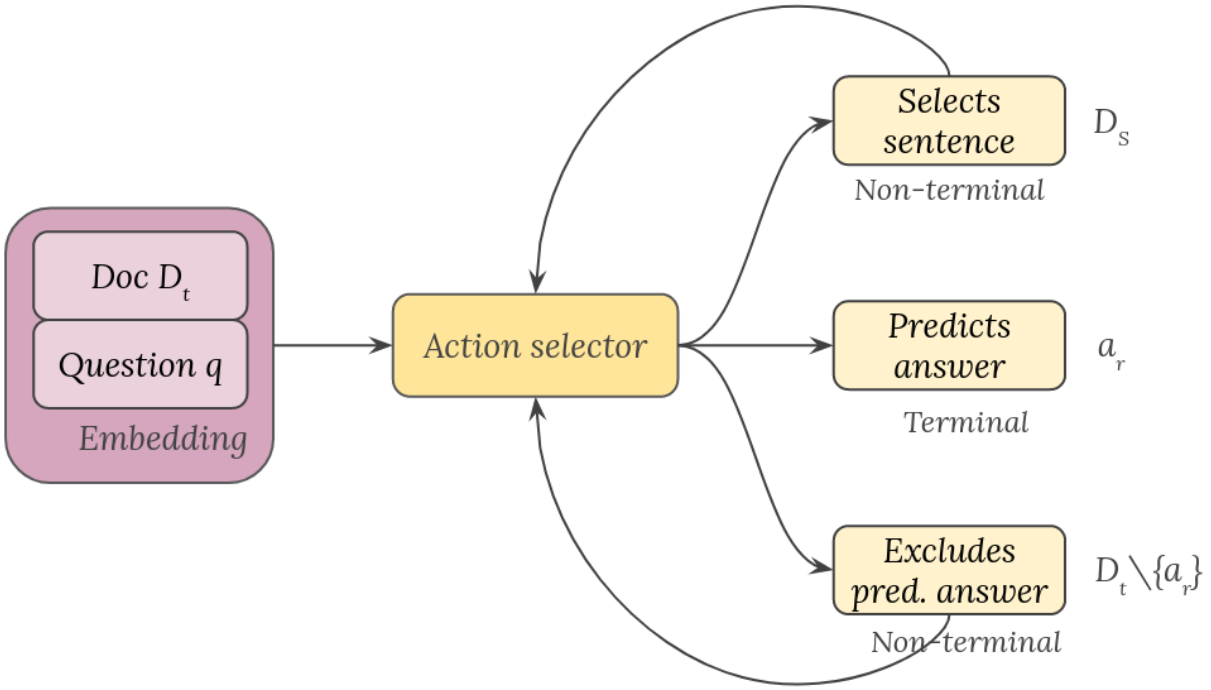


Figure 18: MSCQA architecture. States s_t are defined as the embedding of concatenation of documents D_t and the query q at the step t; at $t = 0$, D_t is just the original document that accompanies q. The agent is an actor-critic that learns to map each s_t to one of three actions. If *Predicts answer* is chosen, the agent simply extracts an answer a_r from D_t. If *Selects sentence* is chosen, it gathers a new sentence from D_s. If the agent selects *Excludes predicted answer*, the predicted answer a_r from D_t is considered a false positive and excluded from the current state definition. Image created by the author, adapted from Wang and Jin (2019).

The challenge of exploring the multi-hop QA problem in a DRL environment was also clearly setted in Ramamurthy, Sifa and Bauckhage (2020)'s work, as part of an effort to fill the lack of suitable DRL frameworks targeting standard text-based NLP tasks, such as sequence classification and sequence tagging. The open-sourced library is called NLPGym, and its code modularization is inspired by the architecture of the OpenAI's Gym library (BROCKMAN et al., 2016). This feature makes it compatible with other frameworks built to work on top of Gym, such as Stable Baselines (HILL et al., 2018), a collection of state-of-the-art DRL agents.

In the case of the multiple-choice QA task, Ramamurthy, Sifa and Bauckhage (2020) develop models to handle two datasets, QASC (8-way) (KHOT et al., 2020) and ARC (4-way) (CLARK et al., 2018), both based on grade school science exams. Since the two

algorithms are analogous, we will focus here on the modeling over QASC. In this case, as described in the subsection 2.1.2, each question q comes with two chained facts f_1 and f_2 and 8 possible choices c_t (from A to H for QASC), of which only 1 is correct. Each episode consists of a QA-pair instance. The MDP states are defined at each timestep t as the embedding g of the tuple (q, f_1, f_2, c_t). To create the embeddings from the transformation g, the authors adopted the pretrained word vectors from *fasttext* (JOULIN et al., 2017), with 300 dimensions and based on 1 million vectors.

Figure 19 shows an example of NLPGym's algorithm applied to a QASC question. The agent starts in state $s_0 = g(q, f_1, f_2, c_A)$, where $q =$ *"Organisms use fat to what?"*, $f_1 =$ *"Organisms use lipids to store energy."*, $f_2 =$ *"Another name for fat is lipid."* and $c_0 := c_A$, the first choice available, "A". At each timestep t, there are two possible discrete actions to the agent: *CONTINUE* or *ANSWER*. If the agent chooses *CONTINUE*, it proceeds to the next state, defined as $s_{t+1} = g(q, f_1, f_2, c_{t+1})$ and receives a reward of 0.0. If the agent chooses *ANSWER*, it selects the key c_t and receives a reward of 1.0 if it is the right answer (in the example, the agent reached the right key, "D", after four timesteps) or a reward of 0.0 otherwise, and the episode ends. Since there are a maximum of 8 choices for each pair, the episode ends on the last possible key, "H", also with an *ANSWER* action from the agent. For the QA-pair present in Figure 19, for example, the episode associated to the QA-pair is:

$$s_0 = g(q, f_1, f_2, c_A), a_0 = CONTINUE, r_1 = 0.0$$

$$s_1 = g(q, f_1, f_2, c_B), a_1 = CONTINUE, r_2 = 0.0$$

$$s_2 = g(q, f_1, f_2, c_C), a_2 = CONTINUE, r_3 = 0.0$$

$$s_3 = g(q, f_1, f_2, c_D), a_3 = ANSWER, r_4 = 1.0$$

As can be seen, in no decision the model takes advantage of past information, and each state contains only the trio of information associated with the QA-pair, q, f_1 and f_2, and the text of the current choice. Although this is not particularly serious here due to the structure of QASC questions themselves, with candidate choices independent of each other and only two available facts, in the more general case of complex questions, it is important to consider the facts sequentially, using accumulated past information to take the next actions (XIONG et al., 2020). Moreover, another potential flaw in the current version of NLPGym is that the QA model also refrains from making any use of its 17M-sentence scientific corpus.

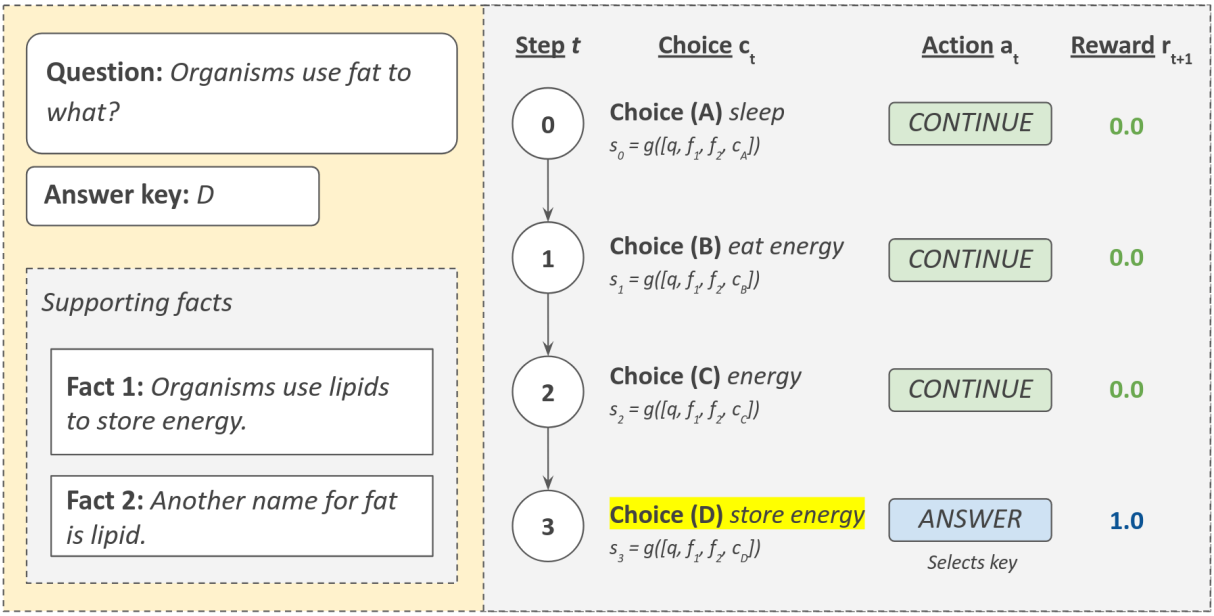


Figure 19: QA module of NLPGym on a multiple-choice question from QASC. The agent starts on a state composed of the question q, the supporting facts f_1 and f_2, and the first candidate answer, A. If it chooses the action *CONTINUE*, it moves to a new state, defined by q, f_1, f_2 and the next available choice, B, and receives a reward of 0.0. If it chooses *ANSWER*, it simply selects the current states's key; if it is the right one, the agent receives a reward of 1.0; otherwise, it receives 0.0 again. Image created by the author.

3.3 Research Gaps

Each DRL work presented in the previous section has its application niches in QA problems, and there is no "one size fits all" approach. There are robust works to handle QA in KBs (SHEN et al., 2018; XIONG; HOANG; WANG, 2017; DAS et al., 2017), to improve the lexical-semantic tradeoff between IR and MRC (WANG et al., 2018), to handle supporting documents of different sizes (WANG; JIN, 2019; CHOI et al., 2017) and also to answer complex multiple-choice questions typical from school science exams (RAMAMURTHY; SIFA; BAUCKHAGE, 2020; KHOT et al., 2020; CLARK et al., 2018).

However, we argue that no DRL work in the literature to date, to the best of our knowledge, addresses the problem of complex open-domain QA in its most common form in real cases: a question accompanied by sequences of documents that can contain the correct answers; all in plain text. In fact, QA work on KBs depends on the prior construction of a laborious schema or ontology that, in the end, is still limited by their own structures. Other influential DRL work on QA, such as R^3 and the MSCQA-based one, deal with open-domain plaintext datasets, but do not focus on the need for compositionality in generating responses to queries that require multiple hops.

Finally, the most adhering case to ours, NLPGym's MCQA module (RAMAMURTHY; SIFA; BAUCKHAGE, 2020) is still very seminal. It does not take into account the compositional and sequential nature of complex questions, and uses a method to build states that only concatenates available facts with given choices. The system has also not been tested on datasets that require more than two hops, because QASC (KHOT et al., 2020) only has two facts per QA-pair. In contrast, HotpotQA has an average of 41 thematic facts (passages) per QA-pair, such as those in the Figure 8. Also, the module was not prepared to take advantage of the available 17M-sentence corpus. Besides that, the NLPGym QA component has not been evaluated on open-domain datasets and relies, by construction, on the dataset structure to be multiple-choice, because it is not able to extract or generate a response directly from the set of facts available. This constraint makes the result of the action *CONTINUE*, for example, not very versatile, because it only lists the next candidate key in alphabetical order, without any more elaborate heuristics. These restrictions are critical to extending the system to less controlled scenarios than high school science exams.

Another limitation present in all these works is the use of contextual embeddings to represent queries, sentences and answers. In general, these systems are built on RNNs and their embeddings are classic, such as GloVe (PENNINGTON; SOCHER; MANNING, 2014) or Word2Vec(MIKOLOV et al., 2013). Ramamurthy, Sifa and Bauckhage (2020) used fasttext (JOULIN et al., 2017) in the QA module, but explicitly considers that the use of contextual embeddings like BERT's (DEVLIN et al., 2018) in the definition of states, for example, probably would lead to substantial improvements.

4 PROPOSED SYSTEMS

As stated before, the goal of our framework is to directly tackle the CODQA problem, without relying on KG nor on a full definition of an interactive game. It also does not rely on QA-pairs being accompanied with a previous set of keys, as in an exam, and, instead, is able to extract the answers right from the accumulated passages. In other words, this work aims to address the main gaps mentioned in the previous section. With this aim, we propose two different systems: two RL environments with slightly different agents. We named these two systems (the set of environment and its respective agent) *StringConcat* and *VectorSum*. Before we dive into both of them, next, we present some basic notations.

4.1 Initial Concepts

Our testbed is the HotpotQA, the text-based CODQA dataset described in the previous chapter. Its training set is composed of 90,447 instances. In our experiments, we used the entire training set. Each episode is defined as an instance $0 \leq j \leq 90,447$ of the dataset, comprised of its respective *question* q^j, a set of all the m^j *available passages* $P^j = \{p_0^j, ..., p_m^j\}$ to answer it, an annotated subset $P_g^j \subset P^j$ of *golden passages*, which is the minimal set of passages the annotators considered relevant to chain in order to answer the proposed question, and an *annotated answer* a_g^j – the subscript "g" stands for "golden" answer. In contrast, the *answer extracted or generated* by the reader module of our agent is called a_r^j.

Upon these basic elements of the dataset, at the step t of an episode j (to recall, an instance of the HotpotQA), we define $P_t^j = \{p_0^j, ..., p_{t-1}^j\}$ as the set of *all passages available up to t* to answer q^j. To properly pass these snippets to the Reader module, we define a *document* $d_t^j = [q^j, p_0^j, p_1^j, ..., p_{t-1}^j] \in D^j$ as a string resulting from the concatenation of the initial question q^j with its all subsequent passages, from p_0^j up to p_{t-1}^j, and D^j as the set of all possible permutations of the elements from $P^j \cup \{q^j\}$. Finally, we generate the state $s_t^j \in \hat{\mathbb{R}}^l$ for our RL agent operates on by applying the transformation $b : D \rightarrow$

$\hat{\mathbb{R}}^l$, which maps a document into its normalized real-valued multidimensional embedding represention, such that $s_t^j = b(d_t^j)$ and $-1 \leq ||s_t^j|| \leq 1, \forall j, t$.

For any episode j or step t, the actions a_t^j available can be drawn from the discrete 3-dimensional set $A = [0, 2] \subset \mathbb{Z}$, where 0 stands for the "*ANSWER*" action, 1 for the *"RETRIEVE"* one, and 2 for the *"CLEAN"* one. The rewards are given by $r_t^j \in \mathbb{R}$.

To render these concepts more tangible, in terms of data structures in a high-level computer language such as Python, q^j, each element of the set $\{p_0^j, p_1^j, ..., p_{t-1}^j\}$, a_g^j and a_r^j are strings, P^j and its different subsets and D^j are sets of strings, d_t^j is a list of strings, s_t^j is a normalized 384-dimensional array, A is a set of integers, a_t^j is an integer and r_t^j is a floating-point number.

Considering that we will adhere to the RL framework henceforth and frequently focus on the intricate dynamics of an arbitrary episode, we will simplify the notation from this point onward by implicitly omitting and concealing the superscript j in the subsequent expressions. Below is a real example that showcases some of the basic key elements found in a dataset instance, already formatted to be converted into the elements of our environment:

- **Question** q: *What type of system does the role-playing game created by writer Jenna Katerin Moran use to determine task resolution?*
- **Annotated answer** a_g: *Point-based system.*
- **Available passages** P (there are 28 of them):

  ["`Diceless role-playing game.` *A diceless role-playing game is a role-playing game which is not based on chance: it does not use randomisers to determine the outcome of events in its role-playing game system.",*

  "`Tabletop role-playing game.` *A tabletop role-playing game (or pen-and-paper role-playing game) is a form of role-playing game (RPG) in which the participants describe their characters' actions through speech.",*

  "`Nobilis.` *The player characters are "Sovereign Powers" called "the Nobilis"; each Noble is the personification of an abstract concept or class of things such as Time, Death, cars, or communication.",*

  ...,

  "`Jenna K. Moran.` *Jenna Katerin Moran, previously Rebecca Sean Borgstrom (born March 3, 1972) is a role-playing game writer."]*

- **Annotated golden passages** P_g: *["Nobilis", "Jenna K. Moran"]*

4.2 Supporting Transformers Models

Below are the two Transformers models used as auxiliary modules in both of our RL environments.

4.2.1 Passage encoder

In order to encode documents in runtime with $b(\cdot)$, we use the Sentence Transformer `all-MiniLM-L6-v2` model, which is designed for mapping English sentences into 384-dimension embeddings. The model is finetuned in a constrastive learning task, and it is specially tailored for semantic search and clustering operations, more efficient and faster than tradicional BERT-like models in metric computations (REIMERS; GUREVYCH, 2019). In this work, we used normalized vectors, between -1 and 1, in order to optimize the learning of the states accumulated by the agent. We also adopted a batch size of 256.

4.2.2 Reader

The reader module is a Longformer Base with 4096 input tokens, previously finetuned on the SQuAD dataset version 1.1 dataset[1] (RAJPURKAR et al., 2016). It works by receiving a question q and its context d_t, both as strings, and extracting the most likely text span from d_t as the answer a_r to q. The reason for a Longformer model instead of a tradicional Transformer lies in the fact that many instances of the HotpotQA dataset have several passages available and require more than 1,000 input tokens to be ingested, as can be seen in the distribuitions in Figure 10. Hence, in the worst case scenario, in which the agent requires lots of passages to answer q, a traditional Transformer such as BERT or DistilBERT would not be able to handle it since they are limited to less than a thousand input tokens.

4.3 RL Agent

For the DRL agent, we use the Stable Baselines 3 framework (HILL et al., 2018), which delivers state-of-the-art implementations of the lastest released DRL agents. For

[1] Off-the-shelf model: ⟨https://huggingface.co/valhalla/longformer-base-4096-finetuned-squadv1⟩.

the *StringConcat* system, our agent is a Deep Q-Network (DQN), with two layers of 64 neuros in each one, a learning rate of 5e-4, $\gamma = 0.99$, a buffer size of $20k$ and $20k$ steps available for the agent gathers experiences before start updating its Q-network. We trained the agent 3 times, for at least $500k$ episodes in each of them.

For the *VectorSum* system, on the other hand, everything remained the same, except for the learning rate, which was updated to 1e-4, the buffer size of $90k$ and the we raised up to $25k$ the number of steps available for the exploration phase.

4.4 States and Actions

The main difference between the *StringConcat* and the *VectorSum* systems lies on the state definition. For the *StringConcat* environment, the set of states S is infinite, and the state at timestep t, $s_t \in \hat{\mathbb{R}}^l$, is given by:

$$s_t = \begin{cases} b([q, p_0]), & \text{if } t = 0, \\ b([q, p_0, p_1, ..., p_{t-1}]), & \text{for } t \geq 1, \end{cases} \tag{4.1}$$

where q represents the complex question, such as *"The Oberoi family is part of a hotel company that has a head office in what city?"* and, in order to speed up the learning process, the first state is also already built-in with p_0, the first passage most similar to q; $b(\cdot) \in \hat{\mathbb{R}}^l$, $l = 384$ in this case, is a dense encoder based on a Transformer model, and $[\cdot]$ is the concatenation operation.

For the *VectorSum* environment, the state is $s_t \in \hat{\mathbb{R}}^l \times \hat{\mathbb{R}}^l$, with also $l = 384$, but given by:

$$s_t = \begin{cases} (b(q), b(p_0)), & \text{if } t = 0, \\ (b(q), b(p_0) + b(p_1) + ... + b(p_{t-1})), & \text{for } t \geq 1, \end{cases} \tag{4.2}$$

At each timestep t, in both environments, our value-based agent has three possible discrete actions: *ANSWER*, *RETRIEVE* and *CLEAN*, as in the work of Wang and Jin (2019). We limited the number of maximum steps the agent can take – or, the maximum length for each episode –, as a parameter called *MAXSTEPS*. In the case of the *StringConcat* env, $MAXSTEPS = 30$; for *VectorSum*, $MAXSTEPS = 20$.

In the first case, the agent triggers a reader module to extract an answer associated to the current state s_t and arrives into the terminal state. In the case of a *RETRIEVE*

action, the agent triggers the IR module, based on MIPS, that searches for the k_R closest set of intermediate passages $P_t^{new} = \{p_i, ..., p_{i+k_R}\}$ available such that their embeddings $\{b(p_i), ..., b(p_{i+k_R})\}$ have the maximum inner product with respect to s_t, conditioned to $P_t^{new} \not\subset P_t$, such that:

$$P_t^{new} = \{p_i \in P : \{\arg\max_{p_i}(s_t \cdot b(p_i))\}_{k_R}\}. \tag{4.3}$$

Thus, the agent advances to the new state $s_{t+1} = b([q, p_0, ..., p_i, ..., p_{i+k_R}])$ in the case of the *StringConcat* env, or to $s_{t+1} = (b(q), b(p_0) + ... + b(p_i) + ... + b(p_{i+k_R}))$, in the case of *VectorSum*, and continues the process.

As a quick reminder, it is worth noting that Equation 4.3 aims to select the passages p_i most similar to s_t, i.e, the vectors $b(p_i)$ with the smallest angle θ to s_t in the embedding space. This means minimizing with respect to θ and, since $\cos\theta \propto s_t \cdot b(p_i)$, maximizing the prior inner product.

If the agent chooses the *CLEAN* action, it will first try to extract an answer a_r from the current passages P_t at its disposal, and then eliminate all of them *containing* a_r. The agent would receive a small positive reward $\beta_C = 0.1$ if the annotated answer a_g is then contained in one or more of the remaining passages, and 0 otherwise. The motivation for this design is to teach the agent to gather the most lean set of passages required to answer q, and, at the same time, to remove passages that could lead to false positive answers. This action is also relevant to handle the traditional input tokens limitation found in most readers modules based on Transformers.

4.5 Rewards

In both *StringConcat* and *VectorSum* environments, the rewards are the same and defined similarly to the model present in Wang and Jin (2019), and depend on the action taken.

For the action ***ANSWER***:

$$R_{ANSWER} = \begin{cases} 2 + 1/|P_t|, & \text{if } F_1(a_r, a_g) = 1, \\ -1/|P_t|, & \text{if } F_1(a_r, a_g) = 0, \\ F_1(a_r, a_g) + 1/|P_t|, & \text{otherwise.} \end{cases} \tag{4.4}$$

For the action ***RETRIVE***:

$$R_{RETRIEVE} = \begin{cases} 1/|P_t|, & \text{if } a_g \text{ is present in the newer passages,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

And for the action ***CLEAN***:

$$R_{CLEAN} = \begin{cases} \beta_C, \beta_C \geq 0, & \text{if } a_g \text{ is present in the remaining } d_t, \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

The insights behind this reward shaping is to prompt the agent to efficiently find the right supporting passages p_i towards the answer a_g, avoiding unnecessary passages, which could, all together, overflow the input token capacity of the reader, even for a Longformer model in a more general scenario. Another consequence of the success of such behavior would be an improvement in the explainability obtained with the output answer a_r.

4.6 Example

To illustrate a typical episode, consider that in the beginning of it, when $t = 0$, the agent starts with the document $d_0 = [q, p_0]$ ($P_0 = \{p_0\}$), which basically consists of the question itself concatenated with its first most similar (aka., closest in the embedding space considering their inner product) passage p_0. For the sake of this example, we assume the agent is in the *StringConcat* environment – if it was in the *VectorSum* environment, the only difference would be in the computation of the states stage. Thus, the first state is this initial document encoded by a Transformer model, $s_0 = b(d_0)$.

At $t = 1$, the agent faces three different options: *ANSWER* the question directly with the available information, *RETRIEVE* more textual passages to have more data for a future answer attempt, or *CLEAN*, in order to remove potentially misleading passages. For the sake of this example, suppose the agent chooses the *RETRIEVE* action. Then, it searches for the k_R closest encoded passages $\{b(p_1), ..., b(p_{k_R})\}$ to s_0 and concatenate their passages p_i with d_0. Hence, it ends up with $d_1 = [q, p_0, p_1, p_2..., p_{k_R}]$ and the new state at this stage is $s_1 = b(d_1)$. If the annotated answer a_r is present at least in one of the *newer retrieved passages*, that is, a_r is a substring of $p_i \subset P_1 = \{p_1, p_2..., p_{k_R}\}$, the agent receives a positive reward $1/|P_1|$; otherwise, it receives 0.

Next, at $t = 2$, consider the agent chooses the action *CLEAN*. In this case, it tries to extract an answer a_r from the current available passages in P_1 and then *remove* all passages

$\{p_i, ..., p_{kC}\}$ from P_1 that *contain the extracted answer* a_r. If the annotated answer a_r is present in the remaining passages (that is, a_r is a substring of $p \in P_2 = P_1 \setminus \{p_i, ..., p_{k_C}\}$), the agent receives a small positive reward β_C; if a_r is not a substring of any element of P_2, it receives 0. Finally, the output state will be $b(d_2)$, where d_2 is the concatenation of q with all elements of P_2.

This process unfolds likewise until the agent reaches a predefined maximum number of steps *MAXSTEPS* or chooses to answer q with the accumulated information so far P_t (choosing the action *ANSWER*). In this last scenario, the agent leverages a Longformer model finetunned on the SQuAD dataset[2] to answer q. If a_r totally matches a_g, it receives a reward of $2 + 1/|P_t|$; if a_r is completelly wrong ($F_1(a, a_g) = 0$), it is punished with $-1/|P_t|$; for any other intermediate case, its reward is $F_1(a, a_g) + 1/|P_t|$.

Then, the episode ends and a new one starts as a sampled QA-pair from the 90k-traning set. We proceed this way up till the end of the predefined number of timestemps. Figure 20 resumes the proposed model learning from the QA-pair presented in the Figure 8.

[2]The model is available in the HuggingFace Hub: ⟨valhalla/longformer-base-4096-finetuned-squadv1⟩

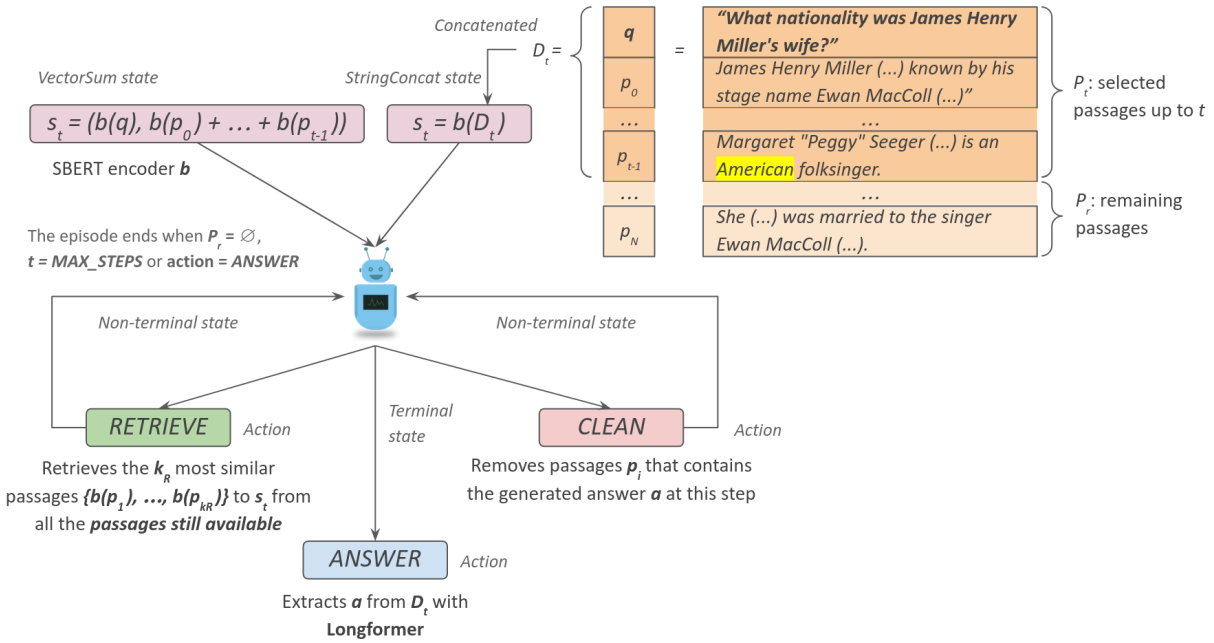


Figure 20: Our proposed systems. We have two different environments, *StringConcat* and *VectorSum*, which differs in the state definition and progression. In the *StringConcat* env, the states are $s_t \in \hat{\mathbb{R}}^l$, made upon the embedding of the concatenation of the passages up to t. On the other hand, in the *VectorSum* env, $s_t \in \hat{\mathbb{R}}^l \times \hat{\mathbb{R}}^l$, and the states are made of embeddings of both the question q and the sum of the embeddings of each passage up to t. The agent starts at s_0 when $t = 0$ and, at each step, it has three available actions. If it chooses *RETRIEVE*, it receives a new set of passages P^{new}, to concatenate to the previous ones in the document D_t. If it chooses *CLEAN*, it removes k_C passages, extracts an answer from the remaining passages, and, if this answer is the annotated one, it receives a positive reward. The agent arrives to the terminal state if the available passages p_i is exhausted for the QA-pair, if the agent reaches a maximum predefined number of steps *MAXSTEPS*, or chooses *ANSWER*. In the latter case, a Longformer model finetuned on the SQuAD dataset is activated to extract an answer a_r from D_t. In this case, the agent receives a new reward proportional to the similiarity between a_r and the ground truth answer a_g. Image created by the author.

5 RESULTS AND DISCUSSION

5.1 Initial Considerations

It is challenging to convert a traditional NLP CODQA dataset into a sort of textual game and then properly guide a DRL agent towards what we first considered a desirable outcome. In the work of Ramamurthy, Sifa and Bauckhage (2020), most of their models, DQN and PPO-based ones, failed in the QA task, not demonstrating significant improvement in learning an effective strategy for answering the questions, even after $50k$ steps in a simpler multiple-choice QA dataset. The reasons for this difficulty, both in our case and in theirs, can be multiple, from the RL realm, such as:

- A poor feature engineering in the design of the observation space;
- A deceptive reward function;
- The absence of a potentially necessary action;
- The typical instability found in many RL models;
- A bad choice of hyperparameters.

Moreover, other sources of noise could be present in the auxiliary NLP modules, such as:

- An embedding or Reader model not trained in the language or in the task they are being used for;
- A Reader module with a very limited input of tokens for the typical size of context it will have to deal with;
- A Reader based on an embedding model of lower semantic richness;
- An inadequate retriever for the task;

- A Transformer embedding model not specifically finetuned with a constrastive learning objective (important for establishing a metric in vector space and allowing for meaningful inner products).

It is still necessary to add to all these factors the classical sample-inneficiency of DRL models, which requires from 100k up to 2-3M iterations to offer a chance to actually learn an useful policy, and the intensive parallel computing needed to run the Transformer models that builds embeddings and make inferences in runtime. To illustrate the practival implications of these factors in our case, it usually used to take more than 2 days to run an experiment with 500k timesteps, even in a dedicated server with 32 CPU-cores, 64 GB of RAM and 2x24 GB NVIDIA GPUs.

Nevertheless, in this work we tried to circumvent all of the above mentioned flaws in both the RL design and in the NLP components. In the RL side, we chose the most stable and reliable implementations of state-of-the-art RL agents available and slightly adapted a set of hiperparameters commonly found in successul cases in the literature of the field. With regard with the states, in both of our systems, we leverage the most direct vector representation: an embedding of the accumulated passages, plus the original question. The actions and respective rewards were designed to estimulate the agent not trying to answer a question without gathering a set of initial extra information, but also to avoid accumulating unnecessary or misleading passages.

In the NLP side, we chose a state-of-the-art embedding model finetuned in a metric learning task, and a Longformer with more input tokens than any amount of passages available in an episode. Both models were pretrained in English, the language of our target dataset. Also, our retriever implements a dot-product between the collected passages and the next seeked ones, in order to leverage the semantic properties of theses vectors.

However, we could not control some downsides. First, our embedding model is developed to encode short sentences, with 256 tokens at most – texts longer than this are truncated, which means a potential loss of information in our use-case, besides the efforts of the agent in cleaning its bucket of passages. Furthermore, there is the difficulty of repeating the experiments several times to test different combinations of actions-rewards designs and hyperparameters.

5.2 Results and Comparisons

The charts in the Figure 21 shows the main statistics accumulated during the training stage of our the systems described in the previous subsections. On the left, in blue tones, there are the plots for the *StringConcat* system, and on the right, in red tones, the ones for the *VectorSum* system. We trained a DQN agent 3 times in both setups.

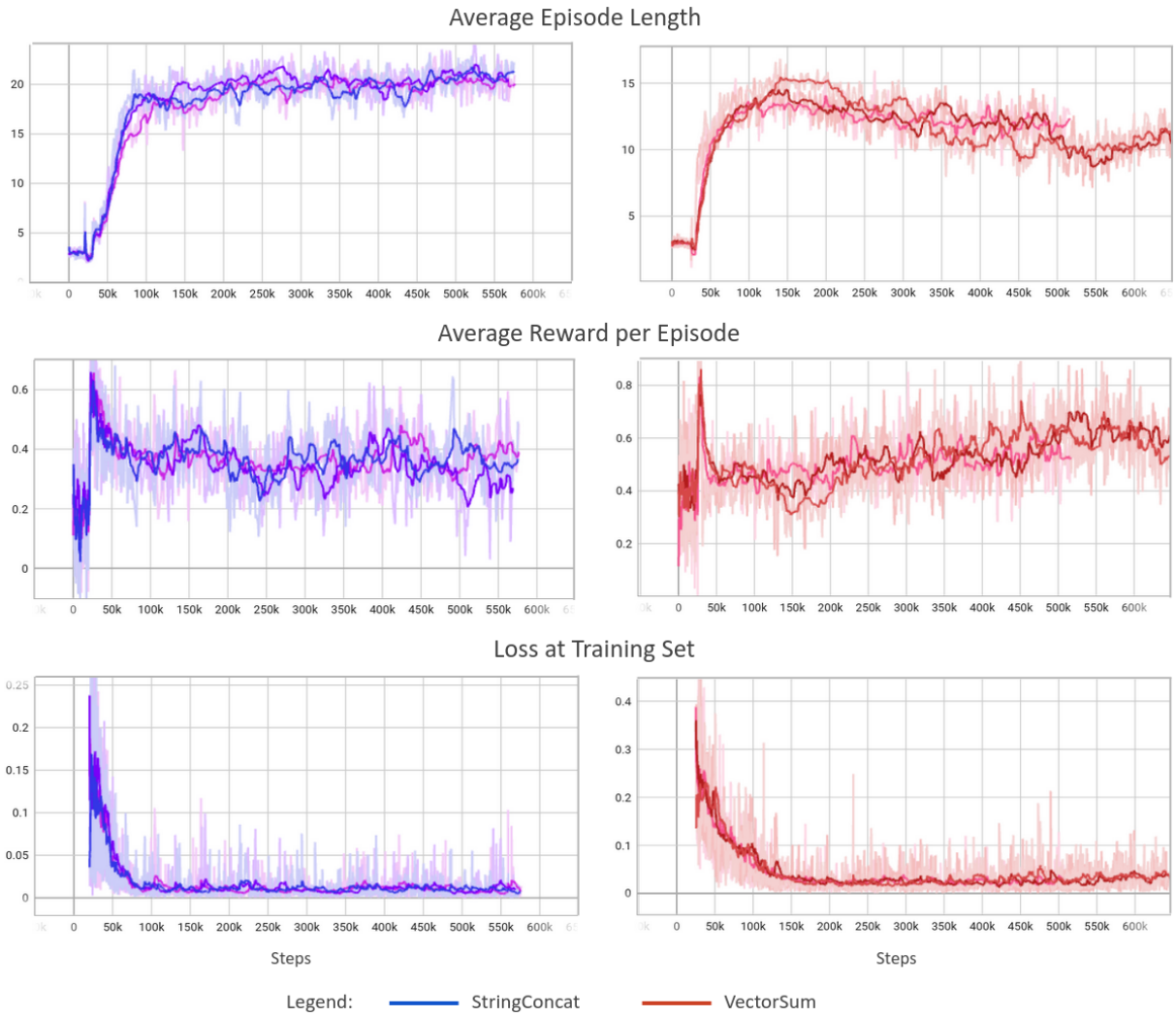


Figure 21: The charts illustrates the training curves of the *StringConcat* and *VectorSum* systems after ∼500k steps. Three training sessions were performed for each system. The average episode length of the *StringConcat* system converges to around 20 after 200k steps, while the *VectorSum* system exhibits a less stable pattern, peaking at 15 before decreasing to approximately 10. Both systems show convergent behavior in the loss charts. The *StringConcat* system maintains an average reward between 0.3 and 0.4, while the *VectorSum* system initially falls to an average reward between 0.4 and 0.5 before surpassing the *StringConcat* average, reaching 0.6. Image produced by the author.

The *Average Episode Length* represents the average number of actions the agent had taken in each episode. In *StringConcat*, this number is caped to $MAXSTEPS = 30$,

and in *VectorSum*, to $MAXSTEPS = 20$. If the agent has not yet chosen the action *ANSWER* and has not yet used all available facts for its query, it only has up to this maximum number of *MAXSTEPS* steps for the episode to end. The last action, therefore, does not have to be just *ANSWER*, but can also be *RETRIEVE* or *CLEAN*. Therefore, one can note that the agent under the *StringConcat* environment steadily converged to an average length of about 20 after ∼200k training steps, while the one under the *VectorSum* configuration, showed a less stable pattern: it first grows up to a peak in ∼15, and then falls to a length of ∼10. Both final results are quite less than their respectives *MAXSTEPS*.

It is also possible to observe a convergent behaviour in both Loss charts, while their *Average Reward per Episode* charts show slightly different patterns. Whereas the *StringConcat* system constantly converges to an average between 0.3 and 0.4, the *VectorSum* system first falls to an average limited by 0.4 and 0.5, but then reaches 0.6, surpassing the *StringConcat* average level.

Comparing the episode length and the reward episode plots in each system, although both curves do not change much after 100k-150k steps for the *StringConcat* system, they do change jointly for the *VectorSum* system: when the agent first reaches its peak of 15 at ∼150k step, the reward per episode is at its lowest point (disregarding the values before 25k, which corresponds to the pure exploration stage). Next, as the episode length decreases, the reward gathered in each episode increases, which suggests the agent is learning how to be more efficient with the document's passages at its disposal, profiting more with less information.

5.3 Results in Dev Set

In order to assess our agent's ability to process and answer complex questions and compare it to a traditional neural end-to-end approach, we randomly took 1k instances from the HotpotQA Dev Set, out of 7405 available. We judged that it was not necessary to use the entire Dev set available, since our benchmark is internal (the end-to-end Longformer model alone that we use as a reader in the ANSWER action of our agents) and there is not, so far, an application similar to ours or NLPGym's of RL techniques to HotpotQA to keep as a basis for comparison. Even NLPGym has not been tested in the HotpotQA dataset, because it was not designed for it.

In addition, we used the Dev set (Fullwiki), because the HotpotQA test set does not

contain the label for each instance (that is, the annotated response), required to compute the NLP metrics.

Our results are listed in the Table 3, compared to the end-to-end Longformer alone, finetuned in the SQuAD dataset. Although the RL-based systems were trained to be economical with the number of passages really required to answer a complex question.

Table 3: Performance Metrics in a 1k-Subset of the HotpotQA Dev Set. Table produced by the author.

Model	Avg F1-score	Avg EM	Passages Used	%Passages
StringConcat	0.058 ± 0.205	0.030 ± 0.171	$\mathbf{7.6 \pm 9.8}$	18%
VectorSum	0.13 ± 0.30	0.079 ± 0.270	20 ± 9.6	47%
Longformer	$\mathbf{0.21 \pm 0.36}$	$\mathbf{0.14 \pm 0.34}$	43 ± 12	100%

In our tests, the strategies of answer-retrieve-clean learned by our RL-based agent was not enough exceed nor beat a reader able to ingest the full amount of contents to answer a question – taking into account the Longformer we used can ingest 4096 input tokens, which is typically more than the necessary to absorb all the passages available to answer a question in the HotpotQA dataset, as shown in histogram at the right of the Figure 10.

One likely reason for this result is that both of our agents probably learned to exploit in some level the rewards from the *RETRIEVE* and *CLEAN* actions, instead of properly answer the question. The Table 4 shows the number of times each action was taken as the last action in each system in the Dev set. Although the *VectorSum* performed better than the *StringConcat*, both often end up the episodes with a *RETRIEVE* or *CLEAN* action, which will certantly lead to a final F1 and EM of zero, since no answer is produced in these cases to be compared to the annotated one. This is a clearly suboptimal outcome from the point of view of a NLP task, but is potentially consistent for a RL problem, once the only way the agent could be punished with a negative score was when actually trying to answer the question, because of the second condition in the reward function 4.4.

Table 4: Last-Action Distribution for Each RL System in the 1k-Subset of the HotpotQA Dev Set. Table produced by the author.

Model	ANSWER	RETRIEVE	CLEAN
StringConcat	442	**155**	**403**
VectorSum	**599**	103	298

This probably explains its advantage the *VectorSum* has over *StringConcat*, once it ends up more often in a last action of *ANSWER*. The Figure 22 shows this result graphically, comparing our RL systems against each other.

Next, an example of episode rollout that actually happened during the training stage. For the question *Which one was an American poet, Frank O'Connor or Edwin Arlington Robinson?*, whose annotated answer was *Edwin Arlington Robinson*, we had the following steps up to the agent extracts an answer:

STEP 0:

- **Number of selected passages so far:** 1
- **Action taken:** *RETRIEVE*
- **Reward:** 0.166
- **Done:** False

STEP 1:

- **Number of selected passages so far:** 6
- **Action taken:** *RETRIEVE*
- **Reward:** 0.091
- **Done:** False

STEP 2:

- **Number of selected passages so far:** 11
- **Action taken:** *CLEAN*
- **Reward:** 0.000
- **Done:** False

STEP 3:

- **Number of selected passages so far:** 3
- **Action taken:** *RETRIEVE*
- **Reward:** 0.125
- **Done:** False

STEP 4:

- **Number of selected passages so far:** 8
- **Action taken:** *ANSWER*
- **Answer extracted:** Edwin Arlington Robinson
- **F1-score:** 1.0
- **Reward:** 2.13
- **Done:** True

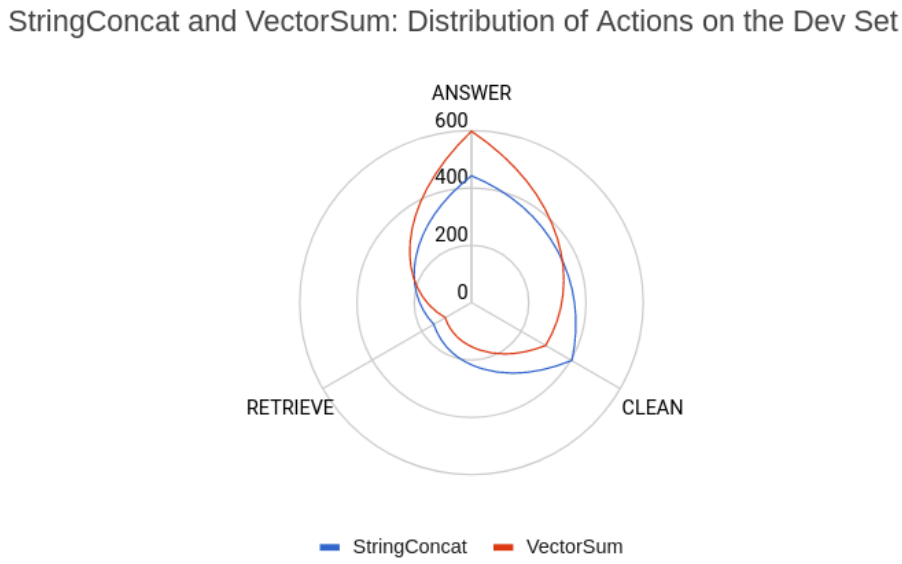


Figure 22: Comparing last-action distribution in the Dev set for both RL-based systems. The fact the *VectorSum* system ends up more often with an *ANSWER* action probably explains its advantage over *StringConcat*. Image produced by the author.

Another reason for the RL-based systems weak results compared to the Longformer is the limited amount of input tokens of our embedding model `all-MiniLM-L6-v2`. Despite it is the state-of-the-art for embedding English sentences intended for the semantic search task we have, it is currently limited to 256 input tokens. This is clearly a major bottleneck for the amount of information the agent observes after just a couple of collected supporting passages, since the average number of tokens is 1415 for a typical full-sized document in an instance of the dataset. It also contrasts strikingly with our Longformer Reader module input token of 4096 tokens.

This probably explains partially the performance gap observed between *StringConcat* and *VectorSum*. For *StringConcat*, with each step forward, more passages are concatenated to generate the document to be vectorized. With each step forward, more passages are concatenated to generate the document to be vectorized. Since the average length of each pass is 41 tokens, after 5 to 6 passes the document usually already has more tokens than the embeddings model is able to support before saturating, truncating and ignoring any information that has accumulated forward. In the case of *VectorSum*, as each state is composed of the original question (which helps to keep the agent aware of the recall), and the sum of each of the already embedded passages, this impact is virtually null in this system.

6 CONCLUSION AND FUTURE WORK

In this work, we tackle a boundary problem in NLP, which is finding apropriate answers for the complex open-domain questions, from a framework still little explored in the literature: a RL setting for the traditional NLP datasets. These datasets are usually addressed by supervised models, specially end-to-end neural models based on the Transformers networks and pretrained in large amounts of texts accross the internet.

Although these models have shown remarkable results in every classical NLP task, such as QA, sentiment analysis, Named Entity Recognition, etc., we argue there is room for RL applications in CODQA tasks mainly due to the fact the intermediate passages (or, "supporting facts") are unknown, and often requires a number of sequential steps, from one passage to the next ones, to answer the question. As a couple of positive side-effects, one can directly optimizes the system to increases an evaluation metric, such as the F_1-score, or reduce the number of evidences collected to the minimal necessary to achieve an accurated answer. These were two of the main intentions of this work.

However, our analysis shows our RL setup is not particularly better than a single Longformer in this QA task, given that the agent is not properly leveraging its *CLEAN-RETRIEVE* pair of actions in an effort to reduce the necessary amount of sentences to answer the question, eliminating the misleading ones, and keeping the most useful ones.

Nonetheless, in real cases, it is common to consult so many documents to answer a complex question that the limit of 4096 tokens can easily be exceeded. At the same time, Transformers networks are not as efficient to process very long inputs due to the quadratic nature of the self-attention mechanism. In this way, it is feasible to consider that RL-based systems can still be improved to the point of being relevant for real tasks of processing complex questions in which it is necessary to manage with some care the amount of support information really needed to answer these questions.

Another important point to consider is the sampling inefficiency of RL models. There really needs to be a substantial advantage in NLP metrics for it to generally be worth

making use of time-consuming RL models composed of heavy Transformers models. If separately these systems can already be slow, together this slowness adds up, even if there is no training of the Transformers networks and they are used only for inference, as we did in this work.

Thus, as some future work ahead, we would like to address these current limitations, trying new ways to run the Transformer models faster, by leveraging ONNX serialization, for example, and search and test different encoders, with a larger input token – or even finetune a Longformer with a contrastive learning objective in an English dataset in order to surpass this 256-token limit. We also strongly believe that a finer reward shaping could lead to a better and more desirable behavior, by more properly aligning the results of scores on the NLP task with those on the RL task.

Since the emergence of the super-powerful chatGPT/GPT-4, it also would be interesting to craft a fine prompt, apply them to our CODQA dataset and compare the performance of these models against the current benchmark results.

REFERENCES

ALLAM, A. M. N.; HAGGAG, M. H. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, v. 2, n. 3, 2012.

André F. A. Paschoal, Paulo Pirozelli, Valdinei Freire, Karina V. Delgado, Sarajane M. Peres, Marcos M. José, Flávio N. Cação, André S. Oliveira, Anarosa A. F. Brandão, and Anna H. R. Costa, F. G. C. Pirá: A Bilingual Portuguese-English Dataset for Question-Answering about the Ocean. In: *30th ACM International Conference on Information and Knowledge Management (CIKM'21)*. [S.l.: s.n.], 2021.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

BELTAGY, I.; PETERS, M. E.; COHAN, A. Longformer: The Long-Document Transformer. 2020. Available at: ⟨http://arxiv.org/abs/2004.05150⟩.

BOLLACKER, K.; EVANS, C.; PARITOSH, P.; STURGE, T.; TAYLOR, J. Freebase: A collaboratively created graph database for structuring human knowledge. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2008. (SIGMOD '08), p. 1247–1250. ISBN 9781605581026. Available at: ⟨https://doi.org/10.1145/1376616.1376746⟩.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. *OpenAI Gym*. 2016.

CAÇÃO, F. N.; JOSé, M. M.; OLIVEIRA, A. S.; SPINDOLA, S.; COSTA, A. H. R.; COZMAN, F. G. *DEEPAGÉ: Answering Questions in Portuguese about the Brazilian Environment*. 2021.

CARMO, D.; PIAU, M.; CAMPIOTTI, I.; NOGUEIRA, R.; LOTUFO, R. Ptt5: Pretraining and validating the t5 model on brazilian portuguese data. *arXiv preprint arXiv:2008.09144*, 2020.

CHEVALIER-BOISVERT, M.; LAHLOU, S.; NGUYEN, T. H.; BAHDANAU, D.; WILLEMS, L.; BENGIO, Y.; SAHARIA, C. Babyai: A platform to study the sample efficiency of grounded language learning. *7th International Conference on Learning Representations, ICLR 2019*, p. 1–13, 2019.

CHOI, E.; HEWLETT, D.; USZKOREIT, J.; POLOSUKHIN, I.; LACOSTE, A.; BERANT, J. Coarse-to-fine question answering for long documents. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, 2017. p. 209–220. Available at: ⟨https://aclanthology.org/P17-1020⟩.

CLARK, P.; COWHEY, I.; ETZIONI, O.; KHOT, T.; SABHARWAL, A.; SCHOENICK, C.; TAFJORD, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

CÔTÉ, M. A.; KÁDÁR, Á.; YUAN, X.; KYBARTAS, B.; BARNES, T.; FINE, E.; MOORE, J.; HAUSKNECHT, M.; El Asri, L.; ADADA, M.; TAY, W.; TRISCHLER, A. TextWorld: A Learning Environment for Text-Based Games. *Communications in Computer and Information Science*, v. 1017, p. 41–75, 2019. ISSN 18650937.

DAS, R.; DHULIAWALA, S.; ZAHEER, M.; VILNIS, L.; DURUGKAR, I.; KRISHNAMURTHY, A.; SMOLA, A.; MCCALLUM, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

DHINGRA, B.; MAZAITIS, K.; COHEN, W. W. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.

DUNN, M.; SAGUN, L.; HIGGINS, M.; GUNEY, V. U.; CIRIK, V.; CHO, K. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

FERRUCCI, D.; BROWN, E.; CHU-CARROLL, J.; FAN, J.; GONDEK, D.; KALYANPUR, A. A.; LALLY, A.; MURDOCK, J. W.; NYBERG, E.; PRAGER, J. et al. Building watson: An overview of the deepqa project. *AI magazine*, v. 31, n. 3, p. 59–79, 2010.

GAO, J.; GALLEY, M.; LI, L. Neural approaches to conversational ai. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. [S.l.: s.n.], 2018. p. 1371–1374.

HENDERSON, P.; ISLAM, R.; BACHMAN, P.; PINEAU, J.; PRECUP, D.; MEGER, D. Deep reinforcement learning that matters. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, p. 3207–3214, 2018.

HERMANN, K. M.; GREFENSTETTE, T. K. E.; ESPEHOLT, L.; KAY, W.; SULEYMAN, M.; BLUNSOM, P. Teaching machines to read and comprehend. In: *Advances in Neural Information Processing Systems (NIPS)*. [s.n.], 2015. Available at: ⟨http://arxiv.org/abs/1506.03340⟩.

HEWLETT, D.; LACOSTE, A.; JONES, L.; POLOSUKHIN, I.; FANDRIANTO, A.; HAN, J.; KELCEY, M.; BERTHELOT, D. WikiReading: A novel large-scale language understanding task over Wikipedia. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016. p. 1535–1545. Available at: ⟨https://aclanthology.org/P16-1145⟩.

HILL, A.; RAFFIN, A.; ERNESTUS, M.; GLEAVE, A.; KANERVISTO, A.; TRAORE, R.; DHARIWAL, P.; HESSE, C.; KLIMOV, O.; NICHOL, A.; PLAPPERT, M.; RADFORD, A.; SCHULMAN, J.; SIDOR, S.; WU, Y. *Stable Baselines*. [S.l.]: GitHub, 2018. ⟨https://github.com/hill-a/stable-baselines⟩.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

JOSHI, M.; CHOI, E.; WELD, D. S.; ZETTLEMOYER, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Vancouver, Canada: Association for Computational Linguistics, 2017.

JOULIN, A.; GRAVE, E.; BOJANOWSKI, P.; MIKOLOV, T. Bag of tricks for efficient text classification. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, 2017. p. 427–431. Available at: ⟨https://aclanthology.org/E17-2068⟩.

KARPUKHIN, V.; OGUZ, B.; MIN, S.; LEWIS, P.; WU, L.; EDUNOV, S.; CHEN, D.; YIH, W.-t. Dense passage retrieval for open-domain question answering. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020. p. 6769–6781. Available at: ⟨https://www.aclweb.org/anthology/2020.emnlp-main.550⟩.

KHOT, T.; CLARK, P.; GUERQUIN, M.; JANSEN, P.; SABHARWAL, A. QASC: A dataset for question answering via sentence composition. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, p. 8082–8090, 2020. ISSN 2159-5399.

KONDA, V. R.; TSITSIKLIS, J. N. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, p. 1008–1014, 2000. ISSN 10495258.

LEWIS, M.; LIU, Y.; GOYAL, N.; GHAZVININEJAD, M.; MOHAMED, A.; LEVY, O.; STOYANOV, V.; ZETTLEMOYER, L. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. p. 7871–7880. Available at: ⟨https://aclanthology.org/2020.acl-main.703⟩.

LEWIS, P.; PEREZ, E.; PIKTUS, A.; PETRONI, F.; KARPUKHIN, V.; GOYAL, N.; KüTTLER, H.; LEWIS, M.; YIH, W.-t.; ROCKTäSCHEL, T.; RIEDEL, S.; KIELA, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. In: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M. F.; LIN, H. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. v. 33, p. 9459–9474. Available at: ⟨https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf⟩.

LUKETINA, J.; NARDELLI, N.; FARQUHAR, G.; FOERSTER, J.; ANDREAS, J.; GREFENSTETTE, E.; WHITESON, S.; ROCKTÄSCHEL, T. A survey of reinforcement learning informed by natural language. *IJCAI International Joint Conference on Artificial Intelligence*, v. 2019-Augus, p. 6309–6317, 2019. ISSN 10450823.

MANNING, C. D. Speech and Language Processing: An introduction to natural language processing. *SPEECH and LANGUAGE PROCESSING An Introduction to Natural Language Processing Computational Linguistics and Speech Recognition*, p. 1–18, 2021. Available at: ⟨http://www.cs.colorado.edu/$\sim$martin/slp.html⟩.

MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. S.; DEAN, J. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2013. p. 3111–3119.

PAULA, A. Felipe Magnossao de; SILVA, R. Fray da; NISHIMOTO, B. E.; CUGNASCA, C. E.; COSTA, A. H. R. Answer selection using reinforcement learning for complex question answering on the open domain. In: *2021 International Symposium on Electrical, Electronics and Information Engineering*. [S.l.: s.n.], 2021. p. 271–276.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. [S.l.: s.n.], 2014. p. 1532–1543.

PLEJIC, B.; VUJNOVIC, B.; PENCO, R. Transforming unstructured data from scattered sources into knowledge. In: IEEE. *2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop*. [S.l.], 2008. p. 924–927.

RAFFEL, C.; SHAZEER, N.; ROBERTS, A.; LEE, K.; NARANG, S.; MATENA, M.; ZHOU, Y.; LI, W.; LIU, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, v. 21, p. 1–67, 2020.

RAJPURKAR, P.; ZHANG, J.; LOPYREV, K.; LIANG, P. SQuad: 100,000+ questions for machine comprehension of text. *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, n. ii, p. 2383–2392, 2016.

RAMAMURTHY, R.; SIFA, R.; BAUCKHAGE, C. Nlpgym–a toolkit for evaluating rl agents on natural language processing tasks. *arXiv preprint arXiv:2011.08272*, 2020.

REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. Available at: ⟨https://arxiv.org/abs/1908.10084⟩.

ROBERTSON, S.; ZARAGOZA, H. *The probabilistic relevance framework: BM25 and beyond*. [S.l.: s.n.], 2009. v. 3. 333–389 p. ISSN 15540669. ISBN 1500000019.

SANH, V.; DEBUT, L.; CHAUMOND, J.; WOLF, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. p. 2–6, 2019. Available at: ⟨http://arxiv.org/abs/1910.01108⟩.

SHEN, Y.; CHEN, J.; HUANG, P.; GUO, Y.; GAO, J. M-walk: Learning to walk in graph with monte carlo tree search. *CoRR, abs*, 2018.

SHILAKES, C.; TYLMAN, J. Enterprise Information Portals and Enterprise Knowledge Portal. 2002.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: pretrained bert models for brazilian portuguese. In: SPRINGER. *Brazilian Conference on Intelligent Systems*. [S.l.], 2020. p. 403–417.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

TALMOR, A.; BERANT, J. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*, 2018.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. *Advances in Neural Information Processing Systems*, v. 2017-Decem, n. Nips, p. 5999–6009, 2017. ISSN 10495258.

WANG, S.; YU, M.; GUO, X.; WANG, Z.; KLINGER, T.; ZHANG, W.; CHANG, S.; TESAURO, G.; ZHOU, B.; JIANG, J. R3: Reinforced ranker-reader for open-domain question answering. In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. [S.l.: s.n.], 2018. p. 5981–5988. ISBN 9781577358008.

WANG, Y.; JIN, H. A deep reinforcement learning based multi-step coarse to fine question answering (mscqa) system. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2019. v. 33, n. 01, p. 7224–7232.

WELBL, J.; STENETORP, P.; RIEDEL, S. Constructing Datasets for Multi-hop Reading Comprehension Across Documents. *Transactions of the Association for Computational Linguistics*, v. 6, p. 287–302, 2018. ISSN 2307-387X.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, Springer, v. 8, n. 3, p. 229–256, 1992.

XIONG, W.; HOANG, T.; WANG, W. Y. 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, 2017. p. 564–573. Available at: ⟨https://aclanthology.org/D17-1060⟩.

XIONG, W.; LI, X. L.; IYER, S.; DU, J.; LEWIS, P.; WANG, W. Y.; MEHDAD, Y.; YIH, W.-t.; RIEDEL, S.; KIELA, D. et al. Answering complex open-domain questions with multi-hop dense retrieval. *arXiv preprint arXiv:2009.12756*, 2020.

YANG, Z.; QI, P.; ZHANG, S.; BENGIO, Y.; COHEN, W. W.; SALAKHUTDINOV, R.; MANNING, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, p. 2369–2380, 2020.