UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CASSIANO MARQUES BARTZ

Empirical Evaluation on Approaches to Transform Tabular Data into Textual Input for QA systems

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Dante Barone
Coadvisor: Mr. Eduardo Cortes

Porto Alegre
October 2022

ACKNOWLEDGEMENT

Words cannot express my gratitude to my family and friends for being by my side. I’m also grateful for my advisor and coadvisor for being understanding and helping me throughout this work.

ABSTRACT

Question answering (QA) systems aim to automatically and precisely answer a specific question provided in natural language over a knowledge base. Although there are models that already work well with regular textual knowledge bases for those systems, when it comes to tabular data this scenario changes. Transforming tabular data into a textual input for a model is a difficult task and this research explores different approaches for this transformation of data by testing on JarvisQA, a prototype QA system which uses BERT pre-trained models to answers questions on top of the Question Answering Benchmark for Scholarly Knowledge (SciQA). SciQA is benchmark developed with the collaboration of researchers from different Universities that leverages the Open Research Knowledge Graph (ORKG).The proposed methodology consists of: (1) creating an approach to generate text from tabular data, (2) running JarvisQA using the text generated by the approach as knowledge base for the benchmark questions and (3) perform an empirical evaluation of the results. The analysis on these approaches, their results and the difficulties faced can help researchers dealing with such scenarios.

Keywords: Question Answering. BERT. JarvisQA. SciQA. ORKG.

Avaliação Empírica em Abordagens para Transformação de Dados Tabulares em Entrada Textual para Sistemas de Pergunta e Resposta

RESUMO

Sistemas de pergunta e resposta tem como objetivo responder automaticamente e precisamente perguntas específicas feitas em linguagem natural usando uma base de conhecimento. Embora existam modelos que funcionam bem com bases de conhecimento textuais para estes sistemas, quando envolve dados tabulares esse cenário muda. A transformação de dados tabulares em dados de entrada textuais para modelos é uma tarefa difícil e essa pesquisa explora diferentes abordagens para essa transformação, testando em cima do protótipo JarvisQA, que utiliza modelos pré-treinados baseados em BERT para responder as perguntas em cima do benchmark *Question Answering Benchmark for Scholarly Knowledge* (SciQA). SciQA é um *benchmark* que foi desenvolvido em colaboração de pesquisadores de diferentes universidades que utiliza o *Open Research Knowledge Graph* (ORKG). A metodologia proposta é a seguinte: (1) criação de uma abordagem para gerar textos através de dados tabulares, (2) utilizar o texto gerado através da abordagem como base de conhecimento do JarvisQA enquanto respondendo as perguntas do benchmark e (3) executar uma avaliação empírica dos resultados. A análise dessas abordagens, seus resultados e as dificuldades encontradas podem ajudar pesquisadores lidando com tais cenários.

Palavras-chave: *Question Answering*, BERT, JarvisQA, SciQA, ORKG.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

QA	Question Answering
ORKG	Open Research Knowledge Graph
SciQA	Question Answering Benchmark for Scholarly Knowledge
SPARQL	Simple Protocol and RDF Query Language
RDF	Resource Description Framework

CONTENTS

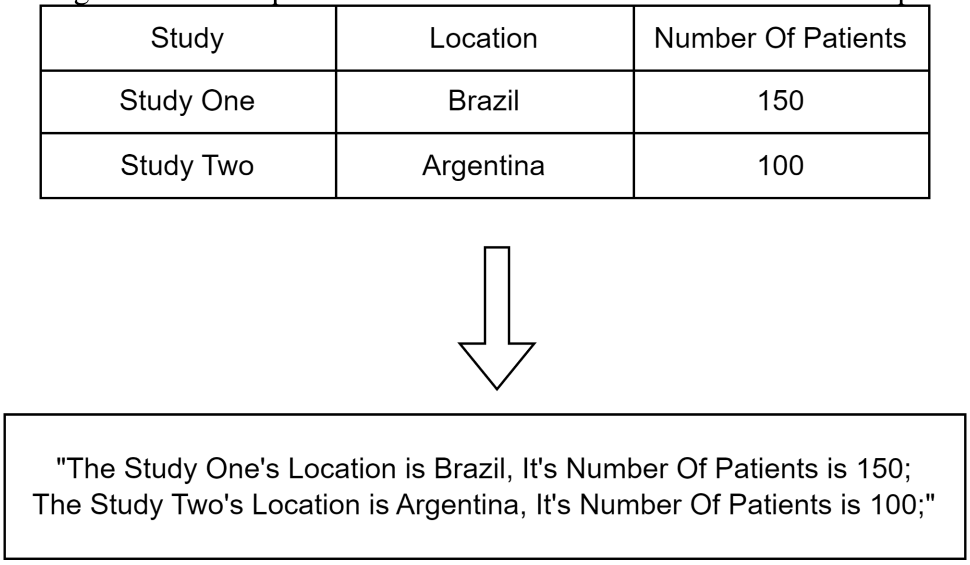
1 INTRODUCTION

Question answering (QA) is a field of study with the objective of answering correctly and automatically questions in natural language. Recently, with the improvements of graphics processing units (GPUs) and parallelism (NARAYANAN et al., 2021), which allows for robust training of models, the field has had significant advancements. Transformers, which are deep learning models, are now being used to answer questions based on a textual input, and while the results have been promising when asking questions on a given contextual input, the research on question answering over tabular data and how to transform this data in text is relatively new. In this monography we are going to research how to transform tabular data into a textual input for fine-tuning deep learning models.

1.1 Motivation

In the last years a few studies emerged trying to address question answering over tables using neural networks, like TableQA (VAKULENKO; SAVENKOV, 2017), which is a prototype that in it's architecture the model is trained with tabular input, or studies that involve search across tables (SUN et al., 2016), for example. On the other hand, the usage of pre-trained models like BERT (DEVLIN et al., 2019) with tabular data is a relatively uncharted territory, and since the model is not prepared for tabular data as is, a transformation to textual data is needed for this kind of usage.

Figure 1.1: Example of Tabular Data transformation into textual input

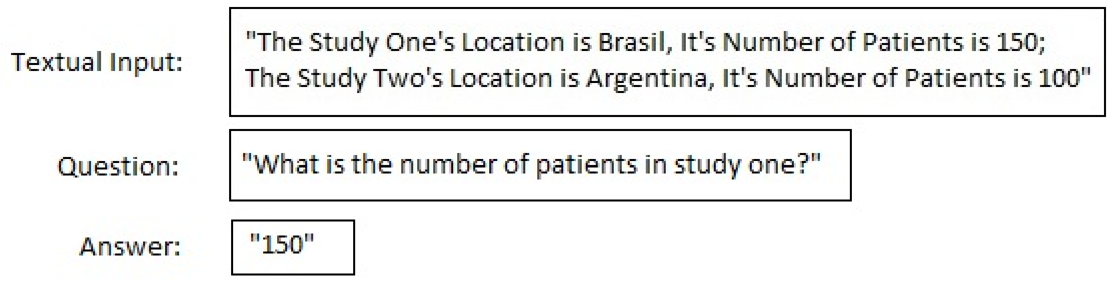


Study	Location	Number Of Patients
Study One	Brazil	150
Study Two	Argentina	100

"The Study One's Location is Brazil, It's Number Of Patients is 150;
The Study Two's Location is Argentina, It's Number Of Patients is 100;"

We can see in Figure 1.1 an example of transforming tabular data into text. On top we see a table which is the contextual information to be queried upon, and below we see the text which was created based on it. Using the generated text as input for the model enables the QA System to answer contextual questions in it.

Figure 1.2: Example of question on top of textual input



Textual Input: "The Study One's Location is Brasil, It's Number of Patients is 150; The Study Two's Location is Argentina, It's Number of Patients is 100"

Question: "What is the number of patients in study one?"

Answer: "150"

We can see in Figure 1.2 an example of question being made on top of the text generated.Although this example shows what the tabular data transformation into text is, it is a simple and basic example, and when questions that would normally require a query in the table are made, if the textual input is too simple, the QA system will be unable to answer it.For example, in this scenario a question about the total number of patients across all studies would not be able to be answered.

1.2 Objective

With the problem presented in the Motivation section, related to more complex questions being made on top of the tabular data, this research proposes to investigate different approaches to generate input texts that have meaningful information attached, so QA Systems are able to answer these questions on top of the provided data. This research also studies the limitations of these approaches, their performance impact and if they bring improvements when compared with the results before the implementation.

1.3 Contribution

The contribution of this research is:

- Analysis on questions which are difficult to answer on regular tabular data transformation in text

- Proposal of approaches to enhance tabular data transformation into text
- Empirical evaluation of these approaches

1.4 Proposed Structure

The structure of this research is organized as follows: Chapter 2 discusses the background of the field covered in this study. Chapter 3 presents the methodology used in the research, along with information of the computational system (hardware and software) used in performance tests. Chapter 4 presents the approaches created in this work and their analysis. Finally Chapter 5 wraps it up with the conclusion of this work.

2 BACKGROUND

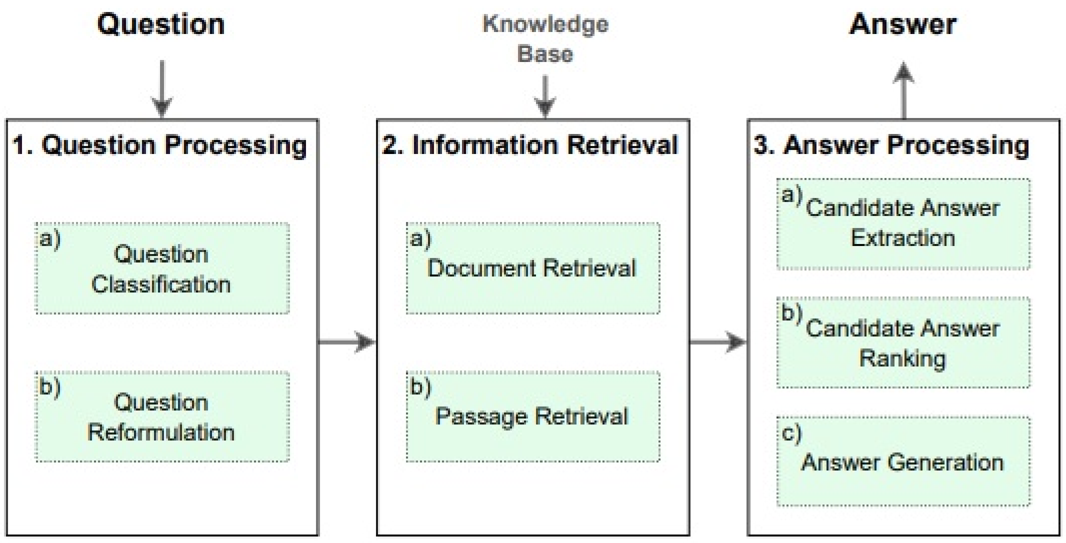
This chapter describes the background for this research. First we present the QA Area, focusing on what it is.Then we present the BERT model, a machine learning framework for natural language processing (NLP), and after that we present the JarvisQA prototype and related works for this research.

2.1 Question Answering

Question Answering (QA) is a field that aims to answer correctly and automatically a question provided by an user in natural language. QA systems may be specific to a certain domain or can be general. While restricted domain QA systems focus on a specialized area, general domain systems answer questions from diverse fields (OLVERA-LOBO; GUTIÉRREZ-ARTACHO, 2011). QA systems also differ in another characteristic, for example the knowledge source which the system is using to query information, which can be (1) documents, where the system query for information in raw text documents, files and websites, or the system can use (2) linked data as a knowledge source, for example knowledge graphs (CORTES; BARONE, Forthcoming).

The questions can be categorized as factoid and non-factoid. Factoid questions require a fact as an answer such as a name or a location, while non-factoid questions require extensive or complex information as the answer. Factoid and non-factoid QA systems present a similar architecture, as seen in Figure 2.1, which contains three main components: Question Processing, Information Retrieval and Answer Processing, with differences appearing inside the components (CORTES; BARONE, Forthcoming).

Figure 2.1: General Architecture of a QA System.



Source: (CORTES; BARONE, Forthcoming)

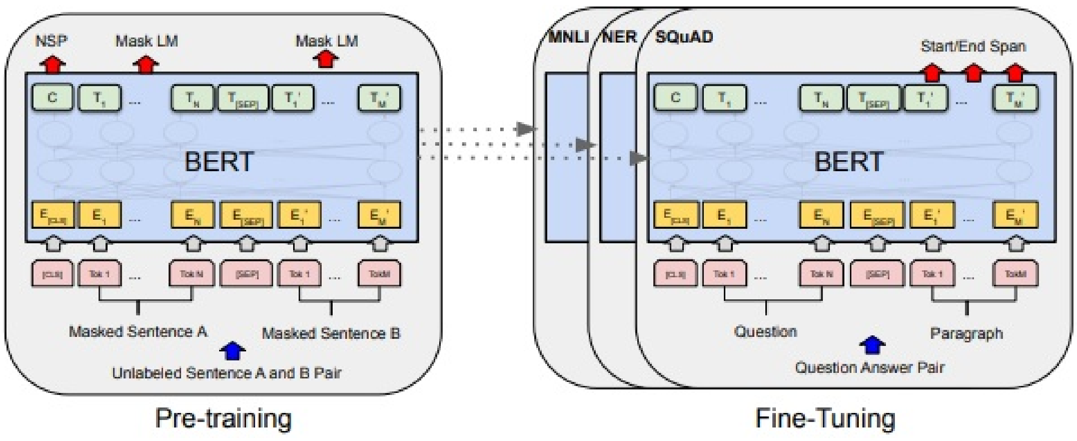
2.2 BERT

BERT (DEVLIN et al., 2018), short for Bidirectional Encoder Representations from Transformers, is a Machine Learning (ML) model for natural language processing. It was developed in 2018 by researchers at Google AI Language and serves as a swiss army knife solution to some of the most common language tasks, such as sentiment analysis and named entity recognition. Aside for the aforementioned tasks, BERT can also be used for Question Answering, Text Prediction, Text Generation and Summarizing.

There are two steps in BERT's framework, pre-training and fine-tuning, which we can see in figure 2.2. During pre-training, the model is trained on unlabeled data, usually a large dataset, and it can learn inner representations of the language of the data used which can be useful for downstream tasks, like text classification for example. For fine-tuning step, the pre-trained model is re-trained using the custom data provided to it, and as a result, the model is updated to account for the characteristics of the domain data supplied.

Recently, substantial work has shown that pre-trained models (PTMs) on large corpus can learn universal language representations, which is beneficial for downstream natural language processing (NLP) tasks and can avoid training a new model from scratch (QIU et al., 2020). Several PTMs are available to use, which varies in language and case/uncased text input.When using a pre-trained BERT model in a QA System, if the system is restricted domain, the model is fine-tuned using data for the specific domain and this avoids large computational costs compared to if the model had to be fully trained, and it will be able to answer questions in this domain.

Figure 2.2: Overall pre-training and fine-tuning procedures for BERT



Source: (DEVLIN et al., 2018). Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.

In this research, we use pre-trained models in a prototype QA System called JarvisQA (JARADEH; STOCKER; AUER, 2020), which fine-tunes the PTM with contextual data for each question, and this data comes in form of tables which the prototype is going to transform into text. After this fine-tuning process, we can answer the question provided.

2.2.1 Tokens Size Limitation

BERT has a limitation of 512 tokens for the input used for fine-tuning, and this size is calculated from the question and input combined. In case more than 512 tokens are fed to the model, it will truncate and only use the first 512 tokens. These tokens are sub-words as BERT uses WordPiece (DEVLIN et al., 2019) to generate them.

This presents a challenge in this study since the objective is to enhance the context for the question, but depending on the size of the original context, there may not be enough space to add information. This scenario will be explored in this research.

2.3 Related Work

In this section we present the related work, consisting of a prototype QA System called JarvisQA, which was used to implement the approaches seen in this research, and the SciQA article, where a benchmark of complex questions was created to challenge current QA Systems.

2.3.1 JarvisQA

JarvisQA is a BERT based system made to answer questions on tabular views of scholarly knowledge graphs. It was developed by Mohamad Yaser Jaradeh, Sören Auer and Markus Stocker in a collaboration between L3S Research Center in the University of Hannover and TIB Leibniz Information Centre for Science and Technology, both in Germany. JarvisQA was presented for a submitted article *Question Answering on Scholarly Knowledge Graphs* (JARADEH; STOCKER; AUER, 2020). It receives a table of questions, which has columns for the question itself, the table name for the context of the question, the type of the question and the expected answer, as can be seen in Table 2.1.

Table 2.1: Example of Questions Table Provided for JarvisQA

Question	Table	Type	Answer
Which system has the worst recall?	R6946	normal	YodaQA
What is the most common location in the studies?	R111045	normal	China
Which ontology has the most classes?	R8342	normal	EXPO
How many studies are published after 2019?	R110393	normal	3

The "Table" column represents the name of the contextual table file in the prototype folder. This table will be used for the fine-tuning.

Table 2.2: Example of Domain for Specific Question

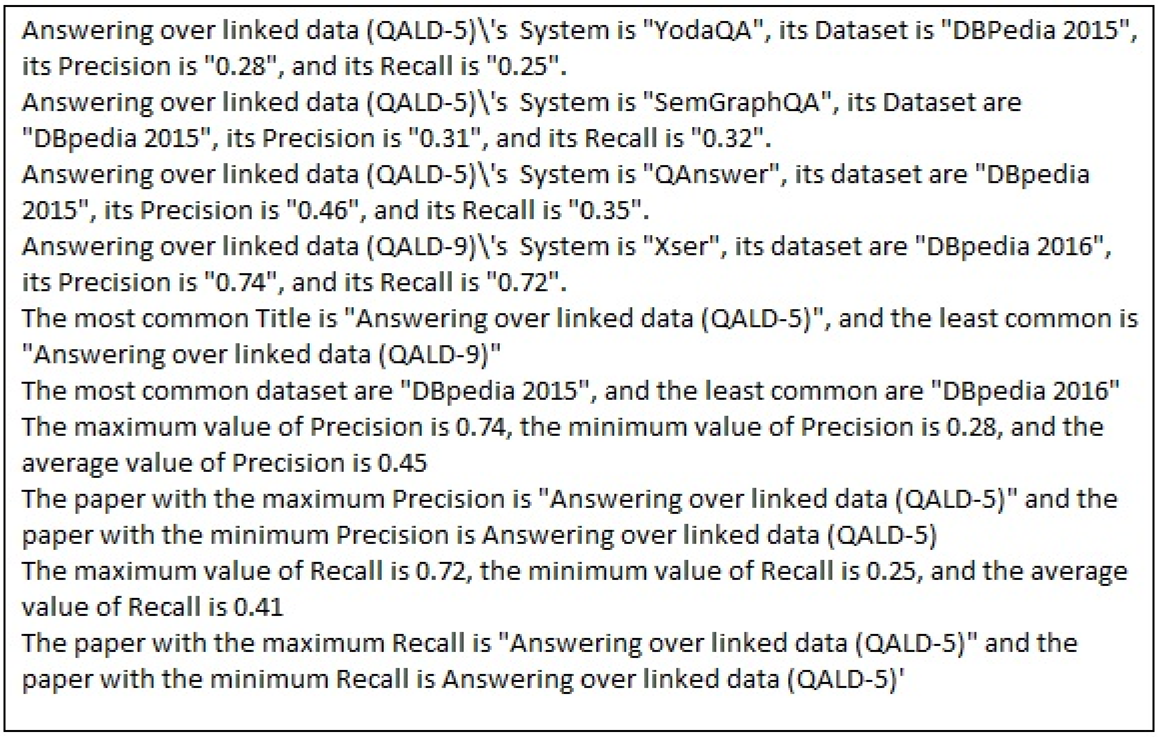
Paper	System	Dataset	Precision	Recall
Answering over linked data (QALD-5)	YodaQA	DBPedia 2015	0.28	0.25
Answering over linked data (QALD-5)	SemGraphQA	DBPedia 2015	0.31	0.32
Answering over linked data (QALD-5)	QAnswer	DBPedia 2015	0.46	0.35
Answering over linked data (QALD-9)	Xser	DBPedia 2016	0.74	0.72

The data showed in this table is fictional. In the context of Table 2.1 this would be the table named R6946.The main objective of the open challenge on question answering over linked data (QALD) is to provide up-to-date, demanding benchmarks that establish a standard against which question answering systems over structured data can be evaluated and compared.(UNGER et al., 2014)

Along with the table of questions, JarvisQA also expects all the tables required for each question listed. These tables can contain any columns and are the domain for the specific question. These tables will be used for the fine-tuning process of the pre-trained models used. An example of such table can be seen in Figure 2.2.

JarvisQA will read the question and fetch the context table file and transform it into a readable text to feed a BERT model for fine-tuning. An example of this transformation has already been presented in Figure 1.1. Along with the basic transformation, the system already has some approaches of enhancing the input text built-in, like adding information about the maximum and minimum values of numeric columns, and most/least common entries in string columns. On Figure 2.3 we can see an example of the tabular transformation using as base the Table 2.2.

Figure 2.3: Example of JarvisQA Tabular Transformation

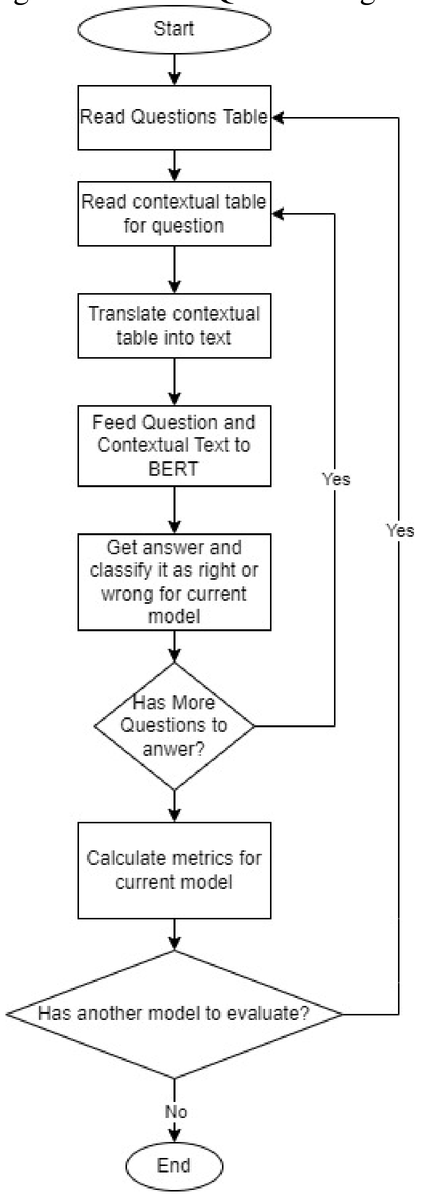


Answering over linked data (QALD-5)\'s System is "YodaQA", its Dataset is "DBPedia 2015", its Precision is "0.28", and its Recall is "0.25".
Answering over linked data (QALD-5)\'s System is "SemGraphQA", its Dataset are "DBpedia 2015", its Precision is "0.31", and its Recall is "0.32".
Answering over linked data (QALD-5)\'s System is "QAnswer", its dataset are "DBpedia 2015", its Precision is "0.46", and its Recall is "0.35".
Answering over linked data (QALD-9)\'s System is "Xser", its dataset are "DBpedia 2016", its Precision is "0.74", and its Recall is "0.72".
The most common Title is "Answering over linked data (QALD-5)", and the least common is "Answering over linked data (QALD-9)"
The most common dataset are "DBpedia 2015", and the least common are "DBpedia 2016"
The maximum value of Precision is 0.74, the minimum value of Precision is 0.28, and the average value of Precision is 0.45
The paper with the maximum Precision is "Answering over linked data (QALD-5)" and the paper with the minimum Precision is Answering over linked data (QALD-5)
The maximum value of Recall is 0.72, the minimum value of Recall is 0.25, and the average value of Recall is 0.41
The paper with the maximum Recall is "Answering over linked data (QALD-5)" and the paper with the minimum Recall is Answering over linked data (QALD-5)'

Using the Table 2.2 as base for the transformation.

In Figure 2.4 we can see the overall JarvisQA flow. The dataset to be used must be within the project folder and consists of a file for the questions table, and in the same location, it must have a folder called *csv* which will contain all the files for the contextual tables. Once the dataset has been provided, the user can define which pre-trained models to run JarvisQA upon, and it will go through the process of reading the questions table, and for each question, translate the associated context table, supplying it to BERT for fine-tuning. Once the question has been provided, JarvisQA will compared the received answer with the expected answer, and record it accordingly. After all questions have been answered, the metrics for the current model are going to be recorded in an output file. This process repeats until all specified models have been evaluated.

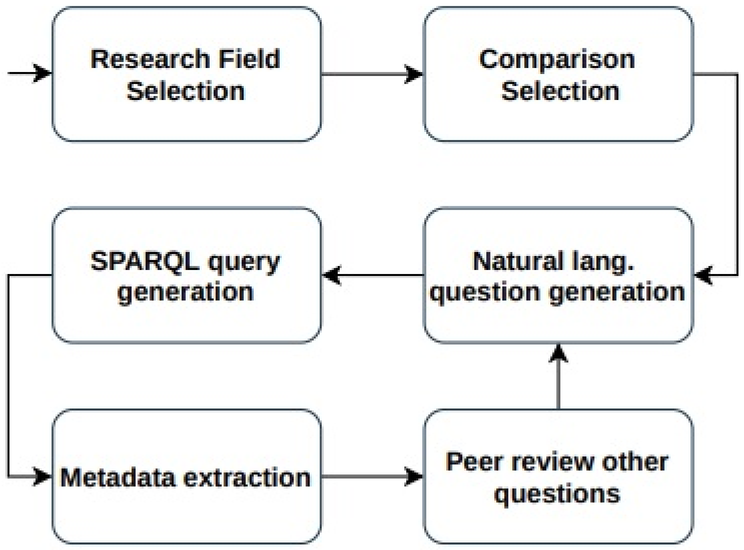
Figure 2.4: JarvisQA Working Flow


Start
Read Questions Table
Read contextual table for question
Translate contextual table into text
Feed Question and Contextual Text to BERT
Get answer and classify it as right or wrong for current model
Has More Questions to anwer?
Yes
Calculate metrics for current model
Has another model to evaluate?
Yes
No
End

In this research we use the JarvisQA system and enhance the translation step for the contextual table. The objective is to generate contextual texts with more information, by trying to predict most common questions that could be asked about the current context, and by doing this, enabling JarvisQA or another QA system with similar architecture to answer more questions.

2.3.2 Question Answering Benchmark for Scholarly Knowledge

Figure 2.5: SciQA benchmark collection workflow



The first part of this research was the collaboration for the creation of the Question Answering Benchmark for Scholarly Knowledge (SciQA benchmark). QA benchmarks and systems are so far mainly geared towards encyclopedic knowledge graphs such as DBpedia and Wikidata (AUER et al., Submitted). In the article, we developed a set of a hundred complex questions, along with their SPARQL queries on top of ORKG, and peer-reviewed each question to guarantee it's correctness. In figure 2.5 you can see the workflow for the creation of the questions.

First a research field was selected, for example *engineering*, and a comparison for this field in ORKG. Comparison is a core type for ORKG content and represents a list of contributions towards a research problem. Then the comparison was analysed for the creation of a question, and for this question, it's SPARQL Query. Then the metadata for the question was collected, like the type of question (factoid or non-factoid) and the query shape (like tree or chain). After the metadata collection, it was passed to peer review to other researchers involved in the article. The objective of the article is to create a challenging benchmark for the next-generation QA systems.

SciQA benchmark leverages the Open Research Knowledge Graph (ORKG) and has a set of a hundrer complex questions which can be answered with the ORKG. JarvisQA was used in a subset of those questions to check how pre-trained models based on BERT performed in such complex questions.

In this research we are going to use JarvisQA again on top of the SciQA benchmark, but applying changes in the step responsible for translating the question context table into text for BERT fine-tuning. The results which were originally obtained in the SciQA Article are going to be used as baseline to compare the approaches developed in this research.

2.3.2.1 Current JarvisQA results with SciQA benchmark

Since JarvisQA only works with tabular data, out of 100 questions which were created for the SciQA benchmark, only 52 are answerable by the system. In the SciQA article (AUER et al., Submitted) the QA system ran in 7 different pre-trained models and the result can be seen in the below table. For the metrics used, Precision is a formula $TP/TP+FP$, where TP stands for True Positive and FP stands for False Positive. Recall is calculated by $TP/TP + FN$, where FN stands for False Negatives. As for F1 Score, it is calculated by $2 * (Recall * Precision)/(Recall + Precision)$. As for *k* values, it is the number of answers requested to the model.

Table 2.3: Evaluation results of running JarvisQA against SciQA benchmark questions.

JarvisQA Setup	Normal						Overall					
	Precision		Recall		F1		Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10	@1	@10	@1	@10	@1	@10
$BERT_{BUS}$	.1905	.2712	.1906	.2713	.1905	.2712	.1364	.1905	.1364	.1905	.1364	.1905
$BERT_{LCS}$	**.1935**	.2542	**.1937**	.2545	**.1936**	.2543	**.1379**	.1786	**.1379**	.1786	**.1379**	.1786
$BERT_{BCS2}$	.1343	.1875	.1344	.1876	.1343	.1875	.0978	.1348	.0978	.1348	.0978	.1348
$BERT_{LUS2}$	.1693	.2883	.1692	.2881	.1692	.2882	.1222	.2024	.1222	.2024	.1222	.2024
$DistBERT_{BUS}$	.1343	.2459	.1343	.2459	.1343	.2459	.0978	.1744	.0978	.1744	.0978	.1744
$ALBERT_{XLS2}$	.1719	**.3393**	.1719	**.3394**	.1719	**.3393**	.1250	**.2346**	.1250	**.2346**	.1250	**.2346**
$ALBERT_{XXLS2}$	.1692	.2459	.1692	.2459	.1692	.2459	.1222	.1744	.1222	.1744	.1222	.1744

JarvisQA setups follow similar notations as introduced in (JARADEH; STOCKER; AUER, 2020). Top performing setup is indicated in bold, second best is underlined

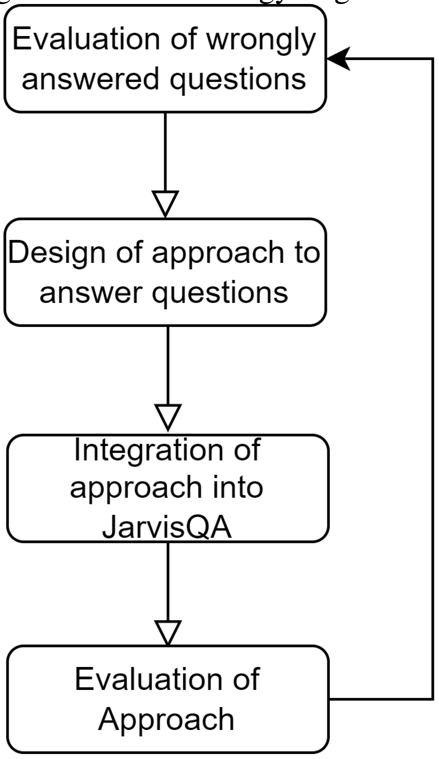
In this study, we are going to focus on $BERT_{LCS}$ and $ALBERT_{XLS2}$ since those were the two models that best performed in *k=1* and *k=10* respectively. By using these two models we can evaluate the created approaches and avoid performance issues, since some evaluations can only run in the CPU and might take hours to complete.

3 METHODOLOGY

In this chapter we present the methodology, the analysis of questions used in this research and the information about the machine used during this work.

The methodology is divided into three stages: (1) Evaluation of questions being incorrectly answered by the QA system prototype in it's current state, (2) design a generic approach that would generate text from the table which would answer one or more of the questions evaluated, (3) integrate this approach in the prototype[1] and (4) evaluate the results of the approach by comparing the results with the baseline and also the performance impact.

Figure 3.1: Methodology stages workflow


Evaluation of wrongly answered questions
Design of approach to answer questions
Integration of approach into JarvisQA
Evaluation of Approach

For the start of this research, a list of all wrongfully answered questions from SciQA benchmark were recorded, along with the answers provided by the model. From there, generic patterns were analysed, for example, questions which would be looking for the sum of an column, questions that would be looking for the highest or lowest value in a column, and/or other patterns. Once these patterns have been analysed, approaches to try and generate text with generic functions were designed, implemented and integrated into JarvisQA. After the implementation, the approach was evaluated with the same metrics

[1]The code and results for the evaluations performed in this research are found in: https://github.com/cassianobartZ/JarvisQA

as the baseline results, Precision, Recall and F1-Score in the SciQA benchmark. Along with those metrics, the time which the QA system took to evaluate the model using the approach was also recorded, to have a baseline of performance cost. All approaches were evaluated by themselves to try and have the most optimal scenario for performance evaluation, with the exception of Split Input approach, which will be discussed further on.

3.1 Analysis on questions

In order to improve the results presented in subsection 2.3.2.1, we need to analyse the questions the QA system is failing to correctly answer. From there, we will check approaches in this research which will focus on enhancing the system's table to text feature in order to supply BERT with more context data.

BERT works really well for simple answer questions, for example, running $BERT_{LCS}$ for the question *What type of data does the system proposed in paper titled "Open Research Knowledge Graph" support?* wields:

```
Expected Answer: Free Text
BERT output: ['score': 0.17584075762448137, 'start': 260, 'end': 272, 'answer': 'Free text']
```

We can see that in the case above BERT managed to answer correctly, but if we take a look at questions that need some kind of calculation on top, we start to see some issues if this calculation is not already done in the context. For the question *What is the total number of patients in the studies?* we get the following result:

```
Expected Answer: 6452
BERT output: ['score': 0.1977839599400255, 'start': 3303, 'end': 3307, 'answer': '2124']
```

We can see that since there was no previous treatment in the context and BERT is only analysing the text, such questions can't be answered yet. Some other examples:

```
Which system has the worst recall?
Expected Answer: YodaQA
BERT output: QALD-5
```

For which country of study overall prevalence of epilepsy is the highest?
Expected Answer: England
BERT output: France

As discussed in the beginning of the chapter, we can see some patterns in the questions shown above. The question *What is the total number of patients in the studies?* is looking for a sum of an specific column, while the question *Which system has the worst recall?* is looking for the smallest number in a column. For these kinds of questions we were able to design approaches to answer them in this research, because given any table, it's **possible** to make **generic** transformations to it in order to extract this kind of information.

But for some complex questions, it is very hard to develop such an generic approach without knowing the question beforehand. For example, for the question *How many studies are published after 2019?*, it would be needed to make a calculation on a certain column, but using only specific cells. These kind of questions prove to be a challenge in the architecture being used in JarvisQA.

3.2 Computational System Used For Evaluations

For the evaluations done in this research, the execution time was recorded to analyse performance impact. The configurations of the computational system used is as follows:

```
Core i7 12700K
32GB DDR5 6000mhz
GeForce RTX 3080 10GB
Windows 10 Pro Version 21H2
Microsoft Visual Studio Code 1.71.2
Python 3.8.10
PyTorch 0.12.0+cu113
```

All the executions were done without background processes running, and when feasible, running in the GPU. Versions of Python and PyTorch are described above. Not all executions were possible in the GPU because of memory allocation issues. Some models, for example $ALBERT_{XXLS2}$, wouldn't run in GPU mode for this machine because it demands more than 10GB of memory, and would fail execution. Without the Split Input approach, presented in this research, other models presented the same error when running in the machine. When possible, all tests were executed in the GPU because of the difference in performance, since GPU is more suited for this kind of application than CPU and delivers consistently better performance (SUN et al., 2019). The record of time was done in the code and saved with the evaluations file exported by the prototype. For the time evaluation, for each approach the prototype ran three times in order to get an average.

4 APPROACHES

In this chapter we will evaluate approaches of pre-processing the contextual tabular data provided for the questions to try and answer some of the wrongfully answered questions as presented in section 3.1.

4.1 Input Split

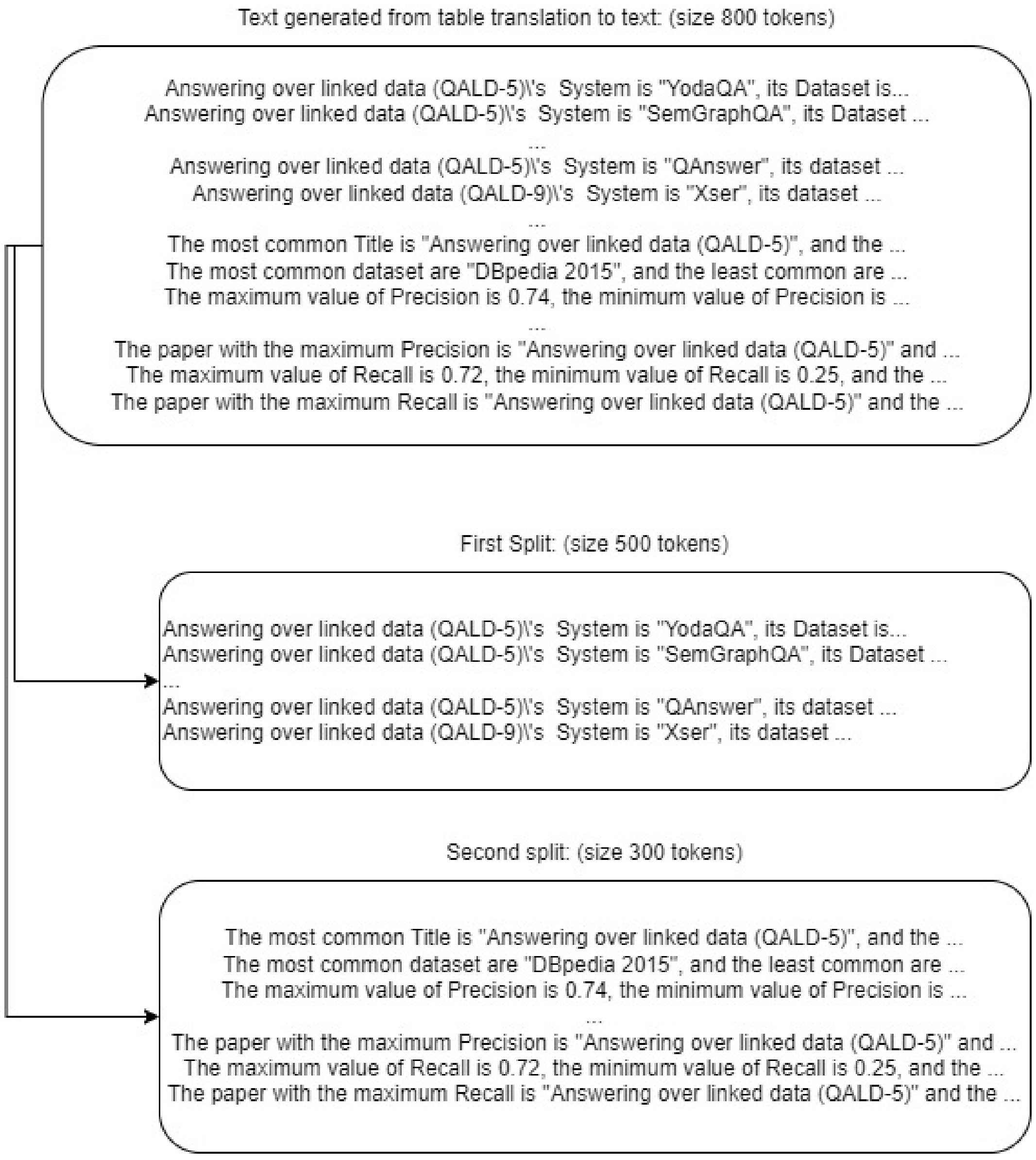
The first approach was implemented not for the reason of trying specifically to answer a group of questions, but because of the previously mentioned tokens limitation of BERT in subsection 2.2.1. It was identified in this research that in some of the questions for the SciQA benchmark, the contextual data table by itself would already be larger than 512 tokens when transformed into text. In this scenario, we wouldn't be able to validate any approach in this research since we would need to fit the table data itself, plus the extra data generated in the approaches in 512 tokens, otherwise any extra information would be ignored by the model.

This situation can happen when the provided contextual table have a large number of rows, columns or both. One example is the same question which was mentioned in section 3.1: *What is the total number of patients in the studies?*. The contextual table provided for this question has a *Title* column, and some of the rows have very long titles. This caused the translated contextual text for this table to have the size of *1402* word tokens, which by itself surpasses the 512 limitation.

Please note that BERT uses sub-word tokens generated by WordPiece (DEVLIN et al., 2019), so the actual size is most certainly higher than that. To be able to enhance the context data with more information to better answer the questions, this approach called *Input Split* is going to detect possible situations were the contextual text will be larger than 512 tokens, and split in different inputs.We can see in Figure 4.1 a visual explanation of what is happening in this approach.

Having *n* splits for a given input, now we need to use all those splits. Instead of fine-tuning the BERT model one time per question, as it previously did, now we have to make *n fine-tunings*, one for each split. In the next subsections we will speak about the implications of this.

Figure 4.1: Split Input Example


Text generated from table translation to text: (size 800 tokens)

Answering over linked data (QALD-5)\'s System is "YodaQA", its Dataset is...
Answering over linked data (QALD-5)\'s System is "SemGraphQA", its Dataset ...
...
Answering over linked data (QALD-5)\'s System is "QAnswer", its dataset ...
Answering over linked data (QALD-9)\'s System is "Xser", its dataset ...
...
The most common Title is "Answering over linked data (QALD-5)", and the ...
The most common dataset are "DBpedia 2015", and the least common are ...
The maximum value of Precision is 0.74, the minimum value of Precision is ...
...
The paper with the maximum Precision is "Answering over linked data (QALD-5)" and ...
The maximum value of Recall is 0.72, the minimum value of Recall is 0.25, and the ...
The paper with the maximum Recall is "Answering over linked data (QALD-5)" and the ...

First Split: (size 500 tokens)

Answering over linked data (QALD-5)\'s System is "YodaQA", its Dataset is...
Answering over linked data (QALD-5)\'s System is "SemGraphQA", its Dataset ...
...
Answering over linked data (QALD-5)\'s System is "QAnswer", its dataset ...
Answering over linked data (QALD-9)\'s System is "Xser", its dataset ...

Second split: (size 300 tokens)

The most common Title is "Answering over linked data (QALD-5)", and the ...
The most common dataset are "DBpedia 2015", and the least common are ...
The maximum value of Precision is 0.74, the minimum value of Precision is ...
...
The paper with the maximum Precision is "Answering over linked data (QALD-5)" and ...
The maximum value of Recall is 0.72, the minimum value of Recall is 0.25, and the ...
The paper with the maximum Recall is "Answering over linked data (QALD-5)" and the ...

Using the Table 2.2 as base for the transformation. This figure shows the Split Input method splitting a 800 token input string into two different inputs.

4.1.1 Implementation

For the implementation of this approach, we go to the *t2t.py* file for JarvisQA and modify the *table_2_text* method. This file is responsible for the translation of tabular data into text for the QA system, and the method mentioned is responsible for generating a readable text for each row of the *csv* context file and combine everything into a input string. The modification made is to test in each row, if the addition of this new row is going to exceed the input size.If the new row makes the input bigger than 500 word tokens, the input is split at this point and a new string is created to continue the process. The number 500 was chosen to leave room for sub-token generation made by WordPiece, but the actual value to be used would require further research. Let's take a look at the code:

```
full_context_text = ''
contexts = []
for row in rows:
  row_text = self.row_2_text(row, header, empty)
  full_context_text_size_test =
    full_context_text + '\n' + row_text + '\n' + extra_info
  if len(full_context_text_size_test.split()) > 500:
    contexts.append(full_context_text+ '\n' + extra_info)
    full_context_text = row_text
    continue
  full_context_text = full_context_text + '\n' + row_text
```

As mentioned in subsection 2.3.1, JarvisQA already makes some processing of the context data, such as average in numeric columns for example. In the above code, *extra_info* is an string with this kind of information. Please note that in this approach we are testing for word tokens, so it is still possible that the input is truncated depending on the number of sub-word tokens generated inside the model.

4.1.2 Results and Analysis

We can see in Table 4.1 that the results for the split are mixed. Let's try to analyse why. With Input Split, JarvisQA is actually calling BERT *n-times*, where *n* corresponds to the number of splits. If we imagine a hypothetical scenario where a question has three inputs and a *k* value of 1, we will have three different answers from BERT with different scores. The answer with the highest score will be selected.

Since now the context is split into different calls, the score can be influenced by this separation of the knowledge. This situation creates the question: **What is the best way to treat the score in a separation scenario like this one?**

Another point is with a value of *top_k* different than one. If BERT is being called three times with a *top_k* of ten, in the end we will have thirty answers, to which, in the case of this study, the ten with the highest score would be returned. This can explain some of the situations were the result was worse, because in some cases it was identified the correct answer was out of the first ten answers. One example is the question *What is the base URL of "The Document Components Ontology"?*, which the correct answer was found in the *nineteenth* position of thirty and was ruled out for not being in the first ten.

Table 4.1: JarvisQA results with Input Split approach

JarvisQA Setup	Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10
$BERT_{LCS}$ (Normal)	.2258 (.1935)	.2459 (.2542)	.2258 (.1937)	.2459 (.2545)	.2258 (.1936)	.2459 (.2543)
$BERT_{LCS}$ (Overall)	.1609 (.1379)	.1744 (.1786)	.1609 (.1379)	.1744 (.1786)	.1609 (.1379)	.1744 (.1786)
$ALBERT_{XLS2}$ (Normal)	.1719 (.1719)	.3158 (.3393)	.1719 (.1719)	.3158 (.3394)	.1719 (.1719)	.3158 (.3393)
$ALBERT_{XLS2}$ (Overall)	.1236 (.1250)	.2195 (.2346)	.1236 (.1250)	.2195 (.2346)	.1236 (.1250)	.2195 (.2346)

Improvements are colored green and worse results are colored red. Between parenthesis is the baseline score running JarvisQA without Split Input

Table 4.2: JarvisQA running time with Input Split approach

JarvisQA Setup	Time Before	Time After
$BERT_{LCS}$	00h38m19s	00h40m47s
$ALBERT_{XLS2}$	01h49m53s	10h16m53s

Execution done in CPU

We can see in Table 4.2 that this approach also comes with a performance cost. Although the difference for $BERT_{LCS}$ is marginal, for $ALBERT_{XLS2}$ we see a drastic increase in execution time. As already mentioned, BERT is going to be called more times and this can be an expensive. In this small subset of 52 questions, we can see a major increase in time for execution, but it is worth to mention that this is directly related to the size of the contexts provided for each question, and the number of questions, so the times will vary for different scenarios.

This execution is the only in this study that needed to be run in **CPU** mode, which is much slower than running in the **GPU**. This is because without the Input Split, the application would crash because of lack of memory to be allocated. This can also be a benefit of the Input Split approach.

Even with this additional performance cost, this approach is valuable to validate the other approaches in this study since it will enable for more information to be added overall in the context. For better results, it is really important to dive deeper into the BERT scores and try to understand how we can use them better for ordering the answers, as this has a direct impact on the metrics. Also it is important to better analyse the sub-word tokens generation, in order to best choose the splitting point, since in this research a value of 500 was used, but for optimal results this needs to be explored deeper.

4.2 Numeric Column Text Explosion

Going forward, the next approaches are going to focus into adding information to the input. This approach which is called Numeric Column Text Explosion has the idea of creating expanded texts for each numeric row. To better understand this approach lets take a look at the table 4.3, which is an example table we will be using to demonstrate some approaches, and for the data in this example, the approach is going to generate the text presented in table 4.4.

Table 4.3: Example of a dataset with a numeric column

Study	Location	Number of Patients
Study One	Brazil	150
Study Two	Argentina	100

Table 4.4: Example of Numeric Explosion Output

Row	Output Text
First Row	The Number of Patients of Study One is 150
First Row	The Number of Patients of Brazil is 150
Second Row	The Number of Patients of Study Two is 100
Second Row	The Number of Patients of Argentina is 100

As you can see from the example, from the two rows present in Table 4.3 it was generated an output of four strings combining the name of all string columns to the values in the numeric column.

4.2.1 Implementation

For the implementation of this approach, we go to the *t2t.py* file for JarvisQA and create a new method called *append_text_for_numeric_column*, which we can see the code below. This method is being called inside the *append_aggregation_info*, which is an existing method in JarvisQA specifically created to add extra information to the input.

```
def append_text_for_numeric_column(inf, numericColumnName, df):
  columns = list(df)

  for indexValue, numericColumnValue in enumerate(df[numericColumn]):
    for indexIterationColumn,iterationColumn in enumerate(columns):
      if is_numeric_dtype(df[iterationColumn]):
        continue
      iterationColumnValue = df.iloc[indexValue][iterationColumn]
      info.append(
        f'The {numericColumnName} of {iterationColumnValue}
        is {numericColumnValue}\n'
      )
  return info
```

As we see from the code, the basic idea is to navigate all rows for the numeric column, and for each other column of the table combine a text with this numeric row value. We are skipping other numeric columns in this code to avoid strings without meaning.

It's worth noting that this method is called for each column, so the total complexity is $O(n^3)$ where *n* is the number of columns.

4.2.2 Results and Analysis

The results for this approach are also mixed, as we can see in table 4.5. We see a decrease in *k=1* across the board and a small increase in *k=10* when comparing to the results for the Split Input. The dataset is too small to make a conclusion on the reason, but what could be observed is that this approach can suffer from the same issue as the Split Input, where a change in BERT scores can lead to a different order of answers. This could explain why we see a decrease in *k=1* but for *k=10* the result is marginally better.

Table 4.5: JarvisQA results with Numeric Column Text Explosion approach

JarvisQA Setup	Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10
$BERT_{LCS}$ (Normal)	.1538 (.2258)	.2500 (.2459)	.1538 (.2258)	.2500 (.2459)	.1538 (.2258)	.2500 (.2459)
$BERT_{LCS}$ (Overall)	.1111 (.1609)	.1765 (.1744)	.1111 (.1609)	.1765 (.1744)	.1111 (.1609)	.1765 (.1744)
$ALBERT_{XLS2}$ (Normal)	.1692 (.1719)	.3571 (.3158)	.1692 (.1719)	.3571 (.3158)	.1692 (.1719)	.3571 (.3158)
$ALBERT_{XLS2}$ (Overall)	.1222 (.1236)	.2469 (.2195)	.1222 (.1236)	.2469 (.2195)	.1222 (.1236)	.2469 (.2195)

Improvements are colored green and worse results are colored red. Between parenthesis is the result with Split Input Approach in Table 4.1

We can also see another downside of this approach, while it brings mixed results, it also brings a lot of computational overhead. This is explained because this approach

Table 4.6: JarvisQA running time with Numeric Column Text Explosion approach

JarvisQA Setup	Time Before	Time After
$BERT_{LCS}$	00h12m02s	00h55m44s
$ALBERT_{XLS2}$	00h12m34s	01h03m47s

Execution done in GPU. Time Before is from running Split Input Approach

adds $N * M * O$ lines to the input, where *N* is the number of numeric columns, *M* is the number of rows and *O* is the number of non-numeric columns. We can see that with bigger tables this escalates very quickly.

Since the dataset is small it is really hard to fully analyse this approach, but looking at this scenario indicates that using this approach does not bring the expected results for the additional computation expense. This can occur because in this specific dataset, we don't see a lot of questions that asks for a specific value linked to some column in the table, so this approach would need to be investigated better with larger datasets.

4.3 Sum for Numeric Columns

Moving on to the next approach, this one focus on adding a simple information for each numeric column, which is the sum of all values in it. The idea behind this approach is to try and answer questions in the format of *'What is the total...?'*.

Table 4.7: Example of output for Sum Approach

Output Text
The total Number of Patients is 250

Example with input from the Table 4.3

The reasoning behind this approach is very simple, as is the implementation which we will see in the next subsection. Since JarvisQA already has a similar logic being used for appending information like *minimum and maximum*, the effort in this approach is not very big.

4.3.1 Implementation

This implementation is really simple, we go to the *t2t.py* file for JarvisQA and inside the existing *append_aggregation_info*, which is an existing method in the prototype system specifically created to add extra information to the input, we add along the information of the sum.

```
def append_aggregation_info(self, df: pd.DataFrame) -> str:
  info = []
  for column in df:
    if is_numeric_dtype(df[column]):
      column_df = df[column].fillna(0)
      max_value = column_df.max()
      min_value = column_df.min()
      avg_value = column_df.mean()
      sum = column_df.sum()
      info.append(f'The maximum value of {column} is {max_value}\n'
        f'the minimum value of {column} is {min_value}\n'
        f'and the average value of {column} is {avg_value:.2f}\n'
        f'the total {column} is {sum}\n')
    ...
  return info
```

This method already existed and the logic for *minimum, maximum and average* already existed before.

4.3.2 Results and Analysis

We can see from the results in Table 4.8 that although there's a slight decrease in *k=1* for $BERT_{LCS}$, we see improvements in *k=10* and also see improvaments across the board on $ALBERT_{XLS2}$. We can also see from Table 4.9 that the performance overhead is marginal. The decrease in *k=1* could be explained once again by ordering of scores, but this would need to be studied better with a larger dataset.

Since the method for adding numeric information like minimum, maximum and average already existed in JarvisQA, the performance impact is negligible and it is so similar that any margin of time can be an external effect, even though measures were taken to minimize this. The final evaluation is this approach has a simple implementation, low performance cost and brings expected results for the target questions.

Table 4.8: JarvisQA results with Sum for Numeric Columns approach

JarvisQA Setup	Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10
$BERT_{LCS}$ (Normal)	.2063 (.2258)	.2881 (.2459)	.2063 (.2258)	.2881 (.2459)	.2063 (.2258)	.2881 (.2459)
$BERT_{LCS}$ (Overall)	.1477 (.1609)	.2024 (.1744)	.1477 (.1609)	.2024 (.1744)	.1477 (.1609)	.2024 (.1744)
$ALBERT_{XLS2}$ (Normal)	.2063 (.1719)	.3571 (.3158)	.2063 (.1719)	.3571 (.3158)	.2063 (.1719)	.3571 (.3158)
$ALBERT_{XLS2}$ (Overall)	.1477 (.1236)	.2469 (.2195)	.1477 (.1236)	.2469 (.2195)	.1477 (.1236)	.2469 (.2195)

Improvements are colored green and worse results are colored red. Between parenthesis is the result with Split Input Approach in Table 4.1

Table 4.9: JarvisQA running time with Sum for Numeric Columns approach

JarvisQA Setup	Time Before	Time After
$BERT_{LCS}$	00h12m02s	00h11m11s
$ALBERT_{XLS2}$	00h12m34s	00h12m43s

Execution done in GPU. Time Before is from running Split Input Approach

4.4 Highest and Lowest Row Explosions

This approach is an enhancement of an existing method inside JarvisQA. If we look at the example shown in Figure 2.3, we see that a logic for highest and lowest values already exist in the code, which can be seen from the string *The maximum value of Precision is 0.74, the minimum value of Precision is 0.28, and the average value of Precision is 0.45*.

Similarly, if the context table contained a column named "Title", JarvisQA would also generate the string *The paper with the maximum Precision is "Answering over linked data (QALD-5)" and the paper with the minimum Precision is "Answering over linked data (QALD-5)"*.

The idea of this approach is to link the information from highest/lowest numeric column values for all other columns in the table, not just "Title". We can see in Table 4.10 and example of what this approach produces when using the Table 4.3 as a base.

Table 4.10: Example of Highest and Lowest Row Explosions Output

Output Text
The Study with the highest Number of Patients is Study One
The Location with the highest Number of Patients is Brazil
The Study with the lowest Number of Patients is Study Two
The Location with the lowest Number of Patients is Argentina

Output for Highest and Lowest Row Explosions approach based on data from Table 4.3

What this new approach brings is an explosion similar as seen in section 4.2, except instead of expanding for all numeric values, which was really expensive in terms of performance and didn't bring much result, this approach is just an expansion for the highest and lowest values.

4.4.1 Implementation

For the implementation of this approach, we go to the *t2t.py* file for JarvisQA and create a new method called *append_text_for_highest_and_lowest_numeric_column*, which we can see the code below. This method is being called inside the same method mentioned before in the other approaches, *append_aggregation_info*.

```
def append_text_for_highest_and_lowest_numeric_column(
    info: str,
    numericColumnName: str,
    df: pd.DataFrame
) -> str:
    columns = list(df)
    column_df = df[numericColumnName].fillna(0)
    rowForHighestNumericValue = df.iloc[column_df.argmax()]
    rowForLowestNumericValue = df.iloc[column_df.argmin()]
    for indexIterationColumn, iterationColumn in enumerate(columns):
        if is_numeric_dtype(df[iterationColumn]):
            continue
        lowestRowIterationColumnValue =
            rowForLowestNumericValue[iterationColumn]
        highestRowIterationColumnValue =
            rowForHighestNumericValue[iterationColumn]
        info.append(
            f'The {iterationColumn} with the highest
            {numericColumnName} is {highestRowIterationColumnValue}\n'
        )
        info.append(
            f'The {iterationColumn} with the lowest
            {numericColumnName} is {lowestRowIterationColumnValue}\n'
        )
    return info
```

This method is called for every column, so it has a complexity of $O(n^2)$ where *n* is the number of columns.

4.4.2 Results and Analysis

The results on Table 4.11 are good. Although we see a slight decrease in *k=1* for $BERT_{LCS}$ like in the other approaches, we see a good increase for *k=10* and also a good increase across the board for $ALBERT_{XLS2}$.

As far as performance cost, we do see a 50% increase in time for this specific scenario as shown in Table 4.12, but if we compare with the results seen in Section 4.2, the increase is not as expressive and we get better results. A possible reason for the better results is that this dataset has questions that are looking for information on highest/lowest values while it does not contain a lot of questions that asks for values linked to an specific column.

From the improvement in the *k=10*, it indicates that this approach could successfully answer one or more questions regarding highest or lowest value present in a table.

Table 4.11: JarvisQA results with Highest and Lowest Row Explosion approach

JarvisQA Setup	Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10
$BERT_{LCS}$ (Normal)	.2063 (.2258)	.3103 (.2459)	.2063 (.2258)	.3103 (.2459)	.2063 (.2258)	.3103 (.2459)
$BERT_{LCS}$ (Overall)	.1477 (.1609)	.2169 (.1744)	.1477 (.1609)	.2169 (.1744)	.1477 (.1609)	.2169 (.1744)
$ALBERT_{XLS2}$ (Normal)	.2258 (.1719)	.3818 (.3158)	.2258 (.1719)	.3818 (.3158)	.2258 (.1719)	.3818 (.3158)
$ALBERT_{XLS2}$ (Overall)	.1609 (.1236)	.2625 (.2195)	.1609 (.1236)	.2625 (.2195)	.1609 (.1236)	.2625 (.2195)

Improvements are colored green and worse results are colored red. Between parenthesis is the result with Split Input Approach in Table 4.1

Table 4.12: JarvisQA running time with Highest and Lowest Row Explosion approach

JarvisQA Setup	Time Before	Time After
$BERT_{LCS}$	00h12m02s	00h18m07s
$ALBERT_{XLS2}$	00h12m34s	00h18m39s

Execution done in GPU. Time Before is from running Split Input Approach

4.5 Combining all Approaches

After taking a look at the previous approaches one by one and comparing it's results against the results seen in Table 4.1 for Split Input approach, it's time to combine all of them together and see how much the score for JarvisQA increased when answering the questions from SciQA benchmark.

The results increased across the board for both models analysed. We can see the biggest improvements in *k=1* in all metrics, with the metrics in $ALBERT_{XLS2}$ almost doubling the original. The increase in *k=10* was not as large, but still considerable.

Table 4.13: JarvisQA Normal Results with All Approaches Combined

JarvisQA Setup	Precision		Recall		F1	
	@1	@10	@1	@10	@1	@10
$BERT_{LCS}$ (Normal)	.2667 (.1935)	.3571 (.2542)	.2667 (.1937)	.3571 (.2545)	.2667 (.1936)	.3571 (.2543)
$BERT_{LCS}$ (Overall)	.1882 (.1379)	.2469 (.1786)	.1882 (.1379)	.2469 (.1786)	.1882 (.1379)	.2469 (.1786)
$ALBERT_{XLS2}$ (Normal)	.3103 (.1719)	.4340 (.3393)	.3103 (.1719)	.4340 (.3394)	.3103 (.1719)	.4340 (.3393)
$ALBERT_{XLS2}$ (Overall)	.2169 (.1250)	.2949 (.2346)	.2169 (.1250)	.2949 (.2346)	.2169 (.1250)	.2949 (.2346)

Improvements are colored green and worse results are colored red. Between parenthesis is the original results in Table 2.3

Table 4.14: JarvisQA Running Time With All Approaches Combined

JarvisQA Setup	Time Before	Time After
$BERT_{LCS}$	00h12m02s	00h18m07s
$ALBERT_{XLS2}$	00h12m34s	00h18m39s

Execution done in GPU. Time Before is from running Split Input Approach Only, because GPU mode on original code won't run for lack of memory.

For the execution time, unfortunately since the machine used for testing was unable to run the original code in GPU mode, we have to compare the time for all approaches against the execution time for Split Input approach. Looking at the execution time, we can see a similar impact as to running the numeric explosion approach as seen in Table 4.6. Since in this testing all approaches were used, we can see an inconsistency here, where the Numeric Explosion approach was used along with the others, but we did not see the same performance impact as to running only that approach by itself. This behavior is hard to pinpoint the reason, and more metrics would have to be collected to generate a better picture of the performance impact these approaches have, and not only execution time.

For the results, since these approaches were designed by analysing the questions, each approach had a different target of questions in mind. This can explain why combining all the approaches increases the metrics so much, because when compared to the baseline, we certainly have a higher number of correct answers.

5 CONCLUSION AND FUTURE WORK

The SciQA benchmark proves to be a really difficult benchmark for the BERT algorithms tested in this research. This is because BERT is fine-tuned with a textual input before trying to answer questions, and the models analysed in this research are not made by design to handle tabular data. While factual and simple questions could be answered, if the question needs some form of specific calculation or processing to be done in the tabular data, it is not going to answer correctly unless the question was predicted in some way when preparing the textual input. This can be extremely challenging for really complex and/or really specific questions.

In this monography we analysed some approaches that not only prepare the input data for simple and factual questions, but also the approaches represent a fairly easy way to process tabular data generically. Let's take a look at some examples from the dataset which wouldn't be able to be answered without previous knowledge of the question:

How many studies do use Chloride as major anion?

How many studies are published after 2019?

As outlined in section 3.1, in the questions above, it would be needed to query the tabular data in an specific manner and then count the occurrences to add in the input text. This is extremely challenging without knowing the question, and to try and predict it in a generic way, the process would generate extremely large inputs because of the data explosions needed. It doesn't seen viable.

Are children examined in the studies?

Is Cobb-Douglas functional applied in the studies?

With the example questions above, BERT seemed to have a difficulty when answering *Is/Are* questions. While the expected answers would be in the *Yes/No* format, BERT seemed to return texts for these questions, sometimes returning the subject of the question back.

What are the objectives for Sepsis prediction?

Last but not least, it was also difficult in this research to deal with List answers.

BERT got really close to answering some of these questions, but often it would miss some of the items or answer all of them but in separated across different answers. To deal with this scenario at the level of input processing would be very difficult, as once again we would need some kind of prediction of the question.

As for the results presented, it is also worth mentioning again that the dataset is small and for that, the changes can have a bigger impact, be it as an increase or as a decrease in metrics. This happens because of the small set of questions, where one more question answered correctly or incorrectly can have a big impact in the calculation. It is though still interesting to see these results, specially when looking at some results that almost doubled the metrics. In the end with a small dataset like this one, the metrics certainly were affected in a drastic way, so it would be best to also test these approaches in larger datasets in the future.

For the execution times, the tests were all executed in the same machine and without background processes running to try and have a stable baseline for a study like this one. Unfortunately it was not possible to generate all metrics wanted because of limitations in the machine. Some of the testing had to be done in CPU mode which uses RAM because the GPU would not have enough memory to be allocated, and this kind of processing is really slower to run in the CPU. As we saw from the combined results, the execution time by itself was not consistent enough to understand all the impact, so this along with the allocation memory problem presents a opportunity to dive deeper into the manner and collect more information, and not only execution time, as with this information we will be able to understand better all the impact of each approach. It was still interesting to see these results, because since BERT models have the size limitation for fine-tuning, the variables which would affect most the execution time are really the size of the context table and the processes being done on translation of the table to text.

In conclusion, while possible to enhance the metrics with the approaches presented, these approaches only tackle simple and easy to predict questions. For the future, more research is needed for approaches that could answer more complex questions in the field of translating tabular data into text, along with more research on top of already presented approaches for improvements.

During this work, a collaboration for the creation of the SciQA benchmark was seen in subsection 2.3.2. This was an internacional collaboration between the TIB – Leibniz Information Centre for Science and Technology (Germany), L3S Research Center from Leibniz University Hannover (Germany), Institute of Informatics from Federal

University of Rio Grande do Sul (Brazil), Department of Informatics and Telecommunications from University of Athens (Greece) and Laboratory of Information Science and Semantic Technologies from ITMO University (Russia). The paper called *SciQA – A Question Answering Benchmark for Scholarly Knowledge* was submitted to Scientific Data (SDATA-22), International Semantic Web Conference (ISWC 2022) and Scientific Reports. For ISWC, although the reviews were positive, it was not published due to the number of papers submitted to the conference. For SDATA-22, it was not accepted because of the scope, as it would need not just the dataset created, but also a tool to facilitate data sharing. At the time of writing this work, we are still waiting for the reply from Scientific Reports.

REFERENCES

AUER, S. et al. **SciQA – A Question Answering Benchmark for Scholarly Knowledge**. Submitted.

CORTES, E. G.; BARONE, D. A. C. **Advancements in Multi-Documents Non-factoid Question Answering**. Forthcoming.

DEVLIN, J. et al. BERT: pre-training of deep bidirectional transformers for language understanding. **CoRR**, abs/1810.04805, 2018. Available from Internet: <http://arxiv.org/abs/1810.04805>.

DEVLIN, J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In: **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 4171–4186. Available from Internet: <https://aclanthology.org/N19-1423>.

JARADEH, M. Y.; STOCKER, M.; AUER, S. Question answering on scholarly knowledge graphs. In: HALL, M. et al. (Ed.). **Digital Libraries for Open Knowledge**. Cham: Springer International Publishing, 2020. p. 19–32. ISBN 978-3-030-54956-5.

NARAYANAN, D. et al. **Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM**. arXiv, 2021. Available from Internet: <https://arxiv.org/abs/2104.04473>.

OLVERA-LOBO, M.-D.; GUTIÉRREZ-ARTACHO, J. Open-vs. restricted-domain qa systems in the biomedical field. **Journal of Information Science**, Sage Publications Sage UK: London, England, v. 37, n. 2, p. 152–162, 2011.

QIU, X. et al. Pre-trained models for natural language processing: A survey. **Science China Technological Sciences**, Springer Science and Business Media LLC, v. 63, n. 10, p. 1872–1897, sep 2020. Available from Internet: <https://doi.org/10.1007%2Fs11431-020-1647-3>.

SUN, H. et al. Table cell search for question answering. In: **Proceedings of the companion publication of the 25th international conference on World Wide Web**. ACM - Association for Computing Machinery, 2016. Available from Internet: <https://www.microsoft.com/en-us/research/publication/table-cell-search-for-question-answering/>.

SUN, Y. et al. **Summarizing CPU and GPU Design Trends with Product Data**. arXiv, 2019. Available from Internet: <https://arxiv.org/abs/1911.11313>.

UNGER, C. et al. Question answering over linked data (qald-5). In: **CLEF**. [S.l.: s.n.], 2014.

VAKULENKO, S.; SAVENKOV, V. Tableqa: Question answering on tabular data. **arXiv preprint arXiv:1705.06504**, 2017.