Natural Language Processing Project

Albin Henriksson
Albin Holmkvist
Ludwig Forsberg
Max Björkander
Sebastian Broman

December 16, 2022

Contents

1 Introduction **2**

2 Method **2**
2.1 Dataset . . . 2
2.2 Implementation . . . 2
2.2.1 Entity and relationship linker . . . 3
2.2.2 Convolutional seq2seq . . . 3
2.3 BART & T5 . . . 3
2.4 Neural graph search . . . 4
2.5 Tokenization . . . 5
2.5.1 None . . . 5
2.5.2 Append-1 . . . 6
2.5.3 Append-2 . . . 6
2.5.4 Replace . . . 6
2.6 Prefixing . . . 6
2.7 Post Processing . . . 6
2.8 Evaluation . . . 7
2.9 ScQL . . . 7

3 Results **7**
3.1 Base Model . . . 8
3.1.1 Fully correct results . . . 8
3.1.2 More detailed statistics . . . 8
3.2 Neural graph search module . . . 9
3.3 scQL . . . 11
3.4 Website . . . 11

4 Discussion **13**
4.1 Accuracy of base model . . . 13
4.2 Neural Graph Search module . . . 13

5 Conclusion **14**
5.1 SPARQL generation . . . 14
5.2 Future work . . . 14

1 Introduction

In today's society technology is used more than ever and its widespread use generates tremendous amounts of data. The data is commonly stored in databases, and to access the data knowledge of how to communicate in a query language is required. The objective of this project is to make databases more accessible by translating an English query into a database query. This will allow any user without prior knowledge of database query languages to interact with the database.

The main interest is to query the semantic web using SPARQL queries. The term "Semantic Web" refers to W3C's vision of the Web of linked data, and it enables people to build vocabularies, write rules for the handling of data and create data stores on the Web. SPARQL is one of the technologies that empower linked data. [1] To find relevant methods the Question Answering over Linked Data (QALD) challenge leaderboard was examined. From the leaderboard, the best-performing techniques were examined closely, and based on that the initial methods were selected.

The chosen approach is to start with natural language to SPARQL queries. It is then hopefully possible to use the knowledge but also the model in order to achieve the same objective for scQL which is a custom query language. The problem can be split into two parts, first mapping each entity and relation to its corresponding URI and then predicting a query from natural language combined with the entity mappings.

The difficult part of this project is that the whole model needs to be state-of-the-art to be able to perform the best on a dataset that is small. This is to be able to meet the customers' demand to in the end be able to have the objective for scQL on a small dataset.

2 Method

This section describes the method of the project.

2.1 Dataset

The original idea was to use the QALD-9 dataset, which consists of 408 training queries and 150 test queries. Each query has a text prompt, a SPARQL query, and the corresponding results to that query. All queries are manually created and reference DBpedia 2016-10. However, we could not find a copy of DBpedia 2016-10 therefore we reran all the queries on DBpedia 2022-10, and only 90 queries worked. We decided to create our own QALD dataset by combining the datasets from all the previous QALD challenges and only keeping the queries that work on DBpedia 2022-10.

The LC-QUAD dataset was also used, it consists of 5000 SPARQL queries and references DBPedia 2016-04. These queries are automatically generated [2]. This dataset had the same problem as QALD-9 and we had to rerun the queries on DBpedia 2022-10 and 1400 queries worked.

The dataset for scQL consists of 893 programmatically-generated queries, each with a text prompt and a corresponding query. The dataset was created late in the project but was known to be small and therefore a small SPARQL dataset was chosen.

All datasets were split into an 80/20 train/test split. Meaning that we use 80 % of the queries for training and the remaining 20 % to evaluate the models.

2.2 Implementation

There are two different implementations of this project and one consists of three different parts,

- **Entity and relationship linker**
- **Seq2seq**
- **Neural graph search module**

These three parts will be put together like a pipeline with the entity and relationship linker first, then the seq2seq, and the last module will be the neural graph search module. The entity and relationship linker will be implemented with Falcon [3]. There will be three different types of seq2seq models that will be implemented which are the convolutional seq2seq [4], BART [5], and T5 [6]. The BART and T5 will also be tested on being the whole standalone model. This is discussed more in

Figure 1: Convolutional seq2seq pipeline.

the subsection about BART and T5. The last part which is the Neural graph search module will be implemented with BERT [7]. These are the different implementations that are used in this project.

The other implementation consists of that the part is going to be standalone models. The models that will be used for this are BART and T5. This implementation is discussed more in Section 2.3.

2.2.1 Entity and relationship linker

An entity and relation tagger maps each entity and relation to its URI which identifies it uniquely in a knowledge graph. Two possible approaches to entity tagging are EARL [8] and Falcon [3]. Entity tagger is especially dependent on training data to get good results.

Falcon will be used for entity and relationship linking. A natural language question is an input to the linker which then predicts which part of the question is entities and which words are in relation to each other. This is done in order to give more information to the seq2seq model and therefore improve the resulting query. The entities and the relations in conjunction with the natural language question are passed to the convolutional seq2seq model for further processing.

Falcon uses two data sources, an Elasticsearch database containing all entities and relations from DBpedia and a connection to DBpedia. The Elasticsearch database allows it to query words from the natural language question and rank them against entities and relations that exist in DBpedia. Queries directly to DBpedia are used to check different triples of entities and relations that exist. [3]

Because of how Falcon works this approach might be difficult to recreate for an scQL implementation.

Another possible approach might be to use an approach similar to genRL for relation-linking which shows good performance even without a database connection [9].

2.2.2 Convolutional seq2seq

A *seq2seq* (sequence to sequence) model in NLP is a model which can generate a sequence of tokens from a different sequence of tokens. A normal use case for this is machine translation, translating from one natural language to another. In this case, what is interesting is to translate the natural language to SPARQL and in extension scQL. This type of model will be the backbone of research in this project as there are many different pre-trained seq2seq models which have much better performance on natural language than can be achieved in the scope of this course. To examples of these models are T5 [6] and BART [5]. Through transfer learning, these models can be made to generate SPARQL or scQL queries instead. With the help of an entity and relation linker, hopefully, this performance can be improved even further.

The convolutional seq2seq model will have tokenized natural language as input. The tokenization is created by the *entity and relationship linker*. The seq2seq model will then predict the output sequence as a query containing tokens, which will be referred to as a *query silhouette*. The *Neural Graph Search module* will be used to mend any potentially broken relations query silhouette output from the seq2seq model.

2.3 BART & T5

BART is a model that combines Bidirectional and Auto-Regressive Transformers for sequence-to-sequence generation created by Facebook AI in 2019 [5]. The model is meant to be fine-tuned to downstream tasks such as translation, question answering, and summarization. The BART model has 406 million parameters.

T5 short for *Text-To-Text Transfer Transformer* is a model published by Google in 2020 [6]. The largest variant of T5 has 11 billion parameters. This model is also meant to be fine-tuned for different

NLP tasks. It is shown that fine-tuning T5 with a small labeled dataset can achieve higher performance than training on the dataset alone.

The BART and the T5 models will be used both as the seq2seq module in the pipeline above and as standalone models. As standalone models, it is meant that a natural language question will be sent into the BART model. Then the query in the dataset will be simplified to be able to train the model easier. This will require that the output from the BART model which is a simplified query be reconstructed into a real query in the end. This standalone pipeline is observed in Figure 2.

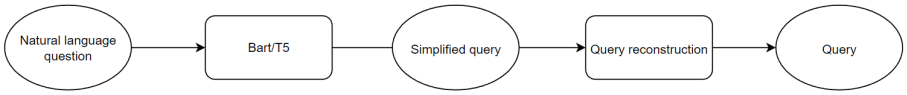

Figure 2: BART or T5 pipeline.

So, the BART and T5 models will fine-tune on a dataset that has simplified queries to hopefully make it easier for the models to learn faster and get better performance.

2.4 Neural graph search

When using an entity linker and predicting with the BART or the T5 model the predicted queries contain reasonable entities, but the same cannot be said for the relationship between the entities. The relation linking task is difficult because of the complexities of natural language [10]. Therefore the neural graph search module is built to address the problem of relation linking. The idea is to feed the model the natural language question along with the subject/object in the question, then the model should learn to predict the correct relation.

To implement the neural graph search model inspiration was taken from Purkayastha et al. [10]. It's built as a BERT-base module, and the architecture is shown in Figure 3.

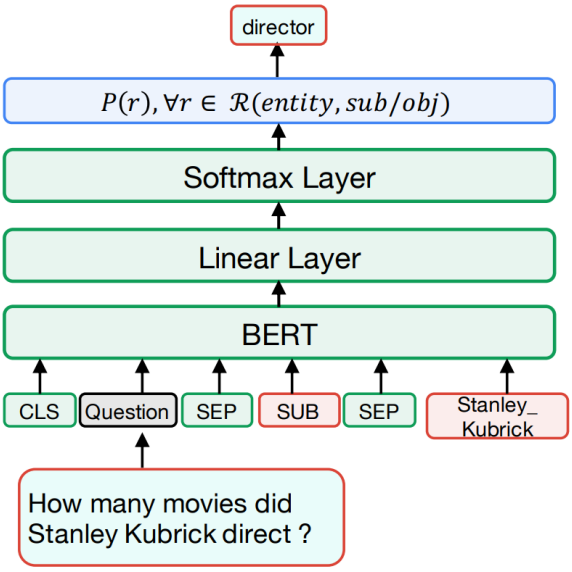

Figure 3: Structure of the neural graph search. Image taken from Purkayastha et al. [10].

The BERT layer outputs an encoding to a linear layer. The linear layer is fully connected to a softmax layer, which has one node for each class that can be predicted, and the value of the node represents the model's confidence for that prediction. The final blue layer in Figure 3 queries DBpedia to find valid relations between an entity and the sub/object. This limits the possible selections to valid relations in DBpedia and is only used when training is completed. During training, the predicted output is the relation with the highest probability from the softmax layer.

The predicted queries are needed to be able to run the neural graph search module. That is because the predicted queries have triplets which are in the format of e.g.

```
SELECT * FROM WHERE {Subject Relation Object},
```

where either subject or object should be a variable and the other two, URIs to specific things in the database. For example, could a query look like this,

```
SELECT ?uri WHERE {
?uri <http://dbpedia.org/property/leaderTitle> <http://dbpedia.org/resource/President_of_France> .
<http://dbpedia.org/resource/cash> <http://dbpedia.org/ontology/currency> ?uri . }
```

In this example, the query has two triplets and every variable is defined as, ?x. So, in the first triplet, the relation is, leaderTitle and object, President_of_France. These triplets are taken from every predicted query and then concatenated into BERT [7] in the format of Figure 4 where [CLS] and [SEP] is BERT-specific input and [CLS] denotes the start while [SEP] denotes a separator. This is the input and the format of the input into the first layer of the neural graph search module.

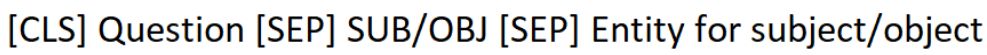

Figure 4: BERT, input format.

The loss function was created by combining cross-entropy loss l_c and a graph search loss l_v as $l = l_c(1-\alpha) + l_{gs}\alpha$ where alpha is a hyperparameter. Cross-entropy loss is calculated by comparing the predicted outcome with the correct outcome and penalizing the probability depending on how far away it is from the correct prediction [11].

The graph search loss punishes predictions of relations that do not exist for the sub/obj in DBpedia. To do this DBpedia is queried for all relations between an entity and the subject/object as $R = \langle e_s,\ r?,\ e_{o/s}\rangle$. Then the graph search loss of a predicted r_{pred} relation is defined as:

$$l_{gs}(r_{pred}) = \begin{cases} 0, \text{if relation exists in } R \\ 1, \text{if relation does not exist in } R \end{cases} \tag{1}$$

To train the model the LC-QuAD training data set was used by taking all the subjects and objects in every correct query and then saving the correct relation to the model to try and predict the correct relation to the subject or object to each query. Note that the question was also saved for each query to be able to create the input format that goes into BERT. The model predicts all the relations in both the training set and the test set to make sure that the model has the possibility to get the correct relation for each query. The test data from LC-QuAD was then used to create the same input into BERT as the training data by dividing each query into subjects or objects.

To evaluate the model the *accuracy* of relation predictions is measured. Accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}}. \tag{2}$$

2.5 Tokenization

To use the entities and relationships from the linker in previous steps, a way to add them to the training data is needed. There are many ways to achieve this but this paper explores the following four variants. The entity linker gives pairs of plain text strings and their corresponding URI, these are then used to give as much information as possible to the Seq2Seq model.

2.5.1 None

For baseline to compare the tokenization strategies the model was trained with no addition of entities or relationships.

2.5.2 Append-1

Append-1 is the first variant of appending all information to the end of the natural language question. The idea behind appending the information is that if the Seq2Seq model still can interpret this information in the end, we keep the normal structure of the initial sentence also. Append-1 works by appending one block per entity or relation that is found by the linker. The blocks have the structure shown in 5.

'PlainText' -res:ResourceName

Figure 5: Structure of the Append-1 blocks.

2.5.3 Append-2

Append-2 is the second append-based variant. In this case, the blocks are simply just the entities or relations, see 6.

res:ResourceName

Figure 6: Structure of the Append-2 blocks.

2.5.4 Replace

Replace is the last tokenization variant tried. In this variant, the question is modified so all occurrences of the plain text of each of the entities or relations are replaced by their matching entity or relation.

2.6 Prefixing

In SPARQL entities and relations are represented by their corresponding URI. This URI can be broken down into two parts, the first part is a prefix which can only be one of the very few values. These are replaced by a shortened form which the Seq2Seq model can interpret as one single word instead of multiple ones, thus reducing the vocabulary and improving performance see 7 for the mapping.

```
prefixes = {
    "http://dbpedia.org/resource/": "res:",
    "http://dbpedia.org/ontology/": "dbo:",
    "http://dbpedia.org/property/": "dbp:",
    "http://www.w3.org/2000/01/rdf-schema#": "rdfs:",
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#": "rdf:",
    "http://dbpedia.org/class/yago/": "yago:",
    "http://www.wikidata.org/prop/direct/": "wdt:",
    "http://www.wikidata.org/entity/": "wd:",
    "http://www.wikidata.org/prop/": "p:",
    "https://w3id.org/payswarm#": "ps:",
    "http://www.wikidata.org/prop/qualifier/": "pq:",
    "http://www.bigdata.com/rdf#": "bd:",
    "http://wikiba.se/ontology#": "wikibase:",
    "http://www.w3.org/2004/02/skos/core#": "skos:",
}
```

Figure 7: Dictionary containing prefixes and their shortened form.

2.7 Post Processing

Since the Seq2Seq models used were designed for natural language and not for SPARQL. The main problem with this is that the tokenizer does not understand the syntax of SPARQL fully. To remedy this, some fixes were implemented as a last part of the pipeline to make as many predicted queries runnable as possible.

From observation, sometimes the BART model does not understand that there should be a space between the question mark, which is normal since usually in natural language question marks are immediately after another character. This could sometimes result in two "words" being combined without a space. To remedy this all instances of "?" are prepended with a space, this makes sense in SPARQL since there should always be a space before every "?" and excess spaces can never impact the query.

Another frequent part of SPARQL queries is a "." surrounded by spaces. This dot has to be surrounded by spaces but sometimes the model generates sections similar to "?xyz." which needs to be replaced by "?xyz .". This can easily be done by finding all instances of "?", finding the next space, and checking if there is a dot just before that space. If there is, a space is added before the dot also.

2.8 Evaluation

The evaluation of the models will be done the same way that was stated in the QALD-9 challenge but with an addition of the metrics on how similar the queries are [12].

Since scQL answers can vary over time they cannot be evaluated the same way as the datasets for SPARQL. It will instead be evaluated by comparing the original query to the generated one.

To specify the evaluation a bit more there will be macro and micro scores of Precision, Recall, and F1 score. Macro means that precision, recall, and F1 score will be evaluated per query and then the average for the metrics over all queries is the final score. Micro means that true positives (TP), false positives (FP) and false negatives (FN) are counted for each query and then the sum of all TP, FP, and FN independently is used in the end to calculate the precision, recall, and F1 score.

For the evaluation stated in the QALD-9 challenge the answers from the database are compared so, if the answer for the predicted query is in the ground truth answers, then it counts as a TP. If the answer for the predicted query is not in the ground truth answers it counts as an FP. In the end, if there are answers in the ground truth which have not been an answer for the predicted query, it counts as FN.

For the evaluation of the query strings the TP, FP, and FN are counted differently. A TP for the query strings is if a word from the predicted query is in the ground truth query. If the predicted query has a word that is not in the ground truth query it is counted as a FP. A FN is then lastly counted as if the ground truth query has a word that is not in the predicted query.

Precision is defined as,

$$Precision = \frac{TP}{TP + FP}, \text{ and} \tag{3}$$

the recall is defined as,

$$Recall = \frac{TP}{TP + FN}, \text{ and} \tag{4}$$

F1-score is defined as,

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \text{ [13]} \tag{5}$$

2.9 ScQL

The final part of the project is to test the performance of the same model trained on scQL. The model will be trained with similar questions as for the previous parts but will have the resulting queries be valid scQL queries. For scQL there is no entity and relation linker meaning none of the tokenization strategies can be used. The scQL dataset did not use any prefixes this model uses complete URI:s.

3 Results

This chapter presents the results of evaluating variations of the pipeline on different datasets.

3.1 Base Model

In the following section, statistics about the pipeline up to, and including, the seq2seq model are presented. During development, BART seemed to perform better than T5. Training the model with T5 was also more computationally expensive making it slower to evaluate. Both pre-trained models still outperformed the standalone seq-2-seq model. Therefore, we focused our evaluations with only BART as the seq2seq part in the pipeline.

3.1.1 Fully correct results

In Table 1, Table 3 the percentage of fully correct queries is presented. This means that the model must return a string that is exactly the same as the gold standard query. In Table 4 and Table 2 however, the percentage of test-data entries where the result of the executed generated query includes exactly the same entities as the executed gold standard query is presented. In all four tables, the green cell marks the optimal epoch number(s) for that tokenization on the given dataset.

Epoch	None	A-1	A-2	R
1	0%	0%	0%	0%
2	0%	0%	0%	0%
3	0%	0%	0%	0%
4	0%	0%	0%	0%
5	0%	0%	0%	0%
6	0%	1.12%	0%	0%
7	0%	0%	0%	1.12%
8	0%	0%	0%	0%
9	1.12%	0%	0%	0%
10	0%	0%	0%	0%

Table 1: String evaluation statistics on QALD.

Epoch	None	A-1	A-2	R
1	2.25%	13.48%	12.36%	12.36%
2	4.49%	8.99%	17.98%	17.98%
3	10.11%	11.24%	14.60%	14.60%
4	10.11%	11.24%	16.85%	16.85%
5	13.48%	17.98%	16.85%	16.85%
6	12.36%	14.60%	14.60%	14.60%
7	11.24%	17.98%	17.98%	17.98%
8	13.48%	14.61%	20.22%	20.22%
9	16.85%	17.98%	17.98%	17.98%
10	12.36%	17.98%	23.60%	23.60%

Table 2: Executed queries evaluation statistics on QALD.

3.1.2 More detailed statistics

In Table 5, Table 6, Table 7 and Table 8 more detailed evaluation metrics such as Precision, Recall and F1 are presented on both Macro and Micro level.

In the previously mentioned tables, the green cells mark the best-performing instance of that metric. In Table 7 it is shown that, for strings, Append-2 after 9 epochs is the best tokenization on all metrics except Precision Micro. In Table 6 it is shown that, for those metrics, Append-1 after 3 epochs is performing better. For the executed case it is shown in Table 6 that Append-1 after 3 epochs is best in terms of the macro evaluation metrics while no tokenization is best in terms of the micro evaluation metrics as shown in Table 5.

Epoch	None	A-1	A-2	R
1	2.33%	9.92%	8.95%	2.92%
2	1.75%	11.09%	10.50%	6.23%
3	2.53%	12.84%	10.12%	8.56%
4	3.50%	12.26%	10.89%	7.00%
5	4.47%	12.45%	10.70%	8.17%
6	4.86%	13.23%	11.67%	8.95%
7	3.70%	12.66%	12.65%	7.78%
8	5.45%	13.23%	11.87%	9.14%
9	5.64%	13.23%	12.65%	9.92%
10	5.25%	12.26%	12.45%	8.37%

Table 3: String evaluation statistics on LC-QuAD.

Epoch	None	A-1	A-2	R
1	8.75%	27.82%	18.09%	8.75%
2	9.92%	30.16%	26.46%	16.73%
3	13.62%	35.21%	27.43%	22.18%
4	17.32%	31.12%	27.04%	20.43%
5	17.51%	31.32%	28.40%	23.35%
6	19.07%	31.52%	29.96%	25.88%
7	17.51%	30.16%	31.91%	24.90%
8	20.62%	29.38%	31.13%	27.24%
9	20.82%	28.99%	32.88%	26.07%
10	20.43%	28.21%	32.68%	26.46%

Table 4: Executed queries evaluation statistics on LC-QuAD.

None (epoch 9)	Strings	Executed
Precision Macro	0.7572238181187599	0.2336005939431804
Recall Macro	0.7535508482006541	0.2398016588434398
F1 Macro	0.7537410702145709	0.23121850577655148
Precision Micro	0.7491743119266056	0.7550957381099327
Recall Micro	0.7460259455508862	0.6321913380736829
F1 Micro	0.7475968140620708	0.6881992821965301

Table 5: String and executed evaluation statistics on LC-QuAD using no tokenization after 9 trained epochs.

Append-1(epoch 3)	Strings	Executed
Precision Macro	0.8002376082629009	0.396077832204907
Recall Macro	0.788359235878692	0.40093088736786964
F1 Macro	0.7925030213519991	0.3929077473732153
Precision Micro	0.7926284437825764	0.7294998987649174
Recall Micro	0.7780010962908825	0.2745142857142836
F1 Micro	0.7852466574458276	0.39891496895937784

Table 6: String and executed evaluation statistics on LC-QuAD using append-1 tokenization after 3 trained epochs.

3.2 Neural graph search module

All results captured for the neural graph search module were only tested by themselves on the correct queries where the correct relations were predicted from the subject or object. The hyperparameters of the results are as observed in Table 9 if nothing else is stated.

An experiment was made where the result is shown in Figure 8. The experiment was made of

Append-2(epoch 9)	Strings	Executed
Precision Macro	0.8003840949366244	0.37356019548862884
Recall Macro	0.7885176593834186	0.38305399548515034
F1 Macro	0.7922460285184943	0.3698258935866408
Precision Micro	0.7895126922364276	0.7133526850507879
Recall Micro	0.7785492417321396	0.5953967292549898
F1 Micro	0.7839926402943883	0.6490590953614244

Table 7: String and executed evaluation statistics on LC-QuAD using no Append-2 tokenization after 9 trained epochs.

Replace(epoch 8)	Strings	Executed
Precision Macro	0.773943598554494	0.3039276717863791
Recall Macro	0.7631454850812833	0.3054994848535596
F1 Macro	0.7662771340364111	0.2981294725095505
Precision Micro	0.7667844522968198	0.34752764713359163
Recall Micro	0.7533345514343139	0.3280537556995401
F1 Micro	0.7599999999999999	0.33751003019508397

Table 8: String and executed evaluation statistics on LC-QuAD using replace tokenization after 8 trained epochs.

Hyper params	Value
Learning rate	0.001
Batch size	8
α	0

Table 9: Standard hyperparameters of the neural graph search model.

different alphas from 0 to 1 and their accuracy of them after 100 epochs was captured. The accuracy with and without graph search is also captured. In the figures after this on the neural graph search module, the α of 0.2 is deemed as the best one.

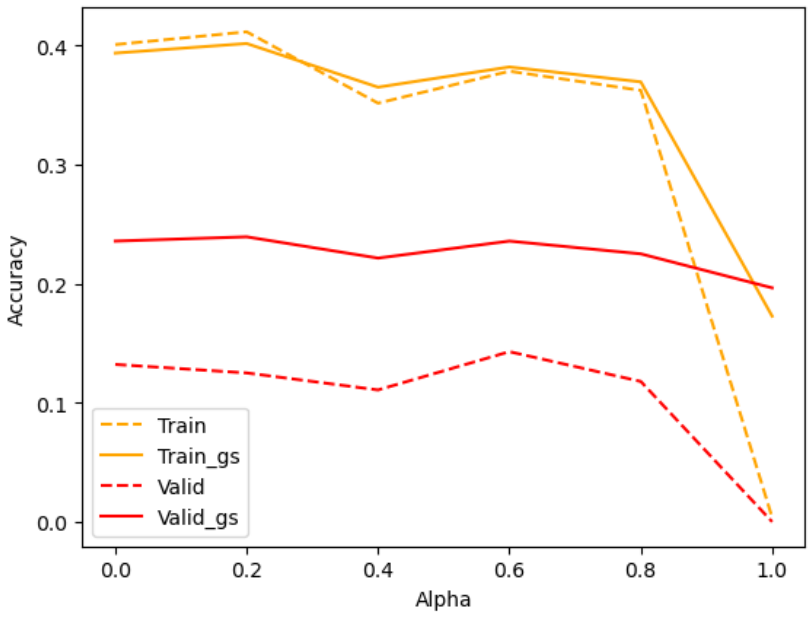

Figure 8: Neural graph search accuracy with different α for 100 epochs of training.

The best result on the neural graph search module is demonstrated in Figure 9 where the model

is fitting well to the training data but not as well to the validation data. This result was captured after 2400 epochs as observed in the figure. The same result as shown in the figure is observed in Table 10 where it is seen that the graph search improves the validation and test accuracy by around ten percentage points.

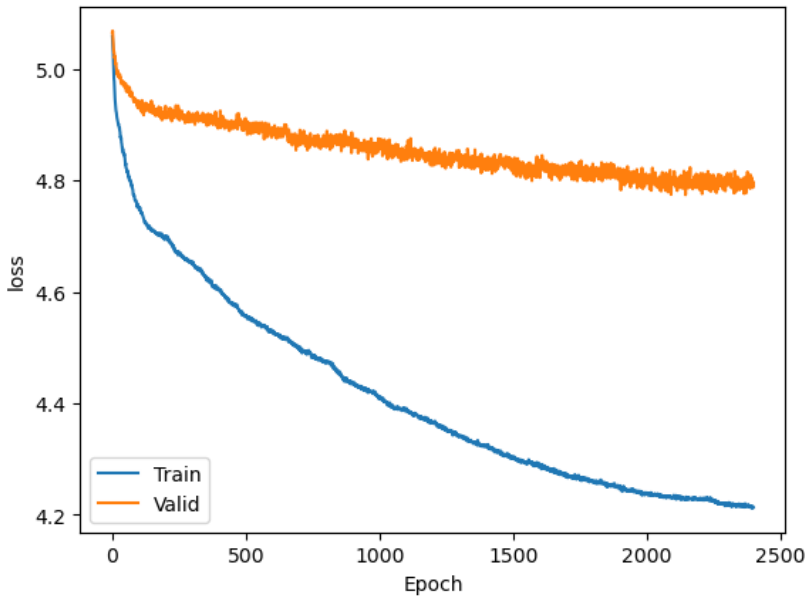

Figure 9: Training and validation accuracy for the neural graph search model with α of 0.2, batch size of 8, and learning rate of 0.0005.

	Train	Valid	Test
Accuracy without GS	**88.5%**	23.6%	26.2%
Accuracy with GS	75.8%	**36.4%**	**37.5%**

Table 10: Neural graph search model accuracy with and without graph search (GS).

3.3 scQL

From the results for SPARQL on QALD and LC-QUAD BART-base was selected as the most appropriate model for scQL. Splitting the scQL dataset into 80 % training samples and 20 % test samples gave the model 714 training queries and 179 test queries.

After fine-tuning the BART-base model for one epoch on the training samples it correctly predicted 4 out of 179 test queries. Meaning it had an accuracy of approximately 2.2%. For ten epochs it correctly predicts 5 out of 179 test queries, with approximately an accuracy of 2.8%. However, in all of the queries, both variants correctly predicted whether it was a SELECT or a PERFORM query.

3.4 Website

A proof-of-concept website was developed in order to test the model from a user's experience. The website is simplistic with only a header, an input field for the question, a button for processing the question, the predicted query, and the result of executing that query. The intended use is to enter a question into the input field, press the button, retrieve the result, and potentially redirect to the requested resources. In Figure 10 four instances of the website are shown.

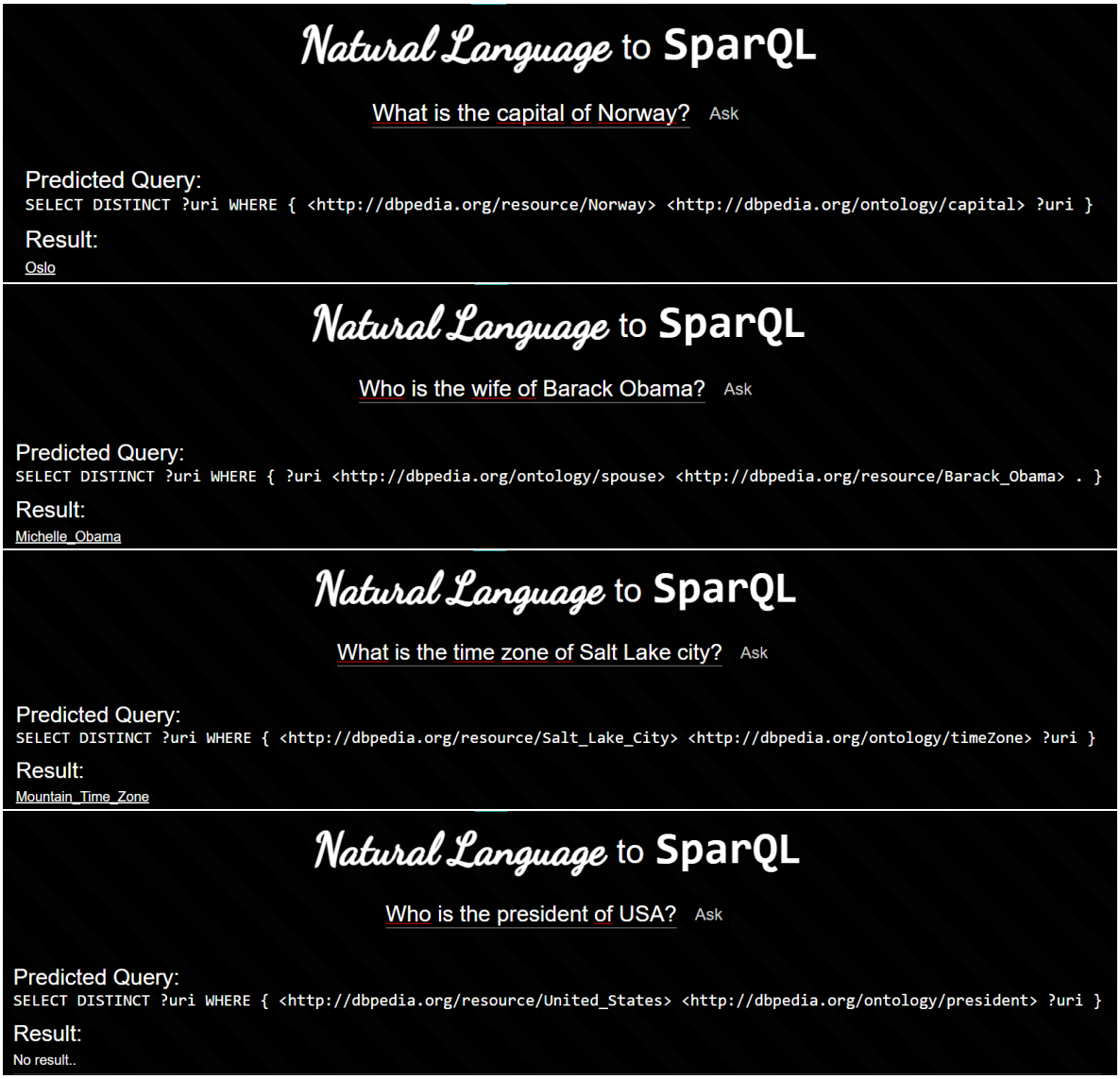

Figure 10: Four examples of questions asked on the website with the given response. The results are links to the corresponding resource for DBpedia.

4 Discussion

The results and method are discussed in this section, using the previous tables as a base.

4.1 Accuracy of base model

Since T5 is a larger model than BART leading to a larger training time. Training T5 for more epochs might have led to better performance but due to computational constraints, we were not able to test this. Using any of these two models might still be preferable compared to a standalone seq-2-seq model since they already have an understanding of the English language.

The resulting model without the Neural Graph Search module performed quite well on the LC-QuAD dataset. The best model had a 35.21% accuracy when comparing executed query results. From a visual inspection of some of the incorrect results, the queries were all executable but usually, the relations were incorrect or the SPARQL triple was shuffled meaning the result was for something else. One observed example of the previously mentioned problem is that for the query "How many states are in the United States" the model formulated the query to answer the query "How things are there which has the United States as their state" which results in nothing since the United States is not a state to anything in DBpedia.

Something which seemingly would have improved accuracy greatly would have been if the neural graph search module would have been connected in the pipeline and evaluated. Since, from observation, many of the incorrect queries had incorrect relations it seems feasible that accuracy would have been improved. The reason this module was not evaluated in combination with the regular model was due to time constraints. Theoretically, if the neural graph search model would have performed as well as it did on test data on the incorrect already predicted queries, the resulting accuracy would be improved to around $(1 - 0.3521) \cdot 0.375 = 0.2429625 = 24.29625\%$ resulting in a total accuracy of 59.50625%. This calculation assumes that the neural graph search model performs at a 37.5% accuracy on the set of incorrectly generated queries. The main reason why this would probably have been an improvement is that most of the incorrect relations were relations missed by the entity and relationship linking and were nonexistent in DBpedia. If an initial check if the relationships existed for the chosen entity, it would be possible to make sure that no correct relations were overwritten by the module.

When evaluating the pipeline on scQL two major parts were missing: the entity and relationship linker, and comparing the results of the query instead of the query itself. The best string evaluations for SPARQL improved from 5.6% to 13.23% when using the append-1 strategy for entities and relations. It should also be fair to assume that we are not interested in a query itself but instead in what the query returns. Even though the best accuracy of string comparisons for LC-QUAD is only 13.23 % the queries generated return the correct response 35.21 % of the time.

4.2 Neural Graph Search module

The neural graph search module in its current state shows good potential for correcting relations. One factor that limits the performance is the dataset used for training. The training data contains about 1400 example queries, and the model has 441 relation classes to predict. If an even distribution of the dataset is assumed each relation has approximately 3 examples, which might be too little to learn a general case for that relation. The neural graph search implemented by Purkayastha et al. [10] uses the entire DBpedia as training examples and predicts relation among 61623 candidates. This scale was not an option for us due to limited computational resources.

A delimitation was that the neural graph search module was only trained on specific triplets. This simplified the process of extracting the triplets from the correct queries and makes sure that the correct triplets are extracted. Also, this is a constraint because the model cannot predict all the correct relations and is therefore a bit worse than it could have been if it could predict the relations of all queries.

The alpha of 0.2 was deemed as the best one because it seemed the best from Figure 8 but also lets the alpha stay at a relatively low value which makes the model prediction the heaviest factor of the loss.

5 Conclusion

This chapter describes the results shortly and what can be brought to a future project in this domain.

5.1 SPARQL generation

From the results, it is apparent that a somewhat decent SPARQL-generating model can be achieved using a pipeline consisting of an entity linker and a seq2seq model with some additional minor fixes after. However, since accuracy is around 35% there are many scenarios where it is not good enough. Additionally, the neural graph search model seems to potentially be able to increase the overall accuracy of the model greatly.

5.2 Future work

For future work, the application of using a neural graph search model connected to the generated output of the SPARQL queries could be interesting. Furthermore, additional seq2seq models or other tokenization techniques could be tried but ultimately it seems like, a new model structure entirely would be needed for a model to perform much better. As it currently stands, the model cannot generate scQL queries well but some of the results show promise and could potentially be improved greatly if the model had an entity and relationship linker it seems like it can have comparable results to the SPARQL results.

References

[1] "Semantic web, howpublished = https://www.w3.org/standards/semanticweb/, note = Accessed: 2022-11-04."

[2] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann, "Lc-quad: A corpus for complex question answering over knowledge graphs," in *International Semantic Web Conference*, pp. 210–218, Springer, 2017.

[3] A. Sakor, I. Onando Mulang', K. Singh, S. Shekarpour, M. Esther Vidal, J. Lehmann, and S. Auer, "Old is gold: Linguistic driven approach for entity and relation linking of short text," 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), pp. 2336–2346, Association for Computational Linguistics, June 2019.

[4] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," 2017.

[5] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "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), pp. 7871–7880, Association for Computational Linguistics, July 2020.

[6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[8] M. Dubey, D. Banerjee, D. Chaudhuri, and J. Lehmann, "Earl: Joint entity and relation linking for question answering over knowledge graphs," 2018.

[9] G. Rossiello, N. Mihindukulasooriya, I. Abdelaziz, M. Bornea, A. Gliozzo, T. Naseem, and P. Kapanipathi, "Generative relation linking for question answering over knowledge bases," in *International Semantic Web Conference*, pp. 321–337, Springer, 2021.

[10] S. Purkayastha, S. Dana, D. Garg, D. Khandelwal, and G. P. S. Bhargav, "Knowledge graph question answering via sparql silhouette generation," 2021.

[11] K. E. Koech, "Cross-entropy loss function." https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e, 2020.

[12] R. Usbeck, R. Gusmita, M. Saleem, and A.-C. Ngonga Ngomo, "9th challenge on question answering over linked data (qald-9)," 11 2018.

[13] T. Kanstrén, "A look at precision, recall, and f1-score." https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec. (accessed: 2022-11-30).