Master of Technology

Practical Language Processing Project

Conversational Natural Language Query of Relational and Non-relational Databases

Team members:

Benjamin Quek Xiang Yi	A0229973M
Chwa Choon Xiang	A0229962R
Gerard Ong Zi Quan	A0229967H

Table of Contents

List of Figures

List of Tables

ABSTRACT

Data-driven insights have become a key focus in organisations to improve operational efficiency and business decisions. Information sources such as reports, tables and documents could be stored in various types of relational and non-relational databases. Business users are required to extract information from these databases; however, they may face difficulty in doing so due to the lack of familiarity with database query languages. This project aims to resolve this issue by developing a solution to convert natural language query to database query statements, which include Structured Query Language (SQL) and Graph Query Language (GQL). This involved text pre-processing methods to generate query pairs from data sources and question templates, and Seq2Seq model was trained to generate the corresponding SQL statements. Automated entity-relation and question-answer generation methods were also used for self-supervised training of relationship classification models for slot filling of GQL statements. A conversional user interface was also developed to obtain user's queries and perform intent classification to the appropriate query module, and to generate natural language responses based on the answers retrieved from the databases. This solution demonstrated the capabilities of natural language processing techniques in establishing a business automation system, for conversational query of information stored in relational and non-relational databases.

1 PROBLEM STATEMENT & OBJECTIVES

1.1 Natural Language Query: Background, Issues & Proposed solution

Data is growing at 2.5 quintillion bytes each day, and is expected to reach 181 zetabytes by 2025 [1, 2]. With the abundance of data generated and collected, companies are increasingly adopting a data-driven approach to improve operational efficiency and gain business insights. Within a company, different types of data may be stored in various methods. Tabular data generated from sources such as financial accounting, manufacturing output and inventory status, are usually structured and could be stored in relational databases such as Oracle, MySQL and PostgreSQL. Companies may also have textual data in the form of reports, documents, or webpages, which are classified as unstructured data and stored in JSON or XML formats. Further information extraction on these unstructured data could yield information such as entity terms and the relationship links between them. This business information could be stored in non-relational graph databases such as Neo4j, DGraph and JanusGraph, as a representation of domain specific ontology of company entities and processes.

While information is stored in databases, methods are required to perform operations and retrieve specific data for business users. This could involve constructing a query to create, read, update, or delete (CRUD) data in the database. Structured Query Languages (SQL) and Graph Query Languages (GQL) are designed to efficiency perform such operations on relational and non-relational databases respectively. However, these query languages are mainly developed for programming tasks, with statement syntax differing from natural language. Hence, business users without programming knowledge would not be familiar with such query languages, and may face difficulty in retrieving data stored within databases. Currently, such data requests are routed to IT professionals which could also place a strain on IT manpower resources given the increasing number of data analytics projects.

To address the issue, our team proposes a no-code solution to assist business users to retrieve information from databases. We present methods to convert natural language queries provided by business users, to the corresponding SQL or GQL statements for database operations. This will be coupled with a conversation user interface (UI) that will handle the user's intent and route the query to the appropriate query language conversion module. Following the retrieval of specific data, it will be parsed to a response generator module to construct a natural language response to the user's query, to improve the conversational experience with the user. In addition, the system is also designed to handle general conversations and question-answering (QA) retrieval from a defined corpus. The proposed conversational UI with text2SQL and text2GQL functions, serves as a business automation solution that enables non-IT users to harness the information within databases more effectively for analytics and question-answering tasks, and to reduce workload strain on IT professionals.

1.2 Project Scope

This project aims to develop a conversational UI with text2SQL and text2GQL functions for automation of information retrieval and question-answering tasks. It consists of the following scope:

- Develop a conversational UI (Chatbot) as the front-end question-answering machine to obtain business user's query and return the generated answers retrieved from the backend databases, both in natural language format. The chatbot will interact with the user to obtain the appropriate intent to parse to the backend query modules, while handling general conversational and question-answering statement retrieval from a defined corpus.

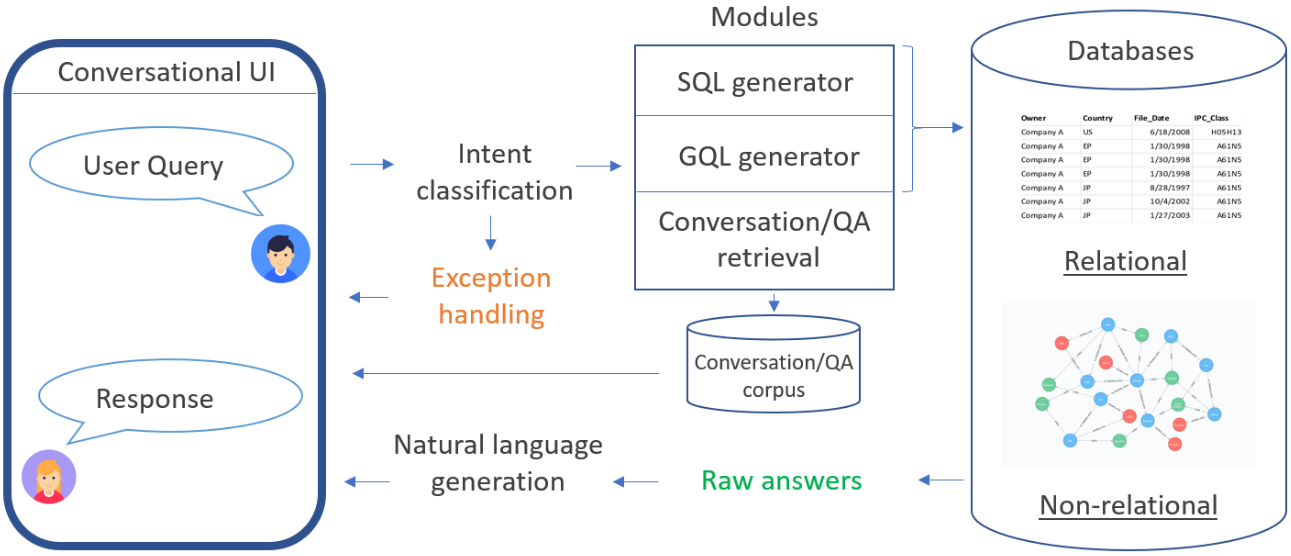
- Develop method for conversion of natural language query to SQL format (text2SQL), to query the backend relational databases. This serves to perform the required SQL operations such as SELECT, WHERE, ORDER BY, to fulfil the answer requested by the user.
- Develop method for conversion of natural language query to Graph Query Language (GQL) format (text2GQL), to query the backend non-relational databases. This serves to perform the search function of entities and their relationship links, to fulfil the answer needed by the user.

The proposed solution includes techniques such as text classification, information extraction, machine translation and text similarity, enabled by machine learning and deep learning methods. Dataset is obtained both from online text repositories and generated through self-supervised approaches.

2 SOLUTION

2.1 System overview

The system overview is shown in Figure ***1***. It consists of a conversional UI front-end module which interacts with users and obtains the query intent. Following the intent classification of the natural language query, it will be parsed to the SQL, GQL or conversation/QA retrieval modules, for processing into an appropriate query format for database operations. The answers will be retrieved from the relational or non-relational databases, which is subsequently used for generation of natural language response and returned to the user as a reply.



Owner	Country	File_Date	IPC_Class
Company A	US	6/18/2008	H05H13
Company A	EP	1/30/1998	A61N5
Company A	EP	1/30/1998	A61N5
Company A	EP	1/30/1998	A61N5
Company A	JP	8/28/1997	A61N5
Company A	JP	10/4/2002	A61N5
Company A	JP	1/27/2003	A61N5

Figure 1: Overview of solution for conversational natural language query of relational and non-relational databases.

The proceeding sections detail the solutions for conversion of natural language to SQL & GQL, and conversational UI system design.

2.2 Natural language to SQL

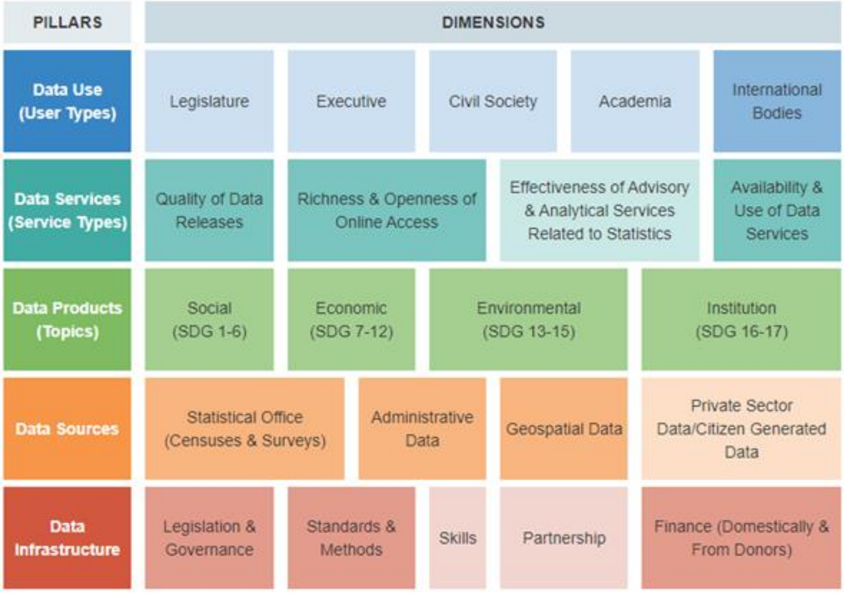
2.2.1 SQL Dataset

The dataset we chose as our representative SQL table to be queried was the Statistical Performance Indicators (SPI) dataset by The World Bank [3].

The Statistical Performance Indicators measure the capacity and maturity of national statistical systems by assessing the use of data, the quality of services, the coverage of topics, the sources of information, and the infrastructure and availability of resources.

They provide stakeholders from policymakers to stock market analysts with the latest data on the country's socioeconomic developments. Formally, the SPI consists of a collection of 22 unique dimensions that cover across five key pillars of a country's statistical performance:

- data use
- data services
- data products
- data sources
- data infrastructure



PILLARS	DIMENSIONS				
Data Use (User Types)	Legislature	Executive	Civil Society	Academia	International Bodies
Data Services (Service Types)	Quality of Data Releases	Richness & Openness of Online Access	Effectiveness of Advisory & Analytical Services Related to Statistics	Availability & Use of Data Services	
Data Products (Topics)	Social (SDG 1-6)	Economic (SDG 7-12)	Environmental (SDG 13-15)	Institution (SDG 16-17)	
Data Sources	Statistical Office (Censuses & Surveys)	Administrative Data	Geospatial Data	Private Sector Data/Citizen Generated Data	
Data Infrastructure	Legislation & Governance	Standards & Methods	Skills	Partnership	Finance (Domestically & From Donors)

Figure 2: Pillars and dimensions of a country's statistical performance.

As seen in the above figure, each of the five key pillars has multiple sub-dimensions/indicators and each contribute to a country's score in their parent pillar. For the dataset, data from 174 countries were captured across a timespan from 2004 – 2019.

2.2.2 Data pre-processing

The dataset was first provided as a .csv file so we had to first convert it into a suitable format that can be queried using SQL, as such we converted it to an SQLite db format such that we can use third-party software such as DB Browser to test the queries generated by our model to ensure that the SQL is executable and produces the correct results.

The database consists of 79 columns, with most of the columns containing integer values and a few containing string values.

String containing columns:

1. Country
2. Iso3c
3. Income
4. Region

The main challenge with Seq2Seq problems such as this with translating natural language to SQL is that there are little readily available datasets containing corresponding pairs of the 2 languages, especially so if targeting a specific industry/use-case.

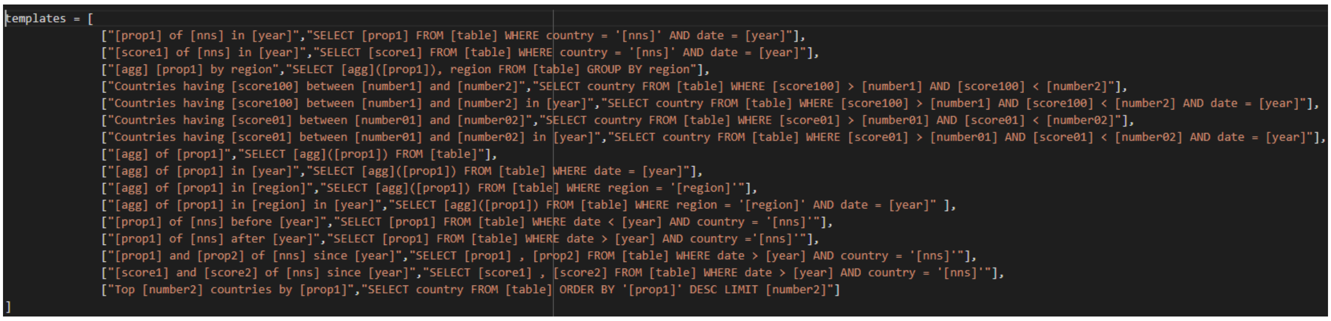
The most well-known open-source natural language-SQL dataset is WikiSQL [4], which consists of 80,654 manually annotated example questions, and their equivalent SQL queries and results across 24,241 tables from Wikipedia. While it is widely used in research and academia as a one-stop dataset for training and evaluating new Natural language querying (NLQ) information retrieval models, we find that for our use case it is better instead to create a pipeline that allows superusers to work with business users to create their own query dataset in an efficient way. This is because although training the model on the WikiSQL database will result in a more generalized model, through some quick testing we found that due to the training data being too general the model does not perform well when used in a more finance-oriented use-case like ours.

Usually, business users have a good idea of what are the most common queries that are made in natural language and superusers have a good understanding of the different columns within a table in the database. We thus propose a template based natural language – SQL query generation approach to create a sufficient use-case specific dataset to train and evaluate the seq2seq model.

When studying the database table, we noticed that the most logical form of questions for humans are to ask for properties related to one or a combination of the following intents:

1. Country
2. Region
3. Year

As such we have created the template based on the idea of users looking for information regarding the three intents [5].



```
templates = [
                ["[prop1] of [nns] in [year]","SELECT [prop1] FROM [table] WHERE country = '[nns]' AND date = [year]"],
                ["[score1] of [nns] in [year]","SELECT [score1] FROM [table] WHERE country = '[nns]' AND date = [year]"],
                ["[agg] [prop1] by region","SELECT [agg]([prop1]), region FROM [table] GROUP BY region"],
                ["Countries having [score100] between [number1] and [number2]","SELECT country FROM [table] WHERE [score100] > [number1] AND [score100] < [number2]"],
                ["Countries having [score100] between [number1] and [number2] in [year]","SELECT country FROM [table] WHERE [score100] > [number1] AND [score100] < [number2] AND date = [year]"],
                ["Countries having [score01] between [number01] and [number02]","SELECT country FROM [table] WHERE [score01] > [number01] AND [score01] < [number02]"],
                ["Countries having [score01] between [number01] and [number02] in [year]","SELECT country FROM [table] WHERE [score01] > [number01] AND [score01] < [number02] AND date = [year]"],
                ["[agg] of [prop1]","SELECT [agg]([prop1]) FROM [table]"],
                ["[agg] of [prop1] in [year]","SELECT [agg]([prop1]) FROM [table] WHERE date = [year]"],
                ["[agg] of [prop1] in [region]","SELECT [agg]([prop1]) FROM [table] WHERE region = '[region]'"],
                ["[agg] of [prop1] in [region] in [year]","SELECT [agg]([prop1]) FROM [table] WHERE region = '[region]' AND date = [year]" ],
                ["[prop1] of [nns] before [year]","SELECT [prop1] FROM [table] WHERE date < [year] AND country = '[nns]'"],
                ["[prop1] of [nns] after [year]","SELECT [prop1] FROM [table] WHERE date > [year] AND country ='[nns]'"],
                ["[prop1] and [prop2] of [nns] since [year]","SELECT [prop1] , [prop2] FROM [table] WHERE date > [year] AND country = '[nns]'"],
                ["[score1] and [score2] of [nns] since [year]","SELECT [score1] , [score2] FROM [table] WHERE date > [year] AND country = '[nns]'"],
                ["Top [number2] countries by [prop1]","SELECT country FROM [table] ORDER BY '[prop1]' DESC LIMIT [number2]"]
]
```

Figure 3: Templates of natural language to SQL queries.

As seen in the above figure, we created a set of 16 natural language – SQL query templates to automate the generation of our training and evaluation dataset. This list is not exhaustive and is designed to be easily expandable with time as business users notice new query patterns in natural language.

The placeholder values within the templates that point to different things:

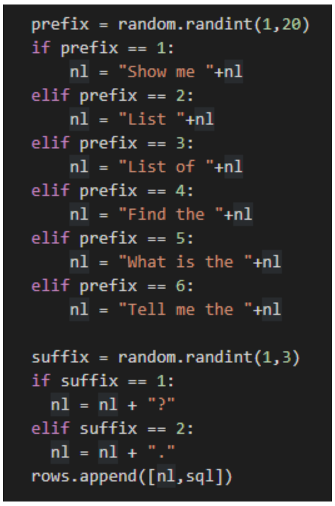
Table 1: Placeholder references for natural language-SQL query templates.

Placeholder	What it refers to
nns	Country column
year	Year column
region	Region column
prop1/prop2	Columns of properties that countries possess that are not “scores”
table	Sepecific SQL table to be queried (spi_index_labelled_vFinal)

agg	Aggregation pairs in both natural language and SQL (e.g. ["average","avg"])
score01	Columns of “score” that have a range of values from 0-1 (e.g., ODIN Open Data Openness Score)
number01/number02	Random integers between 0-1, with number02>number01
score100	Columns of “score” that have a range of values from 0-100 (e.g., SPI Overall Score)
number1/number2	Random integers between 0-100, with number2>number1
score1/score2	All columns of “score”

Besides the first three which represent intents, the above table show the slots of which the queries can take. We categorize the slots first into “score” slots and non “score” slots. For score related slots we further categorized them according to the range of values they can take; integer values of range between 0-1 inclusive and integer values of range between 0-100 inclusive. By recursively and randomly having different combinations of intents and slots we can easily generate natural language to SQL query pairs.

In addition to the above, we add prefixes and suffixes to the natural language queries randomly to make them more human-like.

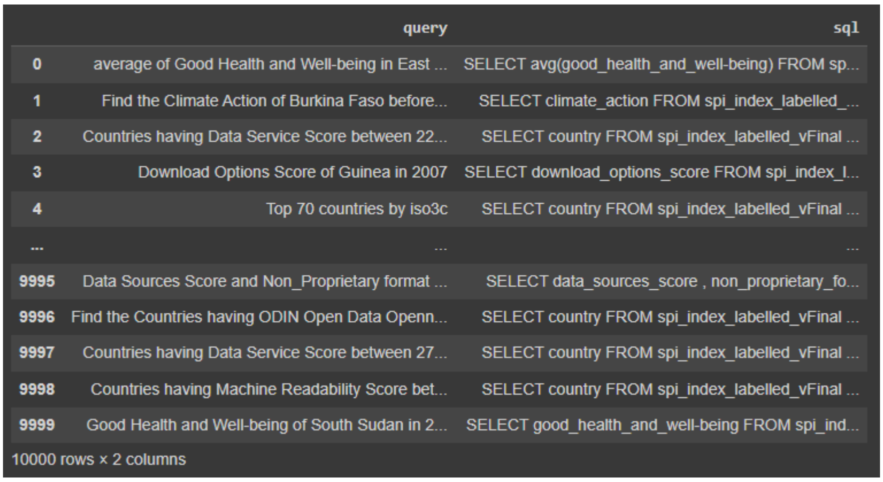


```
prefix = random.randint(1,20)
if prefix == 1:
    nl = "Show me "+nl
elif prefix == 2:
    nl = "List "+nl
elif prefix == 3:
    nl = "List of "+nl
elif prefix == 4:
    nl = "Find the "+nl
elif prefix == 5:
    nl = "What is the "+nl
elif prefix == 6:
    nl = "Tell me the "+nl

suffix = random.randint(1,3)
if suffix == 1:
  nl = nl + "?"
elif suffix == 2:
  nl = nl + "."
rows.append([nl,sql])
```

Figure 4: Set of prefixes and suffixes for natural language queries.

By doing so, we can automatically generate thousands of query pairs that makes sense in both natural language and SQL. For this project we generated 10,000 pairs for model training/evaluation purposes.



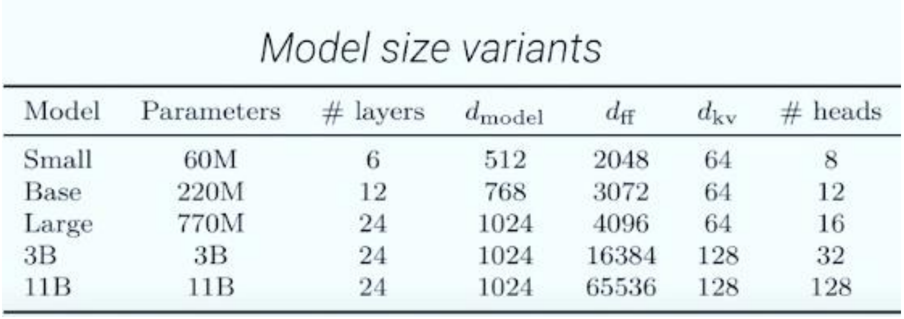
```
                                         query                                              sql
0        average of Good Health and Well-being in East ...   SELECT avg(good_health_and_well-being) FROM sp...
1           Find the Climate Action of Burkina Faso before...    SELECT climate_action FROM spi_index_labelled_...
2         Countries having Data Service Score between 22...    SELECT country FROM spi_index_labelled_vFinal ...
3              Download Options Score of Guinea in 2007   SELECT download_options_score FROM spi_index_l...
4                               Top 70 countries by iso3c    SELECT country FROM spi_index_labelled_vFinal ...
...                                                    ...                                              ...
9995      Data Sources Score and Non_Proprietary format ...     SELECT data_sources_score , non_proprietary_fo...
9996   Find the Countries having ODIN Open Data Openn...    SELECT country FROM spi_index_labelled_vFinal ...
9997      Countries having Data Service Score between 27...    SELECT country FROM spi_index_labelled_vFinal ...
9998       Countries having Machine Readability Score bet...    SELECT country FROM spi_index_labelled_vFinal ...
9999       Good Health and Well-being of South Sudan in 2...   SELECT good_health_and_well-being FROM spi_ind...
10000 rows × 2 columns
```

Figure 5: Examples of generated query pairs for natural language and SQL.

2.2.3 Experimental results

As mentioned earlier, we will not be using the WikiSQL dataset to train/evaluate our model. As such, we will not be using the T5-base-finetuned-wikiSQL model [6] as the data we can generate is far lesser than WikiSQL and finetuning that model on our dataset will not update the weights by much.

Instead, we decided to use the T5-small model [7] to train and evaluate on with our generated query pair dataset. We chose T5 as it was inherently designed to handle text to text tasks such as machine translation due to its nature of converting all text-based language problems into a text-to-text format. We used the small variant of the model as we are constrained by the limited size of the training pairs we generated, and do not wish to use a more powerful variant that will be more prone to overfitting given the data size.

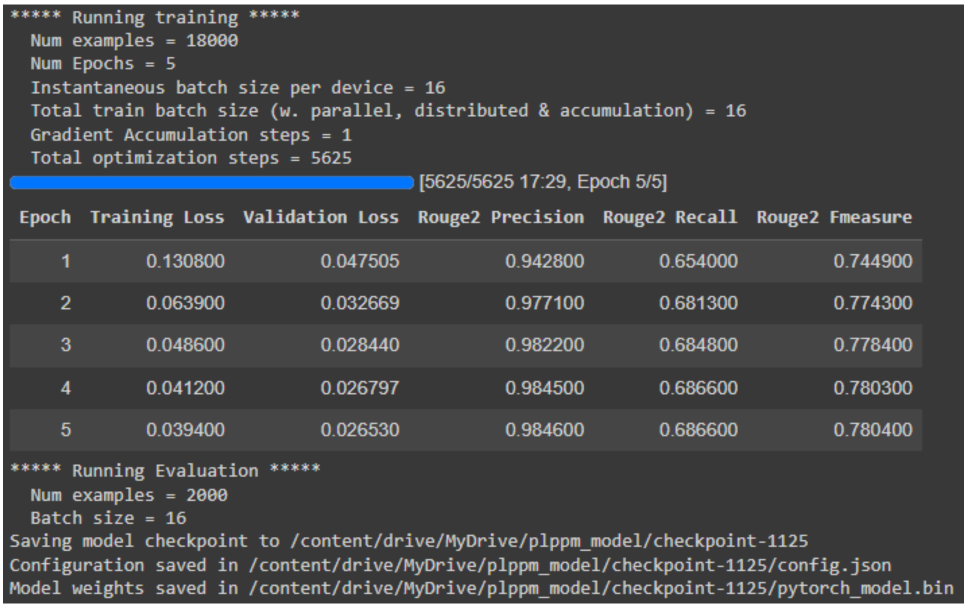


Model size variants

Model	Parameters	# layers	d_{model}	d_{ff}	d_{kv}	# heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

Figure 6: Model size variants for T5 model [7].

We utilized the Huggingface transformers library, specifically the Seq2Seq group of functions to aid with the training of the model using our generated dataset. The settings we use were a train-test split of 0.9-0.1 and 5 training epochs of batch size 16. The results of model training and evaluation are as shown below:



```
***** Running training *****
  Num examples = 18000
  Num Epochs = 5
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 5625
```

[5625/5625 17:29, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Rouge2 Precision	Rouge2 Recall	Rouge2 Fmeasure
1	0.130800	0.047505	0.942800	0.654000	0.744900
2	0.063900	0.032669	0.977100	0.681300	0.774300
3	0.048600	0.028440	0.982200	0.684800	0.778400
4	0.041200	0.026797	0.984500	0.686600	0.780300
5	0.039400	0.026530	0.984600	0.686600	0.780400

```
***** Running Evaluation *****
  Num examples = 2000
  Batch size = 16
Saving model checkpoint to /content/drive/MyDrive/plppm_model/checkpoint-1125
Configuration saved in /content/drive/MyDrive/plppm_model/checkpoint-1125/config.json
Model weights saved in /content/drive/MyDrive/plppm_model/checkpoint-1125/pytorch_model.bin
```

Figure 7: Model training and evaluation for T5-small on the generated dataset.

We have opted to use Rogue [8] (Recall-Oriented Understudy for Gisting Evaluation) as the set of evaluation metrics for our model training. This metric is based on calculating the synaptic overlap between candidate and reference text pieces. In our case we opted to use Rogue2 where the match rate of bigrams between the model output and reference are measured as the SQL queries are naturally short, and we believe bigram matching would suffice.

$$\frac{\text{number of n-grams found in model and reference}}{\text{number of n-grams in reference}}$$

$$\text{Precision} = \frac{\text{count}_{\text{match}}(\text{gram}_n)}{\text{count}(\text{gram}_n)} \qquad \text{Recall} = \frac{\text{number of n-grams found in model and reference}}{\text{number of n-grams in } \textbf{model}}$$

$$\text{F1 score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Mainly the Rogue2 F1 score is used to evaluate the model as it gives us a good measure of model performance that does not rely only on capturing as many words as possible (recall) but also without outputting irrelevant ones (precision). As can be seen in the previous figure, after 5 epochs the model has achieved a Rogue2 F1 score of 78 which is quite good. We would not want to train for further epochs as we can see after epoch 2 the F1 score has started to plateau, and we would not want the model to overfit.

2.3 Natural language to GQL

2.3.1 Overview of GQL

For the intents which are classified as queries to the graph database, it will be parse to a backend module to convert the natural language questions to the relevant graph query language (GQL). This allows business users to make queries and retrieve information from the graph database, without having knowledge on GQL statements. For this project, Neo4j is selected as the graph database and it has its corresponding GQL which is the Cypher query language [9]. In general, the Cypher query language has the format of

```
MATCH (n1:NODE) - [r:relation] → (n2:NODE)
```

where n1 and n2 are the nodes, and r is the relation link between the nodes.

In order to make a query to the Neo4j graph DB to extract specific information, the required nodes and relations have to be provided in the Cypher query. For example as seen in Figure 8 below, if the business user was to enquire for When was Facebook founded?, there needs to be an extraction of the Node1:Facebook and Relation: Inception, and the remaining desired information Node2: 2004 can be obtained from the graph DB.

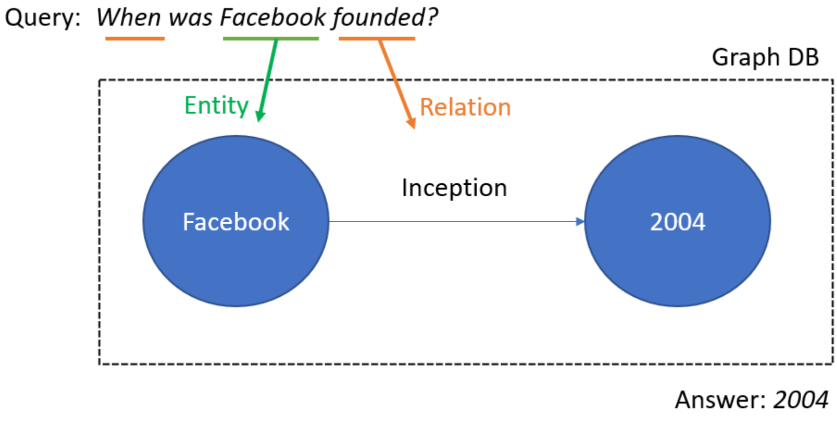


Figure 8: Illustration of information extraction from query statement to corresponding graph DB structure.

The task of the text2GQL module in this project is to convert the natural language query to the corresponding GQL statements, by extracting the relevant information from the user's query. Entity extraction for Node1 may be performed using Named-Entity Recognition (NER) methods, however the Relation to be extracted remains a challenge, which will be addressed in the proposed text2GQL solution.

2.3.2 Dataset for Graph DB

The dataset used for building the Neo4j graph DB is the US Financial Articles dataset [10]. It contains over 300,000 financial articles from January to May 2018, extracted from various news publishers such as CNBC, Bloomberg, Reuters, WSJ, and Fortune, and stored in JSON format. Information extraction was performed on a subset of 10,000 articles, to retrieve data on the financial domain, which includes information on companies, persons, and figures, and the relations between them. As forementioned on the requirements to store data as nodes and links in a graph DB, data in this format must be similarly extracted from the financial articles. This was performed using an autoregressive entity-relation extractor known as REBEL: Relation Extraction By End-to-end Language generation [11]. REBEL (2021) is a Seq2Seq model that provides state-of-the-art method for relation extraction of an input sentence. It utilised BART-large as the base model, which was further fine-tuned on a curated dataset of Wikipedia abstracts and Wikidata entities, and filtered by a Natural Language Inference model (RoBERTa) to reduce noisy annotations. It has achieved leading results for both Relation Extraction (RE) benchmarking datasets (CONLL04, NYT, DocRED & ADE) and Relation Classification (RC) dataset (Re-TACRED). The information extraction and graph DB storage procedures are shown in Figure 9. When a sentence is selected from the financial article, information on the relation, head span and tail span are extracted and represented as a triplet. Using the information extracted from REBEL, it can be stored in the Neo4j graph DB as nodes and links.

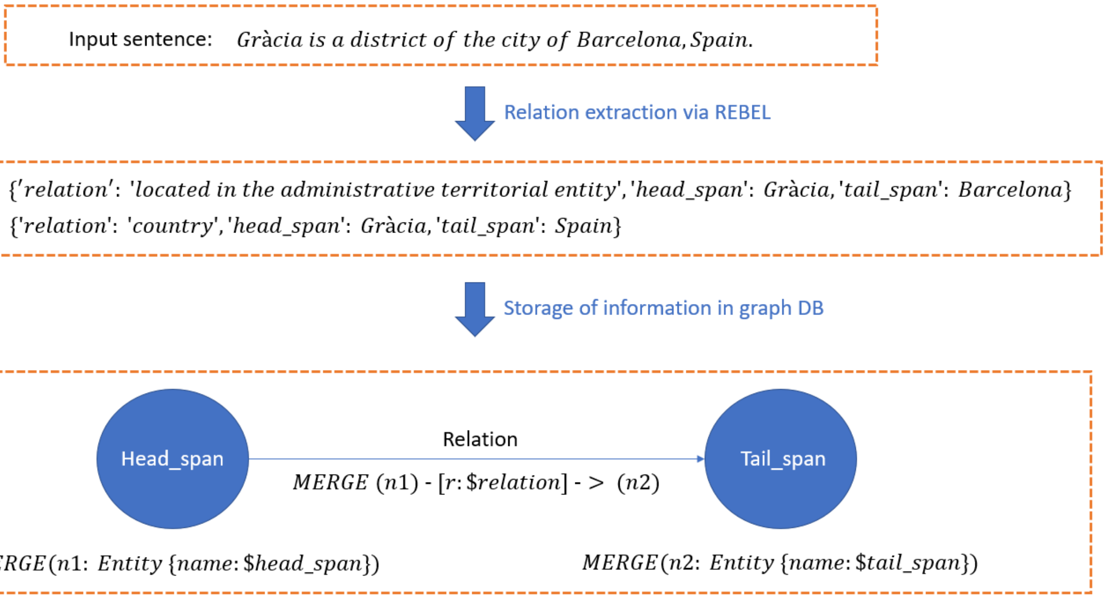

Input sentence: Gràcia is a district of the city of Barcelona, Spain.
Relation extraction via REBEL
(0, 8): {'relation': 'located in the administrative territorial entity', 'head_span': Gràcia, 'tail_span': Barcelona}
(0, 10): {'relation': 'country', 'head_span': Gràcia, 'tail_span': Spain}
Storage of information in graph DB
Head_span
Relation
MERGE (n1) - [r: $relation] - > (n2)
Tail_span
MERGE(n1: Entity {name: $head_span})
MERGE(n2: Entity {name: $tail_span})

Figure 9: Information extraction via REBEL and graph DB storage procedures.

2.3.3 Text2GQL Methodology

The proposed solution of text2GQL in this project, is framed as a task of extracting Node and Relation for a GQL statement. Automated entity-relation extractors such as REBEL may not be suitable as the user query statement does not contain both the headspan and tailspan entities. Rather, the user query only has information on one entity, and the relation has to be inferred from the remaining structure of the query. As of development of this project, there is no well-established dataset containing pairs of natural language to GQL statements available, which may be used to train a model for machine translation. Therefore, the approach in this project is to build a self-supervised model from the US Financial Article dataset, to generate queries that can be further utilised for training of a relation extractor, which is summarised in Figure 10.

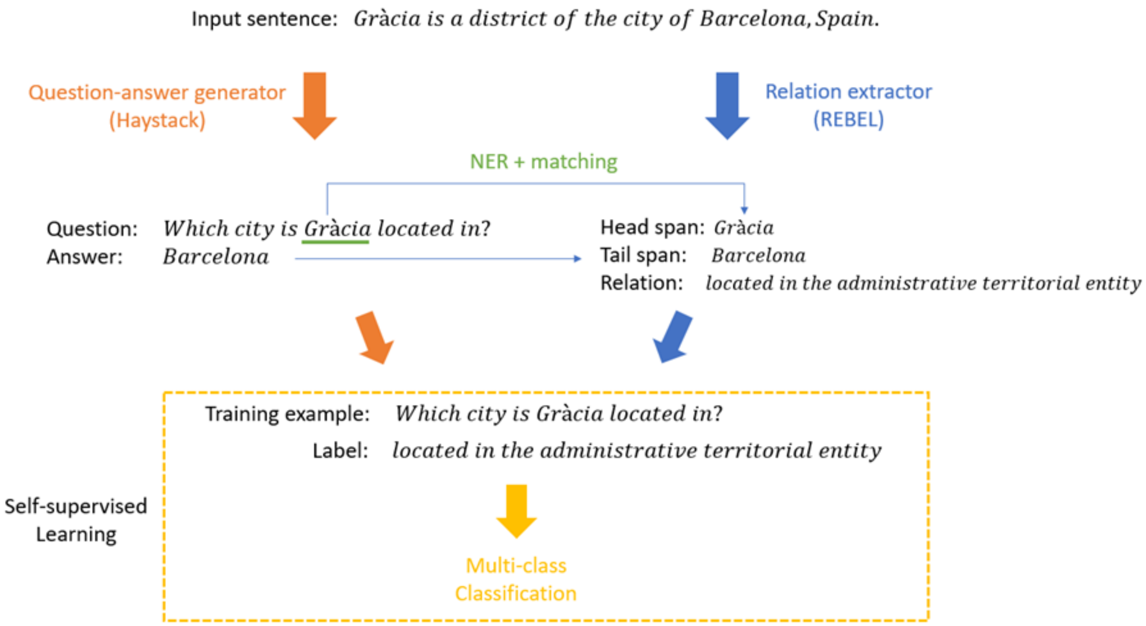

Input sentence: *Gràcia is a district of the city of Barcelona, Spain.*
Question-answer generator (Haystack)
Relation extractor (REBEL)
NER + matching
Question: *Which city is Gràcia located in?*
Answer: *Barcelona*
Head span: *Gràcia*
Tail span: *Barcelona*
Relation: *located in the administrative territorial entity*
Training example: *Which city is Gràcia located in?*
Label: *located in the administrative territorial entity*
Self-supervised Learning
Multi-class Classification

Figure 10: Workflow for generation of self-supervised training examples for classification model training.

First, a question-answer generator is used as an automated method to craft possible queries from a given statement. Haystack is an end-to-end framework to utilise state-of-the-art NLP models for tasks such as question answering, summarization, document search and question generation [12]. For the question-answer generation task, a question generator (T5-base) is initially used to generate a list of questions from a text statement. It will subsequently run a reader model (RoBERTa-base) which was fined-tuned on answering task SQuAD2.0, to generate the likely answers to the questions. As seen in Figure ***10***, for the statement '*Gràcia is a district of the city of Barcelona, Spain*', a possible query '*Which city is Gràcia located in*?' may be generated, along with its corresponding answer value '*Barcelona*'.

From the question-answer pair, two entities may be obtained, one from the question which was processed via named-entity recognition, and another from the answer value. Following the example given, this would correspond to '*Gràcia*' and '*Barcelona*'. The two entities were subsequently matched against the entity-relation triplet extracted by REBEL. If the entities extracted by Haystack and REBEL had the same values, then the REBEL-generated relation will be used as the label to the Haystack-generated question. The assumption made is that only one possible relationship between the two entities exists within the same statement. Upon manual verification of a sample of question-relation pairs, this assumption holds true for most cases, unless the statement is lengthy and has several co-references in it. As such, this method was used to generate a large number of question-relation pairs in an automated manner, and further manual filtering of some pairs was conducted to improve the matching accuracy.

Next, the question-relation pairs were sorted and counted to obtain the most common occurrences of relations. Selected top occurrences of relations were chosen in order to provide a more concise scope for multi-class classification and to ensure that each class had adequate number of training examples. As seen in Figure 11, 6 main relations were selected, which are 'founded by', 'inception', 'parent organization', 'employer', 'headquarters location' and 'located in the administrative territorial entity'. An additional class 'Others' was formed, which comprises of training examples with the following relations: 'diplomatic relation', 'member of political party', 'member of sports team', 'head of government', 'author', 'manufacturer', 'director', 'editor' and 'capital'. These relations were selected to represent a spectrum of possible different question types. The inclusion of the class 'Others' was to handle question types from users that were not in the scope of this system, instead of assigning it to one of the preselected 6 relations. Hence, in total there were 7 classes and 882 training examples for the multi-class classification task. Each class had at least 50 training examples and the data was further split to 80% and 20% for training and test set respectively.

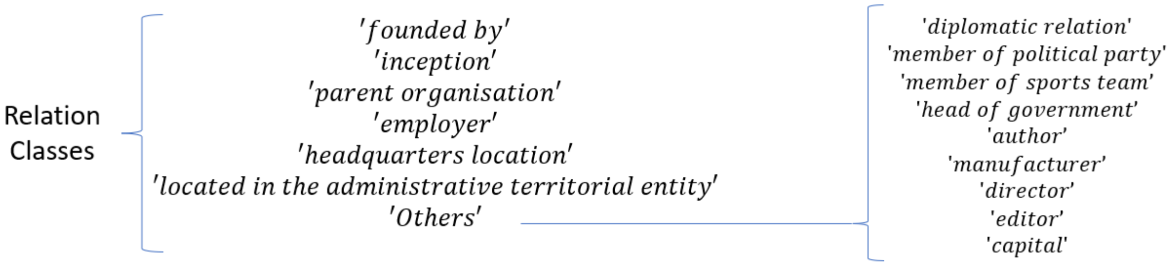
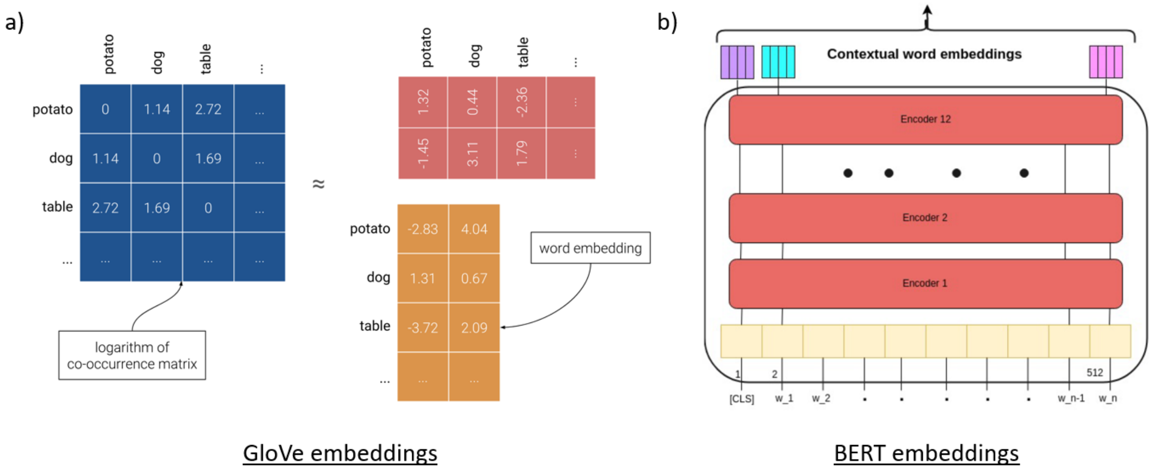


Figure 11: Graph relation classes selected for classification.

For the classification task, the questions were first pre-processed by extracting features via word embeddings. Word embeddings are pre-trained matrices from large text corpus, that capture the semantic meaning of the word, and represented by numerical vectors of multiple dimensions. Two types of word embeddings were explored, GloVe (Global Vectors) and BERT (Bidirectional Encoder Representations from Transformers). GloVe uses global word-word co-occurrence statistics of a corpus, to generate word representations in vector space [13]. BERT utilises the transformer architecture pre-trained on Masked Language Modeling and Next Sentence Prediction tasks, and feature embeddings can be extracted from its hidden states [14]. By representing the questions as feature vectors, the semantic and contextual meaning of the words may be captured as input for the classification model.



GloVe embeddings

BERT embeddings

Figure 12: Model illustrations for a) GloVe [15] and b) BERT embeddings [16].

Table 2: Model parameters for classification of graph relations.

Model	Parameter	Value
GNB	-	-
SVM (classifier)	Kernel Degree C	Polynomial 3 1
k-NN	Neighbours	5
RF	No. of estimators	100

CNN-base	Filters	512
	Kernel size	3
	Activation	ReLU
	Pooling	Max
CNN-dp	Filters	512
	Kernel size	3
	Activation	ReLU
	Pooling	Max
	Dense units	64
	Dropout	0.2
CNN-deep	Filters	128
	Kernel size	2,3,5
	Activation	ReLU
	Pooling	Max
	Dense units	64
	Dropout	0.2
BiLSTM	LSTM units	64
	Dropout	0.2

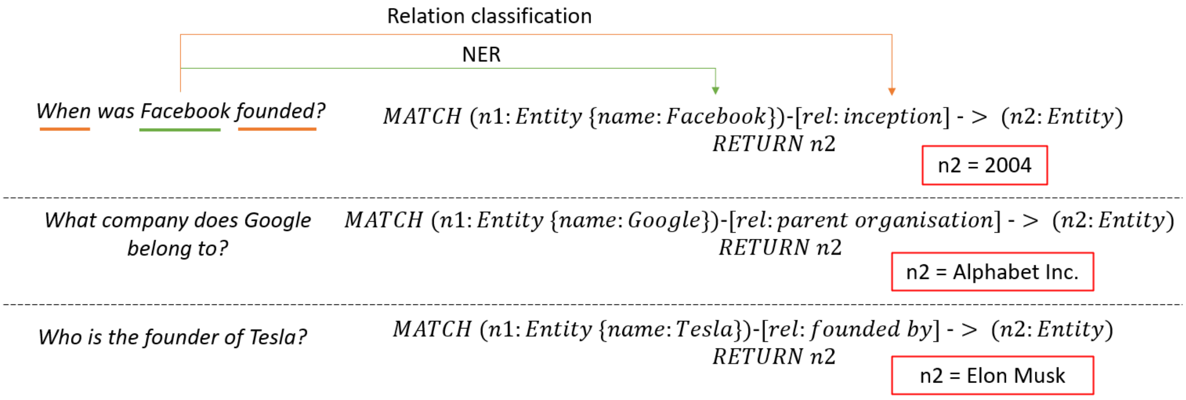
The classification models that were explored includes machine learning based (GNB, SVM, k-NN, RF) and deep-learning based (CNN, BiLSTM) methods. Gaussian Naïve-bayes (GNB) is a probabilistic classifier based on Bayes' Theorem that assumes features are independent of each other. Support vector machine (SVM) is a supervised method which utilises kernels to linearly separate classes of data and constructs a hyperplane that maximises the margin to data points. k-Nearest Neighbours (k-NN) is a memory-based method which assigns the classes based on the k number of data points of closest proximity in vector space. Random forest (RF) is an ensemble learning method that bootstraps multiple decision tree models to reduce variance of the aggregated model. For the machine learning based methods, the sentence vector was obtained by calculating the average of all the word vectors within the sentence. Additionally, to model the inter-dependency of words within the sentence, deep-learning based methods were also explored to further extract features of word sequences. Convolutional Neural Networks (CNN) is a deep learning algorithm that uses filter kernels to convolute with input vectors to extract latent features, which are further piped into dense classifier layers. A total of three variations of CNN models are use: CNN-base, CNN-dp and CNN-deep. CNN-dp includes an additional dense layer with dropout, while CNN-deep consists of three CNN channels with different kernel size. Bidirectional Long Short Term Memory (BiLSTM) is a sequential model which comprises of both forward and backward LSTM computations of a sentence, to model the dependency of data in both directions. The model parameters for the machine-learning and deep-learning based methods are listed in Table 2. The evaluation metrics used for the multi-class classification task are accuracy, F1-score, precision, and recall.

Table 3: Model testing results for classification of graph relations.

Model	**GloVe embeddings**				**BERT embeddings**			
	Accuracy	F1-score	Precision	Recall	Accuracy	F1-score	Precision	Recall
GNB	0.7345	0.7271	0.7424	0.7345	0.7062	0.7089	0.7442	0.7062
SVM	0.6667	0.6619	0.7435	0.6667	0.7966	0.7843	0.8150	0.7966
k-NN	0.8531	0.8476	0.8537	0.8531	0.9040	0.9048	0.9145	0.9040
RF	0.9153	0.9161	0.9226	0.9153	0.9153	0.9188	0.9291	0.9153
CNN-base	0.9266	0.9261	0.9284	0.9266	0.9548	0.9552	0.9566	0.9548
CNN-dp	0.9266	0.9254	0.9340	0.9266	**0.9661**	**0.9665**	**0.9686**	**0.9661**
CNN-deep	0.8814	0.8802	0.8913	0.8814	0.9266	0.9275	0.9304	0.9266
BiLSTM	0.9379	0.9371	0.9384	0.9379	0.9435	0.9439	0.9458	0.9435

The results for the classification models are shown in Table 3. When GloVe embeddings was used as feature vectors, the deep-learning based models (CNN, BiLSTM) generally performed better as compared to the machine learning based models, where the highest accuracy (0.94) achieved was for the BiLSTM model. Similar observation was also seen when BERT embeddings were used, where deep learning models outperformed machine learning models. This could signify the presence of inter-dependency between words in a sentence structure that could aid in determining the predicted classes. When comparing between the embedding methods, utilising BERT embeddings led to better classification performance (0.97 accuracy for CNN-dp model), which signifies that it was able represent word semantics more accurately over GloVe embeddings. This could be attributed to the larger feature space, where BERT embeddings had represented each word over 768 feature dimensions as compared to 100 feature dimensions in GloVe embeddings. For CNN models, it was observed that deep CNN models did not yield better results as compared to shallow CNN models, which could indicate that the classification problem did not require deep complex models. On the other hand, the inclusion of drop-out in the CNN model led to improved results due to reduction of overfitting to the training data.

For inference and deployment to the chatbot, BERT embeddings with CNN-dp classifier was selected, as it resulted in the best performance for both accuracy and F1-score metrics. For a new question that is provided by the user, it will be converted to feature vectors via BERT embeddings, and subsequently predicted for its relation class. Along with the entity extracted using named-entity recognition, the corresponding graph query may be established, as seen from the example in Figure 13 below.


Relation classification
NER
When was Facebook founded?
n2 = 2004
What company does Google belong to?
n2 = Alphabet Inc.
Who is the founder of Tesla?
n2 = Elon Musk

$MATCH\ (n1: Entity\ \{name: Facebook\})\text{-}[rel: inception]\text{ - > }(n2: Entity)$
$RETURN\ n2$

$MATCH\ (n1: Entity\ \{name: Google\})\text{-}[rel: parent\ organisation]\text{ - > }(n2: Entity)$
$RETURN\ n2$

$MATCH\ (n1: Entity\ \{name: Tesla\})\text{-}[rel: founded\ by]\text{ - > }(n2: Entity)$
$RETURN\ n2$

Figure 13: Conversion of natural language query to GQL statements using the proposed method in this solution.

The graph quey will be posted to the Neo4j graph DB to retrieve the value for Node2 (n2), and it will be returned to the module for answer generation.

2.4 Chatbot

A conversational user interface was designed to route the query to the appropriate query language conversion module such as SQL and GQL. To improve conversational experience with users and increase the query diversification, Investopedia financial terms and Chatterbot corpus conversation were integrated into the system. User may not only ask about the corpus exist only in SQL and GQL model, therefore, the added features of conversational & QA corpus retrieval enable the chatbot to serve as a financial knowledgeable expert to answer the queries with completely different intent to the language conversion module that may result as incorrect data reference or exception during the retrieval.

Django REST framework is selected as the backbone of the system to perform natural language understanding on the user query submitted. It does not only handle the user's intent and route the query to the appropriate module but also performs the mapping of the results to the user reply referenced from the template or rule defined. Refer to the following diagram on the detailed system architecture of the system:

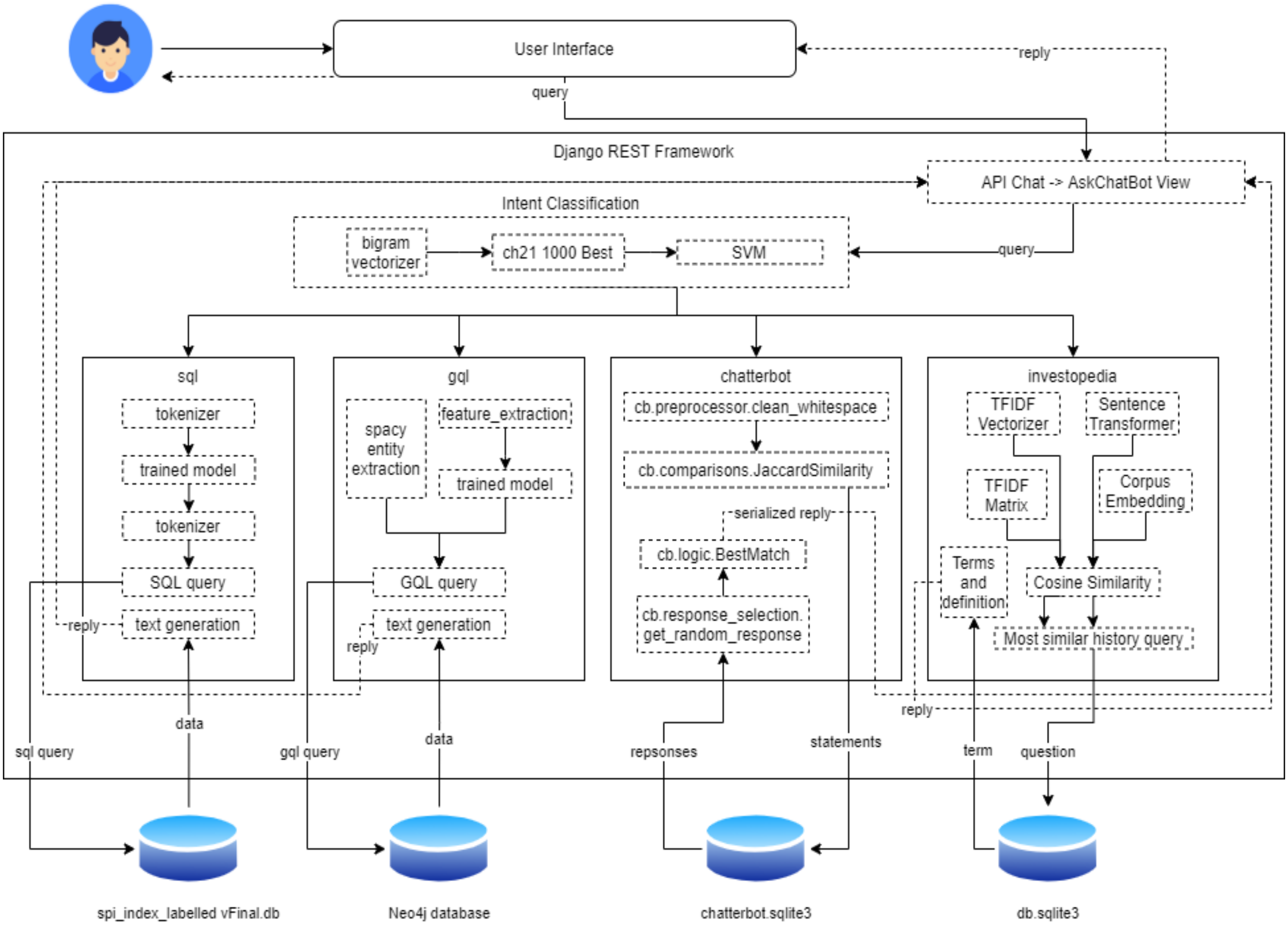


Figure 14: Detailed system architecture of our proposed solution.

A http API Chat is integrated in the REST framework to handle the query submitted through the user interface. It would execute the intent classification to select which language module such a SQL, GQL, Investopedia Terms or Chatterbot conversation to run. The text replied to users from the SQL, GQL and Investopedia Terms module are further mapped into the serialized reply to match with the chatterbot responses in the AskChatBot View. Moreover, the SQL and GQL queries generated from the SQL and GQL language module are also presented together with the generated reply to user in the conversational UI so it is more intuitive to user to understand about the data retrieved from the database.

2.4.1 Intent Classification

Intent classification is crucial in the system to perform natural language understanding at the very beginning stage. The intention of the query can be extracted through the method and selectively pick the correct action to generate the response to users. Due to the unrestricted query submission on the user interface, the system requires many unlikely intentions and exceptional handling to provide user with more flexible responses. Therefore, Investopedia terms and Chatterbot conversation are involved in the system to prevent solitary and frustrating response when there is no data retrieved from the SQL and GQL modules.

2.4.1.1 Data Preparation

The training data in SQL and GQL module were reused and only three thousand are randomly selected from each module. The simulated queries for the Investopedia terms were generated according to the Investopedia corpus, where the Terms and Definitions from Investopedia were extracted from a web scrapper [17]. There were 1082 terms found in the corpus and 3 out of 16 question tags were randomly selected to concatenate with each term and three thousand training data were further selected to perform intention classification.

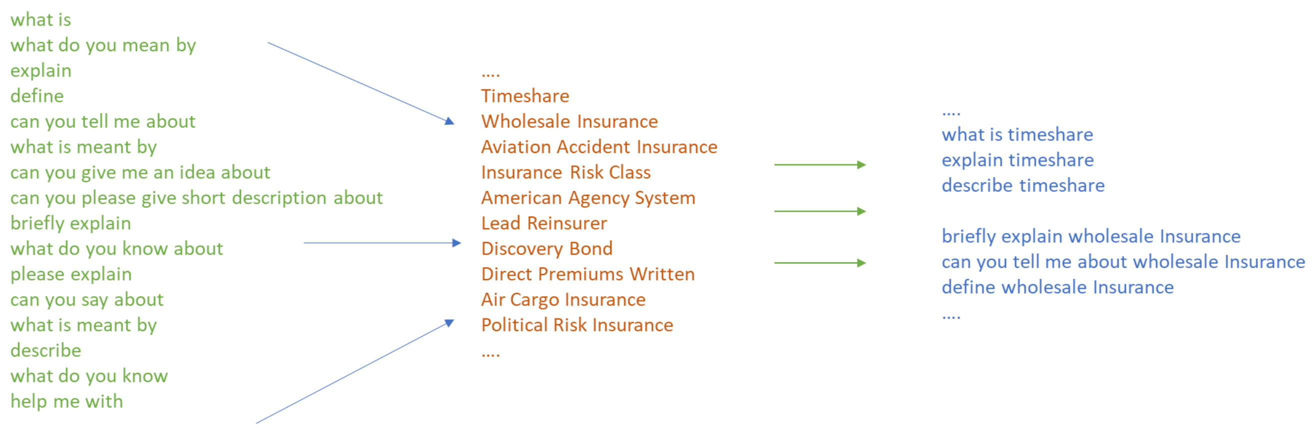
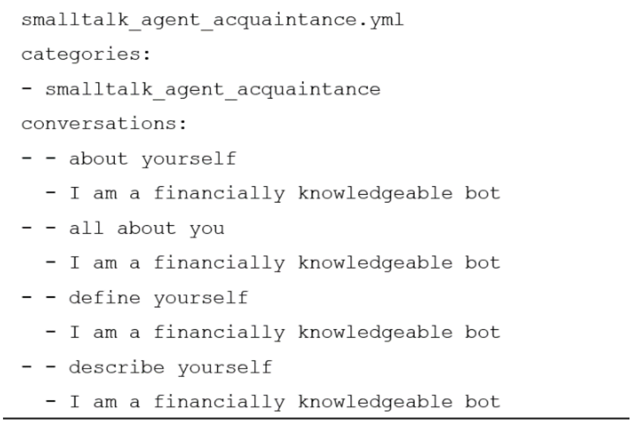


Figure 15: Sample training data for Investopedia Terms.

The chatterbot conversation was trained with yaml files which are referenced from the Kaggle dataset [18]. The utterances in the dataset were treated as the query submitted and only three thousand training data were randomly selected. There were inner intents within the dataset of the chatterbot conversation but the system did not further classify the sub intention as the methodology in the chatterbot API will take care of the conversation.



```
smalltalk_agent_acquaintance.yml
categories:
- smalltalk_agent_acquaintance
conversations:
- - about yourself
  - I am a financially knowledgeable bot
- - all about you
  - I am a financially knowledgeable bot
- - define yourself
  - I am a financially knowledgeable bot
- - describe yourself
  - I am a financially knowledgeable bot
```

Figure 16: Sample yaml file for the training of chatterbot.

Finally, the prepared dataset from four different intents was labelled accordingly and shuffled into train and test data to perform intent classification.

2.4.1.2 Methodology

From the stage of data pre-processing and preparation, we observed that the domain knowledge across the language modules were different so we assumed that the pattern of the queries generated for the training data should be distinct from different intention. To prove the hypothesis, we randomly picked three thousand training data from each intent to perform the training of the intent classification model to ensure less overfitting and give an insight on the how well the intent classification performed on unseen query.

As the model was required to integrate with the chatbot system, we chose machine learning method to perform intent classification to optimize the memory consumption and the speed of inference. Term Frequency – Inverse Document Frequency (TF-IDF) statistical method was selected as the feature extractor of the queries because the frequency of the word tokens in the question may differ from the intent and domain knowledge requested. Unigram and bigram were selected as the n-grams to be extracted across the data corpus and Chi-squared states of non-negative features were used as the feature selection for 1000, 2000, 3000 highest scores, which were selected to evaluate the performance of models with different hyperparameters.

Table 4: Evaluation results of SVM and logistic regression model for intent classification.

	Macro- precision	Macro F1-score
Logistic Regression (CH21 1000 features)	0.99	0.99
SVM (C=1, CH21 1000 features)	0.06	0.1
SVM (C=50, CH21 1000 features)	0.95	0.95
SVM (C=100, CH21 1000 features)	0.97	0.97
SVM (C=500, CH21 1000 features)	**0.99**	**0.99**
SVM (C=5000, CH21 1000 features)	0.99	0.99
Logistic Regression (CH21 2000 features)	0.99	0.99
SVM (C=1, CH21 2000 features)	0.06	0.1
SVM (C=50, CH21 2000 features)	0.97	0.96
SVM (C=100, CH21 2000 features)	0.97	0.97
SVM (C=500, CH21 2000 features)	0.99	0.99
SVM (C=5000, CH21 2000 features)	1.00	1.00
Logistic Regression (CH21 3000 features)	0.99	0.99
SVM (C=1, CH21 3000 features)	0.06	0.1
SVM (C=50, CH21 3000 features)	0.95	0.95
SVM (C=100, CH21 3000 features)	0.97	0.97
SVM (C=500, CH21 3000 features)	0.99	0.99
SVM (C=5000, CH21 3000 features)	1.00	1.00

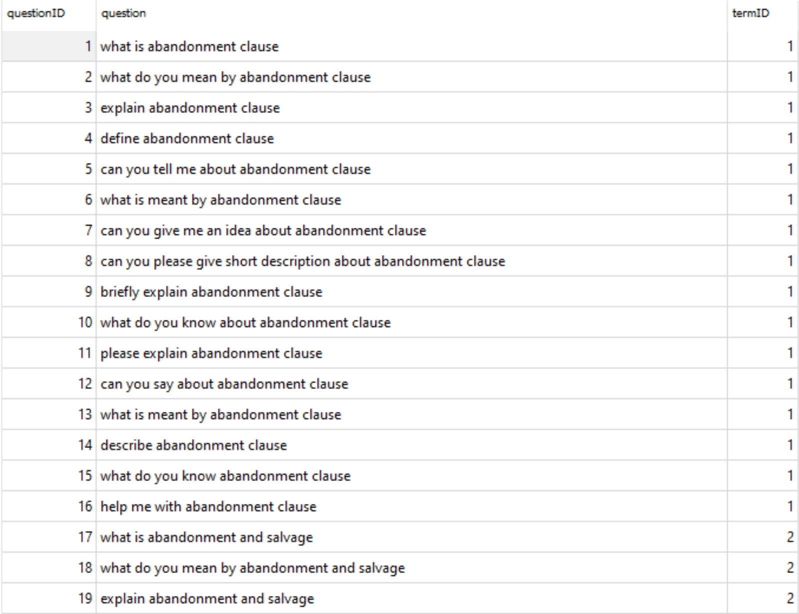
We picked Logistic Regression model with solver 'lbfgs' and Support Vector Machine with 'rbf' kernel. Due to positive performance of the machine learning model on the intent classification, we did not proceed to perform classification with Deep Learning or Transformers model. The hyperparameters of the CH21 feature selector and SVM models were fine-tuned to improve the performance and it was observed that using 1000 bigram features and SVM model with regularization parameter of C=500, produced satisfactory results.

2.4.2 Investopedia Terms Similarity

After the intent classification was executed, the query was parsed to the appropriate language module to retrieve the requested result. Question-answer retrieval of Investopedia Terms corpus is one of the language modules and the terms and definitions were stored in the Django database db.sqlite3. This module selects the most similar query in the Investopedia Terms and Questions table to get the terms ID, and the definition of the terms can be retrieved from the Investopedia Terms and Definitions table as the reply to the user.

2.4.2.1 Data Preparation

The 16 question tags were not only used to generate the training data for the intent classification. It was also concatenated with the 1082 terms found in the Investopedia corpus to form 17312 questions about the terms. The generated questions along with the respective terms mapped were stored in the Investopedia Terms and Questions table to allow the language model to find the most similar question to the text submitted by user in the user interface.



questionID	question	termID
1	what is abandonment clause	1
2	what do you mean by abandonment clause	1
3	explain abandonment clause	1
4	define abandonment clause	1
5	can you tell me about abandonment clause	1
6	what is meant by abandonment clause	1
7	can you give me an idea about abandonment clause	1
8	can you please give short description about abandonment clause	1
9	briefly explain abandonment clause	1
10	what do you know about abandonment clause	1
11	please explain abandonment clause	1
12	can you say about abandonment clause	1
13	what is meant by abandonment clause	1
14	describe abandonment clause	1
15	what do you know abandonment clause	1
16	help me with abandonment clause	1
17	what is abandonment and salvage	2
18	what do you mean by abandonment and salvage	2
19	explain abandonment and salvage	2

Figure 17: Terms and Question table for Investopedia knowledge.

2.4.2.2 Methodology

We applied unsupervised models for the question retrieval to find the most similar question from the Terms and Question table through cosine similarity. The TF-IDF methodology was reused in the system to create word vectorization matrix while Sentence BERT Transformers [19] with pretrained model "msmarco-distilbert-base-v4" was utilised to convert the question bank into a corpus embedding file.

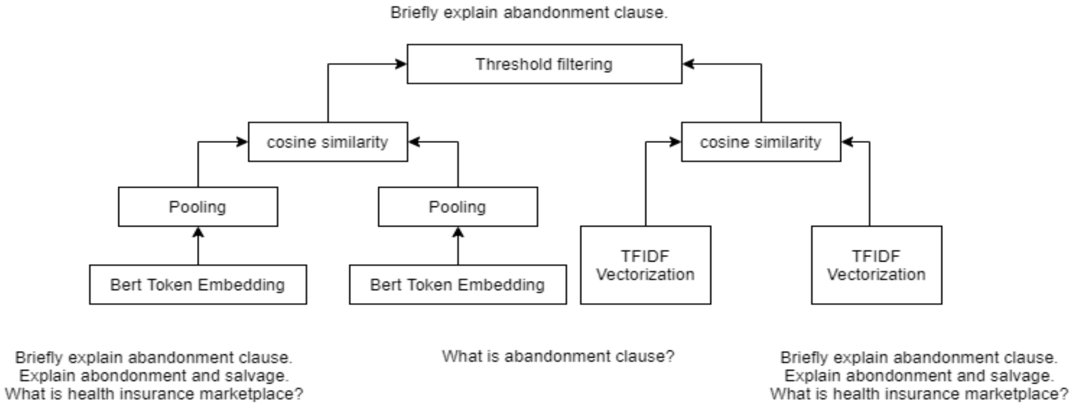

Briefly explain abandonment clause.
Threshold filtering
cosine similarity
cosine similarity
Pooling
Pooling
Bert Token Embedding
Bert Token Embedding
TFIDF Vectorization
TFIDF Vectorization
Briefly explain abandonment clause.
Explain abondonment and salvage.
What is health insurance marketplace?
What is abandonment clause?
Briefly explain abandonment clause.
Explain abondonment and salvage.
What is health insurance marketplace?

Figure 18: Sentence to vector for question similarity comparison.

After the BERT token embedding and TF-IDF matrix were generated, the cosine similarity of the corpus questions to the query submitted were calculated, weighted averaged, and filtered with threshold 0.92. Only the most similar corpus question will be obtained to retrieve the required response to the user.

If there is no TermID detected after the filtering, the system will reply as "Sorry I cannot find the information from the Investopedia Terms dataset."

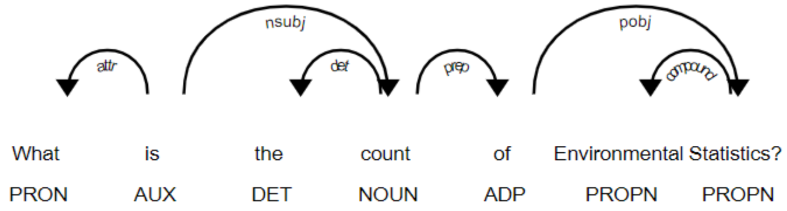
2.4.3 Natural Language Generation

After analysing the output from the SQL and GQL statements to the respective databases, we simplified the response generation through a rule-based approach. The SpaCy English pipeline "en_core_web_sm" [20] was selected as the core knowledge to extract the NER and POS tag of the queries to the SQL and GQL modules. The token labels of the query generated were analysed to learn about the pattern of the training data and to define the rules to handle the majority of writing styles. For the full detailed illustration and example of each rule, they are listed in Appendix A.

2.4.3.1 SQL

There are four rules defined to extract the contextual information from the question:

1. Removal of the interrogative word and token with tag "AUX". Extraction of the rest of the word token before the last punctuation word.
   E.g. "What is the count of Environment Statistics?" - > "the count of Environment Statistics"



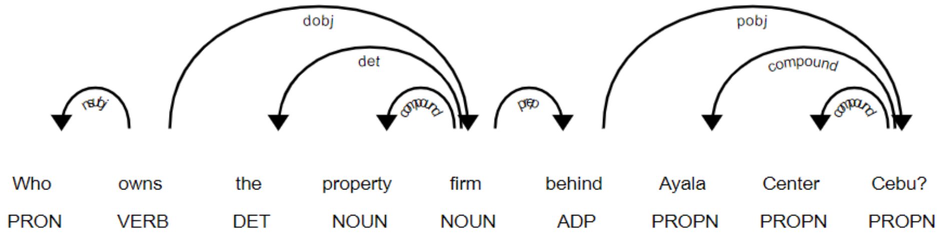
2. If word token with POS tag "VERB" starts from the query and it is a "ROOT" token, the extraction of the contextual sentence starts only from the "DET" word token before the last punctuation word. The rule is only executable after rule 1 is skipped.
3. Retrieval of whole sentence as the contextual sentence until the last token is punctuation when the sentence is started with word token with POS tag "NOUN". The rule is only executable after rule 2 is skipped.
4. Retrieval of whole sentence as the contextual sentence until the last token is punctuation when the sentence is started with the first Named Entity recognised by the English Pipeline. The rule is only executable after rule 3 is skipped.

After applying the rules, the contextual sentence was concatenated with the answer and reply to user (e.g. "the top 3 countries by gender equality: Norway, Italy Austria"). If none of the rules were triggered, the module will provide the reply directly as "Norway, Italy Austria", together with the SQL query generated.

2.4.3.2 GQL

Upon observation of the training data for the GQL module, majority of the queries contain interrogative word token in the beginning of the sentence. Therefore, the rules defined for the GQL module splits the queries to the type of the interrogative sentences:

1. Word Token "Who" scenario:
   a. Replace the word token "Who" with the answer if the interrogative word is followed by "AUX" or "VERB" word token.



b. Removal of the token "Who" and any AUX token after interrogative word. Placement of the answer right after the VERB token or ADP/ADV token.

2. Word Token “When” scenario:
    a. Removal of the token "When" and any AUX token after interrogative word.
    b. AUX tokens "is" "are" "was" "were" are migrated and placed before the first "VERB token".
    c. Placement of the answer at the end of the statement with word token “In” for all the scenarios with interrogative word “When”.

3. Word Token “Which” and “What” scenario:
    a. Ignore of the word token “NOUN” or “PRON” after the interrogative word.
    b. Replace the word token “What” or “Which” with the answer if the interrogative word is followed by “AUX” or “VERB” word token.
    c. Removal of the token "What" or “Which” and any AUX token after interrogative word. Placement of the answer right after the VERB token or ADP/ADV token.

4. Word Token “Where” scenario:
    a. Removal of the token " Where" and any AUX token after interrogative word.
    b. AUX tokens "is" "are" "was" "were" are migrated and placed before the first "VERB token".
    c. Placement of the answer at the end of the statement with word token “In” for all the scenarios with interrogative word “Where”.

If none of the interrogative word token was detected or the above listed scenarios cannot be met, the system will reply the answer directly together with the GQL query generated.

2.4.3.3 Exception handling

There were exceptional handlings built into the system following the classification of query intent. If the SQL or GQL language modules could not generate any meaningful queries, the response of the generated query will “null” and it will be highlighted that no answers could be retrieved from the SPI or US Financial News Articles datasets. Similarly, the system will also provide exceptions to the users, for cases when the modules managed to generate the SQL or GQL query, but the relevant information was not present in the database.

2.4.4 Chatterbot and User Interface

2.4.4.1 Chatterbot

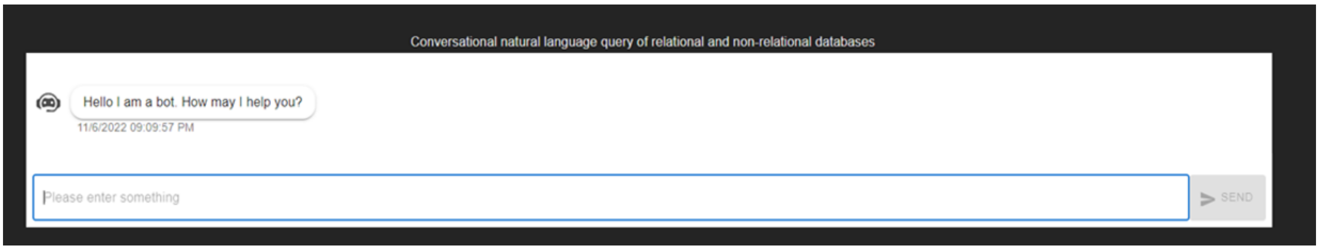
Chatterbot library is served as the communication baseline to handle casual conversations with users. As highlighted in 2.4.1.1, the casual conversations are retrieved from the Kaggle Dataset Small Talk Intent Classification Data [18]. We used the small response of different utterances under the same casual communication intention and trained the chatterbot according to the generated Yaml files.

Due to the obsolete libraries used in the Chatterbot framework, we utilised the entire Python library instead of the guided library installation because of the requirement to modify the code directly due to out-of-date algorithms from the SpaCy and Scikit-Learn. We maintained and ensured existing workable logic were integrated in the Chatterbot, such as JaccardSimilarity, BestMatch and get_recent_repeated_responses and get_random_response, so that the Chatterbot could serve simple casual conversations accurately.

Chatterbot was designed to apply Jaccard Similarity search to find the possible statements which had the same intention as the query submitted. The responses statements were retrieved along with the confidence value which represents the probability of a response being suitable to the input. If there were multiple suitable responses mapped with the input, the algorithm will randomly select a response and reply to the user.

2.4.4.2 User interface

A simple user interface was designed using the ReactJs TypeScript framework to create the system platform for users to submit their questions. The chatbot system will always start with a greeting message and request for questions from users:

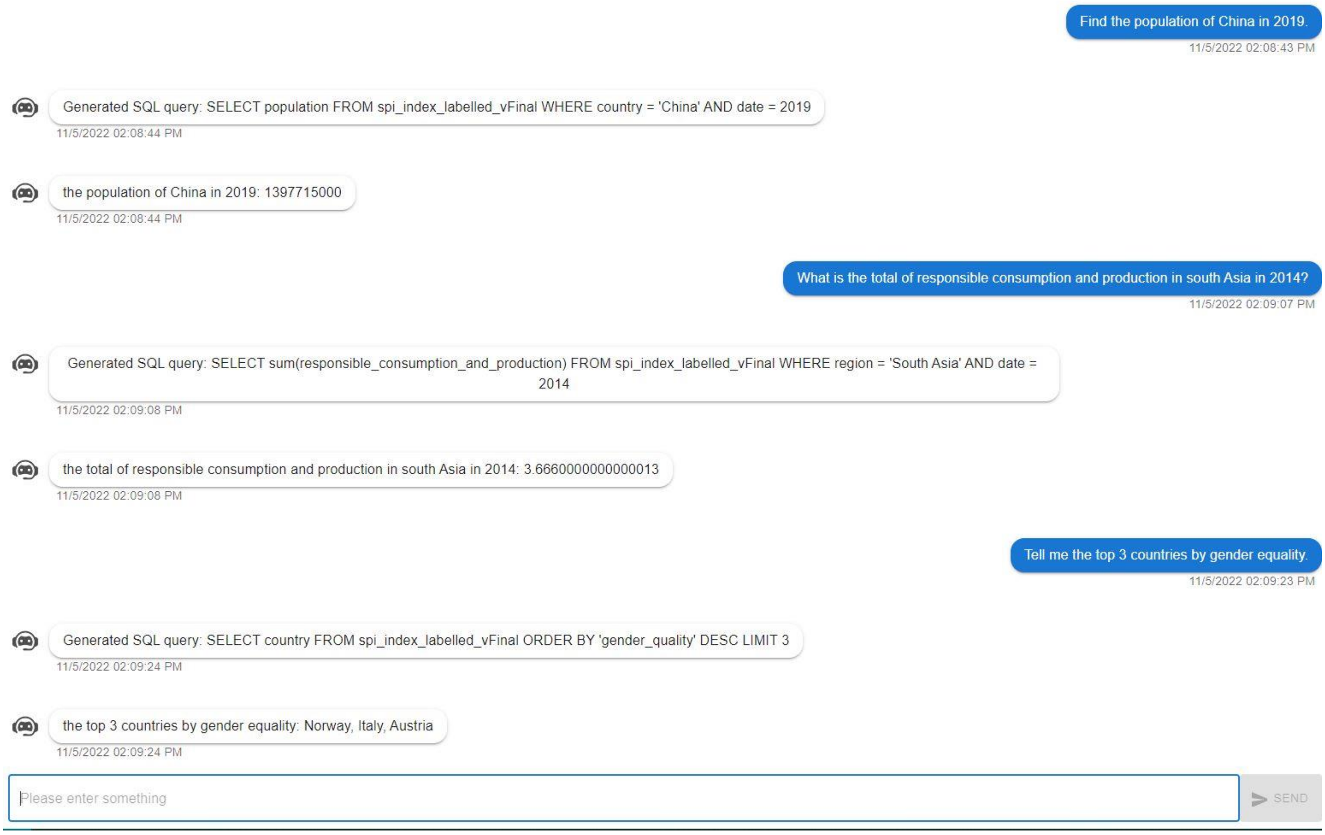


Conversational natural language query of relational and non-relational databases

Hello I am a bot. How may I help you?
11/6/2022 09:09:57 PM

Figure 19: User interface to obtain queries from users.

Users may send any query in the text field and can expect to receive the response in not more than 5 seconds. The date and time of both the response and user request are recorded within the user interface. Users can keep track of any historical chat submitted, provided the web browser is not closed. Demonstrations were conducted on the chatbot system and the responses of the chatbot were recorded. The following figures highlight the performance of the SQL and GQL language modules:



Find the population of China in 2019.
11/5/2022 02:08:43 PM

Generated SQL query: SELECT population FROM spi_index_labelled_vFinal WHERE country = 'China' AND date = 2019
11/5/2022 02:08:44 PM

the population of China in 2019: 1397715000
11/5/2022 02:08:44 PM

What is the total of responsible consumption and production in south Asia in 2014?
11/5/2022 02:09:07 PM

Generated SQL query: SELECT sum(responsible_consumption_and_production) FROM spi_index_labelled_vFinal WHERE region = 'South Asia' AND date = 2014
11/5/2022 02:09:08 PM

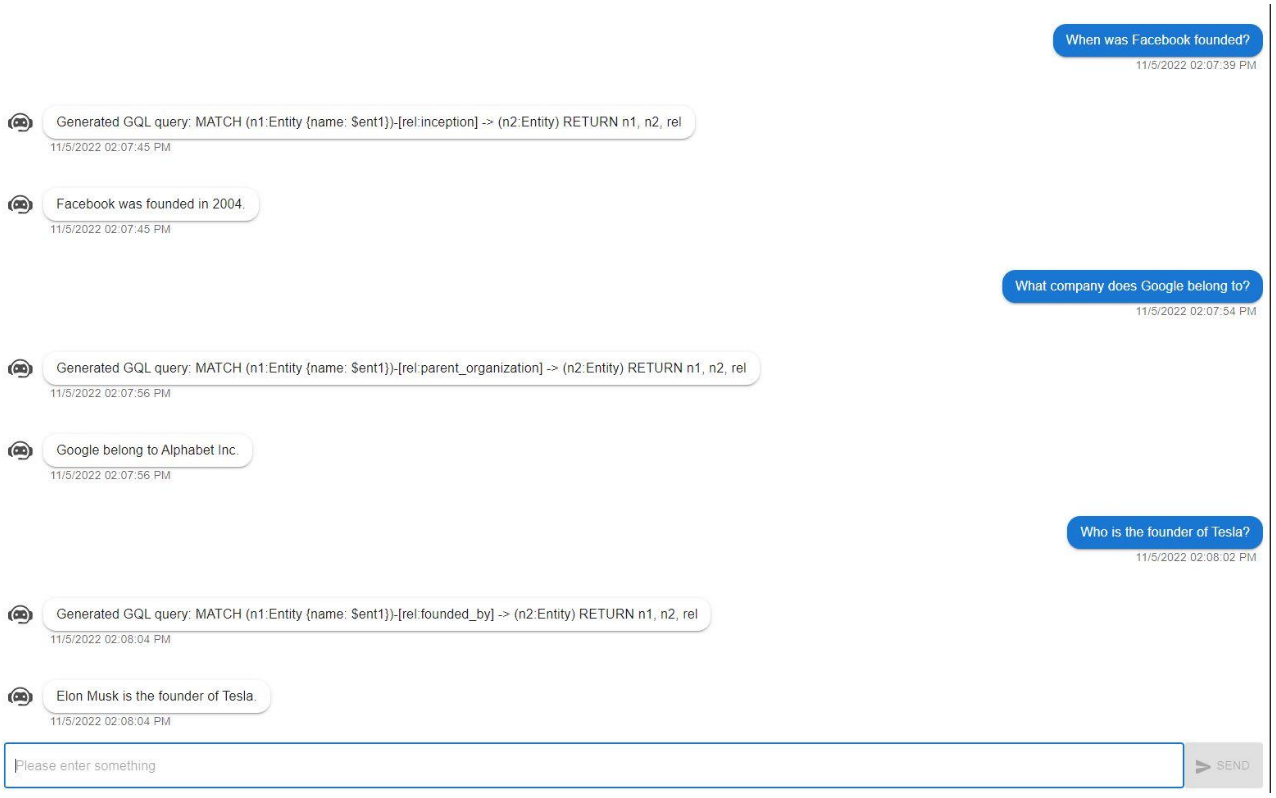
the total of responsible consumption and production in south Asia in 2014: 3.6660000000000013
11/5/2022 02:09:08 PM

Tell me the top 3 countries by gender equality.
11/5/2022 02:09:23 PM

Generated SQL query: SELECT country FROM spi_index_labelled_vFinal ORDER BY 'gender_quality' DESC LIMIT 3
11/5/2022 02:09:24 PM

the top 3 countries by gender equality: Norway, Italy, Austria
11/5/2022 02:09:24 PM

Figure 20: Responses of natural language query to relational database using SQL.



When was Facebook founded?
11/5/2022 02:07:39 PM

Generated GQL query: MATCH (n1:Entity {name: $ent1})-[rel:inception] -> (n2:Entity) RETURN n1, n2, rel
11/5/2022 02:07:45 PM

Facebook was founded in 2004.
11/5/2022 02:07:45 PM

What company does Google belong to?
11/5/2022 02:07:54 PM

Generated GQL query: MATCH (n1:Entity {name: $ent1})-[rel:parent_organization] -> (n2:Entity) RETURN n1, n2, rel
11/5/2022 02:07:56 PM

Google belong to Alphabet Inc.
11/5/2022 02:07:56 PM

Who is the founder of Tesla?
11/5/2022 02:08:02 PM

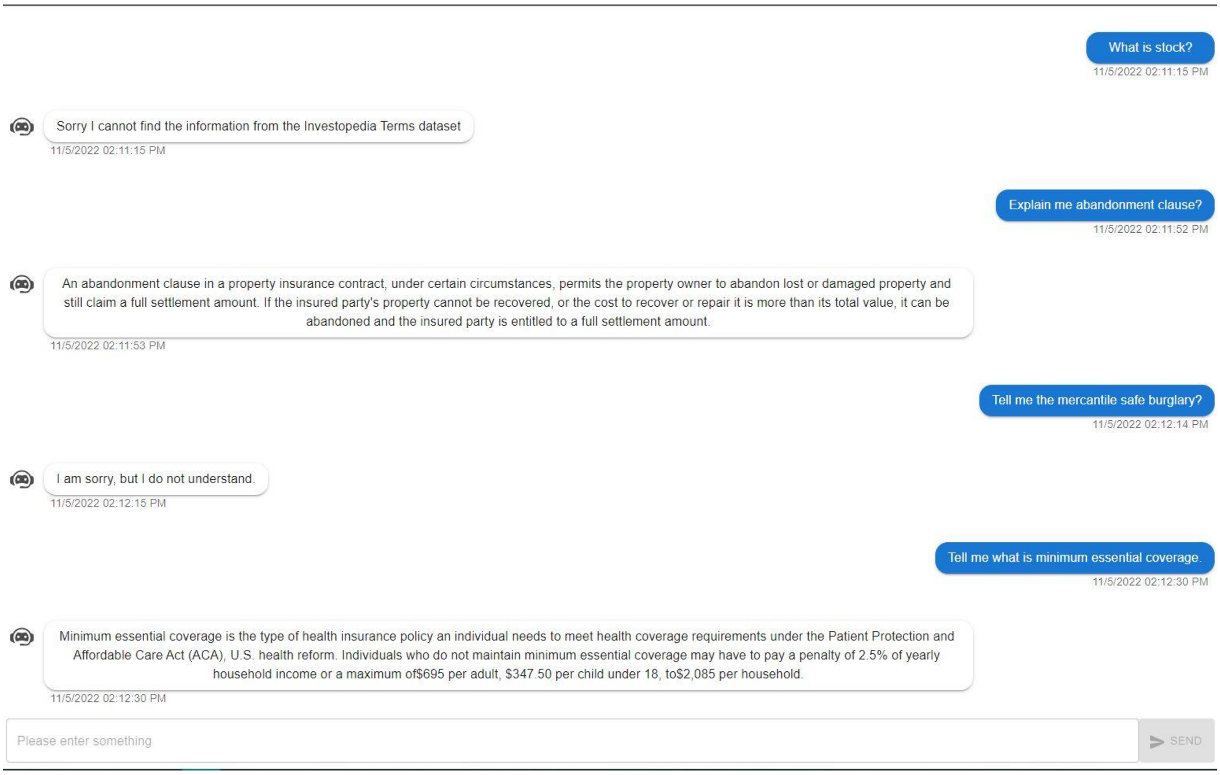
Generated GQL query: MATCH (n1:Entity {name: $ent1})-[rel:founded_by] -> (n2:Entity) RETURN n1, n2, rel
11/5/2022 02:08:04 PM

Elon Musk is the founder of Tesla.
11/5/2022 02:08:04 PM

Figure 21: Responses of natural language query to non-relational database using GQL.

As observed, the rules written in 2.4.3.1 and 2.4.3.2 were applied with the SQL and GQL language modules.

Similarity, Investopedia Terms language module extracts the whole definition of the terms as response to the user. If none of the terms are found in the corpus or the query is not match with any historical query, the system will reply as “Sorry I cannot find information from the Investopedia Terms dataset” as shown below:



What is stock?
11/5/2022 02:11:15 PM

Sorry I cannot find the information from the Investopedia Terms dataset
11/5/2022 02:11:15 PM

Explain me abandonment clause?
11/5/2022 02:11:52 PM

An abandonment clause in a property insurance contract, under certain circumstances, permits the property owner to abandon lost or damaged property and still claim a full settlement amount. If the insured party's property cannot be recovered, or the cost to recover or repair it is more than its total value, it can be abandoned and the insured party is entitled to a full settlement amount.
11/5/2022 02:11:53 PM

Tell me the mercantile safe burglary?
11/5/2022 02:12:14 PM

I am sorry, but I do not understand.
11/5/2022 02:12:15 PM

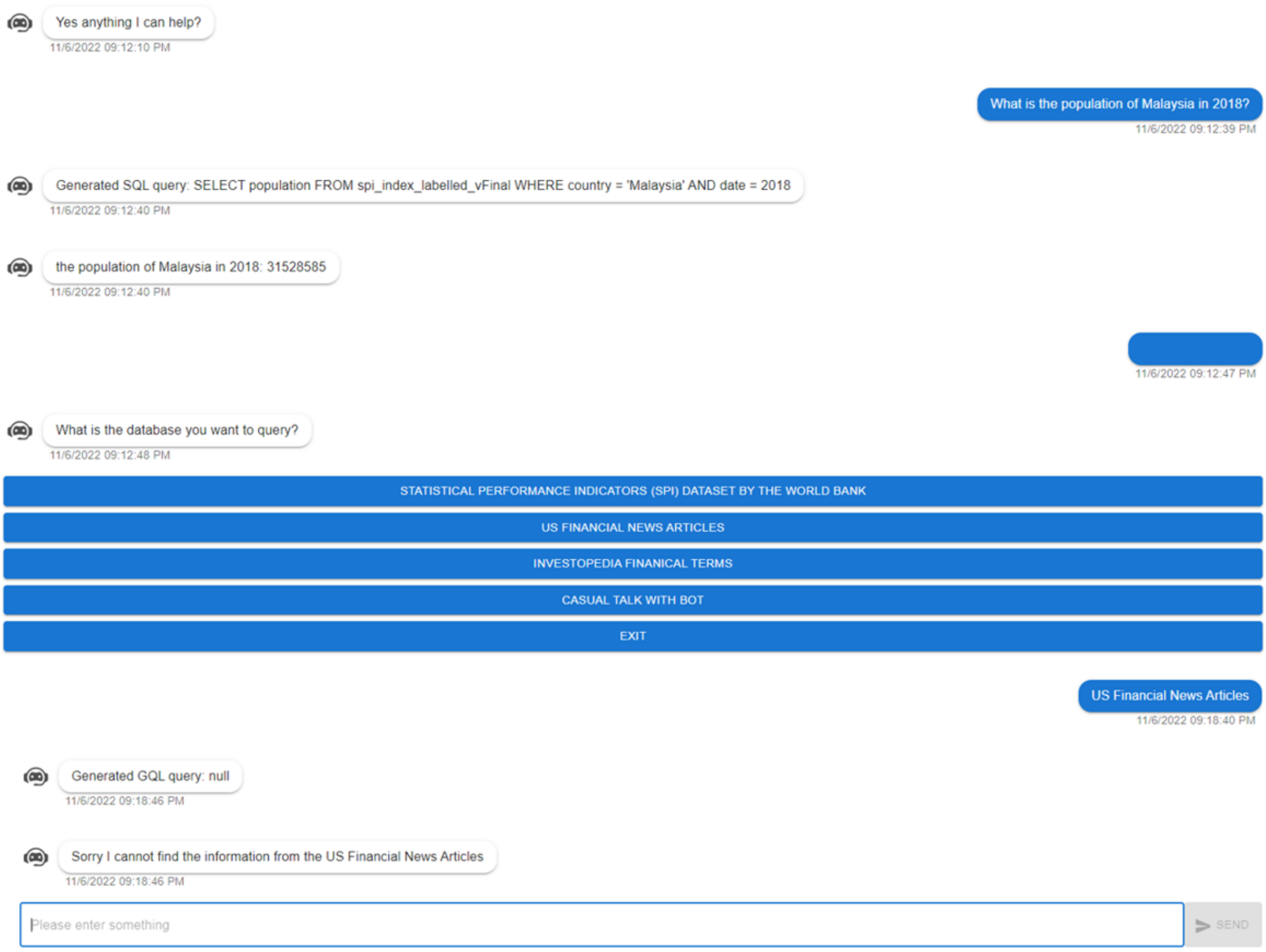
Tell me what is minimum essential coverage.
11/5/2022 02:12:30 PM

Minimum essential coverage is the type of health insurance policy an individual needs to meet health coverage requirements under the Patient Protection and Affordable Care Act (ACA), U.S. health reform. Individuals who do not maintain minimum essential coverage may have to pay a penalty of 2.5% of yearly household income or a maximum of$695 per adult, $347.50 per child under 18, to$2,085 per household.
11/5/2022 02:12:30 PM

Figure 22: Reponses of natural language query to QA corpus.

The chatterbot responses are also managed by the system and only the best matched statement will be returned to user by the chatterbot algorithm. If no suitable response could be found, the chatbot will return the following statement: “I am sorry, but I do not understand”.

The designed chatbot system also has the option to allow users to select any database and corresponding language module to run by submitting a space “ ” value in the text field. The selected intent will be shown in the chat history and it will execute respective language module directly without running the intent classification as illustrated below:



Yes anything I can help?
11/6/2022 09:12:10 PM

What is the population of Malaysia in 2018?
11/6/2022 09:12:39 PM

Generated SQL query: SELECT population FROM spi_index_labelled_vFinal WHERE country = 'Malaysia' AND date = 2018
11/6/2022 09:12:40 PM

the population of Malaysia in 2018: 31528585
11/6/2022 09:12:40 PM

11/6/2022 09:12:47 PM

What is the database you want to query?
11/6/2022 09:12:48 PM

STATISTICAL PERFORMANCE INDICATORS (SPI) DATASET BY THE WORLD BANK
US FINANCIAL NEWS ARTICLES
INVESTOPEDIA FINANICAL TERMS
CASUAL TALK WITH BOT
EXIT

US Financial News Articles
11/6/2022 09:18:40 PM

Generated GQL query: null
11/6/2022 09:18:46 PM

Sorry I cannot find the information from the US Financial News Articles
11/6/2022 09:18:46 PM

Figure 23: User selection of database and intents.

The intent selection will also be triggered if the backend framework encounters any errors or bugs. Users may choose to resubmit the intention or exit the selection as prompted.

3 CHALLENGES & FUTURE WORK

Through the development of our solution, the following challenges & limitations were encountered and the possible future work are suggested:

- Natural language to SQL:

One challenge of this portion was the generation of training pairs of natural language and SQL queries. Although we created a means to automate the generation of thousands of pairs starting with a set of templates, there is still a semi-manual step of generating the templates themselves first.

A possible way to circumvent this would be to use zero-shot learning methods such as those described in the paper: Data Agnostic RoBERTa-based Natural Language to SQL Query Generation [21]. The model is not a sequence to sequence one and only requires the natural language question and table schema. Although a promising alternative as no table data is used during training the model performance lags sequence-to-sequence ones but is an area with potential.

The scope of the project for the SQL translation portion was to provide single table queries based on SQL, to further enhance our solution's functionality we can consider extending to searching over multiple tables with a single query [22].

- Natural language to GQL:

Although the extracted relation and generated queries could be matched using similar pairs of entities, the accuracy of the matching depends highly on the individual capabilities of the automated extraction and generation methods. There are issues such as the presence of multiple entity relations and co-references within a single text sentence, which leads to differences in information extracted even from the same entity pairs.

This was currently addressed by performing manual filtering on the question-relation pairs that were generated; however, future work could focus on developing a language model to identify if the question and relations are derived from similar contexts. For example, an attention-based comparison could be made to verify the similarity of contextual words used to formulate the questions or relations.

- Chatbot

The Django backend framework may encounter the challenges on high memory consumption when there are many models prepared and if databases are served in advance. As we are running the system in the local environment, the system may not react to the user query efficiently due to long calculations by the algorithms.

Furthermore, it was observed that the Chatterbot framework was not compatible with the latest NLP libraries such as NLTK, SpaCy and Scikit-Learn. Future work could explore the capability of RASA AI, to develop the integration of self-made intent classification models with the RASA backend model pipeline.

In future work, possible improvements could involve labelling the training data for slot filling and building an NLG model to handle complex language generation. Rule-based text generation could work perfectly well if the correct scenarios are matched, but it may experience some difficultly handing special use cases.

4 CONCLUSIONS

In this project, we have developed an automated solution for natural language query of relational and non-relational databases. It involved a front-end conversational UI that obtains queries from business users, followed by intent classification to identify the retrieval tasks required, which include converting natural language to SQL or GQL. Seq2Seq model was trained on template generated queries to convert natural language to SQL methods, and achieved a Rogue2 F1 score of 0.78. Whereas for natural language to GQL conversion, entity extraction was coupled with self-supervised relationship classification to obtain the required slots for a GQL statement. The highest accuracy and F1 score of 0.97 were obtained when BERT embeddings were used with CNN-dropout model. Subsequently, the generated SQL and GQL statements were queried to the relational and non-relational databases respectively, to obtain the specific data or information required by the user. The retrieved data was also parsed into a rule-based natural language generator to obtain structured responses as replies to the users. In addition, the conversational UI was also designed to handle general conversation/QA queries through similarity matching against a text corpus. Our solution demonstrated the use of text processing techniques and deep learning methods to address the requirements of natural language queries to retrieve various types of data stored in different formats. This improves the accessibility of data to business users, for more effective data analytics and generation of data-driven insights.

5 BIBLIOGRAPHY

[1] C. Petrov, “25+ Impressive Big Data Statistics for 2022,” Techjury, 14 Oct 2022. [Online]. Available: https://techjury.net/blog/big-data-statistics/#gref. [Accessed 7 Nov 2022].

[2] Statista, “Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025,” 8 Sep 2022. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/. [Accessed 7 Nov 2022].

[3] J. P. U. S. B. S. Hai-Anh H. Dang, “Statistical Performance Indicators and Index : A New Tool to Measure Country Statistical Capacity (English),” *World Bank,* 2021.

[4] C. X. R. S. Victor Zhong, “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning,” *Salesforce Research,* 2017.

[5] P. N. Z. W. R. N. B. X. Alexander R. Fabbri, “Template-Based Question Generation from Retrieved Sentences for Improved Unsupervised Question Answering,” in *Association for Computational Linguistics*, 2020.

[6] M. Romeo, “t5-base-finetuned-wikiSQL,” HuggingFace, [Online]. Available: https://huggingface.co/mrm8488/t5-base-finetuned-wikiSQL?text=translate+English+to+SQL%3A+What+is+the+Rank+of+the+Player+with+362+Matches%3F.

[7] N. S. A. R. K. L. S. N. M. M. Y. Z. W. L. P. J. L. Colin Raffel, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” *Journal of Machine Learning Research 21,* 2020.

[8] J. Briggs, “The Ultimate Performance Metric in NLP,” Towards Data Science, 4 March 2021. [Online]. Available: https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460.

[9] Neo4j, “Cypher Query Language,” [Online]. Available: https://neo4j.com/developer/cypher/. [Accessed 7 Nov 2022].

[10] J. Jeet, “US Financial News Articles,” [Online]. Available: https://www.kaggle.com/datasets/jeet2016/us-financial-news-articles. [Accessed 7 Nov 2022].

[11] P.-L. Huguet Cabot and R. Navigli, REBEL: Relation Extraction By End-to-end Language generation, Association for Computational Linguistics, 2021.

[12] Haystack, [Online]. Available: https://haystack.deepset.ai/. [Accessed 7 Nov 2022].

[13] J. Pennington, R. Socher and C. D. Manning, “GloVe: Global Vectors for Word Representation,” 2014.

[14] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 2019.

[15] H. El Boukkouri, “Arithmetic Properties of Word Embeddings,” 13 Aug 2020. [Online]. Available: https://blog.dataiku.com/arithmetic-properties-of-word-embeddings. [Accessed 7 Nov 2022].

[16] S. Singhal, R. R. Shah, T. Chakraborty, P. Kumaraguru and S. Satoh, “SpotFake: A Multi-modal Framework for Fake News Detection,” 2019.

[17] “Investopedia-scrapper,” 24 Sep 2019. [Online]. Available: https://github.com/vishnuverse/Investopedia-scrapper. [Accessed 7 Nov 2022].

[18] S. Faroz, “Small talk : Intent Classification data,” [Online]. Available: https://www.kaggle.com/datasets/salmanfaroz/small-talk-intent-classification-data. [Accessed 7 Nov 2022].

[19] N. Reimers and I. Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, Association for Computational Linguistics, 2019.

[20] SpaCy, [Online]. Available: https://spacy.io/. [Accessed 7 Nov 2022].

[21] H. S. K. C. Debaditya Pal, “Data Agnostic RoBERTa-based Natural Language to SQL Query Generation,” 2020.

[22] N. G. G. M. Adrián Bazaga, “Translating synthetic natural language to database queries with a polyglot deep learning framework,” *Nature,* 2021.

APPENDIX A

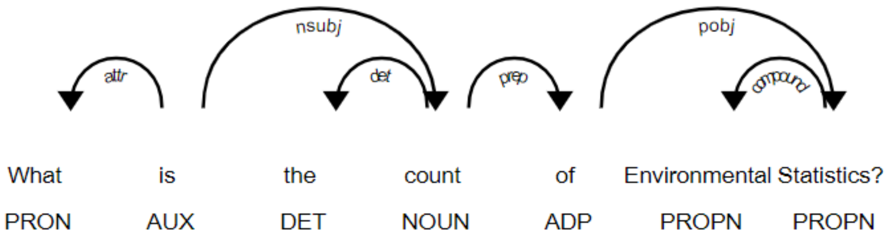
<u>Rules for Natural Language Generation with illustrations:</u>

- **<u>SQL</u>**

There are four rules defined to extract the contextual information from the question:

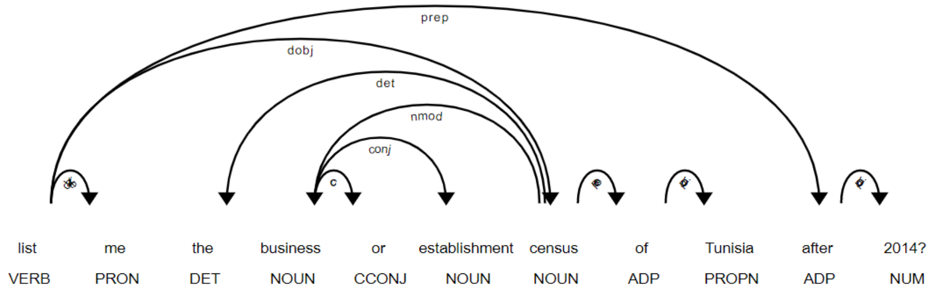
1. Removal of the interrogative word and token with tag “AUX”. Extraction of the rest of the word token before the last punctuation word. Eg: “the count of Environment Statistics” is extracted.

   E.g. “What is the count of Environment Statistics?” - > “the count of Environment Statistics”

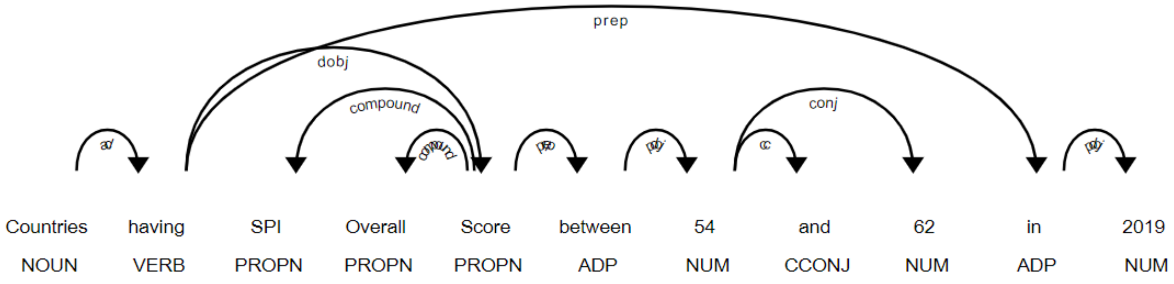


2. If word token with POS tag “VERB” starts from the query and it is a “ROOT” token, the extraction of the contextual sentence starts only from the “DET” word token before the last punctuation word. The rule is only executable after the rule 1 is skipped. Eg: “the business or establishment census of Tunisia after 2014” is extracted.

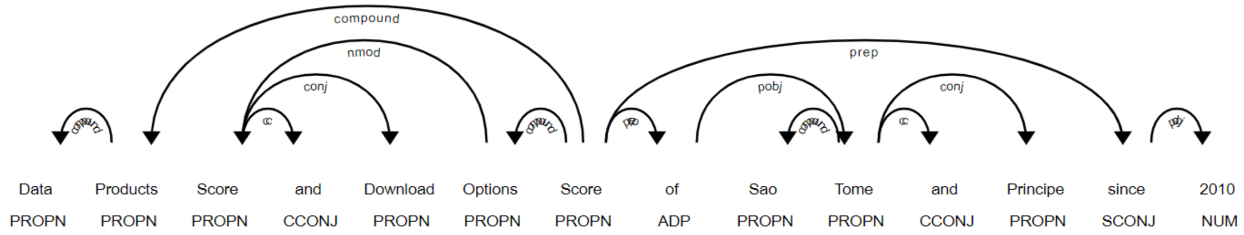
   E.g. “list me the business or establishment census of Tunisia after 2014?” - > “the business or establishment census of Tunisia after 2014”



3. Retrieval of whole sentence as the contextual sentence until the last token is punctuation when the sentence is started with word token with POS tag “NOUN”. The rule is only executable after the rule 2 is skipped. Eg: “Countries having SPI Overall Score between 54 and 62 in 2019” is extracted.



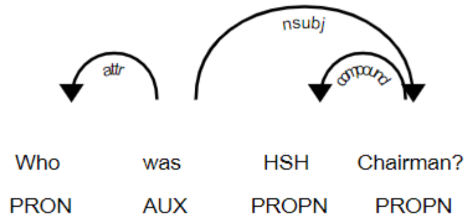
4. Retrieval of whole sentence as the contextual sentence until the last token is punctuation when the sentence is started with the first Named Entity recognised by the English Pipeline. The rule is only executable after the rule 3 is skipped. Eg: “Data Products Score” is entity extracted from the sentence submitted.

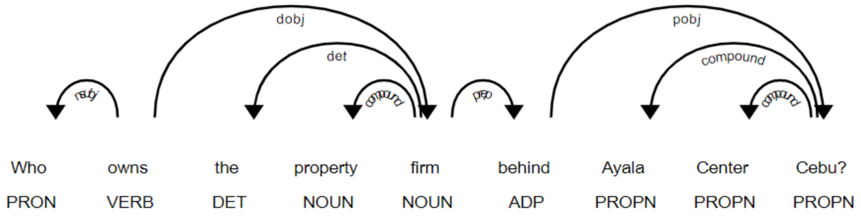


- **GQL**

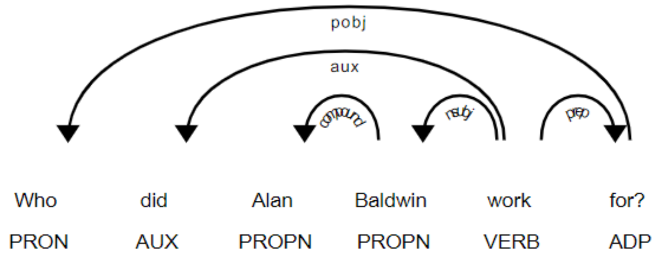
Upon observation of the training data for the GQL module, majority of the queries contain interrogative word token in the beginning of the sentence. Therefore, the rules defined for the GQL module splits the queries to the type of the interrogative sentences:

1. Word Token “Who” scenario:
   a. Replace the word token “Who” with the answer if the interrogative word is followed by “AUX” or “VERB” word token. Refer to the following examples:

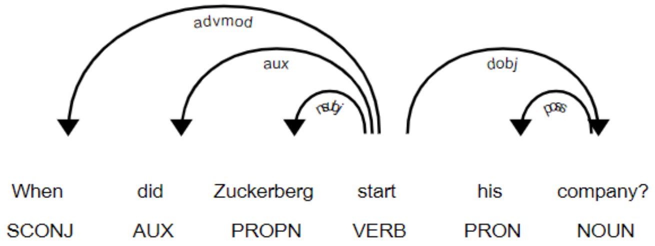




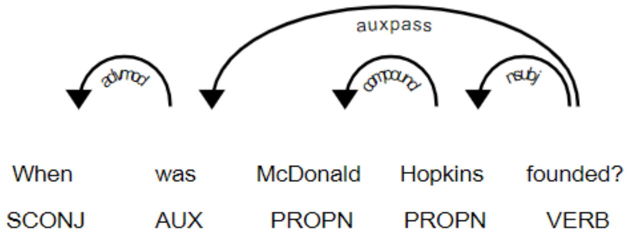
   b. Removal of the token "Who" and any AUX token after interrogative word. Placement of the answer right after the VERB token or ADP/ADV token if the following pattern is met:



2. Word Token “When” scenario:
    a. Removal of the token "When" and any AUX token after interrogative word if the following scenario is met.

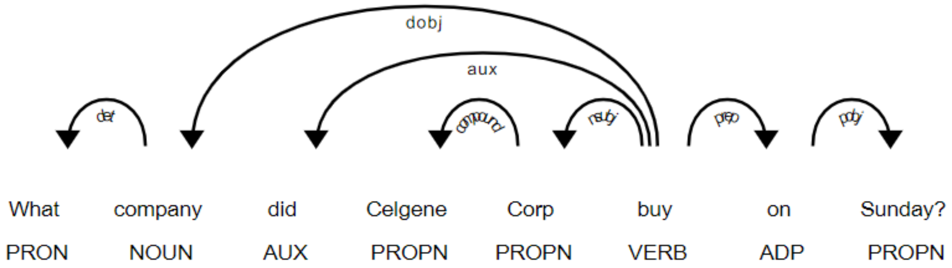


    b. AUX tokens "is" "are" "was" "were" are migrated and placed before the first "VERB token".

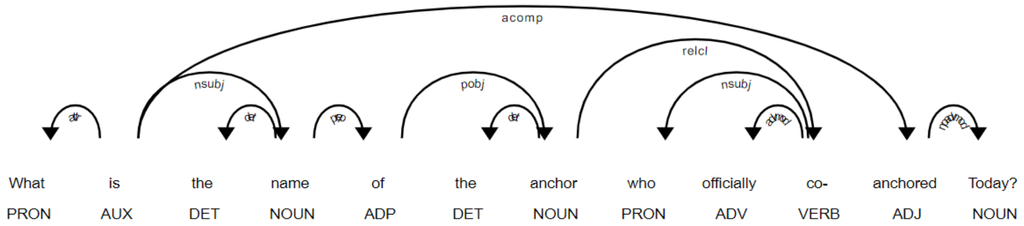


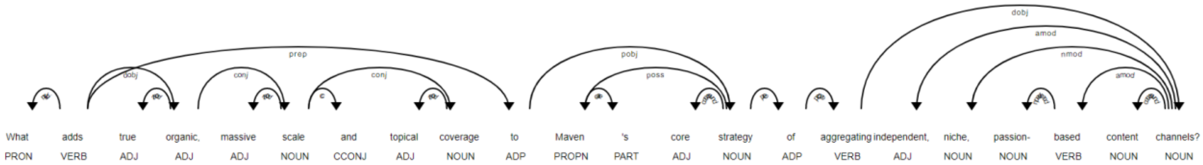
    c. Placement of the answer at the end of the statement with word token “In” for all the scenarios with interrogative word “When”.

3. Word Token “Which” and “What” scenario:
    a. Ignore of the word token “NOUN” or “PRON” after the interrogative word.

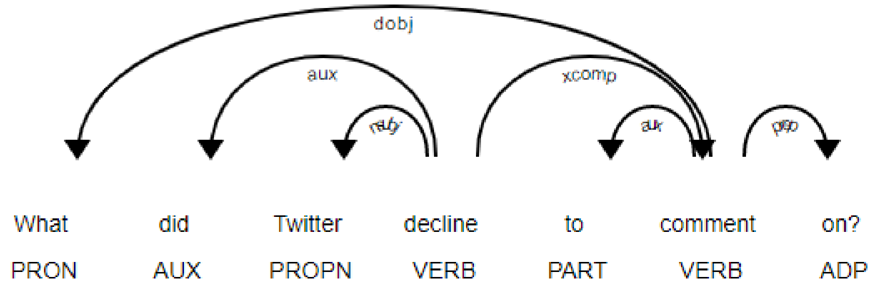


    b. Replace the word token “What” or “Which” with the answer if the interrogative word is followed by “AUX” or “VERB” word token. Refer to the following examples:



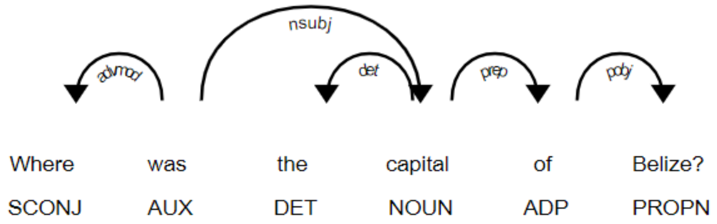


c. Removal of the token "What" or “Which” and any AUX token after interrogative word. Placement of the answer right after the VERB token or ADP/ADV token if the following pattern is met:

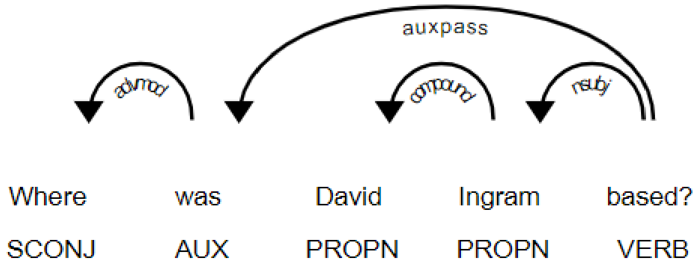


4. Word Token “Where” scenario:

a. Removal of the token " Where" and any AUX token after interrogative word if the following scenario is met.



b. AUX tokens "is" "are" "was" "were" are migrated and placed before the first "VERB token".



c. Placement of the answer at the end of the statement with word token “In” for all the scenarios with interrogative word “Where”.