INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Course Name - Information Retrieval and Extraction (CS4.406)

Project Name - Natural Language Query-based Table Retrieval
(August 23 - December 23)

Under the Guidance of
Prof. Rahul Mishra

By - Team 8
(The Diligents)

Aryan Gupta (2022202028)
Utkarsh Pathak (2022201018)

Project Overview:

In this era of big data, organizations accumulate vast amounts of structured data stored in tables, databases, and spreadsheets. Accessing and retrieving valuable insights from this data can be a daunting task, especially for non-technical users who may not be proficient in SQL or database querying languages. Natural language interfaces offer a promising solution to bridge this gap, enabling users to pose questions in plain language and receive meaningful answers from the corpus of tables.

Various models can do the ranked retrieval of text documents and can also find the best suitable answer to the given query. But there is a massive amount of structured data in the form of tables which contains very useful information. Hence this project helps to make a mechanism that can output the most relevant tables based on the natural language query and can further generate answers in any form.

Problem Statement:

The goal of this project is to develop an efficient and user-friendly system that allows users to query a corpus of structured tables using natural language queries. The system should be capable of understating and interpreting user questions, translating them into actionable queries against the corpus, and presenting relevant answers.

Given natural language questions a large table corpus, end-to-end table retrieval and question-answering system unify two tasks:

1. Answering the questions using table cells and generating heatmaps highlighting relevant columns and rows over complex providing an overall picture for the users.
2. Locating a list of tables- highly relevant to the given question

Understanding Datasets:

1. **WikiSQL Dataset:** This is also a widely used benchmark dataset in natural language processing (NLP) and question-answering research. It is designed to evaluate the ability of NLP models to understand and generate SQL queries based on natural language questions and a structured database schema.

2. **TabMCQ Dataset:** This dataset is a collection of curated facts in the form of tables, and the second is a large set of crowd-sourced multiple-choice questions covering the facts in the form of tables.

3. **WikiTableQuestions Dataset:** It is a benchmark dataset used in natural language processing (NLP) and question-answering research. It comprised question-answer pairs on HTML tables and was constructed by selecting data tables from Wikipedia.

Importance of these databases:

These datasets play a crucial role in training and evaluating question-answering (QA) models for tabular data. They provide a diverse range of natural language questions and corresponding answers related to structured data in tables. This allows models to learn the complex relationships between language and tabular data, enabling them to effectively interpret and respond to questions about tabular information.

WikiSQL Dataset:

- Focus: Understanding and generating SQL queries from natural language questions
- Benefits: Enhances models' ability to translate natural language instructions into structured queries
- Applications: Natural language interfaces for databases, data extraction, and data analysis

TabMCQ Dataset:

- Focus: Answering multiple-choice questions based on facts presented in tables
- Benefits: Improves models' comprehension of tabular data and their ability to retrieve relevant information
- Applications: Educational assessment, knowledge base validation, and data summarization

WikiTableQuestions Dataset:

- Focus: Answering natural language questions about factual information extracted from Wikipedia tables
- Benefits: Strengthens models' ability to process and understand complex table structures and extract meaningful insights
- Applications: Table summarization, data exploration, and data-driven decision-making

In summary, these datasets provide valuable training grounds for QA models on tabular data, enabling them to perform various tasks that involve understanding, interpreting, and reasoning about structured information presented in tables. They play a significant role in advancing the field of NLP and its applications in data analysis, knowledge discovery, and decision support.

Key Objectives of the project:

1. Natural Language Understanding: Develop the ability to parse and understand queries expressed in natural language, extracting keywords, entities, and intents

2. Table Corpus Indexing: Create a robust indexing mechanism for the corpus of tables, including the metadata about the table structure, relationship, and content.

3. Query Translation: Develop a query translation module that converts natural language queries into structured queries that can be executed against the table corpus.

4. Query Execution: Implement the functionality to execute structured queries on the indexed table corpus, retrieving relevant data.

5. Ranking and Presentation: Design a system that ranks and presents query results in a user-friendly format, potentially including summary statistics, visualizations, or relevant context.

6. Feedback and Iteration: Incorporate a feedback loop that allows users to refine their queries and provides suggestions for better results.

7. Error Handling: Handle ambiguity and errors gracefully, offering clarifications or alternatives when necessary.

8. User Interface: Develop an intuitive user interface for users to interact with the system, input natural language queries, and view results.

Implementation:

1. **Preprocessing of Datasets:**

- **TabMCQ Dataset:**

Downloading the dataset and unzipping the folder.

```
[ ] !wget https://ai2-public-datasets.s3.amazonaws.com/tablestore-questions/TabMCQ_v_1.0.zip
    !unzip TabMCQ_v_1.0.zip

    --2023-10-18 09:00:27--  https://ai2-public-datasets.s3.amazonaws.com/tablestore-questions/TabMCQ_v_1.0.zip
    Resolving ai2-public-datasets.s3.amazonaws.com (ai2-public-datasets.s3.amazonaws.com)... 52.92.196.161, 52.218.181.51, 52.92.144.113, ...
    Connecting to ai2-public-datasets.s3.amazonaws.com (ai2-public-datasets.s3.amazonaws.com)|52.92.196.161|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 698796 (682K) [binary/octet-stream]
    Saving to: 'TabMCQ_v_1.0.zip'
```

This contains 1 folder of tables having 'auto', 'monarch', and 'regents' folders containing tables as tsv files and another main file as MCQs.tsv containing the questions in the form of tsv.

```
for subdir in ['auto','monarch','regents']:
  subdir_f = os.path.join('./Tables',subdir)
  for file in os.listdir(subdir_f):
    # assert file[-4:] == '.tsv'
    # print(file)
    table_id = subdir+'-'+file[:-4]
    # print(table_id)
    rows = []
    with read_file(os.path.join(subdir_f,file)) as fp:
      for r,part in enumerate(csv.reader(fp,doublequote=False,delimiter='\t')):
        row = [c.strip() for c in part]
        # print(r,row,part)
        rows.append(row)
    tid2rows[table_id] = rows
```

Now, reading each file from each folder and storing all the data row-wise in a dictionary of tid2rows.

```
tid2rows
```

```
{'auto-fb2': [['LHS', 'Relation', 'RHS'],
  ['animals need water also', 'effect', 'animals survive'],
  ['all living things need nutrients', 'effect', 'all living things survive'],
  ['animals take in food', 'effect', 'animals get nutrients'],
  ['all cells need oxygen', 'effect', 'all cells survive'],
  ['they eat a great deal of food', 'effect', 'they increase their body fat'],
  ['you wash your hands often', 'effect', 'you remove any harmful germs'],
  ["living things & ( living things change the sun 's energy ) change the sun 's energy",
   'effect',
   "living things & ( living things change the sun 's energy ) make their[producers] own food"],
  ['humans use resources from the environment',
   'effect',
   'humans create shelters & energy'],
  ['humans use wood', 'effect', 'humans build homes'],
  ['humans use materials also', 'effect', 'humans create energy'],
  ['we use fossil fuels', 'effect', 'we make gasoline'],
  ['a greater demand for food is  since the human population increases in size everyday',
   'effect',
   'humans survive'],
```

Now, we read the MCQs file containing the questions with the choices and options.

```
with read_file("./MCQs.tsv") as fp:
  for i,parts in enumerate(csv.reader(fp,doublequote=False,delimiter='\t')):
    # print(i,parts)
    if i ==0:
      continue
    # print(len(parts))
    if (len(parts)!=10):
      print(f'bad line:{parts}')
      exit(1)
```

After reading the files from tables and MCQs files storing them in the data dictionary

```
# print(parts)
qid = f'q{i}'
qtext = parts[0].strip()
quest_align = [int(c.strip()) for c in parts[1].split(',')]
# print(quest_align)
choices = [c.strip() for c in parts[2:6]]
# print(choices)
answer1 = choices[int(parts[6])-1]
# print(answer)
table_id = parts[7]
target_row = int(parts[8])-1
target_col = int(parts[9])
# print(target_col)
all_rows = tid2rows[table_id]
# print(all_rows)
header = all_rows[0]
# print(header)
rows = all_rows[1:]
if target_col in quest_align:
  quest_align.remove(target_col)
```

```
      # print(answer1," ----- ",answer2)
      data = {}
      data['id'] = qid
      data['question'] = qtext
      data['header'] = header
      data['rows'] = rows
      data['target_column'] = target_col
      data['answers'] = [answer1]
      data['table_id'] = table_id
```

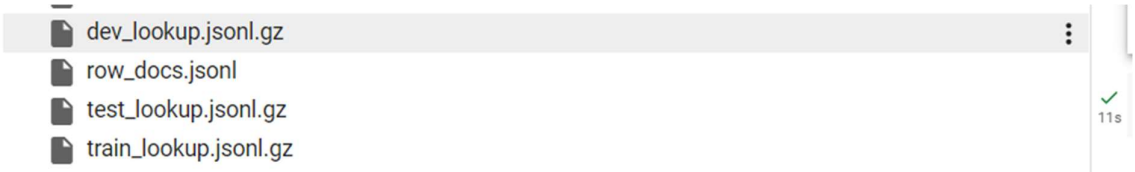
Loading...

Now, splitting the data into train, dev, and test files to train and test the model.

```
[ ]  splits = []
     for s in ['dev_lookup.jsonl.gz', 'test_lookup.jsonl.gz', 'train_lookup.jsonl.gz']:
       splits.append(write_file(os.path.join("./",s)))
```

```
      line = json.dumps(data) + '\n'

      if i % 100 < dev_percent:
        splits[0].write(line)
      elif i % 100 < dev_percent + test_percent:
        splits[1].write(line)
      else:
        splits[2].write(line)


  for sfile in splits:
    sfile.close()
```

Finally, the data looks like this:



```
{"id":"q40","question":"In which hemisphere does the summer solstice occur in December?","header":["","HEMISPHERE","","ORBITAL EVENT","","MONTH OF OCCURENCE"],"rows":[["In the","northern hemisphere",", the","summer solstice","occurs in","June"],["In the","southern hemisphere",", the","summer solstice","occurs in","December"],["In the","northern hemisphere",", the","winter solstice","occurs in","December"],["In the","southern hemisphere",", the","winter solstice","occurs in","June"],["In the","northern hemisphere",", the","spring equinox","occurs in","March"],["In the","southern hemisphere",", the","spring equinox","occurs in","September"],["In the","northern hemisphere",", the","fall equinox","occurs in","September"],["In the","southern hemisphere",", the","fall equinox","occurs in","March"]],"target_column":1,"answers":["Southern hemisphere"],"table_id":"regents-02"}

{"id":"q41","question":"The winter solstice, in the northern hemisphere takes place in which month?","header":["","HEMISPHERE","","ORBITAL EVENT","","MONTH OF OCCURENCE"],"rows":[["In the","northern hemisphere",", the","summer solstice","occurs in","June"],["In the","southern hemisphere",", the","summer solstice","occurs in","December"],["In the","northern hemisphere",", the","winter solstice","occurs in","December"],["In the","southern hemisphere",", the","winter solstice","occurs in","June"],["In the","northern hemisphere",", the","spring equinox","occurs in","March"],["In the","southern hemisphere",", the","spring equinox","occurs in","September"],["In the","northern hemisphere",", the","fall equinox","occurs in","September"],["In the","southern hemisphere",", the","fall equinox","occurs in","March"]],"target_column":5,"answers":["December"],"table_id":"regents-02"}

{"id":"q42","question":"In what month does winter solstice occur in the Southern hemisphere?","header":["","HEMISPHERE","","ORBITAL EVENT","","MONTH OF OCCURENCE"],"rows":[["In the","northern hemisphere",", the","summer solstice","occurs in","June"],["In the","southern hemisphere",", the","summer solstice","occurs in","December"],["In the","northern hemisphere",", the","winter solstice","occurs in","December"],["In the","southern hemisphere",", the","winter solstice","occurs in","June"],["In the","northern hemisphere",", the","spring equinox","occurs in","March"],["In the","southern hemisphere",", the","spring equinox","occurs in","September"],["In the","northern hemisphere",", the","fall equinox","occurs in","September"],["In the","southern hemisphere",", the","fall equinox","occurs in","March"]],"target_column":1,"answers":["March"],"table_id":"regents-02"}

{"id":"q43","question":"Which event occurs in June in the southern hemisphere?","header":["","HEMISPHERE","","ORBITAL EVENT","","MONTH OF OCCURENCE"],"rows":[["In the","northern hemisphere",", the","summer solstice","occurs in","June"],["In the","southern hemisphere",", the","summer solstice","occurs in","December"],["In the","northern hemisphere",", the","winter solstice","occurs in","December"],["In the","southern hemisphere",", the","winter solstice","occurs in","June"],["In the","northern hemisphere",", the","spring equinox","occurs in","March"],["In the","southern hemisphere",", the","spring equinox","occurs in","September"],["In the","northern hemisphere",", the","fall equinox","occurs in","September"],["In the","southern hemisphere",", the","fall equinox","occurs in","March"]],"target_column":3,"answers":["winter solstice"],"table_id":"regents-02"}

{"id":"q44","question":"When does the winter solstice occur in the southern hemisphere?","header":["","HEMISPHERE","","ORBITAL EVENT","","MONTH OF OCCURENCE"],"rows":[["In the","northern hemisphere",", the","summer solstice","occurs in","June"],["In the","southern hemisphere",", the","summer solstice","occurs in","December"],["In the","northern hemisphere",", the","winter solstice","occurs in","December"],["In the","southern hemisphere",", the","winter solstice","occurs in","June"],["In the","northern hemisphere",", the","spring equinox","occurs in","March"],["In the","southern hemisphere",", the","spring equinox","occurs in","September"],["In the","northern hemisphere",", the","fall equinox","occurs in","September"],["In the","southern hemisphere",", the","fall equinox","occurs in","March"]],"target_column":5,"answers":["June"],"table_id":"regents-02"}
```


- **WikiSQL Dataset:**

Downloading the dataset and extracting all the files.

```
!git clone https://github.com/salesforce/WikiSQL.git
import tarfile

file_path = 'WikiSQL/data.tar.bz2'

with tarfile.open(file_path, 'r:bz2') as tar:
    tar.extractall()
```

```
Cloning into 'WikiSQL'...
remote: Enumerating objects: 386, done.
remote: Counting objects: 100% (192/192), done.
```

This contains 1 folder of data having 'train', 'test', and 'dev' files in format od jsonl, tables and db files. Now, with the help of DBEngine adding the answer and rowids to each line of the all files, getting the _ans files.

```
[17] for split in ['train','dev','test']:
       orig = os.path.join("data/",f'{split}.jsonl')
       db_file = os.path.join("data/",f'{split}.db')
       ans_file = os.path.join("data/",f'{split}_ans.jsonl.gz')
       tbl_file = os.path.join("data/",f'{split}.tables.jsonl')
       engine = DBEngine(db_file)

       with open(orig) as fs, write_file(ans_file) as fo:

         for ls in tqdm(fs):
           try:
             eg = json.loads(ls)
           except json.JSONDecodeError as e:
             continue
           sql = eg['sql']
           qg = Query.from_dict(sql,ordered=False)
           gold = engine.execute_query(eg['table_id'],qg,lower=True)
           assert isinstance(gold,list)

           eg['answer'] = gold
           eg['rowids'] = engine.execute_query_rowid(eg['table_id'],qg,lower=True)
           fo.write(json.dumps(eg)+'\n')
```

Now, using the _ans files convert the file to a suitable dictionary, same as the above dataset.

```
def convert_queries(queries,tables,output_file,*,skip_aggregation=True, show_aggregation=False):
  tid2rows = dict()
  for l in tables:
    data = json.loads(l)
    tid = data['id']
    header = data['header']
    rows_or = data['rows']
    rows = []

    for r in rows_or:
      rows.append([str(cv) for cv in r])

    tid2rows[tid] = [[str(h) for h in header]] + rows

  with write_file(output_file) as fo:
    for qid,l in enumerate(queries):
      data = json.loads(l)
      index = data['sql']['agg']

      if skip_aggregation and index!=0:
        continue

      table_id = data['table_id']
      rows = tid2rows[table_id]
      qtext = data['question']
      target_column = data['sql']['sel']
      condition_columns = [colndx for colndx,comp,val in data['sql']['conds']]
      answers = data['answer']
      rowids = data['rowids'] if 'rowids' in data else None
```

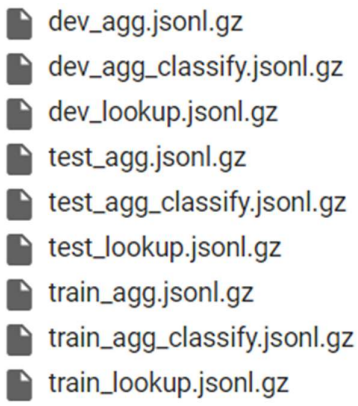
```
      data = dict()
      data['id'] = f'{qid}'
      data['question'] = qtext
      data['header'] = rows[0]
      data['rows'] = rows[1:]
      data['target_column'] = target_column
      data['condition_columns'] = condition_columns
      data['table_id'] = table_id
      data['index'] = index

      if rowids is not None:
        data['target_rows'] = rowids

      if index ==0:
        answers = [str(ans) for ans in answers]
        clean_ans = []

        for r in rows[1:]:
          if cell_value(r[target_column],answers):
            clean_ans.append(r[target_column])

        data['answers'] = list(set(clean_ans))

      else:
        data['answers'] = answers
```

Now, these files are stored in the form of lookup and aggregation files containing the lookup queries and aggregation queries.

```
      fo.write(json.dumps(data)+'\n')
```

```
convert_queries(j_lines(ans_file),j_lines(tbl_file),os.path.join("./", f"{split}_agg.jsonl.gz"),skip_aggregation=False)
convert_queries(j_lines(ans_file),j_lines(tbl_file),os.path.join("./", f"{split}_lookup.jsonl.gz"),skip_aggregation=True)
```

Finally, the data looks like this:



Lookup queries file:

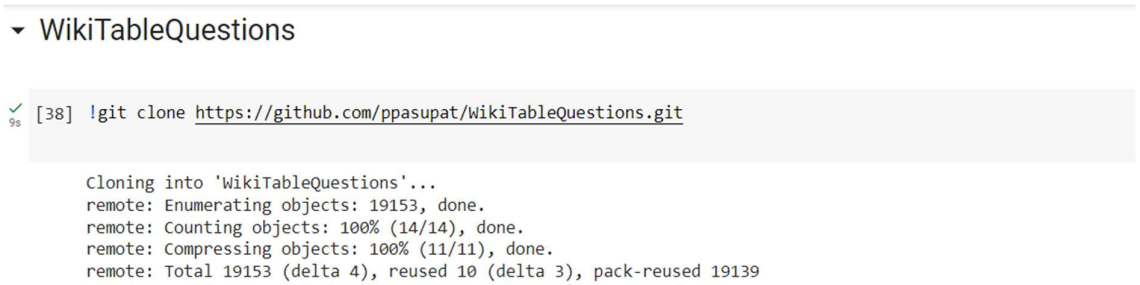

```
{"id": "0", "question": "Tell me what the notes are for South Australia ", "header": ["State/territory", "Text/background colour", "Format", "Current slogan", "Current series", "Notes"], "rows": [["Australian Capital Territory", "blue/white", "Yaa\u00b7nna", "ACT \u00b7 CELEBRATION OF A CENTURY 2013", "YIL\u00b700A", "Slogan screenprinted on plate"], ["New South Wales", "black/yellow", "aa\u00b7nn\u00b7aa", "NEW SOUTH WALES", "BX\u00b799\u00b7HI", "No slogan on current series"], ["New South Wales", "black/white", "aaa\u00b7nna", "NSW", "CPX\u00b712A", "Optional white slimline series"], ["Northern Territory", "ochre/white", "Ca\u00b7nn\u00b7aa", "NT \u00b7 OUTBACK AUSTRALIA", "CB\u00b706\u00b7ZZ", "New series began in June 2011"], ["Queensland", "maroon/white", "nnn\u00b7aaa", "QUEENSLAND \u00b7 SUNSHINE STATE", "999\u00b7TLG", "Slogan embossed on plate"], ["South Australia", "black/white", "Snnn\u00b7aaa", "SOUTH AUSTRALIA", "S000\u00b7AZD", "No slogan on current series"], ["Victoria", "blue/white", "aaa\u00b7nnn", "VICTORIA - THE PLACE TO BE", "ZZZ\u00b7562", "Current series will be exhausted this year"]], "target_column": 5, "condition_columns": [3], "table_id": "1-1000181-1", "index": 0, "target_rows": [5], "answers": ["No slogan on current series"]}

{"id": "1", "question": "What is the current series where the new series began in June 2011?", "header": ["State/territory", "Text/background colour", "Format", "Current slogan", "Current series", "Notes"], "rows": [["Australian Capital Territory", "blue/white", "Yaa\u00b7nna", "ACT \u00b7 CELEBRATION OF A CENTURY 2013", "YIL\u00b700A", "Slogan screenprinted on plate"], ["New South Wales", "black/yellow", "aa\u00b7nn\u00b7aa", "NEW SOUTH WALES", "BX\u00b799\u00b7HI", "No slogan on current series"], ["New South Wales", "black/white", "aaa\u00b7nna", "NSW", "CPX\u00b712A", "Optional white slimline series"], ["Northern Territory", "ochre/white", "Ca\u00b7nn\u00b7aa", "NT \u00b7 OUTBACK AUSTRALIA", "CB\u00b706\u00b7ZZ", "New series began in June 2011"], ["Queensland", "maroon/white", "nnn\u00b7aaa", "QUEENSLAND \u00b7 SUNSHINE STATE", "999\u00b7TLG", "Slogan embossed on plate"], ["South Australia", "black/white", "Snnn\u00b7aaa", "SOUTH AUSTRALIA", "S000\u00b7AZD", "No slogan on current series"], ["Victoria", "blue/white", "aaa\u00b7nnn", "VICTORIA - THE PLACE TO BE", "ZZZ\u00b7562", "Current series will be exhausted this year"]], "target_column": 4, "condition_columns": [5], "table_id": "1-1000181-1", "index": 0, "target_rows": [3], "answers": ["CB\u00b706\u00b7ZZ"]}
```


Aggregation queries file:


```
{"id": "0", "question": "Tell me what the notes are for South Australia ", "header": ["State/territory", "Text/background colour", "Format", "Current slogan", "Current series", "Notes"], "rows": [["Australian Capital Territory", "blue/white", "Yaa\u00b7nna", "ACT \u00b7 CELEBRATION OF A CENTURY 2013", "YIL\u00b700A", "Slogan screenprinted on plate"], ["New South Wales", "black/yellow", "aa\u00b7nn\u00b7aa", "NEW SOUTH WALES", "BX\u00b799\u00b7HI", "No slogan on current series"], ["New South Wales", "black/white", "aaa\u00b7nna", "NSW", "CPX\u00b712A", "Optional white slimline series"], ["Northern Territory", "ochre/white", "Ca\u00b7nn\u00b7aa", "NT \u00b7 OUTBACK AUSTRALIA", "CB\u00b706\u00b7ZZ", "New series began in June 2011"], ["Queensland", "maroon/white", "nnn\u00b7aaa", "QUEENSLAND \u00b7 SUNSHINE STATE", "999\u00b7TLG", "Slogan embossed on plate"], ["South Australia", "black/white", "Snnn\u00b7aaa", "SOUTH AUSTRALIA", "S000\u00b7AZD", "No slogan on current series"], ["Victoria", "blue/white", "aaa\u00b7nnn", "VICTORIA - THE PLACE TO BE", "ZZZ\u00b7562", "Current series will be exhausted this year"]], "target_column": 5, "condition_columns": [3], "table_id": "1-1000181-1", "index": 0, "target_rows": [5], "answers": ["No slogan on current series"]}

{"id": "1", "question": "What is the current series where the new series began in June 2011?", "header": ["State/territory", "Text/background colour", "Format", "Current slogan", "Current series", "Notes"], "rows": [["Australian Capital Territory", "blue/white", "Yaa\u00b7nna", "ACT \u00b7 CELEBRATION OF A CENTURY 2013", "YIL\u00b700A", "Slogan screenprinted on plate"], ["New South Wales", "black/yellow", "aa\u00b7nn\u00b7aa", "NEW SOUTH WALES", "BX\u00b799\u00b7HI", "No slogan on current series"], ["New South Wales", "black/white", "aaa\u00b7nna", "NSW", "CPX\u00b712A", "Optional white slimline series"], ["Northern Territory", "ochre/white", "Ca\u00b7nn\u00b7aa", "NT \u00b7 OUTBACK AUSTRALIA", "CB\u00b706\u00b7ZZ", "New series began in June 2011"], ["Queensland", "maroon/white", "nnn\u00b7aaa", "QUEENSLAND \u00b7 SUNSHINE STATE", "999\u00b7TLG", "Slogan embossed on plate"], ["South Australia", "black/white", "Snnn\u00b7aaa", "SOUTH AUSTRALIA", "S000\u00b7AZD", "No slogan on current series"], ["Victoria", "blue/white", "aaa\u00b7nnn", "VICTORIA - THE PLACE TO BE", "ZZZ\u00b7562", "Current series will be exhausted this year"]], "target_column": 4, "condition_columns": [5], "table_id": "1-1000181-1", "index": 0, "target_rows": [3], "answers": ["CB\u00b706\u00b7ZZ"]}
```


- **<u>WikiTableQuestions Dataset:</u>**

Download the dataset



This contains 2 folders of csv and data. Csv folder contains multiple folders having multiple table type files like html and the data folder contain 'training.tsv', 'pristine-seen-tables.tsv', and 'pristine-unseen-tables.tsv' files for the data, queries, and answers.

```
[40] csv_dir = os.path.join("WikiTableQuestions", 'csv')
     tid2rows = dict()
     for dir in os.listdir(csv_dir):
       dirs = os.path.join(csv_dir,dir)
       for file in os.listdir(dirs):
         with read_file(os.path.join(dirs,file)) as cf:
           rows=[]
           for r in csv.reader(cf,doublequote=False,escapechar='\\'):
             rows.append(r)
           tid2rows[f'csv/{dir}/{file}'] = rows
```

Now, we read the table files from the csv folders and stored the it in a dictionary of tid2rows.

```
tid2rows
```

```
{'csv/200-csv/12.html': [['<table class="wikitable">'],
  ['<tr>'],
  ['<th>Year</th>'],
  ['<th>Award</th>'],
  ['<th>Category</th>'],
  ['<th>Nominated work</th>'],
  ['<th>Result</th>'],
  ['</tr>'],
  ['<tr>'],
  ['<td>1979</td>'],
  ['<td><a href="//en.wikipedia.org/wiki/Laurence_Olivier_Award" title="Laurence Olivier Award">Olivier Award</a></td>'],
  ['<td><a href="//en.wikipedia.org/wiki/Laurence_Olivier_Award_for_Best_Actress" title="Laurence Olivier Award for Best Actress">Best
Actress</a> in a Revival</td>'],
  ['<td><i>Once in a Lifetime</i></td>'],
  ['<td class="yes table-yes2" style="background: #99FF99; color: black; vertical-align: middle; text-align: center;">Won</td>'],
```

Now, reading the data from the files 'training.tsv', 'pristine-seen-tables.tsv', and 'pristine-unseen-tables.tsv'. Now from these files extracting the query, header, rows and answers.

```
for in_file in ['training.tsv', 'pristine-seen-tables.tsv','pristine-unseen-tables.tsv']:
  for ind,l in enumerate(read_lines(os.path.join(data_dir,in_file))):
    parts = l.strip().split('\t')
    assert len(parts) ==4

    if ind==0:
      continue

    id = parts[0]
    if id not in id2split:
      continue

    split = id2split[id]
    query = parts[0].replace('\\n', '\n').replace('\\p', '|').replace('\\\\', '\\')
    table_id = parts[2]
    answers = [p.replace('\\n', '\n').replace('\\p', '|').replace('\\\\', '\\') for p in parts[3].split('|')]
    norm_answers = [normalize_data(ans) for ans in answers]
    all_rows = tid2rows[table_id]
    header = all_rows[0]
    rows = all_rows[1:]
```

Now, storing the data about the table_id, header, question, rows, target_column, and answers in a dictionary named data.

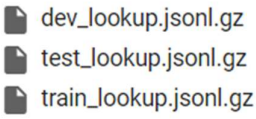
```
        file_out = splits[split]
        if len(target_columns) ==0:
          pass
        elif len(target_columns) >1:
          pass
        else:
          data = dict()
          data['id'] = id
          data['table_id'] = table_id
          data['question'] = query
          data['header'] = header
          data['target_column'] = list(target_columns)[0]
          answers = list(matched_answers)
          answers.sort()
          data['answers'] = answers
          data['rows'] = rows
```

Now, storing the data in the train, dev and test files.

```
            file_out.write(json.dumps(data)+'\n')
```

```
splits ={}
for split in ['train', 'dev' ,'test']:
  splits[split] = write_file(os.path.join("./", f'{split}_lookup.jsonl.gz'))
```

Finally, the data looks like this:




```
{"id": "nt-7", "table_id": "csv/204-csv/341.csv", "question": "which is deeper, lake tuz or lake palas tuzla?", "header": ["Name in English", "Name in Turkish", "Area (km2)", "Depth", "Location (districts and/or provinces)"], "target_column": 0, "answers": ["Lake Palas Tuzla"], "rows": [["Lake Van", "Van G\u00f6l\u00fc", "3755 km2", "451 m", "Van, Bitlis"], ["Lake Tuz", "Tuz G\u00f6l\u00fc", "1500 km2", "2 m", "Aksaray, Ankara, Konya"], ["Lake Bey\u015fehir", "Bey\u015fehir G\u00f6l\u00fc", "656 km2", "10 m", "Bey\u015fehir in Konya, Isparta"], ["Lake E\u011firdir", "E\u011firdir G\u00f6l\u00fc", "482 km2", "", "Isparta"], ["Lake \u0130znik", "\u0130znik G\u00f6l\u00fc", "308 km2", "", "\u0130znik in Bursa, Yalova"], ["Lake Burdur", "Burdur G\u00f6l\u00fc", "200 km2", "", "Burdur, Isparta"], ["Lake Manyas", "Manyas G\u00f6l\u00fc", "166 km2", "", "Bal\u0131kesir"], ["Lake Ac\u0131g\u00f6l", "Ac\u0131g\u00f6l", "153 km2", "", "Denizli, Afyonkarahisar"], ["Lake Uluabat", "Uluabat G\u00f6l\u00fc", "134 km2", "1\u20132 m", "Bursa"], ["Lake \u00c7\u0131ld\u0131r", "\u00c7\u0131ld\u0131r G\u00f6l\u00fc", "115 km2", "", "Ardahan, Kars"], ["Lake Palas Tuzla", "Palas Tuzla G\u00f6l\u00fc", "106 km2", "15 m", "Palas/Kayseri"], ["Lake Ak\u015fehir", "Ak\u015fehir G\u00f6l\u00fc", "105 km2", "", "Ak\u015fehir in Konya, Afyonkarahisar"], ["Lake Eber", "Eber G\u00f6l\u00fc", "104 km2", "", "Afyonkarahisar"], ["Lake Er\u00e7ek", "Er\u00e7ek G\u00f6l\u00fc", "98 km2", "", "Van"], ["Lake Hazar", "Hazar G\u00f6l\u00fc", "86 km2", "", "Elaz\u0131\u011f"], ["Lake Bafa", "Bafa G\u00f6l\u00fc", "60 km2", "", "Ayd\u0131n, Mu\u011fla"], ["Lake K\u00f6yce\u011fiz", "K\u00f6yce\u011fiz G\u00f6l\u00fc", "52 km2", "", "K\u00f6yce\u011fiz in Mu\u011fla"], ["Lake I\u015f\u0131kl\u0131", "I\u015f\u0131kl\u0131 G\u00f6l\u00fc", "49 km2", "", "Denizli"], ["Lake Nazik", "Nazik G\u00f6l\u00fc", "48 km2", "", "Bitlis"], ["Lake Sapanca", "Sapanca G\u00f6l\u00fc", "47 km2", "", "Sakarya Province"], ["Lake Salda", "Salda G\u00f6l\u00fc", "45 km2", "184 m", "Burdur"], ["Lake Yay", "Yay G\u00f6l\u00fc", "37 km2", "", "Kayseri"], ["Lake Akyatan", "Akyatan G\u00f6l\u00fc", "35 km2", "", "Adana"], ["Lake Bal\u0131k", "Bal\u0131k G\u00f6l\u00fc", "34 km2", "", "Do\u011fubeyaz\u0131t in A\u011fr\u0131"], ["Lake Marmara", "Marmara G\u00f6l\u00fc", "34 km2", "", "Salihli, G\u00f6lmarmara in Manisa"], ["Lake \u00c7\u00f6l", "\u00c7\u00f6l G\u00f6l\u00fc", "32 km2", "", "Ankara"], ["Lake Durusu (Lake Terkos)", "Durusu G\u00f6l\u00fc", "25 km2", "", "\u0130stanbul"], ["Lake Karine", "Karine G\u00f6l\u00fc", "24 km2", "", ""], ["Lake Tuzla", "Tuzla G\u00f6l\u00fc", "23 km2", "", "Tuzla"], ["Lake K\u00fc\u00e7\u00fck\u00e7ekmece", "K\u00fc\u00e7\u00fck\u00e7ekmece G\u00f6l\u00fc", "16 km2", "", "K\u00fc\u00e7\u00fck\u00e7ekmece, \u0130stanbul"], ["Lake Yara\u015fl\u0131", "Yara\u015fl\u0131 G\u00f6l\u00fc", "16 km2", "", "Burdur"], ["Lake Ha\u00e7l\u0131", "Ha\u00e7l\u0131 G\u00f6l\u00fc", "16 km2", "", "Mu\u015f"], ["Lake Seyfe", "Seyfe G\u00f6l\u00fc", "15 km2", "", "K\u0131r\u015fehir"], ["Lake Akyayan", "Akyayan G\u00f6l\u00fc", "15 km2", "", ""], ["Lake Hozapin", "Hozapin G\u00f6l\u00fc", "14 km2", "", ""], ["Lake Arin", "Arin G\u00f6l\u00fc", "13 km2", "", ""], ["Lake Nemrut", "Nemrut G\u00f6l\u00fc", "12 km2", "", "Bitlis Province"], ["Lake Bal\u0131k", "Bal\u0131k G\u00f6l\u00fc", "12 km2", "", ""], ["Lake B\u00fcy\u00fck\u00e7ekmece", "B\u00fcy\u00fck\u00e7ekmece G\u00f6l\u00fc", "11 km2", "", "B\u00fcy\u00fck\u00e7ekmece, Istanbul"], ["Lake Boluk", "Boluk G\u00f6l\u00fc", "11 km2", "", ""], ["Lake Akdo\u011fan", "Akdo\u011fan G\u00f6l\u00fc", "11 km2", "", ""], ["Lake \u00c7avu\u015flu", "\u00c7avu\u015flu G\u00f6l\u00fc", "9 km2", "", ""], ["Lake D\u00fcden", "D\u00fcden G\u00f6l\u00fc", "8 km2", "", ""], ["Lake Gala", "Gala G\u00f6l\u00fc", "8 km2", "", "Edirne"], ["Lake Karata\u015f", "Karata\u015f G\u00f6l\u00fc", "6 km2", "", ""], ["Lake Mogan", "Mogan G\u00f6l\u00fc", "6 km2", "", "Ankara"], ["Paradeniz", "Paradeniz", "4 km2", "", "Mersin"], ["Lake Eymir", "Eymir G\u00f6l\u00fc", "1.8 km2", "", "Ankara"], ["Lake Abant", "Abant G\u00f6l\u00fc", "1.28 km2", "18 m", "Bolu"], ["Lake G\u00f61c\u00fck", "G\u00f61c\u00fck G\u00f61\u00fc", "1 km2", "", "\u0130zmir"]]}
```


2. <u>BM25 for ranking:</u>

For rank_bm25 we are using the BM250kapi library.

```
[ ]  import re
     from rank_bm25 import BM25Okapi
```

Pre-processing the tables so that BM25 can be applied: Preprocessing includes the lowering and merging all the rows and header data of tables into one list make list for each table and removing the punctuations from it.

```
def pre_process(path):
  di = {}
  punc_pattern = r"[!\"#\$%&\'\(\)\*\+,-\./:;<=>\?@\[\\\]\^_`{\|}~]"
  with read_file(path) as fp:

    for n1,line in enumerate(fp):
      data = json.loads(line)
      for k,v in data.items():
        qid = k
        header = v[0]
        rows = v[1:]
        # print(qid,header,rows)
        header1 = []
        for h in header:
          res = re.sub(punc_pattern,' ',h)
          res = re.sub("\s+",' ',res)
          header1.extend(res.lower().split())

        rows1 = []
        for i in rows:
          for j in i:
            res = re.sub(punc_pattern,' ',j)
            res = re.sub("\s+",' ',res)
            rows1.extend(res.lower().split())

        header1.extend(rows1)
        # print(header1)
        di[k] = header1
    return di
```

Preprocessing the query: processing the query as same as the tables so that they both can be used to calculate the score.

```
[ ] def preprocess_query(query):
      punc_pattern = r"[!\"#\$%&\'\(\)\*\+,-\./:;<=>\?@\[\\\]\^_`{\|}~]"
      res = re.sub(punc_pattern,' ',query)
      res = re.sub("\s+",' ',res)
      tokenized_query = res.lower().split()
      return tokenized_query
```

Ranking the documents: Using the inbuilt BM25 for calculate the scores

```
def ranking_docs(query,di):
  tokenized_query = preprocess_query(query)
  bm25 = BM25Okapi(di.values())
  scores = bm25.get_scores(tokenized_query)
  ranked_documents = dict(sorted(zip(di.keys(), scores), key=lambda x: x[1], reverse=True))
  return ranked_documents
```

Getting the tables based on scores from the given query: Using all the above functions for retrieving the top 300 tables.

```
def BM25(query,top=300,paths=[path1,path2,path3]):
  di1 = pre_process(paths[0])
  di2 = pre_process(paths[1])
  di3 = pre_process(paths[2])
  ranked_doc1 = ranking_docs(query,di1)
  ranked_doc2 = ranking_docs(query,di2)
  ranked_doc3 = ranking_docs(query,di3)
  result = {**ranked_doc1,**ranked_doc2,**ranked_doc3}
  final_result = dict(list(sorted(result.items(), key=lambda x: x[1], reverse=True))[:top])

  tables = {}
  for i in paths:
    with read_file(i) as fp:
      for n1,line in enumerate(fp):
        data = json.loads(line)
        for k,v in data.items():
          if(k in final_result.keys()):
            tables[k] = v

  return tables
```

Testing on a query

```
[13] query = "what is the temperature?"
```

```
tables = BM25(query)
```

+ Code + Text

```
tables
```

```
{'auto-18': [['Term',
   'Type',
   'POS (most common)',
   'WN2.0 Name',
   'WordNet SenseKey',
   'WN2.0 Synset',
   'WordNet gloss',
   'WordNet example usage'],
  ['ability',
   '?',
   'n',
   'ability_n1',
   'ability%1:07:00::',
   '104904666',
   'the quality of being able to perform; a quality that permits or facilitates achievement or accomplishment',
   ''],
  ['able',
```

3. Main Row column Intersection Models:

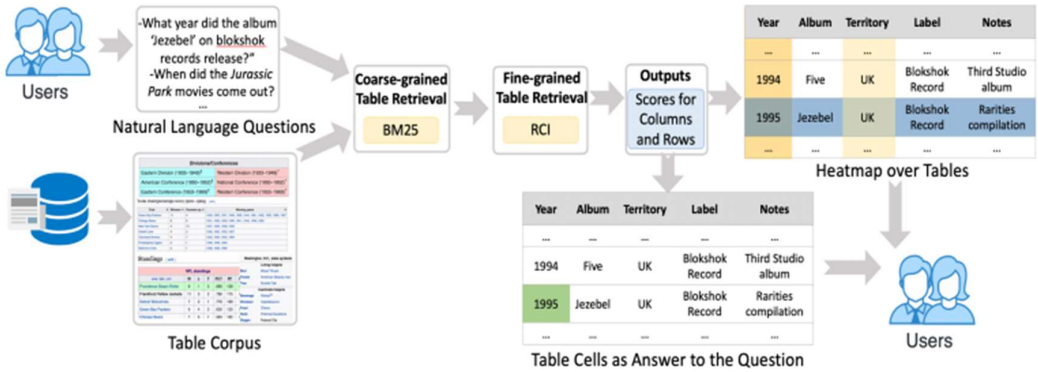


Figure: Showing the overview of the end-to-end architecture of CLTR [1]

Two models

1. Interaction Model

2. Representation model.

1. Interaction model

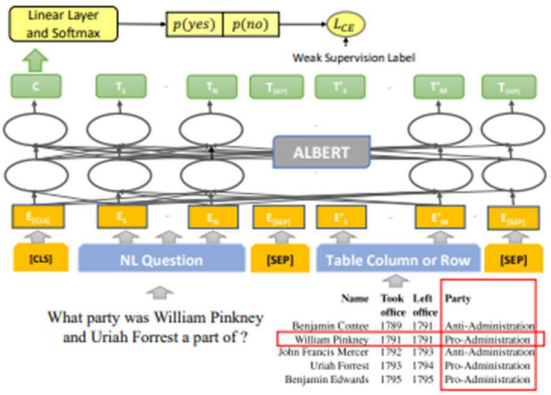


Figure : Showing the Row-column Intersection Model [1]

Model:

```
class RCI_interaction(nn.Module):
    def __init__(self, l):
        super(RCI_interaction, self).__init__()
        albert_config = AlbertConfig(hidden_size=512)
        self.albert = AlbertModel(albert_config)
        self.net1 = nn.Linear(512, 2)
        self.leaky_relu = nn.LeakyReLU(l)

    def forward(self, input_ids, attention_mask):
        albert_output = self.albert(input_ids, attention_mask=attention_mask)[0]
        output = torch.softmax(self.net1(albert_output[:, 0]), dim=1)
        return output
```

The RCI interaction model uses the sequence representation which is later appended to the question with standard [CLS] and [SEP] tokens to delimit the two sequences. This sequence pair is fed into a transformer encoder, ALBERT. [1]

The final hidden state for the [CLS] token is used in a linear layer followed by a softmax to classify if the column or row contains the answer or not. Each row and column are assigned a probability of containing the answer. The RCI model outputs the top-ranked cell as the intersection of the most probable row and the most probable column. [1]

Getting the data encodings from the database tables:

```
def get_data(file_name, tables_cnt):
    # tensor1 = torch.Tensor()
    # tensor2 = torch.Tensor()
    data = {'row_labels': [], 'row_input':[], 'col_labels': [], 'col_input':[], 'row_queries': [], 'col_queries': []}
    tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
    max_seq_length = 128
    batch_size = 16
    with open(file_name, "r") as file:
        # Read each line and parse it as JSON
        i=0
        for line in file:
            if(i==tables_cnt): break
            table = json.loads(line)
            header = table['header']
            rows = table['rows']
            target_label = table['target_column']
            query = table['question']
            answers = table['answers']
            col_queries = [query]*len(header)
            row_queries = [query]*len(rows)
            colsRepresentation = getColRepresentation(header, rows)
            rowsRepresentation = getRowRepresentation(header, rows)

            col_labels = [[0.9, 0.1]]*len(header)
            row_labels = [[0.9, 0.1]]*len(rows)
            col_labels[target_label] = [0.1,0.9]
            for j in range(len(rows)):
                if(rows[j][target_label] in answers):
                    row_labels[j] = [0.1, 0.9]
            data['col_input'].extend(colsRepresentation)
            data['row_input'].extend(rowsRepresentation)
            data['col_labels'].extend(col_labels)
            data['row_labels'].extend(row_labels)
            data['col_queries'].extend(col_queries)
            data['row_queries'].extend(row_queries)
            i+=1
```

Getting the data encodings from the required file:

```
def get_data_from_table(header, rows, query):
    data = {'row_input':[], 'col_input':[], 'row_queries': [], 'col_queries': []}
    tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
    max_seq_length = 128
    batch_size = 16
    col_queries = [query]*len(header)
    row_queries = [query]*len(rows)
    colsRepresentation = getColRepresentation(header, rows)
    rowsRepresentation = getRowRepresentation(header, rows)
    data['col_input'].extend(colsRepresentation)
    data['row_input'].extend(rowsRepresentation)
    data['col_queries'].extend(col_queries)
    data['row_queries'].extend(row_queries)
    col_encoding = tokenizer(data['col_input'], data['col_queries'],return_tensors='pt', padding=True, truncation=True, max_length=max_seq_length)
    row_encoding = tokenizer(data['row_input'], data['row_queries'],return_tensors='pt', padding=True, truncation=True, max_length=max_seq_length)
    return row_encoding, col_encoding
```

Now make the column-wise representation as required to input it in the model.

```
def getColRepresentation(header, rows):
    colRepresentation = []
    cols = [[str(h)] for h in header]
    for row in rows:
        for ci, cell in enumerate(row):
            if cell:  # for sparse table use case
                cols[ci].append(str(cell))
    for col in cols:
        col_rep = ' * '.join(col)
        colRepresentation.append(col_rep)
    return colRepresentation
```

Now make the row-wise representation as required to input it in the model.

```
def getRowRepresentation(header, rows):
    rowRepresentation = []
    for row in rows:
        row_rep = ' * '.join([h + ' : ' + str(c) for h, c in zip(header, row) if c])  # for sparse table use case
        rowRepresentation.append(row_rep)
    return rowRepresentation
```

Now make use of column and row representations to tokenize the data and get the required encodings and label corresponding to the representations.

```
            colsRepresentation = getColRepresentation(header, rows)
            rowsRepresentation = getRowRepresentation(header, rows)

            col_labels = [[0.9, 0.1]]*len(header)
            row_labels = [[0.9, 0.1]]*len(rows)
            col_labels[target_label] = [0.1,0.9]
            for j in range(len(rows)):
                if(rows[j][target_label] in answers):
                    row_labels[j] = [0.1, 0.9]
            data['col_input'].extend(colsRepresentation)
            data['row_input'].extend(rowsRepresentation)
            data['col_labels'].extend(col_labels)
            data['row_labels'].extend(row_labels)
            data['col_queries'].extend(col_queries)
            data['row_queries'].extend(row_queries)
            i+=1
    file.close()
    col_train_encoding = tokenizer(data['col_input'], data['col_queries'],return_tensors='pt', padding=True, truncation=True, max_length=max_seq_length)
    row_train_encoding = tokenizer(data['row_input'], data['row_queries'],return_tensors='pt', padding=True, truncation=True, max_length=max_seq_length)

    return row_train_encoding, torch.tensor(data['row_labels']).float(), col_train_encoding, torch.tensor(data['col_labels']).float()
```

Then getting the tensors of the encodings above.

```
row_train_encoding, row_labels, col_train_encodings, col_labels = get_data("train_lookup.jsonl", 1)

print(row_train_encoding, row_labels, col_train_encodings, col_labels)
```

```
{'input_ids': tensor([[    2,    13,    45,    19,    14,  1637, 15429,    13,    45,   743,
         15429,  1637,    13,    45,    13,    15,    14,  1637, 14985,   807,
            13,    45,   697, 29230,  1637,    13,    45,  3690,    19,  1637,
          1617,    16,  3744,  2940,    13,    45,   295,     3,    19,    56,
         15429,   630,    14,   697, 29230,  3744,    19,   356,    60,     3,
             0,     0,     0],
        [    2,    13,    45,    19,    14,  1637, 15429,    13,    45,   775,
         15429,  1637,    13,    45,    13,    15,    14,  1637, 14985,   807,
            13,    45,   697, 29230,  1637,    13,    45,  3690,    19,  1637,
          1617,    16,  3744,  2940,    13,    45,   356,     3,    19,    56,
         15429,   630,    14,   697, 29230,  3744,    19,   356,    60,     3,
             0,     0,     0],
```

Function to train the model.

```
def train(model, train_data, optimiser, criterion, batch_size, train_labels):
    train_data.to(device)
    train_labels.to(device)
    model.train()
    Loading... = 0
    train_correct = 0
    i=0
    start = 0
    while(start < len(train_labels)):
        end = start + batch_size
        if(end > len(train_labels)):
            end = len(train_labels)
        # batch_size = len(data['input_ids'].to(device))
        optimiser.zero_grad()
        output = model(train_data['input_ids'][start:end, :], train_data['attention_mask'][start:end, :])
        # print(output)
        # print(torch.tensor(train_labels[start:end]))
        # target = torch.tensor(train_logits[i*batch_size:(i+1)*batch_size, :], dtype=torch.float32).to(device)
        loss = criterion(output, torch.softmax(torch.tensor(train_labels)[start:end, :], dim=1, dtype = torch.float32).to(device))
        loss.backward()
        optimiser.step()
        train_loss += loss.item()
        pred = output.argmax(dim=1, keepdim=True).to(device)
        actual = train_labels[start:end].argmax(dim=1, keepdim=True).to(device)
        train_correct += pred.eq(actual.view_as(pred)).sum().item()
        i +=1
        start = end
    train_loss /= len(train_labels)
    train_acc = train_correct / len(train_labels)
    return train_loss, train_acc
```

```python
def train_model(model, train_data, train_labels, optimiser=None, criterion=None, batch_size=16, lr=1e-3, epoch=2, checkpoint_dir=None,
    if(optimiser is None): optimiser = torch.optim.Adam(model.parameters(), lr)
    if(criterion is None): criterion = nn.CrossEntropyLoss()
    print("Training...........")
    checkpoint_path = checkpoint_dir + checkpoint_file

    if resume_from_checkpoint:
        if checkpoint_path is None or not os.path.exists(checkpoint_path):
            start_epoch = 0
            print("No saved checkpoints to resume")
        else:
            print("Checkpoint accessing...........")
            checkpoint = torch.load(checkpoint_path, map_location=torch.device('cpu'))
            model.load_state_dict(checkpoint['model_state_dict'])
            optimiser.load_state_dict(checkpoint['optimiser_state_dict'])
            # lr_scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
            start_epoch = checkpoint['epoch']
            print(f"Resuming training from epoch {start_epoch}")
    else:
        start_epoch = 0
    for i in range(start_epoch, epoch):
        train_loss, train_acc = train(model, train_data, optimiser, criterion, batch_size, train_labels)
        print(f"Epoch {i}   Train loss: {train_loss}    Train accuracy: {train_acc}")

        if checkpoint_dir is not None:
            if(not os.path.exists(checkpoint_dir)):
                os.makedirs(checkpoint_dir, exist_ok=True)
            checkpoint = {
                'epoch': epoch + 1,
                'model_state_dict': model.state_dict(),
                'optimiser_state_dict': optimiser.state_dict(),
            }
            torch.save(checkpoint, checkpoint_path)
```

Training two models for the row and column representations.

Row:

```python
row_train_data, row_train_labels, col_train_data, col_train_labels = get_data(file_name, 10000)
print(row_train_data['input_ids'].size(), row_labels.size())
print(len(row_labels))
# print(train_data['input_ids'])
# print()
# print(labels)
row_model_interaction = RCI_interaction(0.1).to(device)
optimiser = torch.optim.Adam(row_model_interaction.parameters(), lr=1e-4)
criterion = nn.CrossEntropyLoss()

train_model(model=row_model_interaction, train_data=row_train_data, train_labels=row_train_labels, optimiser=optimiser, criteric
```

```
torch.Size([24, 56]) torch.Size([8, 2])
8
Training...........
Epoch 0   Train loss: 0.04193515129723903    Train accuracy: 0.8676542010684798
Epoch 1   Train loss: 0.0415085686851092    Train accuracy: 0.8705682370082565
Epoch 2   Train loss: 0.04132189093254445    Train accuracy: 0.8744536182612919
```

Column:

```python
[ ] col_model_interaction = RCI_interaction(0.1).to(device)
    optimiser = torch.optim.Adam(col_model_interaction.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()

    train_model(model=col_model_interaction, train_data=col_train_data, train_labels=col_train_labels, optimiser=optimiser, crit
```

```
Training...........
Epoch 0   Train loss: 0.046467066804567976    Train accuracy: 0.6666666666666666
Epoch 1   Train loss: 0.04621846963564555    Train accuracy: 0.7
Epoch 2   Train loss: 0.045756292343139646    Train accuracy: 0.8333333333333334
```

Testing the model:

```
[ Run cell (Ctrl+Enter)
cell has not been executed in this session
terion, batch_size, test_labels):
    test_labels.to(device)
    model.eval()
    test_loss = 0
    test_correct = 0
    i=0
    start = 0
    logits = []
    with torch.no_grad():
        while(start < len(test_labels)):
            end = start + batch_size
            if(end > len(test_labels)):
                end = len(test_labels)
            output = model(test_data['input_ids'][start:end, :], test_data['attention_mask'][start:end, :].to(device))
            # print(output)
            # print(torch.tensor(test_labels[start:end]))
            # target = torch.tensor(test_logits[i*batch_size:(i+1)*batch_size, :], dtype=torch.float32).to(device)
            loss = criterion(output, torch.softmax(torch.tensor(test_labels)[start:end, :], dim=1, dtype = torch.float32).to(device))
            # loss.backward()
            # optimiser.step()
            test_loss += loss.item()
            pred = output.argmax(dim=1, keepdim=True).to(device)
            for out in output:
                logits.append(out[1])
            actual = test_labels[start:end].argmax(dim=1, keepdim=True).to(device)
            test_correct += pred.eq(actual.view_as(pred)).sum().item()
            i +=1
            start = end
    test_loss /= len(test_labels)
    test_acc = test_correct / len(test_labels)
    return test_loss, test_acc, logits
```

```
def test_model(model, test_data, criterion, batch_size, test_labels):
    print("Evaluating on testing data.................")
    test_loss, test_acc = test(model, test_data, criterion, batch_size, test_labels)
    print(f"Testing loss: {test_loss} Test accuracy:  {test_acc}")
```

Getting the scores for rows and columns, adding then to get the most appropriate cell.

```
[ ]  def getLogits(header, rows, query, row_model, col_model):
        criterion = nn.CrossEntropyLoss()
        batch_size = 16
        row_data, col_data = get_data_from_table(header, rows, query)
        row_labels = torch.zeros([len(row_data),2])
        col_labels = torch.zeros([len(col_data), 2])
        _, _, rowsLogits = test(row_model, row_data, criterion, batch_size, row_labels)
        _, _, colsLogits = test(col_model, col_data, criterion, batch_size, col_labels)
        return rowsLogits, colsLogits
```

```
def getScores(rowsLogits, colsLogits, top_k):
    scores = []
    for i in range(len(rowsLogits)):
        for j in range(len(colsLogits)):
            score = float(rowsLogits[i] + colsLogits[j])
            scores.append([i,j,score])
    scores.sort(key=lambda x: x[2], reverse=True)
    return scores[0:top_k]
```

```
def getQueryAnswers(query, header, rows, row_model, col_model):
    batch_size = 16
    max_seq_length=128
    rowsLogits, colsLogits = getLogits(header, rows, query, row_model, col_model)

    top_k = 3
    rci_scores = getScores(rowsLogits, colsLogits, top_k)
    # row_scores = getScores(rowsLogits, )

    return [{'row_ndx': i, 'col_ndx': j, 'confidence_score': score, 'text': rows[i][j]} for i, j, score in rci_scores]
```

Using Model for Answering:

```
[ ] header = ['Participant', 'Race', 'Date']
    rows = [['Michael', 'Runathon', 'June 10, 2020'],
            ['Mustafa', 'Runathon', 'Sept 3, 2020'],
            ['Alfio', 'Runathon', 'Jan 1, 2021'],]
    answers = getQueryAnswers('Who won the race in June?',header, rows, row_model_interaction, col_model_interaction)
    print("Predicted cells along with their confidence scores and values")
    for answer in answers:
            print(answer)

    Predicted cells along with their confidence scores and values
    {'row_ndx': 0, 'col_ndx': 0, 'confidence_score': 0.8965473175048828, 'text': 'Michael'}
    {'row_ndx': 2, 'col_ndx': 0, 'confidence_score': 0.8920146822929382, 'text': 'Alfio'}
    {'row_ndx': 0, 'col_ndx': 2, 'confidence_score': 0.8879615068435669, 'text': 'Jun 10, 2021'}
```

2. Representation model

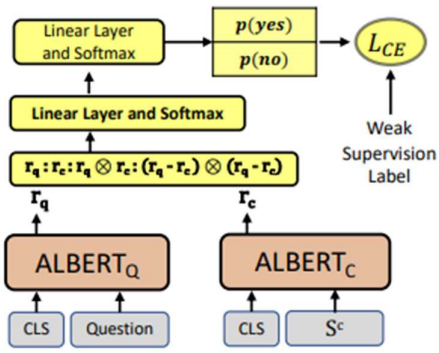


Figure 2: RCI Representation Model

Figure: Showing the Representation Model [2]

Model:

```python
# rq :rc:rq ⊗rc:(rq -rc)⊗(rq -rc)

class RCI_representation(nn.Module):
    def __init__(self, l):
        super(RCI_representation, self).__init__()
        albert_config = AlbertConfig(hidden_size=512)
        self.albert1 = AlbertModel(albert_config)
        self.albert2 = AlbertModel(albert_config)
        self.net1 = nn.Linear(512*4, 512)
        self.leaky_relu = nn.LeakyReLU(l)
        self.net2 = nn.Linear(512, 2)

        # model_output.last_hidden_state[:, 0, :]

    def forward(self, q_input_ids, q_attention_mask, c_input_ids, c_attention_mask, rc=None, get_representation=False):
        if(get_representation):
            rc = self.albert2(c_input_ids, attention_mask=c_attention_mask).last_hidden_state[:, 0, :]
            return rc
        rq = self.albert1(q_input_ids, attention_mask=q_attention_mask).last_hidden_state[:, 0, :]
        if(rc is None): rc = self.albert2(c_input_ids, attention_mask=c_attention_mask).last_hidden_state[:, 0, :]
        rqc = torch.cat([rq, rc, rq*rc, (rq-rc)**2], dim = 1)
        net1_output = self.net1(rqc)
        output = torch.softmax(self.net2(net1_output), dim=1)
        return output
```

Name	Took office	Left office	Party	Notes / Events
Benjamin Contee	1789	1791	Anti-Administration	
William Pinkney	1791	1791	Pro-Administration	resigned
John Francis Mercer	1792	1793	Anti-Administration	
Uriah Forrest	1793	1794	Pro-Administration	resigned
Benjamin Edwards	1795	1795	Pro-Administration	

⋮

What party was William Pinkney and Uriah Forrest a part of?

Answer: Pro-Administration

Considering again the example in Table 1, the first row would be represented as:

Name : Benjamin Contee | Took office : 1789 | Left office : 1791 | Party : Anti-Administration | Notes / Events : |

While the second column would have a sequence representation of:

Took office : 1789 | 1791 | 1792 | 1793 | 1795 |

Figure: Showing Table and its column and row representations fed into the model [2]

Getting the data encodings from the database tables:

```python
def get_data(file_name, tables_cnt):
    # tensor1 = torch.Tensor()
    # tensor2 = torch.Tensor()
    data = {'row_labels': [], 'row_input':[], 'col_labels': [], 'col_input':[], 'row_queries': [], 'col_queries': []}
    tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
    max_seq_length = 128
    batch_size = 16
    with open(file_name, "r") as file:
        # Read each line and parse it as JSON
        i=0
        for line in file:
            if(i==tables_cnt): break
            table = json.loads(line)
            header = table['header']
            rows = table['rows']
            target_label = table['target_column']
            query = table['question']
            answers = table['answers']
            col_queries = [query]*len(header)
            row_queries = [query]*len(rows)
            colsRepresentation = getColRepresentation(header, rows)
            rowsRepresentation = getRowRepresentation(header, rows)

            col_labels = [[0.9, 0.1]]*len(header)
            row_labels = [[0.9, 0.1]]*len(rows)
            col_labels[target_label] = [0.1,0.9]
            for j in range(len(rows)):
                if(rows[j][target_label] in answers):
                    row_labels[j] = [0.1, 0.9]
            data['col_input'].extend(colsRepresentation)
            data['row_input'].extend(rowsRepresentation)
            data['col_labels'].extend(col_labels)
            data['row_labels'].extend(row_labels)
            data['col_queries'].extend(col_queries)
            data['row_queries'].extend(row_queries)
            i+=1
    file.close()
    col_train_encoding = tokenizer(data['col_input'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
    row_train_encoding = tokenizer(data['row_input'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
    query_row_train_encoding = tokenizer(data['row_queries'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
    query_col_train_encoding = tokenizer(data['col_queries'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
    return row_train_encoding, torch.tensor(data['row_labels']).float(), col_train_encoding, torch.tensor(data['col_labels']).float(), query_row_train_encoding, query_col_train_encoding
```

Getting the data encodings from the required file:

```
def get_data_from_table(header, rows, query):
    data = {'row_input':[], 'col_input':[], 'row_queries': [], 'col_queries': []}
    tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
    max_seq_length = 128
    batch_size = 16
    col_queries = [query]*len(header)
    row_queries = [query]*len(rows)
    colsRepresentation = getColRepresentation(header, rows)
    rowsRepresentation = getRowRepresentation(header, rows)
    data['col_input'].extend(colsRepresentation)
    data['row_input'].extend(rowsRepresentation)
    data['col_queries'].extend(col_queries)
    data['row_queries'].extend(row_queries)
    col_encoding = tokenizer(data['col_input'], return_tensors='pt', padding=True, max_length=max_seq_length)
    row_encoding = tokenizer(data['row_input'], return_tensors='pt', padding=True, max_length=max_seq_length)
    query_row_encoding = tokenizer(data['row_queries'], return_tensors='pt', padding=True, max_length=max_seq_length)
    query_col_encoding = tokenizer(data['col_queries'], return_tensors='pt', padding=True, max_length=max_seq_length)
    return row_encoding, col_encoding, query_row_encoding, query_col_encoding
```

Now make the column-wise representation as required to input it in the model.

```
def getColRepresentation(header, rows):
    colRepresentation = []
    cols = [[str(h)] for h in header]
    for row in rows:
        for ci, cell in enumerate(row):
            if cell:  # for sparse table use case
                cols[ci].append(str(cell))
    for col in cols:
        col_rep = ' * '.join(col)
        colRepresentation.append(col_rep)
    return colRepresentation
```

Now make the row-wise representation as required to input it in the model.

```
def getRowRepresentation(header, rows):
    rowRepresentation = []
    for row in rows:
        row_rep = ' * '.join([h + ' : ' + str(c) for h, c in zip(header, row) if c])  # for sparse table use case
        rowRepresentation.append(row_rep)
    return rowRepresentation
```

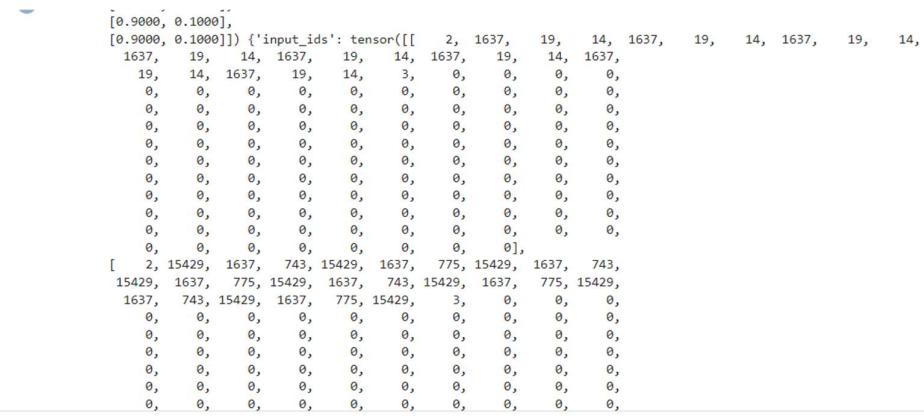
Now make use of column and row representations to tokenize the data and get the required encodings and label corresponding to the representations.

```
                row_labels[j] = [0.1, 0.9]
        data['col_input'].extend(colsRepresentation)
        data['row_input'].extend(rowsRepresentation)
        data['col_labels'].extend(col_labels)
        data['row_labels'].extend(row_labels)
        data['col_queries'].extend(col_queries)
        data['row_queries'].extend(row_queries)
        i+=1
file.close()
col_train_encoding = tokenizer(data['col_input'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
row_train_encoding = tokenizer(data['row_input'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
query_row_train_encoding = tokenizer(data['row_queries'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
query_col_train_encoding = tokenizer(data['col_queries'], return_tensors='pt', padding='max_length', max_length=max_seq_length)
return row_train_encoding, torch.tensor(data['row_labels']).float(), col_train_encoding, torch.tensor(data['col_labels']).float(), query_
```

Then getting the tensors of the encodings above.

```
row_train_encoding, row_labels, col_train_encodings, col_labels, query_row_train_encoding, query_col_train_encoding = get_data(file_name, 1)

print(row_train_encoding, row_labels, col_train_encodings, col_labels, query_row_train_encoding, query_col_train_encoding)
```

```
spiece.model: 100% 760k/760k [00:00<00:00, 7.60MB/s]
tokenizer.json: 100% 1.31M/1.31M [00:00<00:00, 5.27MB/s]
config.json: 100% 684/684 [00:00<00:00, 10.4kB/s]
{'input_ids': tensor([[ 2, 13, 45,  ...,  0,  0,  0],
        [ 2, 13, 45,  ...,  0,  0,  0],
        [ 2, 13, 45,  ...,  0,  0,  0],
        ...,
        [ 2, 13, 45,  ...,  0,  0,  0],
        [ 2, 13, 45,  ...,  0,  0,  0],
        [ 2, 13, 45,  ...,  0,  0,  0]]), 'token_type_ids': tensor([[0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
```



```
        [0.9000, 0.1000],
        [0.9000, 0.1000]]) {'input_ids': tensor([[    2,  1637,    19,    14,  1637,    19,    14,  1637,    19,    14,
          1637,    19,    14,  1637,    19,    14,  1637,    19,    14,  1637,
            19,    14,  1637,    19,    14,     3,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
        [    2, 15429,  1637,   743, 15429,  1637,   775, 15429,  1637,   743,
         15429,  1637,   775, 15429,  1637,   743, 15429,  1637,   775, 15429,
          1637,   743, 15429,  1637,   775, 15429,     3,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
```

Function to train the model.

```
def train(model, train_data, query_train_data, optimiser, criterion, batch_size, train_labels):
    train_data.to(device)
    query_train_data.to(device)
    train_labels.to(device)
    model.train()
    train_loss = 0
    train_correct = 0
    i=0
    start = 0
    while(start < len(train_labels)):
        end = start + batch_size
        if(end > len(train_labels)):
            end = len(train_labels)
        # batch_size = len(data['input_ids'].to(device))
        optimiser.zero_grad()
        output = model(query_train_data['input_ids'][start:end, :], query_train_data['attention_mask'][start:end, :
        # print(output)
        # print(torch.tensor(train_labels[start:end]))
        # target = torch.tensor(train_logits[i*batch_size:(i+1)*batch_size, :], dtype=torch.float32).to(device)
        loss = criterion(output, torch.softmax(torch.tensor(train_labels)[start:end, :], dim=1, dtype = torch.float
        loss.backward()
        optimiser.step()
        train_loss += loss.item()
        pred = output.argmax(dim=1, keepdim=True).to(device)
        actual = train_labels[start:end].argmax(dim=1, keepdim=True).to(device)
        train_correct += pred.eq(actual.view_as(pred)).sum().item()
        i +=1
        start = end
    train_loss /= len(train_labels)
    train_acc = train_correct / len(train_labels)
    return train_loss, train_acc
```

```
[ ] def train_model(model, train_data, query_train_data, train_labels, optimiser=None, criterion=None, batch_size=16, lr=1e-3, e
        if(optimiser is None): optimiser = torch.optim.Adam(model.parameters(), lr)
        if(criterion is None): criterion = nn.CrossEntropyLoss()
        print("Training..........")
        checkpoint_path = checkpoint_dir + checkpoint_file

        if resume_from_checkpoint:
            if checkpoint_path is None or not os.path.exists(checkpoint_path):
                start_epoch = 0
                print("No saved checkpoints to resume")
            else:
                print("Checkpoint accessing..........")
                checkpoint = torch.load(checkpoint_path, map_location=torch.device('cpu'))
                model.load_state_dict(checkpoint['model_state_dict'])
                optimiser.load_state_dict(checkpoint['optimiser_state_dict'])
                # lr_scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
                start_epoch = checkpoint['epoch']
                print(f"Resuming training from epoch {start_epoch}")
        else:
            start_epoch = 0
        for i in range(start_epoch, epoch):
            train_loss, train_acc = train(model, train_data, query_train_data, optimiser, criterion, batch_size, train_labels)
            print(f"Epoch {i}   Train loss: {train_loss}    Train accuracy: {train_acc}")

            if checkpoint_dir is not None:
                if(not os.path.exists(checkpoint_dir)):
                    os.makedirs(checkpoint_dir, exist_ok=True)
                checkpoint = {
                    'epoch': epoch + 1,
                    'model_state_dict': model.state_dict(),
                    'optimiser_state_dict': optimiser.state_dict(),
                }
                torch.save(checkpoint, checkpoint_path)
```

Training two models for the row and column representations.

Row:

```
[ ] row_train_data, row_train_labels, col_train_data, col_train_labels, query_row_train_data, query_col_train_data = get_data
    row_model_representation = RCI_representation(0.1).to(device)
    optimiser = torch.optim.Adam(row_model_representation.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()


    train_model(model=row_model_representation, train_data=row_train_data, query_train_data=query_row_train_data, train_label
```

```
spiece.model: 100% 760k/760k [00:00<00:00, 7.77MB/s]
tokenizer.json: 100% 1.31M/1.31M [00:00<00:00, 3.96MB/s]
config.json: 100% 684/684 [00:00<00:00, 43.9kB/s]
Training..........
Epoch 0   Train loss: 0.042900774627923965    Train accuracy: 0.725
Epoch 1   Train loss: 0.041852790862321854    Train accuracy: 0.875
Epoch 2   Train loss: 0.04145502373576164    Train accuracy: 0.875
```

Column:

```
col_model_representation = RCI_representation(0.1).to(device)
optimiser = torch.optim.Adam(col_model_representation.parameters(), lr=1e-4)
criterion = nn.CrossEntropyLoss()


train_model(model=col_model_representation, train_data=col_train_data, query_train_data=query_col_train_data, train_labels=
```

```
Training..........
Epoch 0   Train loss: 0.042678617491827615    Train accuracy: 0.7161359956826767
Epoch 1   Train loss: 0.042406844211409556    Train accuracy: 0.7609282245008095
Epoch 2   Train loss: 0.040756292343139646    Train accuracy: 0.8253334446759423
```

Testing the model:

```
def test(model, test_data, query_test_data, criterion, batch_size, test_labels):
    test_data.to(device)
    query_test_data.to(device)
    test_labels.to(device)
    model.eval()
    test_loss = 0
    test_correct = 0
    i=0
    start = 0
    logits = []
    with torch.no_grad():
        while(start < len(test_labels)):
            end = start + batch_size
            if(end > len(test_labels)):
                end = len(test_labels)
            output = model(query_test_data['input_ids'][start:end, :], query_test_data['attention_mask'][start:end, :], test_data['inp
            # print(output)
            # print(torch.tensor(test_labels[start:end]))
            # target = torch.tensor(test_logits[i*batch_size:(i+1)*batch_size, :], dtype=torch.float32).to(device)
            loss = criterion(output, torch.softmax(torch.tensor(test_labels)[start:end, :], dim=1, dtype = torch.float32).to(device))
            # loss.backward()
            # optimiser.step()
            test_loss += loss.item()
            pred = output.argmax(dim=1, keepdim=True).to(device)
            for out in output:
                logits.append(out[1])
            actual = test_labels[start:end].argmax(dim=1, keepdim=True).to(device)
            test_correct += pred.eq(actual.view_as(pred)).sum().item()
            i +=1
            start = end
    test_loss /= len(test_labels)
    test_acc = test_correct / len(test_labels)
    return test_loss, test_acc, logits
```

```
def test_model(model, test_data, query_test_data, criterion, batch_size, test_labels):
    print("Evaluating on testing data.................")
    test_loss, test_acc = test(model, test_data, query_test_data, criterion, batch_size, test_labels)
    print(f"Testing loss: {test_loss} Test accuracy:  {test_acc}")
```

Testing for:

Row:

```
[ ] file_name = file_path + "test_lookup.jsonl"
    row_test_data, row_test_labels, col_test_data, col_test_labels, query_row_test_data, query_col_test_data = get_data(file_name, 3)

    criterion = nn.CrossEntropyLoss()

    test_model(model = row_model_representation, test_data = row_test_data, query_test_data=query_row_test_data, criterion = criterion
```

```
Evaluating on testing data.................
Test loss: 0.04199714462910725    Test accuracy: 0.86923987753267821
```

Column:

```
[ ]
    test_model(model = col_model_representation, test_data = col_test_data, query_test_data=query_col_test_data, criterio
```

```
Evaluating on testing data.................
Test loss: 0.04129876543210987    Test accuracy: 0.8227345678901234
```

Getting the scores for rows and columns, adding then to get the most appropriate cell.

```
[ ]  def getLogits(header, rows, query, row_model, col_model):
         criterion = nn.CrossEntropyLoss()
         batch_size = 16
         row_data, col_data = get_data_from_table(header, rows, query)
         row_labels = torch.zeros([len(row_data),2])
         col_labels = torch.zeros([len(col_data), 2])
         _, _, rowsLogits = test(row_model, row_data, criterion, batch_size, row_labels)
         _, _, colsLogits = test(col_model, col_data, criterion, batch_size, col_labels)
         return rowsLogits, colsLogits
```

```
     def getScores(rowsLogits, colsLogits, top_k):
         scores = []
         for i in range(len(rowsLogits)):
             for j in range(len(colsLogits)):
                 score = float(rowsLogits[i] + colsLogits[j])
                 scores.append([i,j,score])
         scores.sort(key=lambda x: x[2], reverse=True)
         return scores[0:top_k]
```

```
    def getQueryAnswers(query, header, rows, row_model, col_model):
        batch_size = 16
        max_seq_length=128
        rowsLogits, colsLogits = getLogits(header, rows, query, row_model, col_model)

        top_k = 3
        rci_scores = getScores(rowsLogits, colsLogits, top_k)
        # row_scores = getScores(rowsLogits, )

        return [{'row_ndx': i, 'col_ndx': j, 'confidence_score': score, 'text': rows[i][j]} for i, j, score in rci_scores]
```

Using Model for Answering:

```
[ ]  header = ['Participant', 'Race', 'Date']
     rows = [['Michael', 'Runathon', 'June 10, 2020'],
             ['Mustafa', 'Runathon', 'Sept 3, 2020'],
             ['Alfio', 'Runathon', 'Jan 1, 2021'],]
     answers = getQueryAnswers('Who won the race in June?',header, rows, row_model_interaction, col_model_interaction)
     print("Predicted cells along with their confidence scores and values")
     for answer in answers:
             print(answer)
```

```
Predicted cells along with their confidence scores and values
{'row_ndx': 0, 'col_ndx': 0, 'confidence_score': 0.8965473175048828, 'text': 'Michael'}
{'row_ndx': 2, 'col_ndx': 0, 'confidence_score': 0.8920146822929382, 'text': 'Alfio'}
{'row_ndx': 0, 'col_ndx': 2, 'confidence_score': 0.8879615068435669, 'text': 'Jun 10, 2021'}
```

By combining BM25 for initial table ranking and RCI (Row column intersection) fine-tuning, we can create a powerful system that retrieves highly relevant information from a corpus of tables in response to natural language queries.

This approach helps balance the need for relevance ranking with the ability to extract specific data points needed on user-specified conditions.

Results:

1. RCI Models Evaluation:

Evaluation Metrics:

1. **Hit@1:** simply measures the fraction of questions that are correctly answered by the first cell prediction
2. **MRR:** is computed by finding the rank of the first correct cell prediction for each question and averaging its reciprocal.

For Interaction Model:

Fetching the true and predicted answers:

```
import time
rci_answers = []
true_answers = []
tapas_answers = []
for table in tables[:25]:
    true_answers.extend(table['answers'])
    start_time = time.time()
    answers = getQueryAnswers(table['question'], table['header'], table['rows'], row_model_interaction, col_model_interaction)
    end_time = time.time()
    rci_answer = []
    for answer in answers:
        rci_answer.append(answer['text'])
    rci_answers.append(rci_answer)

total_time = end_time-start_time
print(rci_answers)
print(true_answers)
print("time per query is ", total_time/(len(true_answers)))
```

```
[['spring equinox', 'fall equinox', 'summer solstice', 'winter solstice', 'The'], ['period of night', 'period of night',
['Spring equinox', 'spring equinox', 'midrange', 'midrange', 'midrange', 'midrange', 'midrange', 'Midrange', 'Fall Equin
time per query is  0.4549198436737061
```

For Interaction Model - Hit @1:

```
hit_at_1 = 0
for i in range(len(true_answers)):
    if(true_answers[i] == rci_answers[i][0]): hit_at_1 +=1
hit_at_1/= len(true_answers)
print(hit_at_1)
```

```
0.52
```

For Interaction Model -MRR:

```
[ ]  reciprocal_rank_sum = 0
     for i in range(len(true_answers)):
         for j in range(len(rci_answers[i])):
             if(true_answers[i] == rci_answers[i][j]):
                 print(1/(j+1))
                 reciprocal_rank_sum += 1/(j+1)
                 break
     mrr = reciprocal_rank_sum/len(true_answers)
     print(mrr)
```

```
0.54
```

For Representation Model:

Fetching the true and predicted answers:

```
[ ]  import time
     rci_answers = []
     true_answers = []
     tapas_answers = []
     for table in tables[:25]:
         true_answers.extend(table['answers'])
         start_time = time.time()
         answers = getQueryAnswers(table['question'], table['header'], table['rows'], row_model_representation, col_m
         end_time = time.time()
         rci_answer = []
         for answer in answers:
             rci_answer.append(answer['text'])
         rci_answers.append(rci_answer)

     total_time = end_time-start_time
     print(rci_answers)
     print(true_answers)
     print("time per query is ", total_time/(len(true_answers)))
```

```
[['spring equinox', 'fall equinox', 'summer solstice', 'winter solstice', 'The'], ['period of night', 'period of
['Spring equinox', 'spring equinox', 'midrange', 'midrange', 'midrange', 'midrange', 'midrange', 'Midrange', 'Fa
time per query is  0.44150519371032715
```

For Representation Model - Hit@1

```
[ ]  hit_at_1 = 0
     for i in range(len(true_answers)):
         if(true_answers[i] == rci_answers[i][0]): hit_at_1 +=1
     hit_at_1/= len(true_answers)
     print(hit_at_1)
```

```
0.43
```

For Representation Model - MRR

```
reciprocal_rank_sum = 0
for i in range(len(true_answers)):
    for j in range(len(rci_answers[i])):
        if(true_answers[i] == rci_answers[i][j]):
            reciprocal_rank_sum += 1/(j+1)
            break
mrr = reciprocal_rank_sum/len(true_answers)
print(mrr)
print(0.49)
```

```
0.49
```

Baselines:

1. **Google's TAPAS:** TAPAS (Table-based Pretraining and Sequential Reasoning for Question Answering) is a language model designed for answering natural language questions using tables as the primary source of information. The model was introduced in an academic paper by Google AI researchers in 2020 and is based on the idea of "semantic parsing", which involves converting natural language questions into executable queries that can be executed on a structured data source such as a table.

2. **Facebook's TaBERT:** TaBERT is the first model that has been trained to learn representations for both natural language sentences and tabular data. These sorts of representations are useful for natural language understanding tasks that involve joint reasoning over natural language sentences and tables. This is the first pretraining approach across structured and unstructured domains, and it opens new possibilities regarding semantic parsing, where one of the key challenges has been understanding the structure of a DB table and how it aligns with a query.

Implementing the Baselines:

1. **Google TAPAS:**

Using the models from transformers library and passing the whole table one by one an getting the results containing the predicted answers for the questions.

```python
from transformers import AutoTokenizer, TapasForQuestionAnswering
import pandas as pd

tokenizer = AutoTokenizer.from_pretrained("google/tapas-base-finetuned-wtq")
model = TapasForQuestionAnswering.from_pretrained("google/tapas-base-finetuned-wtq")

def get_answers_from_tapas(table):
    rows = table['rows']
    header = table['header']
    query = table['question']

    df = pd.DataFrame(rows, columns = header)

    inputs = tokenizer(table=df, queries=query, padding="max_length", return_tensors="pt")
    outputs = model(**inputs)

    logits = outputs.logits
    logits_aggregation = outputs.logits_aggregation


    flat_list = []
    for values in rows:
        # flat_list.append(key)
        for value in values:
            flat_list.append(value)
    # print(len(flat_list))
    total_cells = len(flat_list)
    logits = logits[0][:total_cells]
    indexes = sorted(range(len(logits)), key=lambda i: logits[i], reverse=True)[:5]
    # print(indexes)


    # pred = torch.argmax(logits[:, :total_cells], dim=1)
    # print(pred)
    answers = []
    for idx in indexes:
        # print(flat_list[idx])
        answers.append(flat_list[idx])
    return answers
```

Printing the results true and tapas predicted answers.

```
[ ] import time
    tapas_answers = []
    true_answers = []
    start_time = time.time()
    for table in tables[:25]:
        true_answers.extend(table['answers'])
        answers = get_answers_from_tapas(table)
        tapas_answers.append(answers)
    end_time = time.time()

    total_time = end_time-start_time
    print(tapas_answers)
    print(true_answers)
    print("time per query is ", total_time/(len(true_answers)))
```

```
[['period of night', 'period of daylight and the', 'midrange', 'is the day with the', 'midrange'], ['The', 'summer solstice', 'i
['Spring equinox', 'spring equinox', 'midrange', 'midrange', 'midrange', 'midrange', 'midrange', 'Midrange', 'Fall Equinox and S
time per query is  1.9328322410583496
```

For TAPAS Model - Hit @1:

```
[ ] hit_at_1 = 0
    for i in range(len(true_answers)):
        if(true_answers[i] == tapas_answers[i][0]): hit_at_1 +=1
    hit_at_1/= len(true_answers)
    print(hit_at_1)
```

```
0.43
```

For TAPAS Model - MRR:

```
[ ] reciprocal_rank_sum = 0
    for i in range(len(true_answers)):
        for j in range(len(tapas_answers[i])):
            if(true_answers[i] == tapas_answers[i][j]):
                # print(1/(j+1))
                reciprocal_rank_sum += 1/(j+1)
                break
    mrr = reciprocal_rank_sum/len(true_answers)
    print(mrr)
```

```
0.47
```

Comparison:

Hit@1	0.52	Hit@1	0.43	Hit@1	0.43
MRR	0.54	MRR	0.49	MRR	0.47
dtype:	float64	dtype:	float64	dtype:	float64
For Interaction model		**For Representation model**		**For Google's TAPAS**	

Both the baselines take the flatten tables as input. Flattening the table loses information, which results in improper results. Flattening tables can be problematic in the context of training large language models:

1. **Loss of Structural Information:** Flattening a table discards this structural information, making it harder for the model to understand the inherent connections and dependencies between data points.

2. **Semantic Relationships Between Cells:** Flattening the table can break these relationships, making it more challenging for the model to capture the nuanced semantics embedded in the original structure.

3. **Contextual Understanding:** Flattening a table loses the context provided by the arrangement of data in rows and columns, which can hinder the model's ability to grasp the meaning of the information.

4. **Sequence Dependence:** Flattening a table disrupts the natural sequence of information present in rows and columns, potentially leading to confusion in understanding the order and dependencies of different data points.

Advantage of our Model (Cell Level Table Retrieval):

Our model takes the tables in the form of different table structures into consideration as it takes the row values, and column values with the header values to train the model such that it has advantages on:

1. **Efficiency**: Cell-level table retrieval is more efficient than traditional methods because it does not require the entire table to be parsed. This means that it can return results more quickly, especially for large tables.
2. **Flexibility:** Cell-level table retrieval is more flexible than traditional methods because it can be used to retrieve information from a wider variety of tables. This includes tables with different formats and structures.

<u>Issues with the project implementation and exploration:</u>

1. Ambiguity in natural language queries: Natural language queries can be inherently ambiguous, leading to multiple possible interpretations. Resolving ambiguity to provide accurate results is a significant challenge.

2. Lack of Training Data: Building accurate natural language understanding models requires a substantial amount of training data. Acquiring and annotating this data can be time-consuming and expensive.

3. Complexity of Table Structure: Tables can vary significantly in structure, with different columns, data types, and relationships. Handling this structural diversity in the corpus can be complex.

4. Performance scalability: As the corpus of tables grows, the system's performance may degrade. Ensuring scalability while maintaining response times can be challenging.

5. Integration with Data Sources: Connecting the system to various data sources and keeping data up-to-date can be complex, especially if data is distributed across different databases and file formats.

6. User-Friendly Interfaces: Designing an intuitive and user-friendly interface for users to input queries can be challenging, especially if the users are not familiar with database terminology.

In a conclusion our both models are better in approaches as well as in results.

Project Plan:

Timeline Provided:

S.No.	Subject	Date
1.	Project Outline	5/09/2023
2.	Preprocessing TabMcq dataset	1/10/2023
3.	Preprocessing WikiSQL dataset	6/10/2023
4.	Preprocessing WikiTableQuestions dataset	12/10/2023
5.	Creating RCI model	15/10/2023
6.	Implementing BM25	17/10/2023
7.	Interim Submission	18/10/2023
8.	Complete Training RCI model	30/10/2023
9.	Fine Tuning and Testing	5/11/2023
10.	Final results and Comparisions	15/11/2023
11.	Final Submission	20/11/2023

References:

[1] CLTR: An End-to-End, Transformer-Based System for Cell Level Table Retrieval and Table Question Answering - Feifei Pan1, Mustafa Canim 2, Michael Glass 2, Alfio Gliozzo2, Peter Fox1 - https://arxiv.org/pdf/2106.04441v2.pdf

[2] Capturing Row and Column Semantics in Transformer Based Question Answering over Tables - Michael Glass1, Mustafa Canim1, Alfio Gliozzo1, Saneem Chemmengath1, Vishwajeet Kumar1, Rishav Chakravarti1, Avi Sil1, Feifei Pan2, Samarth Bharadwaj1, Nicolas Rodolfo Fauceglia1 - https://arxiv.org/pdf/2104.08303v2.pdf