1. **Introduction**:

The main input of a text-to-SQL system is a Natural Language query (NLQ) and the database (DB) that the NLQ is posed on. The first step (whenever employed) is Schema Linking which aims at the discovery of possible mentions of database elements (tables, attributes, and values) in the NLQ. These discovered schema links, along with the rest of the inputs, will be fed into the neural network that is responsible for the translation.

There are several approaches for NL to SQL translation. These approaches can be broadly categorized into 5 different groups:

1. *Seq2Seq Models:*
- use neural networks to directly map NL questions to SQL queries
- Learn the alignment between the input and output sequences.
- Eg,RESD-SQL

2. *Tabular language models:*
- combine text information with tabular data representations, allowing for more accurate translation.
- Drawback is large amounts of training data and computational resources for pretraining
- Eg, TaBERT, Salesforce BRIDGE

3. *Few-shot prompting using large language models:*
- leverage pre-trained language models and utilize few-shot prompting techniques to generate SQL queries from NL questions

4. *Sketch-based slot-filling:*
- utilize predefined templates(sketches) to guide the SQL query generation
- Eg, MIT_NL2SQL, SQLNET,

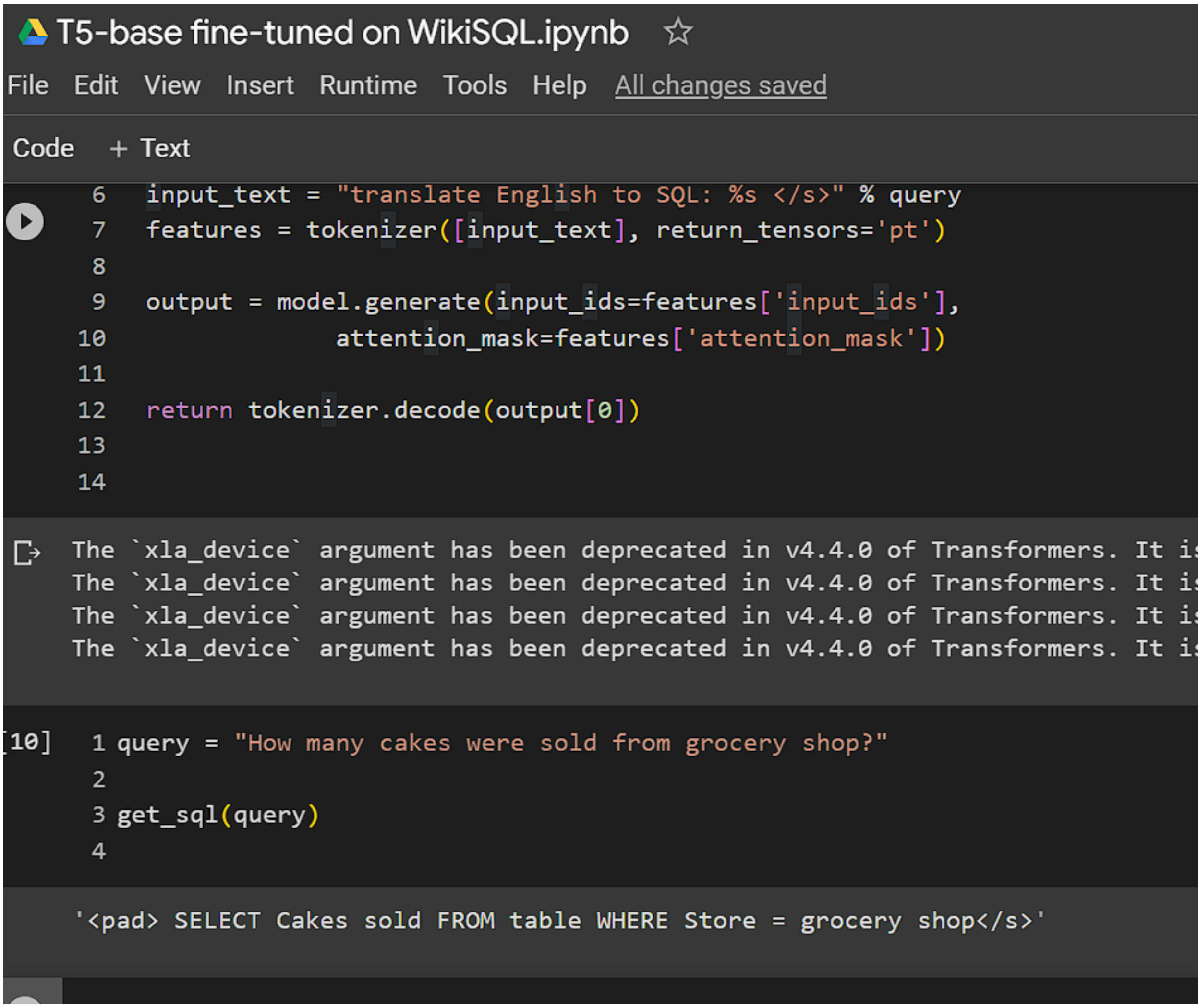
5. *generalization-based models:*
- Generate SQL query for unseen NL questions using a limited set of annotated SQL queries
- focus on the problem of generating SQL queries without relying on a large amount of labeled training data
- Eg: Generate and Rank (GAR) .

2. Tasks:

3.1 Task 1 T5 Based model(from HF):

Approach:
- Finetuned on wikisql dataset
- Only Demerit is that it doesnt consider database schema as input, so it needs to be finetuned on the database everytime otherwise, poor results generated.

Results:



-

3.2 Task 2: SQLnet

Objective:

- Uses a semantic parsing framework that learns to directly map NL questions to corresponding SQL queries
- Works by dividing the translation problem into subtasks, each corresponding to components of the SQL query. E.g., identifying the query's **SELECT** clause (columns to retrieve), the **aggregation** function (e.g., COUNT, MAX), the conditions (e.g., WHERE clause), and the order-by clause.

Approach:

- a sequence-to-set model as well as the column attention mechanism to synthesize the query based on the sketch
- https://github.com/xiaojunxu/SQLNet

Challenges Faced:

- Very old Repository(2018) so environment not compatible and no support from author for the issues raised on github.

3.3 Task 3: RESDSQL

Objective:

- (Ranking-enhanced Encoding plus a Skeleton-aware Decoding framework for Text-to-SQL)
- belongs to the category of "tabular language models
- adopts a two-step approach: first step, it generates an intermediate structured representation "abstract syntax tree" (AST)
- The AST is a structured representation that encodes the logical structure of the SQL query.
- Second step: maps the AST to the final SQL query.

Approach:

- two key components: ranking-enhanced encoding and skeleton-aware decoding

- Ranking-enhanced Encoding:
    - prioritize and select relevant tables and columns from the schema that are most likely to be used in the SQL query generation.
    - Considers various factors such as the lexical match between the question and schema elements, semantic relevance

- Skeleton-aware Decoding:
    - Generates the SQL query from the encoded representation of the input question and schema.
    - In a skeleton-aware decoding framework, the decoding process is guided by a predefined template or "skeleton" that provides a structure for the generated SQL query.
    - The skeleton includes placeholders for the specific details that need to be filled in, such as tables, columns, conditions, and aggregations.
    - The decoding model focuses on generating the skeleton first, ensuring that the overall structure of the SQL query is correctly captured. Once the skeleton is generated, the model then fills in the specific details using the available information from the encoded question and schema.

Challenges Faced:

- The library versions mentioned in the requirement section, not compatible to each other(SpaCy)

```
C:\Users\husain\AppData\Local\Programs\Python\Python310\lib\site-packages\spacy\util.py:895: UserWarning:
 [W094] Model 'en_core_web_sm' (2.2.0) specifies an under-constrained spaCy version requirement: >=2.2.0.
 This can lead to compatibility problems with older versions, or as new spaCy versions are released, beca
use the model may say it's compatible when it's not. Consider changing the "spacy_version" in your meta.j
son to a version range, with a lower and upper pin. For example: >=3.5.3,<3.6.0
  warnings.warn(warn_msg)
Traceback (most recent call last):
  File "C:\Users\husain\OneDrive - iitgn.ac.in\Documents\GitHub\RESDSQL\NatSQL\table_transform.py", line
885, in <module>
    _tokenizer = get_spacy_tokenizer()
  File "C:\Users\husain\OneDrive - iitgn.ac.in\Documents\GitHub\RESDSQL\NatSQL\natsql2sql\preprocess\Toke
nString.py", line 249, in get_spacy_tokenizer
    nlp = spacy.load("en_core_web_sm")
  File "C:\Users\husain\AppData\Local\Programs\Python\Python310\lib\site-packages\spacy\__init__.py", lin
e 54, in load
    return util.load_model(
  File "C:\Users\husain\AppData\Local\Programs\Python\Python310\lib\site-packages\spacy\util.py", line 44
2, in load_model
    return load_model_from_package(name, **kwargs)  # type: ignore[arg-type]
  File "C:\Users\husain\AppData\Local\Programs\Python\Python310\lib\site-packages\spacy\util.py", line 47
8, in load_model_from_package
    return cls.load(vocab=vocab, disable=disable, enable=enable, exclude=exclude, config=config)  # type:
 ignore[attr-defined]
  File "C:\Users\husain\AppData\Local\Programs\Python\Python310\lib\site-packages\en_core_web_sm\__init__
.py", line 12, in load
```

- It requires GPU with pytorch being compatible with CC>=3.7
- GPU in research park have CC=3.5.
- Also tried to run without GPU, but shows CUDA error:

```
C:\Users\Lenovo\anaconda3\envs\RESDSQL\lib\site-packages\torch\cuda\__init__.py:122: UserWarning:
    Found GPU0 NVIDIA GeForce GT 710 which is of cuda capability 3.5.
    PyTorch no longer supports this GPU because it is too old.
    The minimum cuda capability supported by this library is 3.7.

  warnings.warn(old_gpu_warn % (d, name, major, minor, min_arch // 10, min_arch % 10))
Traceback (most recent call last):
  File "schema_item_classifier.py", line 465, in <module>
    total_table_pred_probs, total_column_pred_probs = _test(opt)
  File "schema_item_classifier.py", line 409, in _test
    model = model.cuda()
  File "C:\Users\Lenovo\anaconda3\envs\RESDSQL\lib\site-packages\torch\nn\modules\module.py", line 688, in cuda
    return self._apply(lambda t: t.cuda(device))
  File "C:\Users\Lenovo\anaconda3\envs\RESDSQL\lib\site-packages\torch\nn\modules\module.py", line 578, in _apply
    module._apply(fn)
  File "C:\Users\Lenovo\anaconda3\envs\RESDSQL\lib\site-packages\torch\nn\modules\rnn.py", line 189, in _apply
    self.flatten_parameters()
  File "C:\Users\Lenovo\anaconda3\envs\RESDSQL\lib\site-packages\torch\nn\modules\rnn.py", line 175, in flatten_parameters
    torch._cudnn_rnn_flatten_weight(
RuntimeError: CUDA error: no kernel image is available for execution on the device
CUDA kernel errors might be asynchronously reported at some other API call,so the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.
(RESDSQL)
```

- Tried on IITGN 12 GB GPU but shows memory error

```
(RESDSQL) [husainmalwat@localhost RESDSQL]$ sh scripts/inference/infer_text2sql.sh base spider
366it [00:22, 16.43it/s]
100%|████████████████████████████████████████| 23/23 [00:20<00:00,
1.12it/s]
Namespace(batch_size=16, db_path='./database', dev_filepath='./data/preprocessed_data/resdsql_test.json', device='0,1', epochs=128, gradient_descent_step=4,
learning_rate=3e-05, mode='eval', model_name_or_path='t5-3b', num_beams=8, num_return_sequences=8, original_dev_filepath='./data/spider/dev.json',
output='./predictions/Spider-dev/resdsql_base/pred.sql', save_path='./models/text2sql-t5-base/checkpoint-39312', seed=42,
tables_for_natsql='NatSQL/NatSQLv1_6/tables_for_natsql.json', target_type='sql', tensorboard_save_path='tensorboard_log/text2sql',
train_filepath='data/preprocessed_data/resdsql_train_spider.json', use_adafactor=False)
  0%|                                                                    | 0/23 [00:00<?, ?it/s]
Traceback (most recent call last):
  File "text2sql.py", line 352, in <module>
    _test(opt)
  File "text2sql.py", line 293, in _test
    model_outputs = model.generate(
```

```
  File "/home/husainmalwat/anaconda3/envs/RESDSQL/lib/python3.8/site-packages/torch/autograd/grad_mode.py", line 27, in decorate_context
    return func(*args, **kwargs)
  File "/home/husainmalwat/anaconda3/envs/RESDSQL/lib/python3.8/site-packages/transformers/generation/utils.py", line 1524, in generate
    return self.beam_search(
  File "/home/husainmalwat/anaconda3/envs/RESDSQL/lib/python3.8/site-packages/transformers/generation/utils.py", line 2883, in beam_search
    model_kwargs["past_key_values"] = self._reorder_cache(model_kwargs["past_key_values"], beam_idx)
  File "/home/husainmalwat/anaconda3/envs/RESDSQL/lib/python3.8/site-packages/transformers/models/t5/modeling_t5.py", line 1815, in _reorder_cache
    layer_past_state.index_select(0, beam_idx.to(layer_past_state.device)),
RuntimeError: CUDA out of memory. Tried to allocate 192.00 MiB (GPU 0; 10.76 GiB total capacity; 9.69 GiB already allocated; 65.12 MiB free; 9.79 GiB reserved
in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Management
and PYTORCH_CUDA_ALLOC_CONF
```

Final Results:

- ### NL Questions and tokens:


```
[
    {
        "db_id": "concert_singer",
        "question": "How many singers do we have?",
        "question_toks": [
            "How",
            "many",
            "singers",
            "do",
            "we",
            "have",
            "?"
        ]
    },
    {
        "db_id": "course_teach",

        "question": "Show names of teachers and the courses they are arranged to teach in ascending alphabetical order of the teacher's
name.",
        "question_toks": [
            "Show",
            "names",
            "of",
            "teachers",
            "and",
            "the",
            "courses",
            "they",
            "are",
            "arranged",
            "to",
            "teach",
            "in",
            "ascending",
            "alphabetical",
            "order",
            "of",
            "the",
            "teacher",
            "'s",
            "name",
            "."
        ]
    }
]
```


- ### SQL Queries:

```
select count ( * ) from singer
select teacher.name , course_arrange.course_id from course_arrange join teacher on course_arrange.teacher_id = teacher.teacher_id
order by teacher.name asc
```

- **Agents DB:**

AGENTS.sqlite

AGENTS > AGENTS.sqlite

Tables (3): AGENTS, CUSTOMER, ORDERS — Records: 5

	AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
1	A001	John Smith	New York	0.15	1234567890	USA
2	A002	Jane Doe	London	0.12	9876543210	UK
3	A003	Robert Johnson	Los Angeles	0.1	5551234567	USA
4	A004	Maria Garcia	Madrid	0.08	9123456789	Spain
5	A005	Mohammed Ahmed	Dubai	0.2	9719876543	UAE

Tables (3): AGENTS, CUSTOMER, ORDERS — Records: 5

	CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDIN...	PHONE_NO	AGENT_CODE
1	C001	John Doe	New York	Manhattan	USA	1	5000	1000	3000	3000	1234567890	A001
2	C002	Emma Johnson	London	Westminster	UK	2	8000	2000	4000	4000	9876543210	A002
3	C003	Robert Smith	Los Angeles	Hollywood	USA	3	10000	3000	6000	4000	5551234567	A003
4	C004	Maria Rodriguez	Madrid	Centro	Spain	2	6000	1500	3000	4500	9123456789	A004
5	C005	Hiroshi Suzuki	Tokyo	Shibuya	Japan	1	3000	2500	2000	500	555-5555	A005

AGENTS > AGENTS.sqlite

Tables (3): AGENTS, CUSTOMER, ORDERS — Records: 5

	ORD_NUM	ORD_AMOUNT	ADVANCE_AM...	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
1	1	500	100	2023-06-01	C001	A001	Order 1 Description
2	2	1000	200	2023-06-02	C002	A002	Order 2 Description
3	3	1500	300	2023-06-03	C003	A003	Order 3 Description
4	4	2000	400	2023-06-04	C004	A004	Order 4 Description
5	5	2500	500	2023-06-05	C005	A005	Order 5 Description

- **NL Ques:**

RESDSQL > data > spider > {} dev.json > {} 0 > db_id

```json
[
    {
        "db_id": "AGENTS",
        "question": "Show the name of agents and consumers in ascending alphabetical order of the Agent's name.",
        "question_toks": [
            "Show",
            "the",
            "name",
            "of",
            "agents",
            "and",
            "consumers",
            "in",
            "ascending",
            "alphabetical",
            "order",
            "of",
            "the",
            "Agent",
            "'",
            "s",
            "name",
            "."
        ]
    }
]
```

- **SQL Query/ output:**

RESDSQL > predictions > Spider-dev > resdsql_base > pred.sql

```sql
select agent_name from agents order by agent_name asc
```