Natural Language to SQL: Code Generation using Deep Learning

Aishwarya Malgonde amalgonde@umass.edu
Atharva Nijasure anijasure@umass.edu
Dishank Deven Jhaveri djhaveri@umass.edu
Mrunal Kurhade mkurhade@umass.edu

1 Problem statement

As the amount of data generated by businesses continues to grow exponentially, the need for efficient database management and querying tools becomes increasingly important. However, the process of writing SQL queries to extract information from relational databases can be challenging and time-consuming, especially for non-technical users who may not be familiar with the intricacies of SQL syntax.

To address this challenge, there has been a growing interest in the field of Semantic Parsing to develop tools that enable users to generate SQL code from natural language (NL) queries. Such tools have the potential to greatly enhance the usability and accessibility of relational databases, allowing users to interact with them more intuitively and efficiently.

In this project we have applied the approach of RESDSQL (Li et al., 2023b) to our custom models, for generating SQL code from NL queries. We tried to leverage state-of-the-art natural language processing (NLP) techniques within our computational capability, such as pre-trained contextualized word embedding and fine-tune existing encoder-decoder models with LSTM, using different sequence-to-sequence (seq2seq) models, graph based encoder networks and to map NL queries to SQL code. Later on, we also decided to explore prompting techniques, given that they did not require much computational costs, and the quality of results was much robust. All of our work could be found in our git repository https://github.com/AtharvaNijasure/TextToSQL_AdvNLP. Our own implementations are inside the *notebooks/our_scripts/* folder

2 What we proposed vs. what we accomplished

Items accomplished:

- Baseline model
- Employing RESDSQL approach
- Implementing and training hybrid models
- Explored encoder-decoder with attention mechanism
- Implemented the cross-entropy loss
- Implemented Contrasive Loss
- Error Analysis

Items not accomplished:

- Encoders with GNN
- Encoders with Graph Attention networks
- Experiments with contrastive loss: We implemented it but could not run experiments with it

Items added:

- Prompt engineering

3 Related work

The NL to SQL task has garnered significant attention from the database community and NLP researchers, leading to notable advancements in the field. A comprehensive survey by Deng et al. (Deng et al., 2022) offers a thorough comparison of datasets, model approaches, and results.

Iyer et al. (Iyer et al., 2017) introduced a seq2seq architecture for NL to SQL, which influenced our utilization of T5 and BERT encoders

in constructing seq2seq models. However, challenges persist due to structured syntax requirements and the involvement of multi-hop queries and tables. To address these issues, we explored techniques such as beam search from Wang et al. (Wang et al., 2018) and conducted experiments in prompt engineering, specifically query execution.

To tackle cross-generalization and multi-hop reasoning, Li et al. (Li et al., 2023c) proposed graph-aware layers that encode semantic and structural information using a relational GNN block. We considered modifications incorporating attention from RGAT (Busbridge et al., 2019) and initially explored the use of a multi-relational graph as discussed in Chen et al. (Chen et al., 2021) due to the presence of multi-hop reasoning and foreign keys.

The challenges also involve ensuring valid SQL syntax and proper schema in the output, where schema refers to table and column names.To ensure a valid SQL syntax and proper schema in the output presents additional challenges. Xuan et al. (Xuan et al., 2021) addressed this by training a schema-aware denoising seq2seq model with erosion and shuffle noising techniques. They employed a clause-sensitive execution-guided decoding strategy.

In the decoder component, Li et al. (Li et al., 2023a) proposed a two-step approach involving schema ranking and filtering using a cross encoder. Their method achieved state-of-the-art results on the Spider dataset.

However, all of the aforementioned methods necessitate substantial computing power, particularly in the encoder component or when generating encodings that represent the relationships between the database schema and semantic entities. Hence, to compute within available resources, powerful prompting techniques as described in (Nye et al., 2021) could be used. Consequently, prompt engineering techniques, such as chain of thought and few-shot learning, have also been explored in DINSQL (Pourreza and Rafiei, 2023a). The DIN-SQL network's output is used to fine-tune GPT-4. Given the relation mapping problem present in our task, chain of thought prompting methods in (Wei et al., 2022) proved to be valuable resources.

4 Dataset

We have use the SPIDER 1.0 dataset introduced by (Yu et al., 2018). It consists of 200 databases with multiple tables, 10181 questions and 5693 unique and complex SQL queries covering 138 different domains. SPIDER overcomes the shortcomings of other datasets such as -

ATIS (Shivakumar et al., 2019), **Geo**, **Academic**: Each of them contains only a single database with a limited number of SQL queries, and has exact same SQL queries in train and test splits.

WikiSQL (Zhong et al., 2017): The numbers of SQL queries and tables are significantly large. But all SQL queries are simple, and each database is only a simple table without any foreign key.

Spider 1.0 is the first cross-domain NL-to-SQL dataset, with multiple databases, consisting multiple tables, foreign keys and complex SQL queries.

The original Spider 1.0 dataset has provided a train and dev set, with no database overlaps between them. To ensure fair evaluation of our model's performance, we have created a separate development set from the original Spider 1.0 train set, without any overlap with the provided dev set. We have then used the original dev set as the official test set for our experiments. This approach allows us to accurately assess the generalization ability of our model to unseen data, while still utilizing the complete Spider 1.0 dataset for training. The number of samples for our task can be seen in Table 1.

Table 1: Spider Data Split

Set	Size
Train	6304
Dev	1034
Test	696

4.1 Data preprocessing

Our approach involved integrating the preprocessing techniques from Spider and RESDSQL while customizing them to meet our specific needs.

In Spider's preprocessing, a fixed string called "value" is used to replace numerical or quoted string values. However, we chose to exclude this step from our preprocessing pipeline. Our intention was to investigate whether the models could learn to directly copy and retain these values from the natural language (NL) input itself.

On the other hand, RESDSQL's preprocessing involves working with the raw format of input

Table 2: Sample Data

Type	Text
NL query	”How many heads of the departments are older than 56 ?”
SQL ouput	”SELECT count(*) FROM head WHERE age 56”
Preprocessed input	”how many heads of the departments are older than 56 ?”
Preprocessed SQL	”select count (*) from head where age 56”
SQL skeleton	”select count (_) from _ where _”
DB schema	”head : name , age , born state , head id | department : ranking , ...”

strings. To ensure consistency and comparability, we opted to normalize both the NL inputs and SQL queries. This normalization step allowed us to standardize the format and structure of the data, facilitating more effective training and evaluation processes.

We utilized the RESDSQL scripts without modifications for generating the SQL skeleton. For the purpose of extracting the ordered database schema, we made predictions on the entire dataset after training the schema classifier. In Table 2, we can observe an sample of the Spider data.

5 Baselines

Our baseline model is a seq2seq model with an encoder-decoder architecture (Iacob et al., 2020). The encoder component employs a bidirectional Long Short-Term Memory (LSTM) network to encode the input query, while the decoder component is an LSTM network responsible for generating the SQL query based on the encoded vector.

We selected this baseline model due to its simplicity and effectiveness in handling sequence generation tasks. By leveraging the bidirectional LSTM, the model can capture both past and future contextual information, enhancing its understanding of the input query. The decoder LSTM then utilizes this encoded information to generate accurate and contextually relevant SQL queries. The input to the encoder was the preprocessed input and the target for the decoder was the preprocessed SQL. This model did not have access to the SQL skeleton and the DB schema.

Table 3: Baseline Hyperparameters

Name	Value
Learning rate	**0.0005**, 0.001
Batch size	64
Embedding size	200, **300**
Hidden layer size	100, **200**

During the training process, we performed hyperparameter tuning to optimize the performance of the baseline model. The specific hyperparameters we focused on included learning rate, LSTM hidden dimension and the embedding size. The values we experimented with are listed in Table 3. The values which gave the best result are highlighted in the table. We employed random search to explore different combinations of hyperparameter values and identify the optimal configuration.

To avoid overfitting and evaluate the model's generalization ability, we did hyperparameter tuning on the development set and made the best selection. Results on the test set are shown in Table 3 and discussed in the result analysis section. It is important to note that we strictly avoided tuning any hyperparameters on the test set to prevent any biased or over-optimized results. This baseline model allows us to assess the effectiveness and improvements achieved by more advanced architectures and techniques in our NLP task.

6 Approach

6.1 Seq2Seq Models

In the Ranking-enhanced Encoding plus a Skeleton-aware Decoding framework for Text-to-SQL (RESDSQL) approach, the encoder is injected with the natural language query along with the ordered DB schema items. The schema linking in the encoder is conducted beforehand to filter out most of the irrelevant schema items in the database schema, which can alleviate the difficulty of the schema linking for the seq2seq model. For such purpose, we also trained an cross-encoder (also called as schema classifier) to classify the tables and columns simultaneously based on the input question, and then rank and filter them according to the classification probabilities to form a ranked schema sequence. In this approach, the decoder first generates the skeleton and then the actual SQL query. Since

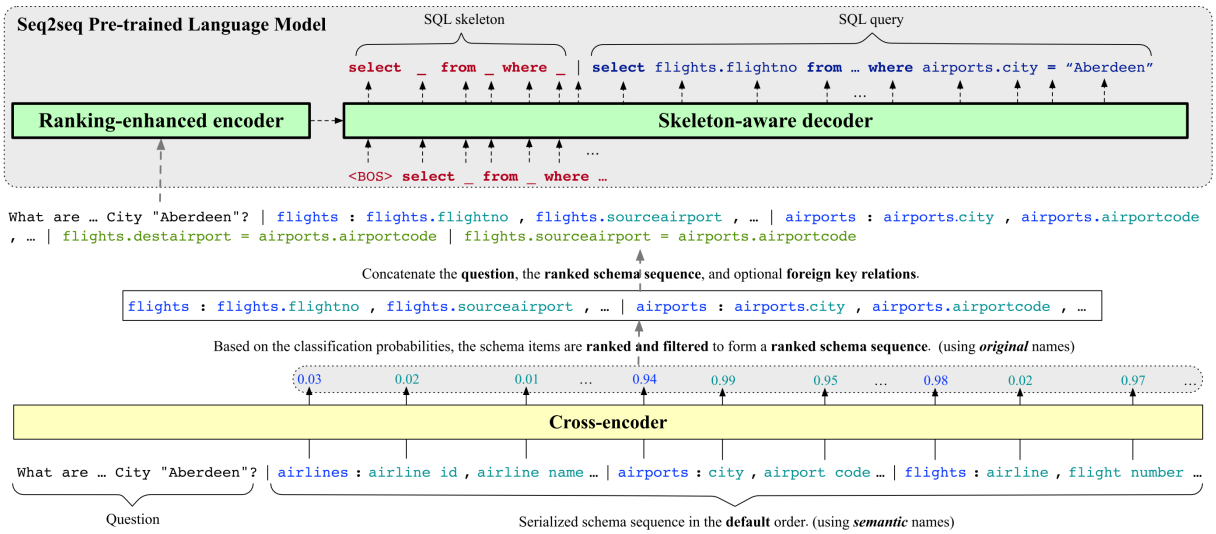

Figure 1: An overview of the ranking-enhanced encoding and skeleton-aware decoding framework. We train a cross-encoder for classifying the schema items. Then we take the question, the ranked schema sequence, and optional foreign keys as the input of the ranking-enhanced encoder. The skeleton-aware decoder first decodes the SQL skeleton and then the actual SQL query. Image source: RESDSQL

skeleton parsing is much easier than SQL parsing, the decoder is trained such that it first generates the skeleton, which could implicitly guide it to generate the subsequent SQL query. Figure 1 illustrates the approach employed in our project.

In this project, we solve the NL-to-SQL task using different transformer-based models, namely T5, BERT, and GPT2. We adopted a transfer learning approach where we fine-tuned these pre-trained models on the Spider dataset. Our objective was to leverage the capabilities of these models and investigate their effectiveness in addressing our task.

For each model, we followed a similar approach. We had the raw data preprocessed. In the training pipeline, we initiated the process by tokenizing both the input text and the target text using the tokenizer associated with each specific model. We then fine-tuned the models using a seq2seq setup, where the models were trained to generate target text given the input text.

During the fine-tuning process, we used a combination of hyperparameters to optimize the performance of the models. This included selecting appropriate hyperparameters such as learning rate, batch size, and maximum sequence length.

For evaluating the performance of each model, we utilized two evaluation metrics: exact match (EM) and execution accuracy. However, during our experiments, we observed that both metrics yielded identical values. Therefore, in this report, we have chosen to report the results using the EM metric. For the schema classifier, we use Area Under ROC Curve (AUC) to evaluate its performance. Additionally, we conducted a qualitative analysis by manually inspecting the generated target text for a subset of examples to assess the model's ability to capture the semantics and context of the input text.

Models from RESDSQL:

- T5-to-T5
- Schema Classifier: During the execution of the script, we encountered certain issues. Upon investigation, we discovered an error in the implementation of the Focal Loss function. As a consequence, the loss values were becoming 'nan' (not-a-number). So we decided to develop our custom implementation of the focal loss function. We trained schema classifier both using contrasive loss and focal loss but in both cases due to computational limitation our classifier remained under trained.

Models implemented by us:

- Baseline (*3_train_baseline.ipynb*)
- BERT-to-T5 (*6_BERT+T5+train_v3.ipynb*)
- GPT2-to-T5 (*8_GPT2+T5+train.ipynb*)

- Contrastive Loss: In order to enhance the training of the schema classifier and ranker, we considered using contrastive loss.

We have leveraged the torch library in Python for all our implementations. And we did all the training on Google Colaboratory. Some of the challenges we faced were -

- We couldn't experiment with large batch sizes for the BERT and GPT2 models while training with the RESDSQL approach, because of large input and target sequences.
- Performing training and evaluation parallelly caused OOM errors (out-of-memory), so evaluated the checkpoints after the training was completed.
- Due to the memory limitations on Google Colab, we were unable to conduct experiments with larger T5 models beyond "t5-small." Therefore, our training for the T5-to-T5 model was restricted to the "t5-small" variant, and we did not explore the possibilities offered by larger models such as "t5-base," "t5-large," and "t5-3b." Similarly for GPT2 and BERT models.
- we tried to come up with a relational graph network but the pre-processing required to consume a lot of resources hence couldn't move forward with this network.

6.2 Prompt Engineering

Due to the difficulty in constructing encodings that capture the relational meaning between semantic entities, such as real-world table names and the database schema, we explored the potential of leveraging large language models to address this challenge. We subsequently recognized that by employing suitable chain of thought prompting techniques, it was possible to convert the given natural language text with schema input into the desired SQL output.

Hence, we explored prompting approach by reading (Pourreza and Rafiei, 2023b) which resulted in much better results. We experimented with zero-shot prompting at first using only the SQL schema and gpt-3.5. This approach while providing decent results was not better than existing neural net approaches.

We then used the langchain library to create SQL agents that use chain of thought prompting to generate accurate sql queries. The Langchain library provides an agent that uses Chain of thought prompting (Wei et al., 2023) to generate accurate results. Through this approach 'chain of thought'—a series of intermediate reasoning steps—significantly improves the ability of large language models to perform complex reasoning. Through this approach, the model performs significantly better when compared to zero-shot prompts.

Primarily, the 'chain of thought' approach permits models to break down complex issues into simpler steps, efficiently allocating resources to tasks that necessitate more logical steps. Furthermore, it offers an intelligible insight into the model's operations, highlighting its reasoning path and facilitating debugging, although fully understanding the model's computations is still a challenge. It is applicable to tasks like math problems, common sense reasoning, symbolic manipulation, and potentially any human-solvable language-based tasks. Thus we believed it was the best prompting technique for natural language to sql detection. Lastly, this reasoning technique can be easily triggered in large, ready-to-use language models by incorporating chain-of-thought examples in few-shot prompts.

Langchain agents provide an interface that allows the model to prompt itself using "observations" and "thoughts" This allows the model to come up with the correct SQL query and prove so by executing the query against the database. The library also allows us to interchange Large Language Models with relative ease.

7 Experiments

We conducted the following series of experiments:

1. **Normal Seq2seq**
   (max_input_len = 43, max_output_len = 127)
   (a) Baseline
   (b) T5-to-T5 (bs = 32, lr = 3e-5)
   (c) BERT-to-T5 (bs = 32, lr = 1e-4)
   (d) GPT2-to-T5 (bs = 32, lr = 1e-4)
2. **RESDSQL Seq2seq**
   (max_input_len = 512, max_output_len = 256)
   (a) T5-to-T5 (bs = 16, lr = 3e-4)
   (b) BERT-to-T5 (bs = 4, lr = 1e-5)
   (c) GPT2-to-T5 (bs = 4, lr = 1e-5)

3. **Schema Classifier** (bs = 2, lr = 3e-3)

In Normal Seq2seq experiments, the encoder input was the normalized input query and the decoder target was normalized sql query. In RESDSQL Seq2seq experiments, the normalized text input was appended with the ranked database schema, and the normalized SQL query was prepended with the SQL skeleton. In Schema Classifier, we trained the classifier to rank the tables and columns based on the input question based on classification probabilities to form a ranked schema sequence. Table 4 list the model names which we used for our experiments.

Table 4: Model versions

Model	Version
T5	t5-small
BERT	bert-base-uncased
GPT2	gpt2
Schema Classifier	roberta-large

8 Result analysis

8.1 Seq2seq Models

The schema classifier achieved an overall AUC of 64%. The low AUC could be attributed to several factors. One possible reason is the limited training duration of only 4 epochs. Training the model for a longer period may allow it to learn more complex patterns and improve its performance. It is also possible that the model architecture or hyperparameters may not have been optimized for the specific task, leading to suboptimal performance.

The performance of the seq2seq models trained using different approaches is summarized in Table 5. All the seq2seq models show better performance than the baseline model, which is expected. The results clearly indicate that adopting the RESDSQL approach has significantly enhanced the performance of all the seq2seq models, with the most notable improvement observed in the T5-to-T5 model. It's worth noting that the models were evaluated for approximately 15 epochs, suggesting that they may not have reached their full training potential. Further training iterations could potentially yield even better results. This is evident from the decreasing trend in the loss curve depicted in Figure 2, indicating that there is still room for further improvement.

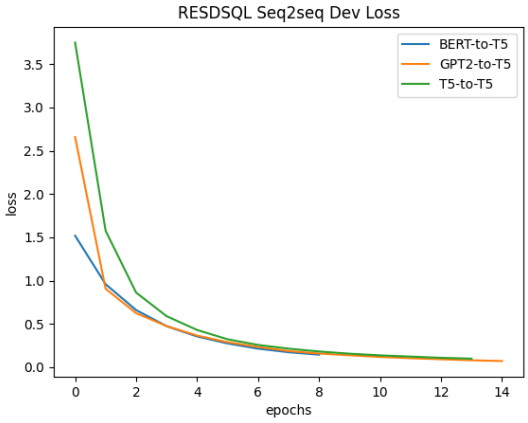

Figure 2: RESDSQL Seq2seq loss on the development set

8.2 Prompting

We noticed that GPT-3.5 was significantly better on the spider dataset as compared to open-source models (gpt4all-l13b-snoozy, vicuna-13b-1.1-q4.2). This may be because gpt-3.5 likely has the spider dataset in its training data as compared to other open-source models. On our test data set gpt-3.5 was better but the difference between the open source models was not very significant. We created a new test dataset for this very reason, realizing that GPT-3.5 was likely trained on the Spider Dataset before. We had an exact match of 78% on the Spider Dataset using GPT3.5 and an exact match of 66% on (gpt4all-l13b-snoozy) and an exact match of 64% on (vicuna-13b-1.1-q4.2). On our test set we noticed that gpt3.5 was significantly slower but had a similar exact match of 75% while the two open source models (gpt4all-l13b-snoozy, vicuna-13b-1.1-q4.2) had an exact match of 72% and 70% respectively.

9 Error analysis

For the purpose of manual error analysis, we compared 50 outputs from each of the three seq2seq models, while considering the database schema present in the input. The initial LSTM baseline model performed poorly on the given dataset. The baseline BiLSTM models generated numerous repetitive and incorrect SQL queries. We selected only the top models for manual analysis, which were trained with both the database schema and SQL skeleton for the decoder and had the same input with the database schema. Our findings revealed that BERT-based encoders per-

Table 5: Experimental Results

Encoder	Decoder	SQL skeleton	DB Schema	EM (%)
Bi-LSTM	LSTM	No	No	0.86
T5	T5	No	No	6.9
GPT2	T5	No	No	4.6
BERT	T5	No	No	3.7
T5	T5	Yes	Yes	**25.8**
GPT2	T5	Yes	Yes	13.5
BERT	T5	Yes	Yes	8.7

formed poorly on this task, followed by GPT2 encoders and then T5 encoders for the resdsql decoder. Refer Table 5 for the results.

There were several common patterns observed across all three networks, particularly in areas such as correctly identifying column and table names, incorrect SQL syntax, and a few spelling errors. In the case of BERT-based encoders, many of the SQL outputs failed to establish connections between the database schemas and semantic entities. This limitation was anticipated, which is why we think Attention-based Graph Neural Network (GNN) modules could solve this problem. Furthermore, BERT-based encoders lacked the logical structural flow necessary for generating SQL statements. GPT2-based encoders exhibited issues with incomplete and repetitive outputs, as well as incorrect structural relationships between the database schema and semantic entities in the natural language question. Among our trained models, T5 encoders performed the best with fewer logical or SQL syntactical errors, but they did not excel in accurately capturing the semantic and schema relations as anticipated.

Here are a few examples that illustrate the performance errors encountered in these networks:

1. Incomplete:

```
Input : "Show the name of drivers in
    descending order of age | DB schema
    :..."
Actual SQL: "select name from driver
    order by age desc"

Baseline: "select name from"

Correct in all three models.
```

2. Wrong Structure:

```
Input: "What are the party emails
    associated with parties that used
    the party form that is the most
    common? | DB schema : ..."
Actual SQL: "select parties.party_email
    from parties join party_forms on
    parties.party_id = party_forms.
    party_id where party_forms.form_id =
     ( select form_id from party_forms
    group by form_id order by count ( *
    ) desc limit 1 )"

Baseline: "select t2 . party from party
    from party from party ..."

BERT: "select party_email from parties
    where party_form = ( select
    party_form from parties group by
    party_form order by count ( * ) desc
     limit 1 )"

GPT2: "select email_address from
    email_address group by email_address
     order by count ( * )"

T5: "select party_email from parties
    where party_form = ( select
    party_form from parties group by
    party_form order by count ( * ) desc
     limit 1 )"
```

3. Confused in Table / Column Names:

```
Input : "Find the last name of the
    individuals that have been contact
    individuals of an organization. | DB
     schema : ..."
Actual SQL: "select distinct individuals
    .individual_last_name from
    individuals join
    organization_contact_individuals on
    individuals.individual_id =
    organization_contact_individuals.
    individual_id"

BILSTM: "select t1 . t1 . t1 . t1 ..."

BERT: "select distinct services.
    service_name from services join
    people_in_events on services.
    service_id = people_in_events.
    service_id"

GPT2: "students.student_id from people
    join student_contact_"

T5: "select distinct individuals.
    individual_last_name from
    individuals join
    organization_contact_individuals on
    individuals.individual_id =
```

Table 6: Manual error analysis summary of 200 outputs, 50 for each model

Model	Wrong Structure	Repetition	Incomplete	Confused in column / table names	EM (%)
Baseline	13	29	4	3	2.0
BERT	33	0	0	17	0.0
GPT2	15	3	13	16	6
T5	8	0	0	20	44

```
organization_contact_individuals.
individual_id"
```

As you can see, the same output exhibits different issues in different models. However, the T5 model is more robust than others in at least getting the SQL skeleton somewhat right. The problem remains with recognizing the relation between the database schema and semantic entities present in natural language. Hence, after obtaining such results, we considered using pre-trained large language models with prompt engineering to mitigate the database schema and column name resolution issues.

10 Contributions of group members

Each member of the group contributed the following to this project:

- Aishwarya: Preprocessed Spider data; built BERT-to-T5, GPT2-to-T5 and baseline models; trained baseline, T5-to-T5, BERT-to-T5, GPT2-to-T5 and schema classifier; produced final results and results for error analysis; wrote sections - Dataset, Baseline, Experiments and Seq2seq models (Approach + Result Analysis)

- Atharva: Did research on all the papers present in related section, helped in preprocessing the data, build models and input for sql-skeleton based networks, helped in manual analysis of error results, building T5-to-T5, BERT-to-T5 model. Built loss functions for contrasive loss in classifier network. Tried to implement and incorporate GNN type models to pre-process the data, Tried to implement and build classifier ranker module, trained module for few epochs. Both of these networks didn't figure out in the analysis because of shortage of computational power and no documentation of these codes. Helped in shifting our approach from fine tuning pre-trained network based models to prompt engineering approach. Managed code on git repository. Wrote Related work, Problem Statement and helped in Conclusion, Approach, Error Analysis

- Dishank: Built Langchain agents to prompt 3 different Large Language Models on the Spider Dataset and our own test set. Used Langchain's Zero Shot Chain of Thought Prompting approach to get better results compared to our original proposal. Tested the following LLM's: OpenAI (Codex code-davinci-002), GPT4ALL (gpt4all-13b-snoozy) and Vicuna(Vicuna-13b). Wrote section 2 and 6.2.

- Mrunal: Tried to build the schema classifier on preprocessed data but didn't work out due to the lack of computing power, prepared the sample dataset to analyze the prompting engineering approach, helped in the result and error analysis. Wrote conclusion, problem statement, and proposed vs accomplished sections, approach and error analysis.

11 Conclusion

In this project we focused on the NL-to-SQL task. We employed a variety of techniques, including pre-processing, hybrid model architectures and prompt engineering. We explored different models with encoder-decoder architectures, such as T5-to-T5, BERT-to-T5, and GPT-to-T5 and fine-tuned them on the Spider dataset. The models were trained to generate target SQL queries from input natural language text using a seq2seq setup. We improved the accuracy of SQL query generation using Langchain's zero shot chain of thought prompting method. The model's performance was evaluated using exact match (EM) and execution accuracy metrics.

While the schema classifier achieved an overall AUC of 64%, its performance could be further enhanced with a longer training period and an optimized architecture. The results showed that the RESDSQL approach significantly improved the performance across all seq2seq models, with the most notable improvement observed in the T5-to-T5 model.

We encountered certain limitations during our project. Due to resource constraints, we couldn't experiment with larger pretrained models, and we faced memory issues while training and evaluating the models in parallel. These limitations could be addressed in future work by utilizing more powerful hardware. There is still room for further exploration and improvement, such as extending the training duration, optimizing hyperparameters, and investigating alternative model architectures.

12 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - Yes, ChatGPT

If you answered yes to the above question, please complete the following as well:

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which part of the proposal is associated with which prompt.
  - your response here
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - We mostly used it for paraphrasing our text
  - Sometimes for checking syntax of code.

References

Busbridge, D., Sherburn, D., Cavallo, P., and Hammerla, N. Y. (2019). Relational graph attention networks. *arXiv preprint arXiv:1904.05811*.

Chen, M., Zhang, Y., Kou, X., Li, Y., and Zhang, Y. (2021). r-gat: Relational graph attention network for multi-relational graphs. *arXiv preprint arXiv:2109.05922*.

Deng, N., Chen, Y., and Zhang, Y. (2022). Recent advances in text-to-SQL: A survey of what we have and what we expect. In *COLING*, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Iacob, R., Brad, F., Apostol, E. S., Truică, C.-O., Hosu, I., and Rebedea, T. (2020). Neural approaches for natural language interfaces to databases: A survey. pages 381–395.

Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J., and Zettlemoyer, L. (2017). Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*.

Li, H., Zhang, J., Li, C., and Chen, H. (2023a). Decoupling the skeleton parsing and schema linking for text-to-sql. *arXiv preprint arXiv:2302.05965*.

Li, H., Zhang, J., Li, C., and Chen, H. (2023b). Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql.

Li, J., Hui, B., Cheng, R., Qin, B., Ma, C., Huo, N., Huang, F., Du, W., Si, L., and Li, Y. (2023c). Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *arXiv preprint arXiv:2301.07507*.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. (2021). Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

Pourreza, M. and Rafiei, D. (2023a). Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Pourreza, M. and Rafiei, D. (2023b). Din-sql: Decomposed in-context learning of text-to-sql with self-correction.

Shivakumar, P. G., Yang, M., and Georgiou, P. (2019). Spoken language intent detection using confusion2vec. *arXiv preprint arXiv:1904.03576*.

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang, P.-S., Mao, Y., Polozov, O., and Singh, R. (2018). Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.

Xuan, K., Wang, Y., Wang, Y., Wen, Z., and Dong, Y. (2021). Sead: end-to-end text-to-sql generation with schema-aware denoising. *arXiv preprint arXiv:2105.07911*.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing

and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.