LLMs Go To SQL School

Yifan Wang
Northeastern University

Abstract

Evaluating SQL queries automatically is complex due to the intricate nature of the language and the various of valid solutions for any given query. In this paper, we present a cutting-edge approach that employs LLMs, utilizing models such as BERT, RoBERTa, and Big Bird. We experimented with various methods of structuring the training data, including using answers only, combinations of questions and answers, and integrating schema with questions and answers to enhance the model's generalization capabilities. By training these models on diverse datasets of SQL queries and their corresponding labels, we develop a system that can autonomously grade new SQL queries with high accuracy. Our experiments indicate that this approach not only achieves exceptional accuracy in grading but also significantly outperforms prior leading models. This study underscores the potential of using advanced pre-trained language models to improve the efficiency and accuracy of SQL query evaluation.

Introduction

Grading multiple SQL queries is inherently time-consuming and complex, a common issue in many areas of computer science education. The variety of correct ways to construct SQL queries and the sheer volume of submissions can make manual grading particularly cumbersome and daunting. The challenge intensifies when attempting to automate this process.

Automated SQL query grading systems can significantly reduce time and effort, although they often struggle to consistently assess diverse query types. These systems typically fall into two categories: static analysis systems and dynamic analysis systems. Static analysis systems assess queries by comparing the structure of a submitted query to predefined answer-key queries without executing the queries. Dynamic analysis systems, in contrast, execute the submitted queries against fixed datasets and compare the outcomes to those of the answer-keys. This method primarily evaluates the results, which may not accurately reflect the correctness of a query, especially in cases where different queries yield identical results, such as producing an empty table.

Despite their availability, many automated systems lack the sophistication needed for precise and consistent grading. This paper suggests enhancing pre-trained language models to interpret and model SQL statements more effectively, aiming to overcome these challenges. By fine-tuning models like BERT on a dataset of SQL queries and their respective labels, we develop a system capable of accurately grading new queries. Leveraging the extensive pre-training of these models on vast text corpora enables them to understand the intricate syntax and structures of SQL, thereby overcoming the limitations of traditional grading systems.

Background

Theoretical Framework

The theoretical basis for our project is rooted in natural language processing (NLP), particularly the advancements introduced by transformer architectures such as BERT (Bidirectional Encoder Representations from Transformers), RoBERTa (a robustly optimized version of BERT), and Big Bird.

Tools and Libraries

The implementation of these transformer models in our project is facilitated by the Hugging Face Transformers library, a platform that provides pre-trained models that can be customized and fine-tuned for various NLP tasks. This library is built on top of PyTorch, a leading deep learning framework that provides flexibility and speed in building and training models. By leveraging these tools, we can efficiently apply complex transformer models to the task of SQL query grading.

Related work

The static analysis approach, as originally outlined by Aho et al., uses a matrix of relational expressions to categorize SQL queries into equivalence classes. Another method by Štajduhar et al. employs string similarity metrics to compare student submissions against correct answers.

In dynamic analysis, systems like SQLator, SQLify, and AsseSQL assess the accuracy of submitted queries by comparing their results with correct outputs. SQL Tester, for instance, conducts detailed, case-sensitive comparisons. The

XData system, introduced by Chandra et al., integrates dynamic analysis to identify errors with static analysis to gauge query precision, implementing edit-based grading for partial credits.

Deep learning has become increasingly prominent in enhancing grading accuracy and efficiency. This includes using LSTM paired with Grey Wolf Optimizer for short answer grading, as developed by S. Hassan et al., and a combination of CNN and BiLSTM for evaluating handwritten responses by M. A. Rahaman et al. Recent advancements feature BERT, pre-trained on domain-specific resources for short answer grading by C. Sung et al., a deep learning system with attention mechanisms for scoring requests for proposals, and a hybrid approach that combines BERT with feature engineering for automated essay evaluation by Prabhu et al. These developments underscore the capability of deep learning to significantly improve the accuracy and consistency of automated grading across various types of educational assessments.

Project description

Dataset description and preprocessing

Our study utilizes a dataset comprising SQL statements aimed at automating grading, sourced from both a publicly accessible database and supplemented with our internal data. The schema of the public dataset includes tables featuring anonymized student submissions, feedback, grades, labels, and other relevant details. We have enriched this dataset by merging specific fields and incorporating prefixes to each segment to enhance the model's learning of various information types, similar to the preprocessing in BERT's training involving special tokens like [CLS] and [SEP]. Figure 1 shows the label distribution within our dataset, which consists of 12,987 unique student submissions, with 53.9% of these submissions marked as correct.

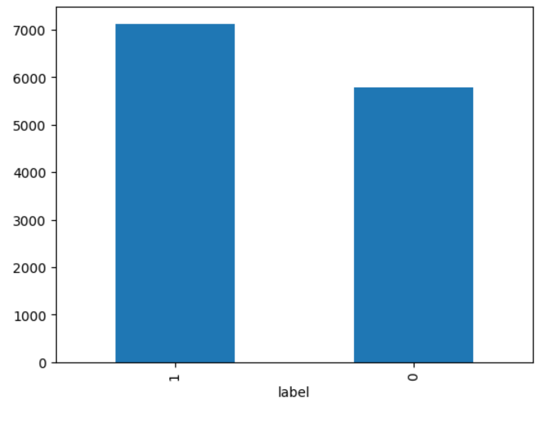
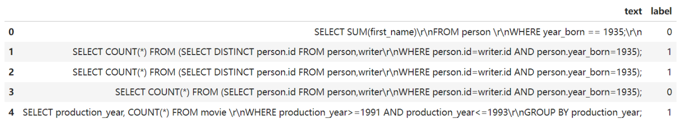


Figure 1: Label Distribution

Fine-tuning

We fine-tuned several models using the Hugging Face Trainer API, each employing distinct strategies for structuring the training data. These approaches included organizing the data by answers only, pairing questions with their corresponding answers, and integrating schema details with questions and answers. Each configuration was designed to optimize the model's learning and generalization capabilities across various SQL query assessments.

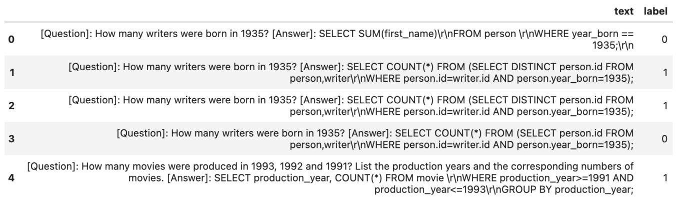
No Context Initially, we structured the training data as depicted in Figure 2, which straightforwardly pairs submission answers with their corresponding labels. We then fine-tuned the BERT base uncased model on this dataset, followed by cross-validation. Afterward, we tested the model on queries derived from various schemas.



	text	label
0	SELECT SUM(first_name)\r\nFROM person \r\nWHERE year_born == 1935;\r\n	0
1	SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
2	SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
3	SELECT COUNT(*) FROM (SELECT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	0
4	SELECT production_year, COUNT(*) FROM movie \r\nWHERE production_year>=1991 AND production_year<=1993\r\nGROUP BY production_year;	1

Figure 2: No Context

Question and Answer Next, we revised the structure of our training data, combining information about the question and answer and adding the prefixes [Question]: and [Answer]: respectively, as shown in Figure 3. We fine-tuned both BERT base uncased and RoBERTa base models with this updated data format. Following fine-tuning, we performed cross-validation on the BERT model, then tested both models on queries from various schemas. Additionally, we conducted an error analysis to assess and refine model performance further.



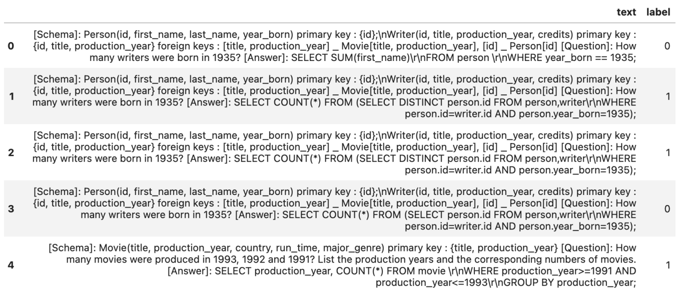
	text	label
0	[Question]: How many writers were born in 1935? [Answer]: SELECT SUM(first_name)\r\nFROM person \r\nWHERE year_born == 1935;\r\n	0
1	[Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
2	[Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
3	[Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	0
4	[Question]: How many movies were produced in 1993, 1992 and 1991? List the production years and the corresponding numbers of movies. [Answer]: SELECT production_year, COUNT(*) FROM movie \r\nWHERE production_year>=1991 AND production_year<=1993\r\nGROUP BY production_year;	1

Figure 3: Question and Answer

Schema, Question and Answer Following previous modifications, we further adjusted the training data structure to include a new prefix [Schema]: before the existing [Question]: and [Answer]: prefixes, combining information about schemas, questions, and answers from the database, as shown in Figure 4. We switched to using the Big Bird model for fine-tuning with this comprehensive data organization. After fine-tuning, we directly tested the Big Bird model on queries from various schemas.

Prompt Engineering

In our project, we employ prompt engineering with the GPT-3.5 Turbo model to automate the validation of SQL queries. The system is set up to simulate an interaction where the model acts as a SQL expert tasked with assessing queries based on provided schemas. User prompts request a binary evaluation—0 or 1—on the syntactic and logical correctness



	text	label
0	[Schema]: Person(id, first_name, last_name, year_born) primary key : {id};\nWriter(id, title, production_year, credits) primary key : {id, title, production_year} foreign keys : [title, production_year] _ Movie[title, production_year], [id] _ Person[id] [Question]: How many writers were born in 1935? [Answer]: SELECT SUM(first_name)\r\nFROM person \r\nWHERE year_born == 1935;	0
1	[Schema]: Person(id, first_name, last_name, year_born) primary key : {id};\nWriter(id, title, production_year, credits) primary key : {id, title, production_year} foreign keys : [title, production_year] _ Movie[title, production_year], [id] _ Person[id] [Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
2	[Schema]: Person(id, first_name, last_name, year_born) primary key : {id};\nWriter(id, title, production_year, credits) primary key : {id, title, production_year} foreign keys : [title, production_year] _ Movie[title, production_year], [id] _ Person[id] [Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT DISTINCT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	1
3	[Schema]: Person(id, first_name, last_name, year_born) primary key : {id};\nWriter(id, title, production_year, credits) primary key : {id, title, production_year} foreign keys : [title, production_year] _ Movie[title, production_year], [id] _ Person[id] [Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) FROM (SELECT person.id FROM person,writer\r\nWHERE person.id=writer.id AND person.year_born=1935);	0
4	[Schema]: Movie(title, production_year, country, run_time, major_genre) primary key : {title, production_year} [Question]: How many movies were produced in 1993, 1992 and 1991? List the production years and the corresponding numbers of movies. [Answer]: SELECT production_year, COUNT(*) FROM movie \r\nWHERE production_year>=1991 AND production_year<=1993\r\nGROUP BY production_year;	1

Figure 4: Schema, Question and Answer

of the queries.the prompt used for interacting with the model is structured as follows:

"Given the database schema and the SQL query, determine if the query is syntactically and logically correct. Return 0 if it is not correct, and return 1 if it is correct. Return 0 or 1 only without any explanation. Context: {x}"

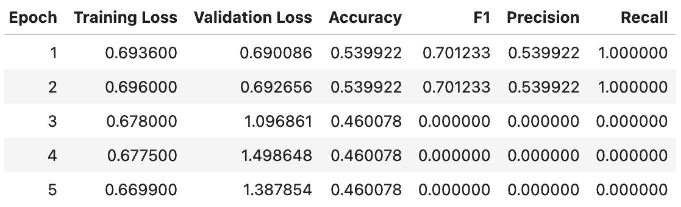
In this prompt, x is a placeholder that is replaced with specific SQL queries and schema details during the execution. This format ensures the model evaluates the queries strictly based on their syntactic and logical attributes in relation to the provided schemas.

Empirical results

In this section, we will detail the experiments conducted, describe the configurations used, and present the corresponding results. Additionally, we will provide an analysis of these results to interpret the outcomes and implications of the experiments in a formal and comprehensive manner.

No Context, Bert

In this experiment, we utilized a dataset derived from the Pandas DataFrame, displayed in Figure 2, which was tokenized and split into an 80/20 train-test ratio. We fine-tuned a 'bert-base-uncased' model specifically adapted for SQL classification over five epochs with a batch size of four, employing Hugging Face's 'Trainer' API to facilitate training and evaluation. Model performance was assessed after each epoch using standard metrics like accuracy, precision, recall, and F1-score, captured during testing. The results, which illustrate the model's performance across these metrics, are comprehensively presented in Figure 5.



Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.693600	0.690086	0.539922	0.701233	0.539922	1.000000
2	0.696000	0.692656	0.539922	0.701233	0.539922	1.000000
3	0.678000	1.096861	0.460078	0.000000	0.000000	0.000000
4	0.677500	1.498648	0.460078	0.000000	0.000000	0.000000
5	0.669900	1.387854	0.460078	0.000000	0.000000	0.000000

Figure 5: Results for No Context, Bert

Analyzing the results in Figure 5, we observe signs of overfitting, characterized by a decrease in training loss accompanied by an increase in validation loss over successive epochs. While the model maintains steady accuracy during the first two epochs, a significant drop in performance metrics—accuracy, F1, precision, and recall—is seen from the third epoch, indicating the model's diminishing generalization capabilities. This is further substantiated by test results on a separate set of 2000 SQL queries from different schemas, where only 1.6% were correctly labeled, reinforcing that the model failed to capture sufficient information to accurately classify a diverse range of SQL queries.

The observed results could be attributed to the randomness inherent in the selection of the training and test sets. To mitigate this and obtain a more reliable performance estimate, we implemented cross-validation with five folds, training the model five separate times and calculating the average score. The outcomes of this process, aimed at providing a more robust evaluation, are depicted in Figure 6.

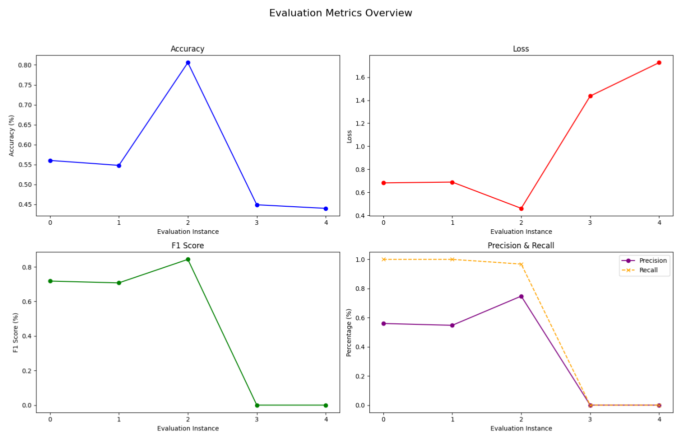


Figure 6: Cross-validation Results for No Context, Bert

The model's performance, as indicated by an average accuracy of 0.5604 and an average F1 score of 0.4539, suggests that it has not learned sufficiently from the dataset, resulting in poor classification outcomes. This inadequate learning points to the need for further model optimization or data preprocessing to ensure the model can capture and generalize from the information within the dataset more effectively.

Question and Answer, Bert

Subsequently, we experimented with incorporating question information into the bert-base-uncase model's training process. By including this additional context, we observed that the majority of the tokenized queries were shorter than the maximum length of 512 tokens allowed by the BERT model. This ensured that truncation did not result in significant information loss. The distribution of the tokenized query lengths, confirming this finding, is illustrated in Figure 7.

In the next phase of our experimentation, we retained a configuration analogous to the initial setup but introduced a few critical changes to the training parameters to better accommodate the modified input data. Specifically, we increased the batch size to 16, hypothesizing that a larger batch

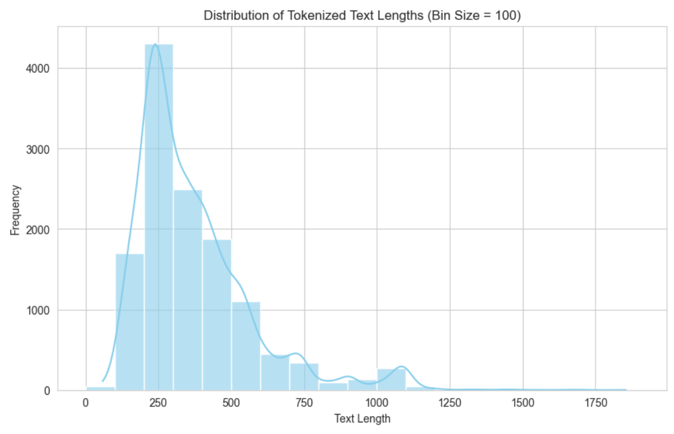
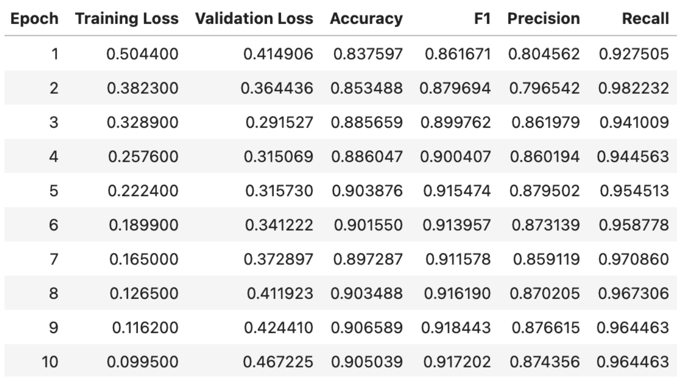


Figure 7: Distribution of Tokenized Text Lengths

would offer more stability during gradient descent and potentially improve the learning process. Additionally, we doubled the number of training epochs to 10, aiming to provide the model with ample opportunity to learn from the augmented dataset, which now included question information alongside the queries. The outcomes of the training, under the revised parameters of an increased batch size and extended number of epochs, are presented in Figure 8.



Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.504400	0.414906	0.837597	0.861671	0.804562	0.927505
2	0.382300	0.364436	0.853488	0.879694	0.796542	0.982232
3	0.328900	0.291527	0.885659	0.899762	0.861979	0.941009
4	0.257600	0.315069	0.886047	0.900407	0.860194	0.944563
5	0.222400	0.315730	0.903876	0.915474	0.879502	0.954513
6	0.189900	0.341222	0.901550	0.913957	0.873139	0.958778
7	0.165000	0.372897	0.897287	0.911578	0.859119	0.970860
8	0.126500	0.411923	0.903488	0.916190	0.870205	0.967306
9	0.116200	0.424410	0.906589	0.918443	0.876615	0.964463
10	0.099500	0.467225	0.905039	0.917202	0.874356	0.964463

Figure 8: Results for Question and Answer, Bert

The training results indicate a consistent improvement in training loss over 10 epochs, decreasing from 0.5044 in the first epoch to 0.0995 in the last. The validation loss also decreases initially but starts to rise after the seventh epoch, suggesting early signs of overfitting. Accuracy improves across epochs, peaking at 90.15% in the sixth epoch before a slight drop-off, indicating the model may have reached its optimal state before epoch seven.

Following individual training sessions, we implemented cross-validation to validate the model's stability, with these results depicted in Figure 9. Cross-validation yielded an average accuracy of 89.97% and an average F1 score of 0.9124, demonstrating a substantially enhanced performance compared to previous iterations. Despite this improvement, when tested on an independent set of 2000 SQL queries from different schemas, the model's correct labeling rate was 77.5%. While this is a significant improvement over prior results, it still suggests room for optimization to further improve the model's generlization ability on varied schemas.

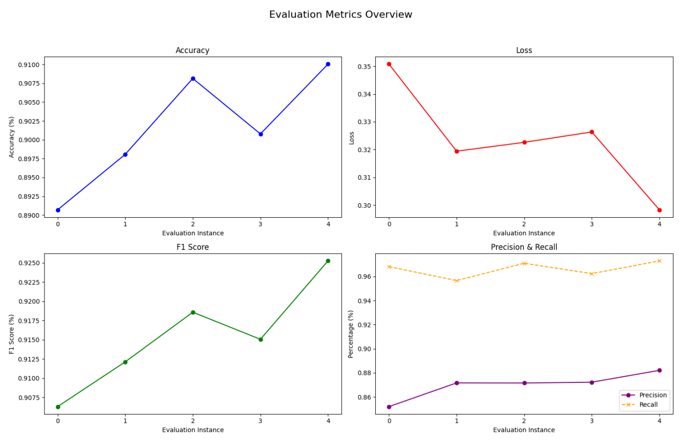


Figure 9: Cross-validation Results for Question and Answer, Bert

Following the cross-validation process, we conducted an error analysis to delve deeper into the model's performance nuances. The confusion matrix, which provides insights into the types of prediction errors made, is illustrated in Figure 10. Additionally, we examined the instances of incorrect predictions that resulted in the highest loss, with these particular cases being highlighted in Figure 11.

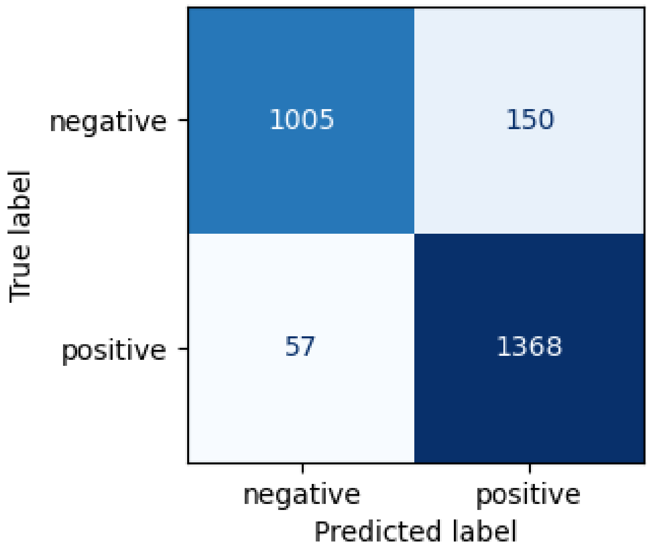
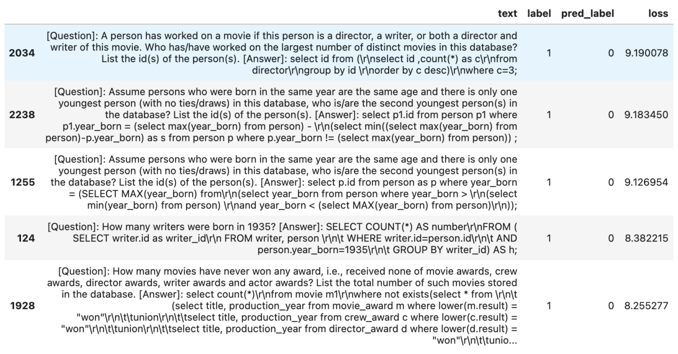


Figure 10: Confusion Matrix

The confusion matrix indicates a proficient model, with a substantial count of true positives and true negatives. The presence of more false positives than false negatives suggests a tendency of the model to incorrectly predict some negative instances as positive. This aspect necessitates a closer examination of the model's decision-making to refine its predictive accuracy, especially in distinguishing negative cases.



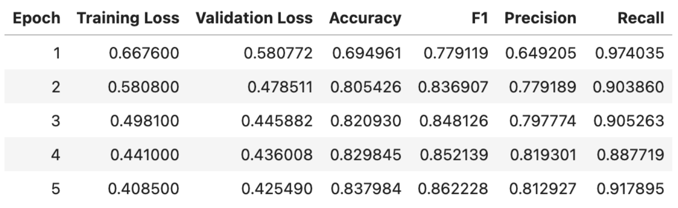
	text	label	pred_label	loss
2034	[Question]: A person has worked on a movie if this person is a director, a writer, or both a director and writer of this movie. Who has/have worked on the largest number of distinct movies in this database? List the id(s) of the person(s). [Answer]: select id from (\r\nselect id ,count(*) as c\r\nfrom director\r\ngroup by id \r\norder by c desc)\r\nwhere c=3;	1	0	9.190078
2238	[Question]: Assume persons who were born in the same year are the same age and there is only one youngest person (with no ties/draws) in this database, who is/are the second youngest person(s) in the database? List the id(s) of the person(s). [Answer]: select p1.id from person p1 where p1.year_born = (select max(year_born) from person) - \r\n(select min((select max(year_born) from person)-p.year_born) as s from person p where p.year_born != (select max(year_born) from person)) ;	1	0	9.183450
1255	[Question]: Assume persons who were born in the same year are the same age and there is only one youngest person (with no ties/draws) in this database, who is/are the second youngest person(s) in the database? List the id(s) of the person(s). [Answer]: select p.id from person as p where year_born = (SELECT MAX(year_born) from\r\n(select year_born from person where year_born > \r\n(select min(year_born) from person) \r\nand year_born < (select MAX(year_born) from person)\r\n));	1	0	9.126954
124	[Question]: How many writers were born in 1935? [Answer]: SELECT COUNT(*) AS number\r\nFROM (SELECT writer.id as writer_id\r\n FROM writer, person \r\n\t WHERE writer.id=person.id\r\n\t AND person.year_born=1935\r\n\t GROUP BY writer_id) AS h;	1	0	8.382215
1928	[Question]: How many movies have never won any award, i.e., received none of movie awards, crew awards, director awards, writer awards and actor awards? List the total number of such movies stored in the database. [Answer]: select count(*)\r\nfrom movie m1\r\nwhere not exists(select * from \r\n\t (select title, production_year from movie_award m where lower(m.result) = "won"\r\n\t\tunion\r\n\t\tselect title, production_year from crew_award c where lower(c.result) = "won"\r\n\t\tunion\r\n\t\tselect title, production_year from director_award d where lower(d.result) = "won"\r\n\t\tunio...	1	0	8.255277

Figure 11: Wrong Prediction with Highest loss, Bert

The model tends to mispredict with longer inputs, likely because segments exceeding the 512-token limit were truncated. Given the data structure—prefixing inputs with [Question]: and [Answer]:—it's probable that truncation impacted the answer segment, leading to information loss. This truncation could explain the poorer performance on extended inputs. Prior to exploring larger models capable of handling longer sequences, we opted to experiment with RoBERTa to assess any performance improvements with its architecture.

Question and Answer, RoBERTa

We proceeded to fine-tune the RoBERTa-base model, utilizing both questions and answers in the training data. The outcomes of this fine-tuning process are presented in Figure 12.

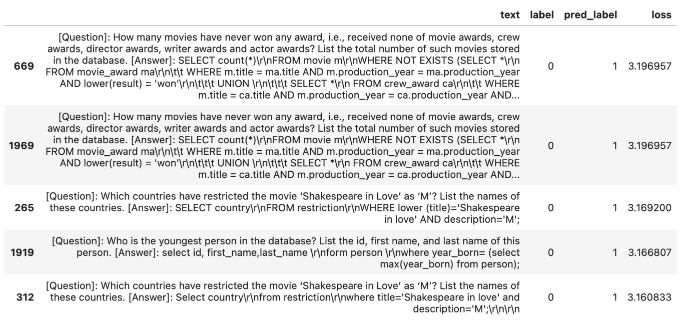


Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.667600	0.580772	0.694961	0.779119	0.649205	0.974035
2	0.580800	0.478511	0.805426	0.836907	0.779189	0.903860
3	0.498100	0.445882	0.820930	0.848126	0.797774	0.905263
4	0.441000	0.436008	0.829845	0.852139	0.819301	0.887719
5	0.408500	0.425490	0.837984	0.862228	0.812927	0.917895

Figure 12: Results for Question and Answer, RoBERTa

The fine-tuning results for RoBERTa-base show a decreasing trend in training and validation loss across epochs. However, the performance metrics suggest RoBERTa-base underperformed relative to BERT-base, with a final accuracy of approximately 83.8% and an F1 score of 86.2%. This indicates that despite its advanced architecture, RoBERTa-base did not achieve the same level of effectiveness on this SQL classification task as the BERT-base model.

Following the training phase, an error analysis was carried out to examine the incorrect predictions that incurred the highest loss, detailed in Figure 13.



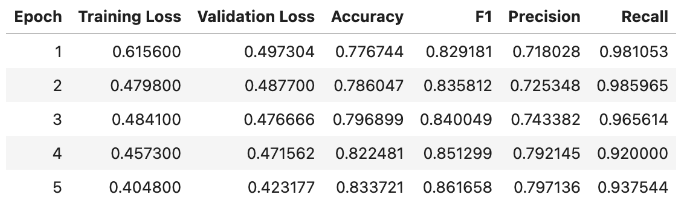
	text	label	pred_label	loss
669	[Question]: How many movies have never won any award, i.e., received none of movie awards, crew awards, director awards, writer awards and actor awards? List the total number of such movies stored in the database. [Answer]: SELECT count(*)\r\nFROM movie m\r\nWHERE NOT EXISTS (SELECT *\r\n FROM movie_award ma\r\n\t\t WHERE m.title = ma.title AND m.production_year = ma.production_year AND lower(result) = 'won'\r\n\t\t\t UNION \r\n\t\t\t SELECT *\r\n FROM crew_award ca\r\n\t\t WHERE m.title = ca.title AND m.production_year = ca.production_year AND...	0	1	3.196957
1969	[Question]: How many movies have never won any award, i.e., received none of movie awards, crew awards, director awards, writer awards and actor awards? List the total number of such movies stored in the database. [Answer]: SELECT count(*)\r\nFROM movie m\r\nWHERE NOT EXISTS (SELECT *\r\n FROM movie_award ma\r\n\t\t WHERE m.title = ma.title AND m.production_year = ma.production_year AND lower(result) = 'won'\r\n\t\t\t UNION \r\n\t\t\t SELECT *\r\n FROM crew_award ca\r\n\t\t WHERE m.title = ca.title AND m.production_year = ca.production_year AND...	0	1	3.196957
265	[Question]: Which countries have restricted the movie 'Shakespeare in Love' as 'M'? List the names of these countries. [Answer]: SELECT country\r\nFROM restriction\r\nWHERE lower (title)='Shakespeare in love' AND description='M';	0	1	3.169200
1919	[Question]: Who is the youngest person in the database? List the id, first name, and last name of this person. [Answer]: select id, first_name,last_name \r\nform person \r\nwhere year_born= (select max(year_born) from person);	0	1	3.166807
312	[Question]: Which countries have restricted the movie 'Shakespeare in Love' as 'M'? List the names of these countries. [Answer]: Select country\r\nfrom restriction\r\nwhere title='Shakespeare in love' and description='M';\r\n\r\n	0	1	3.160833

Figure 13: Wrong Prediction with Highest loss, RoBERTa

Unlike the BERT model which showed a tendency towards false negatives as the primary source of high-loss errors, the error pattern for the RoBERTa model was notably different, with false positives accounting for the majority of high-loss instances. Interestingly, the magnitude of loss associated with these high-error predictions in RoBERTa was less than that observed in BERT's case. This difference suggests that while RoBERTa may be more prone to incorrectly classifying negative instances as positive, the errors it does make are less severe than those made by BERT, possibly indicating a more conservative prediction approach or differences in how each model processes and learns from the data.

Schema, Question and Answer, Big Bird

We then proceeded to fine-tune the BigBird-RoBERTa-base model, which supports a maximum sequence length of 4096 tokens, allowing for the processing of longer inputs. With this capability, we incorporated schema information into the training data to provide the model with a richer learning context. The results of this fine-tuning, which reflect the model's performance with the additional schema data, are shown in Figure 14.



Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.615600	0.497304	0.776744	0.829181	0.718028	0.981053
2	0.479800	0.487700	0.786047	0.835812	0.725348	0.985965
3	0.484100	0.476666	0.796899	0.840049	0.743382	0.965614
4	0.457300	0.471562	0.822481	0.851299	0.792145	0.920000
5	0.404800	0.423177	0.833721	0.861658	0.797136	0.937544

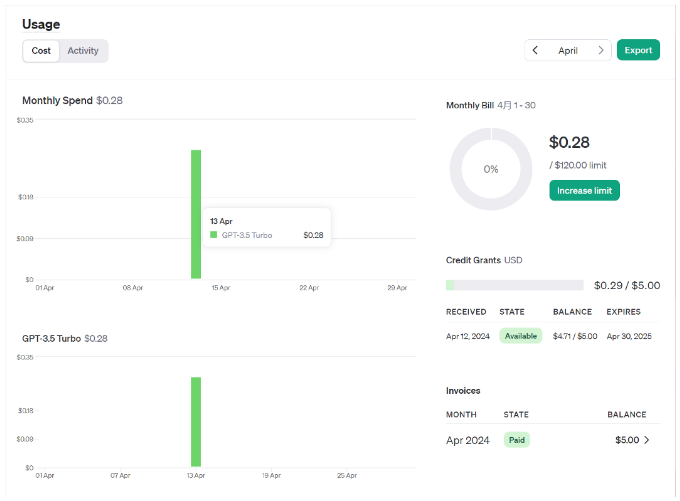
Figure 14: Results for Schema, Question and Answer,Big Bird

The fine-tuning results of the BigBird-RoBERTa-base model exhibit a steady decrease in training loss across five epochs, with validation loss also trending downwards. Despite the model achieving a lower accuracy compared to the earlier BERT and RoBERTa models, when it was tested against a diverse set of 2000 SQL queries from various schemas, it correctly labeled 85.6% of them. This notable result suggests that BigBird-RoBERTa's ability to handle longer inputs—and thus retain more contextual information like table names, primary keys (PK), foreign keys (FK), and other attributes—enhances its generalization capacity. Due to the quadratic complexity of the self-attention mechanism in the transformer model (n^2), we opted out of cross-validation to manage computational demands. The model's robust performance on the separate test set indicates that the additional context from schema information contributes positively to its ability to generalize across different SQL schemas.

Prompt Engineering, GPT-3.5 turbo

In our approach to utilizing GPT-3.5 Turbo for SQL query evaluation, we engaged in prompt engineering. A custom function was created to dispatch structured prompts, crafted to instruct the model to act as a SQL expert. These prompts were paired with user queries, incorporating schema, question, and answer information.

The model was prompted to evaluate the syntactic and logical validity of the SQL queries and to provide a binary judgment—0 for incorrect and 1 for correct—without further explanation. Upon testing the model's responses, we achieved an accuracy of 70.75%, which shows the model's adeptness at classifying queries accurately. This testing process incurred a cost of $0.28, indicating a cost-effective approach to automating SQL query evaluation. The usage for this Prompt Engineering process is displayed in Figure 15.



RECEIVED	STATE	BALANCE	EXPIRES
Apr 12, 2024	Available	$4.71 / $5.00	Apr 30, 2025

MONTH	STATE	BALANCE
Apr 2024	Paid	$5.00

Figure 15: Usage of Prompt Engineering

Broader Implications

This project's application of AI for grading SQL queries has significant implications for educational efficiency, potentially allowing educators to focus on more nuanced teaching tasks. It illustrates AI's capacity to interpret complex information, which could translate to broader programming and syntax-based evaluations.

However, this also raises questions about the role of AI in education, particularly regarding the balance between automated feedback and essential human interaction in learning. It underscores the need to address access disparities and data privacy concerns, ensuring that such technologies serve to enhance, not hinder, educational equity and ethical standards. As AI becomes more integrated into pedagogical tools, it's crucial to navigate these challenges thoughtfully to maximize benefit and minimize unintended consequences.

Conclusions / Future Directions

In conclusion, the project has yielded insightful results in the automated evaluation of SQL queries. We found that incorporating schema, question, and answer information into the Big Bird model outperformed GPT-3.5 Turbo for this specific task, showcasing impressive generalization capabilities. Notably, all the models discussed throughout the project have been published on the Hugging Face Model Hub. They've garnered significant attention, as evidenced by around 50 downloads, indicating the project's relevance and utility within the community. These models can be accessed through this link, enabling further exploration and use by others in the field.

Throughout the course of this project, I've gained a deeper understanding of the architecture and mechanisms underpinning Large Language Models (LLMs). I learned to navigate the Hugging Face Transformers library, which is an invaluable toolkit for anyone working with state-of-the-art language models. My skills in utilizing PyTorch, an open-source machine learning library, were also enhanced, providing the flexibility and power needed to train complex models.Additionally, the project offered practical experience with the OpenAI API, which is a robust platform for interacting with models like GPT-3.5 Turbo. Through this API, we were able to apply prompt engineering techniques and analyze the results of model responses.

These skills are crucial for the development and fine-tuning of models to suit specific tasks and will be invaluable for future projects.

Looking to the future, the recent release of the Llama 3 model by Meta, known for its proficiency across various tasks, including coding questions, presents an exciting opportunity. The next steps would involve experimenting with this open-sourced model to see if it can further improve on the task at hand. Additionally, expanding the dataset could provide a richer training environment and potentially enhance the model's accuracy and robustness.

For students embarking on DS 5983 or similar projects, my advice would be to start by familiarizing with the Hugging Face Transformers library early on. Understanding transformer architecture is fundamental to effectively leveraging these tools. Moreover, immersing in the latest research and reading papers in the field can spark ideas and provide a deeper understanding of advanced techniques, which are instrumental in driving innovation and success in such projects.

GitHub Link

SQL-grading