Natural Language Models for Data Visualization Utilizing nvBench Dataset

Shuo Wang and Carlos Crespo-Quinones

DATASCI 266: Natural Language Processing
UC Berkeley School of Information
{shuo.wang, carlos.d.crespo}@ischool.berkeley.edu

Abstract

Translation of natural language into syntactically correct commands for data visualization is an important application of natural language models and could be leveraged to many different tasks. A closely related effort is the task of translating natural languages into SQL queries, which in turn could be translated into visualization with additional information from the natural language query supplied[1]. Contributing to the progress in this area of research, we built natural language translation models to construct simplified versions of data and visualization queries in a syntax called Vega Zero first proposed by Luo, Yuyu, et al[2]. In this paper, we explore the design and performance of these sequence to sequence transformer based machine learning model architectures using large language models such as BERT as encoders to predict visualization commands from natural language queries, as well as apply available T5 sequence to sequence models to the problem for comparison.

Introduction

Data visualization is an integral part of data analysis and reporting, highly skilled professionals spend a significant portion of their working hours constructing data queries and turning them into charts and graphs. The potential application of machine learning models to understand the underlying data and translate natural language queries into data visualization would greatly enhance the efficiency and productivity of many data analysis tasks. Not only would such tools enhance our existing ability to understand and communicate with our data, it could also help us uncover previously hidden information.

Some immediate applications include: creating bar, line and scatter plots of sql tables, transform and group data with a table and display the aggregated data and joining and combining tables of data to extract information and display visually.

Previous research in this area has included explorations of transforming natural language into SQL commands[1, 3] with deep learning models and natural language to visualization interfaces[4]. NvBench is a new dataset created to facilitate the research to further integrate the process of natural language to data query and data query to visualizations[2] . Leveraging this data set and the ncNet transformer model architecture[5], we further explore the various modeling possibilty in our work.

In this paper, we have used the nvBench as our train, validation and test dataset to create BERT encoder based multi-head attention transformer models and compare the performance of the new models with the performance of the ncNet model (also a sequence to sequence transformer model). Our goal is to explore the possibility of a generalized natural language to visualization process where the ability of the system to handle natural language inputs is not limited by the input training data. We have provided below the main results of our research and detailed description and analysis of the input data, model architecture and model performance. From our reseach, the BERT encoder based sequence to sequence transformer model was able to achieve an overall accuracy of 79%, confirming our hypothesis of the possibility to leverage transferred learning of BERT models to data visualization tasks. Applying a pretrained CodeT5 model realizes even more impressive results, achieving overall accuracy of 97%.

Background

Natural language to visualization research has been conducted for decades, some of these efforts include early approaches to manually program rule-based logic for the handling of anticipated user inputs in multi-model systems[4] and more recent attempts to incorporate machine learning and optimization techniques[6]. However these systems are limited in functionality by the ability of the designer of the systems to anticipate the possible inputs from users. What if the user uses a word that the system has never seen? What if the natural language input was written in an idiomatic way that was not anticipated by the training data? These limitations could potentially be addressed with the recent advances in large language models such as BERT and GPT through

transfer learning.

Recent research exists for applying BERT model to the task of data visualization[7] by converting natural language into vector representation embeddings and use these embeddings to predict various ingredients needed for the visualization of tabular data such as chart type, data columns and aggregation type. However, the structured approach to the problem inherently limits the expressive power of the system to generate complex visualizations.

A new dataset, nvBench[2], was made publicly available for research purposes in 2021. The dataset contains pairs of natural language queries and data visualization commands in vega zero syntax[5]. A self-contained sequence-to-sequence model, ncNet[5], created for and trained on the dataset was also made available. We base our research on this dataset and the ncNet transformer model, modifying the architecture to use BERT and other language models to train and test generalizable natural language to data visualization models, eventually applying the pre-trained T5 model to the problem and achieve superior results.

Methods

Data

nvBench The dataset contains 7247 visualization queries (labels). Each query contains a visualization type and data retrieval command meant to be run against an underlying database with multiple tables, which contain the actual data to be displayed. Below is an example of a visualization query:

> Visualize BAR SELECT JOB_ID , SUM(MANAGER_ID) FROM employees WHERE salary BETWEEN 8000 AND 12000 AND commission_pct != "null" OR department_id != 40 GROUP BY JOB_ID ORDER BY JOB_ID ASC

In this example, the chart type is “BAR”, followed by the SQL statement for data retrieval. Figure 1 shows the rendering result of the query.

The example visualization query is mapped to several natural language queries, from which we train our models:

> 1. For those employees whose salary is in the range of 8000 and 12000 and commission is not null or department number does not equal to 40, show me about the distribution of job_id and the sum of manager_id , and group by attribute job_id in a bar chart, I want to sort in ascending by the bar.

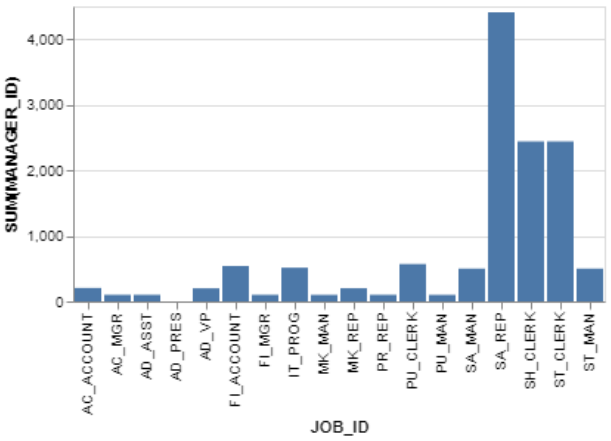

Figure 1: *Rendering: Visualize BAR SELECT JOB_ID , SUM(MANAGER_ID) FROM employees WHERE salary BETWEEN 8000 AND 12000 AND commission_pct != "null" OR department_id != 40 GROUP BY JOB_ID ORDER BY JOB_ID ASC.*

> 2. For those employees whose salary is in the range of 8000 and 12000 and commission is not null or department number does not equal to 40, a bar chart shows the distribution of job_id and the sum of manager_id , and group by attribute job_id, and could you order x axis in asc order?

In total, there are 25762 natural language queries, averaging 3.55 natural language queries for every visualization query.

Augmented Data Due to the complexity of SQL syntax, a simplified version of the visualization query is used in place of the original query, called vega zero[5]:

> mark bar data employees encoding x hire_date y aggregate count hire_date transform filter salary between 8000 and 12000 and commission_pct != "null" or department_id != 40 bin x by month

Where the visualization query is flattened into a sentence with special markers designating locations of information, please refer to [5] for complete syntax. In our research, we follow the same syntax rules.

Since this syntax only supports single table queries, it is not possible to predict queries where the joining of tables are necessary, so we only train and test our models on a subset of nvBench dataset, with 2988 visualization queries for training, 186 visualization queries for validation and 625 visualization queries for test.

The natural language queries are also augmented to include the template of vega zero queries:

<N> For those employees whose salary is in the range of 8000 and 12000 and commission is not null or department number does not equal to 40 , draw a line chart about the change of department_id over hire_date , display by the X from low to high . </N> <C> mark [T] data employees encoding x [X] y aggregate [AggFunction] [Y] color [Z] transform filter [F] group [G] bin [B] sort [S] topk [K] </C> <D> employees <COL> hire_date salary department_id last_name first_name </COL> <VAL> Bull Lex Seo Bell Chen Lee Gee Banda King Baer Fay </VAL> </D>

The natural language query is enclosed in <N> </N> markers, the vega zero template is enclosed in <C> </D> markers and the locations of vega zero query fields are marked by special tokens: [T], [X], [Y], [Z], [F], [G], [B], [S] and [K]. These are the syntax used by the original ncNet sequence to sequence transformer models and we follow the same rules for our models as well. Finally, every natural language and visualization query pair in the dataset is augmented with two versions, one with the chart type kept as placeholder, [T], and another with the chart type filled in (bar, line, etc). The motivation is to train the model to predict the visualization query whether the chart type is explicitly specified or not.

Data Loader The augmented data were provided in the Github repo for the ncNet paper[5] , which we use unaltered. However, the original data loader no longer works the latest version of Pytorch, we rewrote the data loader to be used for training and testing the ncNet model and our own models.

Model

ncNet The ncNet[5] model is a sequence to sequence multi-head transformer model, where the encoder transforms the natural language queries into embeddings combining position and token type information, and apply multiple layers of multi-head attention transformers to incorporate contextual information into the encoded embeddings. Shown in Figure 2, The decoder then takes in the encoded embeddings and the embeddings of the previously predicted tokens to further transform the predicted tokens with attention to the encoded embeddings. We use the ncNet model as our baseline, and replace the encoder with various different architectures.

One important fact to note is that the ncNet model uses all words occurring in the training, testing and validation dataset as vocabulary. Therefore letting the possibility of natural language query inputs.

We also removed the attention mask (referred to as attention forcing in Luo 2022[5]) used in ncNet to create another baseline for our experiments, since attention masks are not applicable to our models.

BERT Model We then proceeded to replace the ncNet encoder with the BERT model encoder, as shown in Figure 3. Because the BERT model was pre-trained with a much larger vocabulary, the hope was that this knowledge would transfer to the new model, ncBERT, once we fine-tuned it. We trained an additional BERT encoder model with convolutional layers added to the BERT embeddings, in order to distill relevant information from BERT model embeddings before decoding, as the BERT embedding has a much larger dimension that may contain information irrelevant to the data visualization task. The convolutional layers incorporated model is shown in Figure 4 .

Combining ncNet and BERT Model Furthermore, we created an encoder combining both the ncNet and BERT encoders, we achieved this by reshaping the embeddings from BERT to the same dimension as ncNet encoder and concatenating them together. The architecture of the model could be found in Figure 5. The motivation of creating this model was to analyze the effect of BERT embedding on the baseline model and explore the potential of enhancing the original encoder without completely replacing it. However, the implication of this approach was that words that did not exist in the original ncNet encoder still could not be used, further research needs to be done to remove this limitation.

CodeT5 Finally, a series of experiments were conducted by fine-tuning two different families of the pre-trained T5 model on the nvBench dataset. The first set of models was conducted using variations of the original T5 model [8] by Raffel et al. and the second set of experiments was based on the Code-T5 model family [9] by Wang et al., which is a variation of the T5 model pre-trained for code specific tasks. The experiments included using models of different parameter sizes and modifying the input by including or not, the query templates along with the input NL request. The results are very encouraging, where we achieved 98% prediction accuracy and 88% guided search accuracy using the “large” (700M parameter) version of the CodeT5 model and using the same input sequence as the other models which included both the template and the NL request.

Evaluation

The models are trained and tested with training dataset of size 25238, validation dataset of size 1430 and test dataset of size 4920. Each model is run over

Table 1: *Model test accuracy.*

Model	Query	Query+Chart	Overall
ncNet	95%	96%	96%
ncNet w/o AF	95%	96%	96%
nvBERT	79%	79%	79%
nvBERT_CNN	79%	79%	79%
nvncNetBERT	89%	89%	89%
codeT5 w/o TPT	97%	n/a	97%
codeT5	97%	97%	97%
codeT5 Large	98%	98%	98%

at least 5 epochs with a learning rate of 0.0005 and the resulting losses are recorded. Then the accuracy of the models are evaluated by running predictions with the models over the test dataset, with the natural language query tokens and the first n-1 tokens of the label as input (the label tokens are masked so that successive predictions of label are not affected by future label tokens), counting the total number of correct predictions and dividing it by the total number of predictions.

Finally, we evaluate the models one more time with a guide search algorithm to predict. The algorithm starts the prediction with the start of the sentence token, then repeatedly predicts the next label token with guidance until the end of the sentence token is reached. This evaluation provides a more realistic measure of the model's usability. The accuracy from this evaluation is defined as the total number of complete label matches over the total number of test labels.

Results and Discussion

Accuracy

After running 5 epochs with learning rate 0.0005 and keeping the model version with the lowest validation loss, the test accuracy of the models are reported in Table 1. The accuracy of a model is defined as the probability of predicting the next word correctly given part of the label query.

The table shows three columns, the accuracy of the models on queries where chart type is not specified (Query), the accuracy of the models on queries where chart type is specified (Query+Chart) and the overall accuracy (Overall).

As we can see from the results, the original ncNet models produced test accuracies in excess of 95%. While the BERT based transformer models were able to achieve close to 80% accuracy. This is attributable to the fact that the BERT model contains a much larger vocabulary than the ncNet model trained only on the vocabulary of tokens present in the dataset.

Furthermore, the removal of attention forcing in the ncNet model does not appear to affect the test accuracy, suggesting that its absence in nvBERT models had no bearing on the accuracy results of the nvBET models. The accuracy of nvBERT models with and without convolutional layers does not appear to affect the accuracy at all, suggesting minimal impact of the convolutional layers. Finally the combined ncNet and BERT encoder produced results that improved upon the nvBERT only models, suggesting that were a model able to combine the ncNET and BERT encoder while remaining generalizable, the results would improve along with the benefits of transfer learning.

CodeT5 models achieved superior results across the board, surpassing both the original ncNet model as well as the BERT encoder integrated models. What is more exciting is the fact that, even without templates as input, codeT5 models are still able to achieve comparable results.

Besides the overall accuracy difference, the general trend is that the accuracy for "query only tests are slightly" lower than accuracy for "query and chart type" tests. This is expected since the query only tests require the model to predict an additional field. However, given that the accuracy differences are small, the chart type prediction is generally accurate.

Guided Search Accuracy

We next look at the accuracy of predicting the entire label query. The algorithm of guided search, as used by the original ncNet model, proceeds as follows: starting with the start of sentence token for the label, successively predict the top five candidates. Within the top five candidates, the top candidate is chosen unless it does not belong to the possible words for the current position being predicted, in which case the rest of the candidates are iterated over to find a suitable choice. The predict is counted as correct if the resulting predicted sentence matches the label sentence completely. Table 2 shows the results for each model.

From looking at the results, it's immediately clear that the lower accuracy in the BERT based models are amplified through the guided search process, resulting in a low overall accuracy of 7.8%. This effect also shows up in the difference between query only and query with chart type accuracy, where the small difference in Table 1 becomes much larger in Table 2.

The codeT5 models again show great results. However, the model trained without template now shows reduced accuracy, suggesting that template input

Table 2: *Model test accuracy with guided search.*

Model	Query	Query+Chart	Overall
ncNet (baseline)	62%	72%	67%
ncNet w/o AF	65%	75%	70%
nvBERT	3%	13%	8%
nvBERT_CNN	2%	12%	7%
nvncNetBERT	21%	33%	27%
codeT5 w/o TPT	83%	n/a	83%
codeT5	86%	87%	86%
codeT5 Large	89%	88%	88%

helps the guided search to improve the final prediction results.

Sample Analysis

nvBERT Given the large discrepancy between overall accuracy and the guided search accuracy due to the flaws of the respective metrics, we need a better understanding of the true performance of the new models with BERT encoders. One of the things to observe is that in the predicted query, many of the words are part of the template and rarely changes. These words need to be separated out because the correction prediction of them mostly requires the model to simply be position aware. As an example:

> Query: <N> Plot how many booking start date by grouped by booking start date as a bar graph , order Y in ascending order . </N> <C> mark [T] data apartment_bookings encoding x [X] y aggregate [AggFunction] [Y] color [Z] transform filter [F] group [G] bin [B] sort [S] topk [K] </C> <D> apartment_bookings <COL> booking_start_date </COL> <VAL> </VAL> </D>
>
> Label: mark bar data apartment_bookings encoding x booking_start_date y aggregate count booking_start_date transform sort y asc bin x by weekday <eos>

Words such as "mark", "data", "encoding", "aggregate" and "transform" always appear in the label and in fixed positions, the model is able to predict them almost perfectly. Words that are also part of the template such as "x", "y", "color", "filter", "group", "bin", "sort" and "topk" may not occur in the label or may occur more than once, are harder to predict.

Table 3 shows the count of words that are part of template vs non-template, on average, there are more template words than non-template words, which partially account for why word prediction accuracy is much higher than guided search accuracy.

Table 3: *Count of words that are template and non-template.*

Count	TPT	Non-TPT	Total
mean	10.5	8.5	18.9
min	6	4	12
max	13	22	34

Table 4: *Incorrect prediction for easy template, hard template, total template and non-template prediction count.*

Incorrect	Easy	Hard	TPT	Non-TPT
mean	0	0.6	0.6	3.4
min	0	0	0	1
max	1	4	4	12

Table 4 shows the summary statistics for the count of incorrect predictions made by the nvBERT model on the easy template words, hard template words and non-template words. The rate of incorrect prediction is much higher for hard template words than easy template words, as expected. The third column shows the statistics for incorrect prediction of non-template words, curiously the minimum of incorrect prediction is 1, which indicates that every prediction has at least one incorrectly predicted word, yet the guided search did result in completely correct predictions. The cause behind this apparent discrepancy is that for the guided search, predicted words are checked for whether it is possible for it to occupy the position. Below shows an example:

> Query: <N> List all the possible ways to get to attractions , together with the number of attractions accessible by these methods in a bar chart , and sort names in asc order please . </N> <C> mark [T] data tourist_attractions encoding x [X] y aggregate [AggFunction] [Y] color [Z] transform filter [F] group [G] bin [B] sort [S] topk [K] </C> <D> tourist_attractions <COL> name how_to_get_there </COL> <VAL> walk bus </VAL> </D>
>
> Label: mark bar data tourist_attractions encoding x how_to_get_there y aggregate count how_to_get_there transform group x sort x asc <eos>
>
> Prediction: mark bar data employees encoding x how_to_get_there y aggregate count how_to_get_there transform group x sort x asc <eos>

Although the prediction has incorrectly predicted "employees" instead of "tourist_attractions", "employees" is not the name of the table under consideration, therefore guided search chose the word that

Table 5: *nvBERT prediction accuracy breakdown.*

Easy	Hard	TPT	Non-TPT	Total
100%	89%	94%	60%	79%

Table 6: *nvBERT chart type prediction accuracy breakdown.*

Query	Query+Chart
19%	19%

matches the table name instead, resulting in a correct prediction.

Table 5 shows the breakdown of nvBERT model prediction accuracy by category. As expected, template words prediction accuracy is much higher than non-template words.

Finally, Table 6 shows the accuracy of the nvBERT model at predicting the chart type, "arc", "bar", "line" or "point". The accuracy is broken down by whether the chart type is already provided by the source query, interestingly, providing chart type in source query does not improve the accuracy of predicting chart type, which is also reflected in Table 1. However, the provided chart type does help guided search to determine whether the predicted chart type is valid or not, thereby improving the guided search accuracy 2.

CodeT5 For inference, the standard model decoder was used. Even though certain words such as table names and available column choices were encouraged via including them with the model input, no output tokens were forced. With this in mind the results from fine-tuning the CodeT5 model has been phenomenal, achieving a final token accuracy of 98%. The model showed a good ability to predict the general structure of the query with a 96% success rate predicting query keywords (i.e. "count", "group", "transform", etc.) and in 93% of the cases, was able to correctly identify the column names. A results review shows that in the majority of the cases, the error was semantic or abbreviation related. For example, for a label column name of 'market_val' the model would predict 'market_value'. In other cases the error was more related to the way the question was asked and the column name. For instance the model would predict "profit" instead of "profit_dollars" because the question would ask for profits without mentioning dollars.

When comparing exact query matches as shown in Table 7, the CodeT5 model consistently achieved high performance across difficulty levels, suggesting that the complexity of language and label query structure is not a significant limiting factor in its performance. A quick examination of the results show that in many

Table 7: *CodeT5 performance breakdown by difficulty level.*

Hardness	Success	Failure	Success Rate
Easy	1451	147	91%
Medium	1869	233	89%
Hard	605	135	81%
Extra Hard	384	52	88%

Table 8: *CodeT5 success rate before and after error correction.*

Before Correction	After Correction
83%	88%

of the cases the errors were systemic in nature and could easily be corrected by a series of logical filters. Some of these errors involved unmatched '%' signs which are used for wildcards. For instance, to match values with the letter "a" the pattern would be '%a%'. The other most common error involved a lack of a white-space preceding the unequal operator '!='. Please refer to the appendix for examples. In fact, fixing this one error improved the query accuracy from 83% to 88% as can be seen in Table 8. The rest of the errors were more nuanced in nature and involved commonly being unable to correctly predict the transform sections which tended to be the most complex. In this area the CodeT5 model only achieved a 90% success rate.

Conclusion

Through our research and analysis, we have demonstrated the potential of transfer learning through the use of the BERT model as encoder for natural language to visualization tasks, specifically applied to the nvBench dataset. With the result of the CodeT5 model trained and tested on the nvBench dataset we see that the problem could be adequately modeled as a natural language to code translation problem. With more sophisticated architecture and more comprehensive data visualization language, it is clear that natural language to data visualization translation could be done very effectively and in a general manner with existing large language models.

Of course, there are still many areas that remain to be explored. First, a more generic visualization language needs to be developed to accommodate the more complex requests, the design of this language should be guided by the effectiveness with which it could be predicted by language models. Second, more database design should take into account the need for visualization, and integrate machine learning translation as part of the research and development process. We hope our research would

contribute to the progress of these initiatives.

References

[1] Victor. Zhong et al. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103v7 [cs.CL]*, November 2017.

[2] Yuyu. Luo et al. nvbench: A large-scale synthesized dataset for cross-domain natural language to visualization task. *arXiv:2112.12926v1 [cs.HC]*, December 2021.

[3] Tao. Yu et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv:1909.05378v1 [cs.CL]*, September 2019.

[4] Kenneth. Cox et al. A multi-modal natural language interface to an information visualization environment. *International Journal of Speech Technology*, 2001.

[5] Yuyu. Luo et al. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics, vol. 28, no. 1, pp. 217-226*, January 2022.

[6] Jillian. Aurisano et al. Articulate2: Toward a conversational interface for visual data exploration. *IEEE VIS '16 (Poster paper)*, January 2016.

[7] Can. Liu et al. Advisor: Automatic visualization answer for natural-language question on tabular data. *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, January 2021.

[8] Colin. Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

[9] Yue. Wang et al. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *Conference on Empirical Methods in Natural Language Processing*, 2021.

APPENDIX

Model Architectures

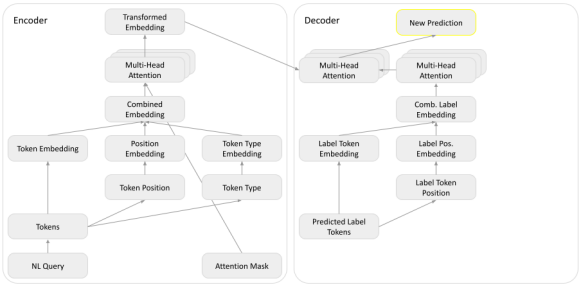

Figure 2: *Architecture of ncNet model.*

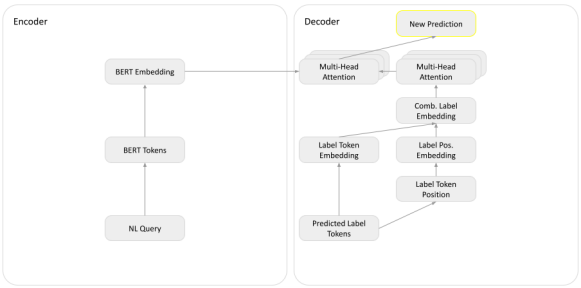

Figure 3: *Architecture of nvBERT model.*

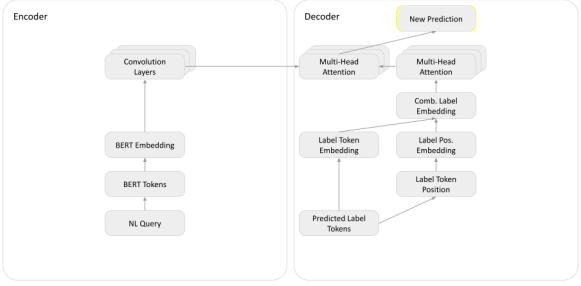

Figure 4: *Architecture of nvBERT_CNN model.*

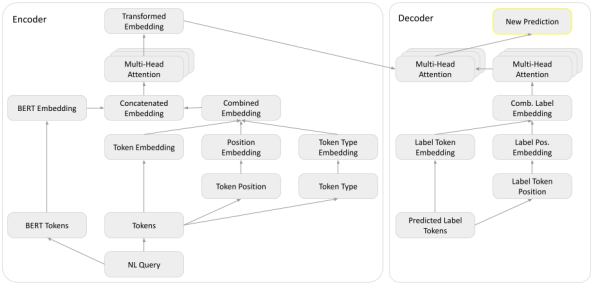

Figure 5: *Architecture of nvncNetBERT model.*

CodeT5 Error Examples

We further examine some of the cases where the codeT5 model fails to predict the label exactly:

> Label: mark bar data customer encoding x cust_name y aggregate none acc_bal transform filter cust_name like '%a%' sort y desc
>
> Prediction: mark bar data customer encoding x cust_name y aggregate none acc_bal transform filter cust_name like '%a' sort y desc

In the first example, the predicted vega zero query is missing a "%" sign after "%a".

> Label: mark bar data employees encoding x hire_date y aggregate count hire_date transform filter salary between 8000 and 12000 and commission_pct != "null" or department_id != 40 sort y asc bin x by 'weekday
>
> Prediction: mark bar data employees encoding x hire_date y aggregate count hire_date transform filter salary between 8000 and 12000 and commission_pct!= "null" or department_id!= 40 sort y asc bin x by weekday

In the second example, a space is missing before "!=". We have systematically corrected these mistakes for our final success rate. Table 8 shows the improvement of results.

Training and Loss

The results of train and validation loss for ncNet, ncNet without attention forcing, nvBERT, nvBERT_CNN and nvncNetBERT are shown for the first 5 epochs in Figure 6. The BERT encoder models show much smaller drop in train loss from epoch to epoch, due to transfer learning that occurs. However the BERT encoder models do not show significant drop in validation loss, suggesting that most of the fine-tuning has happened in the first epoch.

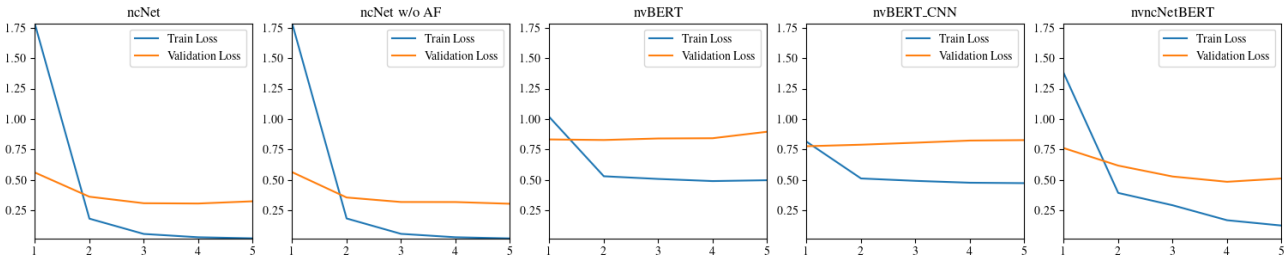

Figure 6: *Train and validation loss of models.*