PA026 AI Project Documentation

Fine-tuning of a pre-trained T5 or other model for natural language to corpus query language conversion.

Ota Mikušek (UČO: 514440)

July 29, 2025

1 Goal

This project aims to provide a method to translate natural language into Corpus Query Language (CQL). The focus of this project is on fine-tuning models for the translation task. The models in question are T5 and similar.

For this purpose, a dataset of 10,046 records containing corpus query language queries is used. The dataset contains only queries for `preloaded/bnc2_tt31` corpus. Also, the dataset provides only an English language input.

2 Solution

Models selected for this project are `T5`, `MT5`, `Gemma2`, and `Llama 3.2`. These models were fine-tuned with the provided dataset. `T5`, and `MT5` were fine-tuned and extended with Backus–Naur form (BNF) grammar, limiting models' output only to valid CQLs. `Gemma2`, and `Llama 3.2` were fine-tuned using Low-Rank Adaptation (LoRA).

3 Enviroment Setup

The environment is set up using `make`. Jupyter Lab is used to manage the training process. A demo with the best resulting models is provided as two Python HTTP servers.

3.1 Jupyter Lab

For running Jupyter notebooks, the environment is set up using the command in the main directory:

```
make jupyter0
```

The command will set up a Python virtual environment with Jupyter Lab. A Jupyter Lab is run and accessible on port 8890.

A token is required to access Hugging Face models. The token may be given to the Python virtual environment by running the command:

```
make login
```

The token must provide access to models:

- `google/gemma-2-2b`
  https://huggingface.co/google/gemma-2-2b
- `meta-llama/Llama-3.2-1B`
  https://huggingface.co/meta-llama/Llama-3.2-1B
- `google/mt5-base`
  https://huggingface.co/google/mt5-base
- `google-t5/t5-base`
  https://huggingface.co/google-t5/t5-base

Each folder name in the main directory specifies which model and how it is fine-tuned. Folders commonly contain four Jupyter notebooks. `playground.ipynb` is used to fine-tune a model. Model states are periodically stored. `validation.ipynb` selects the best model by running validation tests on all fine-tuned model versions. `evaluation.ipynb` is used to get the best model. The model is run on test set inputs, storing the results. `evaluation_cql.ipynb` compares model outputs with gold outputs. This notebook is separated from the evaluation notebook because metrics for model evaluation may change. This way, metrics need to be updated in this notebook. The model does not have to be rerun.

All notebooks are expected to run on exactly one visible CUDA GPU. GPUs used for this project were NVIDIA T4.

3.2 Setting up Demo

Folder `demo` contains two programs that give access to the eight best models created during the fine-tuning. These programs can again be run using `make` in their respective folder by the command:

```
make run
```

This command will set up a Python virtual environment for each respective program. After both programs start, a web application is started on URL `localhost:6130/`.

4 Training

The dataset was divided into three parts based on target CQLs, ensuring that no data used for validation or evaluation appears in the training set. The split was as follows: 80% for training, 16% for validation, and 4% for evaluation. During training, validation, and evaluation, one epoch refers to a complete pass through the respective subset of the dataset.

4.1 Dataset Exploration

The dataset provides access to 10,046 records of CQLs. Each CQL has multiple natural language translations generated by ChatGPT, based on rule-based translation of the original CQL. The number of generated translations for one CQL ranges from approximately 8 to 13. The goal of the generated translations was to provide diversity to the rule-based translation. The models were fine-tuned only on these generated translations. The rule-based translations were not used.

What can be noticed is that sometimes the expansion of rule-based translation to natural language using the ChatGPT Is not correct. For example:

```
[lemma="(set|group)"&tag="NN"][lemma="of"]?[tag="N.*"][tag="VVP"]
```

was translated using rulebased translation as:

> [token which attribute 'lemma' matches regex '(set—group)' AND attribute 'tag' matches regex 'NN'] FOLLOWED BY [token which attribute 'lemma' matches regex 'of'] (optional) FOLLOWED BY [token which is a noun] FOLLOWED BY [token which attribute 'tag' matches regex 'VVP']

This translation is correct. However, the translation generated by ChatGPT is wrong:

> All nouns with the lemma set or group followed optionally by the lemma of and then by a noun followed by a verb phrase.

The incorrect part is transformation of **regex 'VVP'** to **a verb phrase**. The VVP means: verb, present, not 3rd person. Even though the dataset contains this type of error, I still hope that it will be possible to fine-tune the models to a reasonable quality.

4.2 T5 models

The T5 model was trained as a seq2seq model. The model was initialized with default T5 weights and trained for over 33 epochs.

In addition, another model was created by limiting the next generated token by BNF grammar. This mechanism was employed in the validation phase. The BNF grammar limits model output to generate tokens that may result in valid CQL. This grammar is stored and used in `cql_checker.py`. The grammar is parsed using the lark[1] package.

4.3 MT5 models

The MT5 model was trained the same way as the T5 model. Another variant of the model was again created by applying the grammar in validation. The MT5 model was selected due to it's capability to work in multiple languages. This is in contrast to the T5 model, which was originally trained on only English data.

4.4 Gemma2 and Llama

Gemma2 and Llama models were trained using LoRA. Since I did not have any experience prior to this project with LoRA, the r values were chosen by feeling. Llama was trained with $r \in \{1, 2\}$ and Gemma2 with $r \in \{4, 8\}$

4.5 Metrics

During validation, the best model was selected based on the best BLEU metric result with weights 0.25 on 1-4 n-grams.

During the evaluation, multiple metrics were evaluated. These metrics can be organized into syntactic and semantic categories.

4.5.1 Syntactic Metrics

These metrics evaluate the similarity of the model CQL string with the gold CQL string. In general, a higher score means CQL is more likely to look like it was made by a human. These metrics are: `n_grams_{1,2,3,4,5,6}`, which is an average of respective n-grams that are present in both model and gold CQL, divided by the number of all n-grams in gold CQL. `bleu` score is used as another metric with weights 0.25 on 1-4 n-grams.

[1]`https://pypi.org/project/lark/`

4.5.2 Semantic Metrics

These metrics compare results returned by a corpus manager for model CQL and gold CQL. Here, in general, a higher score means the CQL result is actually relevant to the user input. This is in contrast with syntactic metrics, where one CQL could be similar to the other, but the evaluation may return very different results. These metrics are: `precision`, which is the number of hits sheared by model CQL and gold CQL, divided by the number of hits in model CQL. `recall`, which is the number of hits sheared by model CQL and gold CQL, divided by the number of hits in gold CQL. `f1` metric computed as:

$$\texttt{f1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

`max IoU` (maximal intersect over union) is the average of the maximal value for each hit in model CQL with gold CQL. This way, when, for example, we are looking for all sentences in passive, gold CQL requires whole sentences, but the model provides a CQL that finds only the structures that make the sentence passive, at least some score is given, because a subpart was found, instead of 0 in case of `precision` and `recall`.

`max IoU +-{1,2}` works in the same way as `max IoU`, but tolerance is extended to off by one or two errors. This may detect situations when, for example, gold CQL searches for all ends of sentences by end of sentence structures, but the model CQL searches for punctuation at the end of sentences.

5 Results

During validation, the best models were selected based on the highest BLEU score. This validation was made on validation data. In the case of Gemma and Llama models, the validation was made only on a random sample of 1,000 validation records.

There was an idea that during generation, the generated CQLs will be checked against the Sketch Engine API. However, such frequent checks quickly result in the block by the API because FUP limits were exceeded. Therefore, this method was not implemented.

The best models created that can be seen in Tables 1 and 2 are available on the URL: `epimetheus2:6130/`. I personally tested these models with the following prompts, and it seems that all models perform badly. However, often at least one model is correct and fine-tuning improves the model's ability to respond correctly, Prompts:

Any four-letter word.
I need an example of a negative sentence.
I need a verb with negation.
Any word.
Libovolné slovo.
Word door followed by a handle.
Word door followed by a handle in a sentence.
Word door followed by a lemma handle within a sentence.

The whole project was made public on the Faculty GitLab.[2]

[2] `https://gitlab.fi.muni.cz/xmikusek/aiproject`

6 Conclusion

The results revealed that the fine-tuning process improved the model's performance, especially for the Gemma2 model. The gemma2 model outperformed all models in all metrics. Models that were using `lark` package for limiting the generated tokens did not perform better than without limitation, which should not happen, and there is possibly an error in the grammar. This will be part of future work.

In the end, this project shows that improvement in natural language to CQL translation can be made with fine-tuning for T5 and similar models.

7 Future Work

I found out that `lark`, which I am using to limit models' outputs only to valid CQL, does not work properly. `lark` sometimes returns an error that no valid continuation exists, even when there is a valid continuation. Here is an example:

```
cql_checker.CQLChecker().parse('[wo')
```

which ends with:

```
lark.exceptions.UnexpectedCharacters:  No terminal matches 'w'
```

but, here is a valid continuation that the grammar accepts:

```
cql_checker.CQLChecker().parse('[word="an"]')
```

The higher value of R for Llama models could also be tested. It is questionable if the lower value in results, in contrast to gamma models, could be caused by the lower value of R.

model name	precision	recall	f1	max IoU	max IoU +-1	max IoU +-2	sentence_precision	sentence_recall
gemma2_baseline	20.49	20.16	20.09	21.09	21.27	21.34	21.40	21.07
gemma2_r4	72.61	72.63	72.25	73.06	73.25	73.34	73.69	74.01
gemma2_r8	70.97	70.69	70.45	71.38	71.53	71.61	71.89	72.07
llama_baseline	1.63	1.61	1.57	1.66	1.67	1.67	1.67	1.68
llama_r1	0.21	0.21	0.21	0.24	0.24	0.25	0.26	0.25
llama_r2	0.26	0.24	0.25	0.28	0.28	0.28	0.28	0.26
t5	63.42	63.55	63.09	63.90	64.20	64.32	64.72	64.85
t5_logit	40.31	38.67	38.64	42.80	43.88	44.32	45.70	42.75
mt5	0.02	0.00	0.00	0.02	0.02	0.02	0.02	0.00
mt5_logit	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1: Semantic Metrics (in %)

model name	n_grams_1	n_grams_2	n_grams_3	n_grams_4	n_grams_5	n_grams_6	bleu
gemma2_baseline	85.29	63.95	48.40	40.25	33.48	26.95	39.64
gemma2_r4	95.87	90.54	85.19	80.95	76.81	73.35	80.77
gemma2_r8	95.20	89.36	83.87	79.59	75.34	71.62	79.18
llama_baseline	47.43	25.60	17.17	10.71	6.32	4.56	9.65
llama_r1	75.61	61.73	52.95	45.04	38.82	33.45	46.55
llama_r2	66.39	53.83	45.87	38.40	33.69	29.44	38.25
t5	95.24	88.74	82.36	77.43	72.47	68.77	77.65
t5_logit	87.73	72.19	62.97	56.37	49.46	42.94	56.29
mt5	75.57	63.66	47.24	43.27	39.94	36.31	29.72
mt5_logit	61.70	44.35	35.87	31.67	27.54	22.59	21.99

Table 2: Syntactic Metrics (in %)