CS5740: Assignment 4

The report for this assignment is structured. Please fill in content only in the appropriate places. Other submitted content cannot be evaluated or graded. Do not add a cover page.

This assignment focuses on supervised sequence prediction, specifically the task of translating natural language instructions into SQL queries. You will explore three different approaches for this task: fine-tuning a pre-trained encoder-decoder transformer model (specifically, the T5 model), training a model with the same architecture from scratch and using diverse prompt engineering techniques with a large language model (LLM). As the same instruction can be expressed in different ways in SQL, you will evaluate each system by comparing whether the generated SQL queries return the same database records as the ground-truth by using the F1 metric. During the assignment, you will evaluate these approaches, perform qualitative error analysis and report results.

This assignment includes an early milestone (5pt). To complete the milestone and receive the relevant points, you will need to achieve an F1 score of at least 0.5 on the test set with a finetuned T5 model by the milestone deadline above. Because training from scratch should use basically the same training code (except the initial model weights), this should put you in a good position for the training from scratch component as well.

GitHub Classroom Setup Follow the *starter repository invitation* link. Enter your Cornell NetID and continue. GitHub Classroom will provide a link to a private repository for you to clone to your computer. All code should exist in the repository. *Only the code that is present in the master branch of your repository by the due date will be evaluated as part of your submission!*

Submission Please see the submission guidelines at the end of this document.

Starter Repo

All data for the assignment can be found under the `data/` directory. The database you will be evaluating queries on is `flight_database.db`, with the `flight_database.schema` file containing the database schema. The `ents` (entities) section in the schema lists the 25 tables in the database, such as "airline", "restriction", "flight", etc. The schema also gives information about the columns in the table, such their type and whether they're indexed.

The text-to-SQL data, on the other hand, is split into training, development and held-out test sets. The files with the `.nl` extension contain the natural language instructions, while the files with the `.sql` extension contain the corresponding, ground-truth SQL queries. The starter code contains various utility functions for evaluation (under `utils.py`) to abstract away the details of interacting with the SQL database.

A skeleton of training code for T5 is provided in `train_t5.py` and a skeleton of prompting code for Gemma is provided in `prompting.py`.

Task

In this assignment, you will be exploring the sequence prediction task of converting natural language instructions for flight booking to SQL queries with the three separate approaches detailed below.

Tasks 1 and 2: Working with the T5 Architecture In your first two tasks, you will be working with the small variant of the T5 encoder-decoder architecture (Raffel *et al.*, 2019)[1]. The encoder will ingest natural language queries (i.e., the input sequence) while the decoder will predict the corresponding SQL queries (i.e., the output sequence). Your first task will be to finetune the pretrained T5 architecture while your second task will be to train the exact same architecture from scratch (i.e., from randomly initialized weights). You will submit entries to the leaderboard for both tasks separately. We provide you a training loop code skeleton where you will only need to implement model initialization (using the relevant Huggingface transformers implementation[2]), the evaluation loop and data processing.

For either task, simply implementing data processing with the existing T5 tokenizer and varying simple hyperparameters should lead to good baseline results. You may, however, choose to experiment with data processing or architecture details to push performance higher. During finetuning, for instance, a common design choice is to freeze part of the model parameters (i.e., only finetune a subset of the parameters). Likewise, for training from scratch, you could find experimenting with data processing leading to better outcomes. If you find that the tokenizer is ill-suited for SQL code, for instance, you could choose to design your own tokenizer for SQL and learn new embeddings and language modeling output heads for the decoder.[3]

In the assignment report, we ask you about your strategies for data processing, tokenization and architecture details (such as freezing parameters). If you experiment with these dimensions and find them to improve performance, we will expect ablation experiments supporting your findings in the results section.

Task 3: Prompting & In-context Learning For this task, you will experiment with in-context learning (ICL) using an LLM. You can use two LLMs: instruction-tuned Gemma 1.1 2B model and/or instruction-tuned CodeGemma 7B model.[4] To get access to the models, you will need to log into your Hugging Face account, review the conditions on the model's page[4] and access the model's content. The 7B model is slower for inference and development, so you may want to do most development with the 2B model, and use 7B only in later stages. However, there is no requirement to use both models, and you could do all development and report your final results on the 2B model. We will not distinguish between the two models on the leaderboard. You can trade-off inference time with performance by quantizing the model.

The LLM is frozen here, and will perform the generation task while only conditioned on the text input (prompt). You will design prompts to experiment with zero- and few-shot prompting. In the zero-shot case, you can provide instructions in the prompt, but it doesn't include examples that show the intended behaviour. In the few-shot case, you will also include examples showing the intented behaviour. You need to at least try $k = 0, 1, 3$, where k is the number of examples. You are encouraged to try other values. For few-shot prompting, you will also need to experiment with different ways of selecting the examples, observe how the design choice affects the performance, and how sensitive performance is to the selection of ICL examples. In the prompt, you can also provide additional context and indications, for instance, about the task, the database, or details about the intended behavior.

Optionally, you may also experiment with chain-of-thought[5] prompting to increase performance, but you are not required to.

In the assignment report, you should provide the best prompt you designed, and ablation experiments that show how the design choices about different parts of the best prompt influenced the performance. The ablation experiments should empirically show the effectiveness of your choices, at a meaningful granularity level. For instance, did specific instruction(s), contextual information, or clause order influence your performance? Note: these are for example purposes, and you should include the ablations that are significant for your own prompt.

If you are interested, we recommend experimenting with your prompts with state-of-the-art LLMs (e.g.,

[1] https://arxiv.org/pdf/1910.10683

[2] https://huggingface.co/google-t5/t5-small

[3] These are ideas to get you thinking, rather than specific suggestions to try. We did not fully experiment with these approaches.

[4] https://huggingface.co/google/gemma-1.1-2b-it / https://huggingface.co/google/codegemma-7b-it

[5] Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Wei *et al.*, 2022)

GPT-4, Claude, Gemini) to get an impression of the performance of the best models out there. **Do not submit these results to the leaderboard, and do not include them in the report. This is just for your own intellectual curiosity and broader understanding.**

Evaluation Details For the task of language-to-SQL generation, since different SQL queries can correctly respond to the same natural language instruction, evaluation is commonly performed by executing the SQL queries directly on the database. The evaluation code in the starter repository will provide you with three separate metrics: Record F1, Record Exact Match and SQL Query Exact Match (EM).

Record F1 is the metric that we will use in the leaderboard and computes the F1 score between database records produced by model-generated and ground-truth SQL queries. Record EM, on the other hand, is a stricter metric checking whether the produced records match exactly, similar to an accuracy score. Finally, SQL Query EM will tell you whether the SQL queries themselves match exactly. As the latter two can help with debugging and error analysis, we will ask you to also report them on the development set component of the results section.

Submissions

After implementing your methods and performing experiments, you will summarize your findings in a report. The report template provides the requirements. Your code will be submitted automatically with your repository. **Please submit your code as you develop it. Incremental small commits over time provide proof of work.**

Leaderboard Instructions We will maintain a leaderboard for the test set, with submissions ranked according to the F1 score. You will submit output files for all three tasks separately. As it can be resource intensive to run SQL queries, you will submit two files for each task: (a) the models' SQL query outputs in the `results/` folder; and (b) the database records associated with the SQL queries under the `records/` folder. The `save_queries_and_records` function in `utils.py` will help you with this. Please refer to the `README.md` file in the code skeleton for more details on formatting.

To verify the correctness of your output format, you may use the `evaluate.py` script. For instance, to evaluate submission files on the dev set, run:

```
python evaluate.py
  --predicted_sql results/t5_ft_dev.sql
  --predicted_records records/t5_ft_dev.pkl
  --development_sql data/dev.sql
  --development_records records/ground_truth_dev.pkl
```

Performance Grading Performance will be graded for all systems (llm, t5ft, t5scr), but with significantly more emphasis on your best performing system.

Let $F1_{\text{best}} = \max(F1_{\text{llm}}, F1_{\text{t5ft}}, F1_{\text{t5scr}})$ be the F1 score of the best performing approach, and $F1_{\text{second}}$ and $F1_{\text{third}}$ be the F1 score of the two other approaches. The scoring curve function is:

$$f(x) = 3 \times \big(\sigma(x \times 2.5) - 0.5\big) \ ,$$

where σ is the sigmoid function. The performance grade is:

$$f(F1_{\text{best}}) \times 35 + f(F1_{\text{second}}) \times 5 + F1_{\text{third}} \times 5$$

Development Environment and Third-party Tools All allowed third-party tools are specified in the `requirements.txt` file in the assignment repository.[6] The goal of the assignment is to gain experience with specific methods, and therefore using third-party tools and frameworks beyond these specified is not allowed. You may only `import` packages that are specified in the `requirements.txt` file or that come with Python's Standard Library. The version of Python allowed for use is 3.10.x. Do not

[6] https://realpython.com/lessons/using-requirement-files/

use older or newer version. We strongly recommend working within a fresh virtual environment for each assignment. For example, you can create a virtual environment using conda and install the required packages:

```
conda create -n cs5740a4 python=3.10
conda activate cs5740a4
pip install -r requirements.txt
```

Leaderboard We will consider the most recent leaderboard result, and it must match what you provide in your report. Please be careful with the results you submit, and validate them as much as possible on the development set before submitting. If your results go down, they go down. This aims to approximate testing in the wild (and in good research). The leaderboard refresh schedule is: every 8 hours. Each refresh will use what your repository contains at that point. Because our scripts take time to run, the exact time we pull your repository might be a bit after the refresh time. So please avoid pushing results that you do not wish to submit.

Submission, Grading, and Writeup Guidelines Your submission on Gradescope is a writeup in PDF format. **The writeup must follow the template provided. Do not modify, add, or remove section, subsection, and paragraph headers. Do not modify the spacing and margins.** The writeup must include at the top of the first page: the names of the student, NetID, and the URL of the Github repository. We have access to your repository, and will look at it. Your repository must contain the code in a form that allows to run it form the command line (i.e., Jupyter notebooks are not accepted).

The following factors will be considered: your technical solution, your development and learning methodology, and the quality of your code. If this assignment includes a leaderboard, we will also consider your performance on the leaderboard. Our main focus in grading is the quality of your empirical work and implementation. Not fractional differences on the leaderboard. We value solid empirical work, well written reports, and well documented implementations. Of course, we do consider your performance as well. The assignment details how a portion of your grade is calculated based on your empirical performance.

Some general guidelines (not only specific to this assignment) to consider when writing your report and submission:

- Your code must be in a runnable form. We must be able to run your code from vanilla Python command line interpreter. You may assume the allowed libraries are installed. Make sure to document your code properly.
- Your submitted code must include a `README.md` file with execution instructions.
- Please use tables and plots to report results. If you embed plots, make sure they are high resolution so we can zoom in and see the details. However, they must be readable to the naked eye (i.e., without zooming in). Specify exactly what the numbers and axes mean (e.g., F1, precisions, etc).
- It should be made clear what data is used for each result computed.
- Please support all your claims with empirical results.
- All the analysis must be performed on the development data. It is OK to use tuning data. Only the final results of your best models should be reported on the test data.
- All features and key decisions must be ablated and analyzed.
- All analysis must be accompanied with examples and error analysis.
- Major parameters (e.g., embedding size, amount of data) analysis must include sensitivity analysis. Plots are a great way to present sensitivity analysis for numerical hyper parameters, but tables sometimes work better. Think of the best way to present your data.
- If you are asked to experiment with multiple models and multiple tasks, you must experiment and report on all combinations. It should be clear what results come from which model and task.

- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.
- Make figures clear in isolation using informative captions.

This assignment was created by Omer Gul, Anne Wu, and Yoav Artzi.

Updated on August 12, 2024