Natural Language to SQL

CSE 5525: Assignment 3

Due: March 11th, 2026 at 11:59 PM

Goals

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.

The report for this assignment is structured and has additional requirements for experiments. It is provided with the code: `a3-report-template.zip`. PLEASE READ THIS FILE EARLY TO ENSURE YOU ARE AWARE OF THEM. Please fill in content only in the appropriate places.

Hugging Face Primers

- Transformers Quickstart (load models, tokenize, generate).
- Seq2Seq fine-tuning tutorial (uses T5): mirrors our training loop needs.
- T5 model docs: `T5ForConditionalGeneration`, usage examples.
- Train a new tokenizer from an existing one
- Decoding strategies (greedy vs beam, sampling, etc.).
- `google/gemma-1.1-2b-it`
- `google/codegemma-7b-it`
- Chat templates & `tokenizer.apply_chat_template` (useful for instruction-tuned models).
- BitsAndBytes quantization in Transformers (using `BitsAndBytesConfig`).

Dataset and Code

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 as 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`.

Please read `README.md` for exact file formats for the submission.

Part 1 & 2: Working with the T5 Architecture

In Parts 1 & 2, 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). We provide you with a training loop code skeleton where you will only need to implement model initialization (using the relevant Hugging Face 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.

Part 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

[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`

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. 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 intended 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., GPT-5, Claude, Gemini) to get an impression of the performance of the best models out there. **However, do not submit these results, or 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 to evaluate your submission 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.

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

Submission

You will upload your code to Gradescope for the assignment "HW3: Natural Language to SQL". Please create a .zip file of your submission which includes all the data, code, and a .pdf of the report. **Please include all the results and records generated from your experiments in the respective directories. Also, when you are uploading your submission please remember to exclude any saved models or checkpoint files.** When you are zipping up your files, make sure to select all the files and then zip, and not the folder containing the files.

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 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 points will be:

$$f(F1_{\text{best}}) \times 35 + f(F1_{\text{second}}) \times 10 + F1_{\text{third}} \times 5$$

This will give you a score out of 50, which will be rounded up to the nearest multiple of 5 as your final score. For example, scores in (45, 50] will get 50 points, scores in (40, 45] will get 45 points, and so on. The remaining 50 points will be for the report.