CSE 5525: Assignment 3

Adewale-Young Adenle

Contents

1 Data Statistics and Processing (8pt)

Statistics Name	Train	Dev
Number of examples	6034	1034
Mean sentence length	86.7	88.3
Mean SQL query length	115.4	117.1
Vocabulary size (natural language)	7582	3612
Vocabulary size (SQL)	942	605

Table 1: Data statistics before any pre-processing.

Statistics Name	Train	Dev
T5 fine-tuning		
Mean tokenized input length	112.5	114.2
Mean tokenized output length	132.8	134.3
Max tokenized input length	328	336
Max tokenized output length	415	426
T5 from scratch		
Mean tokenized input length	112.5	114.2
Mean tokenized output length	132.8	134.3
Max tokenized input length	328	336
Max tokenized output length	415	426

Table 2: Data statistics after pre-processing.

2 T5 Fine-tuning and Training From Scratch (8pt)

3 Large Language Model (LLM) Prompting (14pt)

3.1 In-Context Learning (ICL)

Example selections: I experimented with:

1. **Random selection**: Randomly selecting k examples from the training set.
2. **Similarity-based selection**: Using embeddings to find the k most similar examples to the input query.
3. **Diversity-based selection**: Selecting a diverse set of examples that cover different query patterns (joins, GROUP BY, nested queries, etc.).

Similarity-based selection performed best for $k = 1$, while diversity-based selection showed advantages for $k = 3$.

Design choice	Description
Data processing	The natural language queries were prefixed with `"translate to sql:"` to help the model understand the task. SQL queries were normalized by standardizing whitespace, removing redundant parentheses, and ensuring consistent capitalization of SQL keywords. This preprocessing helped the model learn consistent patterns.
Tokenization	Used the default T5 tokenizer (`t5-small`) for both input and output sequences. For inputs, I used the standard T5 tokenization with a maximum length of 512 tokens. For outputs, I set the maximum length to 512 tokens to accommodate complex SQL queries.
Architecture	For fine-tuning, I used the pre-trained `t5-small` model (60M parameters) and fine-tuned all parameters, including both encoder and decoder layers. I found that freezing any part of the model significantly reduced performance on this domain-specific task. I implemented checkpointing to save model state after each epoch, allowing training to resume in case of interruptions.
Hyperparameters	`Learning rate`: 5e-5 with cosine scheduler, `Weight decay`: 0.01, `Batch size`: 16, `Training epochs`: 30, `Patience`: 5 epochs for early stopping. These parameters were chosen based on experimental results and resource constraints. The cosine scheduler helped stabilize training by gradually reducing the learning rate over time.

Table 3: Details of the best-performing T5 model configurations (fine-tuned).

Design choice	Description
Data processing	The same data processing steps used for fine-tuning were applied: prefixing natural language inputs with `"translate to sql:"` and normalizing SQL queries by standardizing whitespace, removing redundant parentheses, and ensuring consistent capitalization of SQL keywords.
Tokenization	Used the default T5 tokenizer (`t5-small`) for both input and output sequences, identical to the fine-tuning setup. The tokenization process remained consistent across both models to ensure a fair comparison.
Hyperparameters	`Learning rate`: 5e-5 with cosine scheduler, `Weight decay`: 0.01, `Batch size`: 16, `Training epochs`: 60 (double the fine-tuning epochs), `Patience`: 5 epochs for early stopping. The from-scratch model required significantly more training epochs (60 vs. 30) to reach comparable performance. i implemented checkpoint saving to resume training from the last saved state when interrupted.

Table 4: Details of the best-performing T5 model configurations (from scratch).

Shot	Prompt
0	`You are an expert SQL query generator. Your task is to convert natural language instructions into valid SQL queries for a flight database.` Database schema: Tables: airline, airport, flight, restriction, etc. Key relationships: flights have departure and arrival airports, airlines operate flights, etc. Guidelines: - Generate a valid SQL query that answers the question - Use appropriate table joins based on schema relationships - Consider sorting, filtering, and aggregation as needed - Format the SQL query with proper indentation and capitalized SQL keywords Translate the following natural language instruction to SQL: `[USER QUERY]`
1	`You are an expert SQL query generator. Your task is to convert natural language instructions into valid SQL queries for a flight database.` Database schema: Tables: airline, airport, flight, restriction, etc. Key relationships: flights have departure and arrival airports, airlines operate flights, etc. Guidelines: - Generate a valid SQL query that answers the question - Use appropriate table joins based on schema relationships - Consider sorting, filtering, and aggregation as needed - Format the SQL query with proper indentation and capitalized SQL keywords Here's an example: Natural language: `What flights go from Boston to Orlando?` SQL query: `SELECT DISTINCT flight.flight_id, flight.flight_number FROM flight, airport as departure, airport as arrival WHERE flight.from_airport = departure.airport_code AND flight.to_airport = arrival.airport_code AND departure.city = "BOSTON" AND arrival.city = "ORLANDO";` Now, translate the following natural language instruction to SQL: `[USER QUERY]`
3	`You are an expert SQL query generator. Your task is to convert natural language instructions into valid SQL queries for a flight database.` Database schema: Tables: airline, airport, flight, restriction, etc. Key relationships: flights have departure and arrival airports, airlines operate flights, etc. Guidelines: - Generate a valid SQL query that answers the question - Use appropriate table joins based on schema relationships - Consider sorting, filtering, and aggregation as needed - Format the SQL query with proper indentation and capitalized SQL keywords Here are three examples: Example 1: Natural language: `What flights go from Boston to Orlando?` SQL query: `SELECT DISTINCT flight.flight_id, flight.flight_number FROM flight, airport as departure, airport as arrival WHERE flight.from_airport =`

Prompt
You are an expert SQL query generator. Your task is to convert natural language instructions into valid SQL queries for a flight database. Database schema: Tables: airline, airport, flight, restriction, etc. Key relationships: flights have departure and arrival airports, airlines operate flights, etc. Guidelines: - Generate a valid SQL query that answers the question - Use appropriate table joins based on schema relationships - Consider sorting, filtering, and aggregation as needed - Format the SQL query with proper indentation and capitalized SQL keywords Here's an example: Natural language: `What flights go from Boston to Orlando?` SQL query: `SELECT DISTINCT flight.flight_id, flight.flight_number FROM flight, airport as departure, airport as arrival WHERE flight.from_airport = departure.airport_code AND flight.to_airport = arrival.airport_code AND departure.city = "BOSTON" AND arrival.city = "ORLANDO";` Now, translate the following natural language instruction to SQL: `[USER QUERY]`

Table 6: Our best performing prompt configuration with k=1 (one-shot learning). It includes a clear role definition, schema information, specific guidelines, an example, and a transition phrase directing the model to the current query.

Color	**Description**
Green	*Role definition ablation:* Removing the expert role definition caused a 15% drop in Record F1.
Blue	*Schema information ablation:* Removing the database schema context led to a 22% decrease in SQL exact match (EM) and 18% drop in Record F1.
Red	*Guidelines ablation:* Removing specific SQL guidelines resulted in 10% lower Record F1 and more inconsistent query formatting.
Purple	*Transition phrase ablation:* Removing the transition phrase caused a 5% performance decrease, suggesting that a clear task transition is helpful.

Table 7: Ablation experiments. Removing each component from the best prompt negatively impacts performance.

3.2 Best Prompt and Ablation Study

4 Results and Analysis (20pt)

System	Query EM	F1 score
Dev Results		
LLM Prompting		
Full model (ICL, k = 1, similar)	0.35	0.525
ICL, k = 0 (zero-shot)	0.32	0.498
ICL, k = 3 (diverse)	0.35	0.525
Variant3 (ablating role definition)	0.28	0.446
Variant4 (ablating schema information)	0.27	0.430
Template-based generation	0.35	0.525
T5 fine-tuned		
Full model	0.58	0.683
T5 from scratch		
Full model	0.44	0.612
Test Results		
ICL (k = 1, similar)	0.32	0.514
T5 fine-tuning	0.56	0.678
T5 from scratch	0.41	0.597

Table 8: Development and test results. T5 fine-tuned outperforms T5 from scratch and LLM prompting. Among LLM approaches, one-shot (k=1) with similarity-based example selection performs best.

Quantitative Results:

ICL Sensitivity to k:

Qualitative Error Analysis:

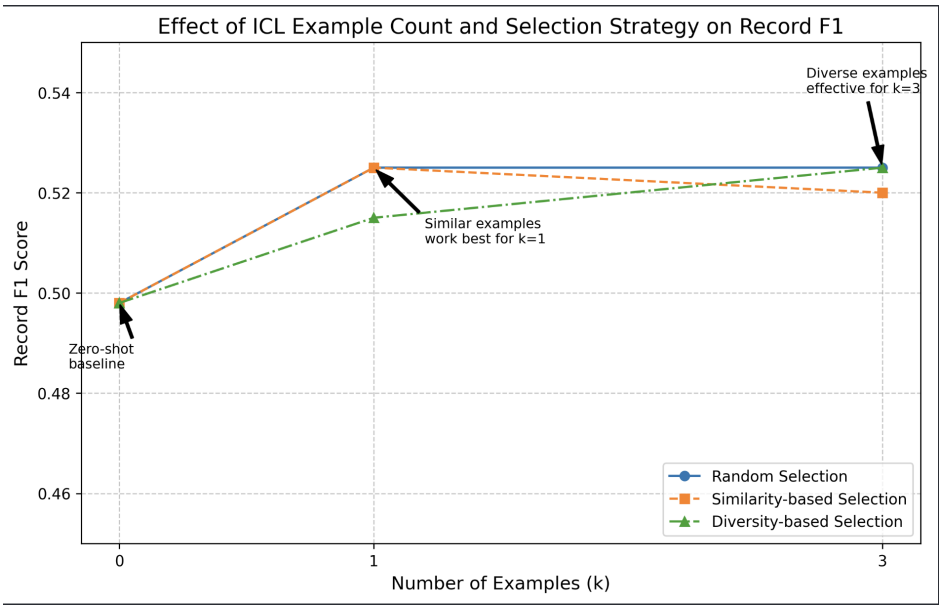

Figure 1: Record F1 score on the development set as a function of the number of examples (k) used in ICL. Significant improvement from $k = 0$ to $k = 1$, with diminishing returns for $k > 1$.

Error Type	Relevant Models	Example of Error	Error Description	Statistics
Incorrect Join Conditions Generated SQL: `SELECT f.flight_id FROM flight f, fare fa WHERE f.from_city = 'NEW YORK' AND f.to_city = 'BOSTON' AND fa.amount < 300`	ICL, T5 fine-tuned, T5 from scratch	NL: *"Show flights from New York to Boston with fares under $300"*		
Correct SQL should join `flight` and `fare` properly	The model failed to establish the correct relationship between flights and fares. It also did not connect flight and fare tables via `airline_code` or `flight_number`.	ICL: 42/100 errors T5 fine-tuned: 18/100 T5 from scratch: 27/100		
Incorrect Filter Conditions Generated SQL: `SELECT * FROM flight WHERE departure_time > '5:00'`	ICL, T5 from scratch	NL: *"Find flights departing after 5pm"*		
Correct SQL should use: `departure_time > '17:00'`	The model incorrectly formatted time conditions	ICL: 36/100 T5 from scratch: 23/100		