

# Case Study: Fine Tuning Text2SQL Model

---

## TL;DR (Executive Summary)

- **Objective:** Fine-tune a  $\leq 3B$  parameter LLM to generate **executable SQL** from natural-language questions under **limited GPU resources (6GB VRAM)**.
- **Approach:** Used **Falcon3-1B-Base** and **Qwen2.5-Coder-1.5B-Instruct** with **QLoRA (NF4 + double quantization)** and **LoRA adapters ( $r=32$ )** to enable efficient fine-tuning.
- **Dataset:** Trained on **BIRD (bird23-train-filtered)**, leveraging the provided **evidence** field and applying **distribution-preserving sampling (40% - 50%)** based on evidence length.
- **Training Setup:** Small batch size with gradient accumulation (effective batch size = 16), conservative learning rate (2e-5), and 3 epochs to ensure stability.
- **Results:** Finetuned Qwen model performs better than finetuned Falcon3 model. For Qwen, the SQL **syntax validity improved from 0.19% (baseline) to 89.02%** on unseen test data, as measured using `sqlglot`.
- **Key Limitation:** The model still exhibits **schema hallucination** (e.g., incorrect table/column names), despite syntactically valid SQL.
- **Future Improvements:** Execution-based evaluation, AST-level equivalence checks, higher LoRA capacity, inclusion of `k_proj`, and exploration of larger models (e.g., StarCoder2-3B).

## Objective

The goal is to train an LLM model (<3B parameters) for Text2SQL purposes using QLoRA. The notebook is designed to run with **a minimum of 6GB RAM of GPU**.

Although accuracy is not the primary goal, we expect the model **to generate syntactically valid and executable SQL queries** from natural-language questions.

## Model Preparation

QLoRA is a variant of LoRA in which we strictly set the weight data types to a given precision to reduce RAM usage during training.

LoRA itself is an optimization finetuning method where we freeze the weight pretrained model  $\mathcal{W}$ , add a weight matrix  $\Delta \mathcal{W}$  with rank  $r$  (which should be much smaller than  $d_{in}$  and  $d_{out}$ ) and only train the matrix  $\Delta \mathcal{W}$ .

# Practical Setup

## 1. Model

### a. Qwen/Qwen2.5-Coder-1.5B-Instruct

Link: <https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B-Instruct>

**Reason:** This model has been previously trained and popularly used for coding tasks, and the size parameter itself is relatively small compared to the specified requirements in the PDF file.

### b. tiiuae/Falcon3-1B-Base

Link: <https://huggingface.co/tiiuae/Falcon3-1B-Base>

**Reason:** This model contains smaller parameters than Qwen. If we use smaller parameters, we can allocate more memory for data training.

## 2. QLoRA: BitAndBytes configuration

Link: <https://medium.com/@niraj.h/deep-dive-into-quantization-in-qlora-1f82e04a4673>

### Reason:

- `bnb_4bit_quant_type="nf4"` : The weight distribution of a neural network tends to follow a normal distribution. Therefore, NF4 is a preferred choice because it provides memory efficiency through its distribution, offering 16-bit performance via a non-linear mapping with 4-bit memory.
- `bnb_4bit_use_double_quant=True` : This saves additional VRAM by quantizing the quantization constants themselves from 32-bit down to 8-bit. Often paired with NF4 quantization.

## 3. LoRA Configuration

Link: <https://www.cognativ.com/blogs/post/what-is-lora-llm-low-rank-adaptation-in-language-models/310>

### Reason:

- `r=32` : The model can be categorized as a small LLM; therefore, choosing rank = 32 is a good start.
- `lora_alpha=64` : Setting `lora_alpha` to  $2 \times r$  is a commonly adopted heuristic that balances training stability and adaptation strength. In this setup, `lora_alpha=64` ensures that the injected LoRA updates are large enough to meaningfully steer the model toward the Text2SQL task, while still maintaining stable optimization under mixed-precision and quantized weights.

- `target_modules` : In a small RAM setup, `q_proj` and `v_proj` is the practical choice, as they retrain the model's attention mechanism. Ideally, we should also include `k_proj` in the fine-tuning to achieve better results.
- `lora_dropout=0.05` : A small value such as 0.05 helps prevent overfitting, especially given the reduced dataset size after sampling. This is particularly important in low-rank fine-tuning, where the adapter capacity is limited and the model can otherwise over-specialize to the training data.

## Attention Setup

The objective of the finetuning is to have a model that produces executable SQL syntax. Therefore, we need to define the labels as valid SQL syntax only. This will prevent from echoing the input due to the nature of `AutoModelForCausalLM`.

## Dataset Preparation

Link: <https://huggingface.co/datasets/birdsql/bird23-train-filtered>

To train the model in Text2SQL task, we pick `birdsql/bird23-train-filtered`. This dataset is a subset of the original BIRD dataset (retaining 70% of the original dataset).

The dataset is appropriate for the fintech use case because it includes `evidence` as additional context.

### EDA Analysis

We check the data distribution across categories, SQL query complexity, and `evidence`. This analysis is crucial because our setup has limited resources, meaning we need to sample the data while preserving its distribution. Therefore, we sample 40 - 50% from the dataset.

Based on the quick analysis on each aspect, maintaining the distribution of the data based on `evidence` is crucial because it reveals the difficulty of the task. We bucket the evidence lengths and sample from each bucket at 40-50%, preserving the data distribution.

## Training Setup Preparation

### 1. Batch Size

- `TRAINING_BATCH_SIZE = 2`
- `VALIDATION_BATCH_SIZE = 2`

A small per-device batch size is required due to GPU memory limitations when fine-tuning a 1.5B-parameter model, even with QLoRA and quantization. A batch size of 2 allows the model, optimizer states, and LoRA adapters to fit within memory without triggering out-of-memory errors.

Using the same batch size for training and validation ensures consistent memory usage and avoids unexpected failures during evaluation.

## 2. Gradient Accumulation

- `TRAINING_GRADIENT_ACCUMULATION_STEPS = 8`

To compensate for the small per-device batch size, we apply gradient accumulation. This effectively simulates a larger batch size without increasing memory usage.

The effective batch size becomes:

$$\begin{aligned} \$\$ \\ 2 \times 8 &= 16 \\ \$\$ \end{aligned}$$

This improves gradient stability and convergence behavior while remaining compatible with the available hardware.

## 3. Learning Rate

- `TRAINING_LEARNING_RATE = 2e-5`

A conservative choice for adjusting the learning rate in a deep learning model. A higher learning rate could lead to overshooting during training.

## 4. Number of Epochs

- `TRAINING_NUMBER_EPOCH = 3`

Given a reduced dataset and the task's scope, 3 epochs should be sufficient for model fine-tuning. This should illustrate the stability of the training.

## 5. Logging Strategy

- `TRAINING_LOGGING_STEP = 10`

Logging every 10 steps provides sufficient visibility, allowing early detection of divergence or instability while keeping training efficient.

## 6. Checkpointing and Evaluation Strategy

- TRAINING\_SAVE\_STRATEGY = "epoch"
- TRAINING\_EVALUATION\_STRATEGY = "epoch"

Given the size of the training set, it is reasonable to save checkpoints and perform a validation test after each epoch, allowing us to monitor the trend of the loss function.

## Test Data Evaluation

We are using test data to evaluate the model's performance after finetuning.

### Example - Finetuned Owen Model

**Question:** What is the cheapest order price of the book "The Little House"?

**Evidence:** "The Little House" is the title of book; cheapest order price refers to Min(price)

```
-- Expected SQL
SELECT MIN(T2.price)
FROM book AS T1
INNER JOIN order_line AS T2
ON T1.book_id = T2.book_id
WHERE T1.title = 'The Little House'
```

```
-- Generated SQL
SELECT T1.price
FROM books AS T1
INNER JOIN orders AS T2
ON T1.book_id = T2.book_id
WHERE T1.title = 'The Little House'
ORDER BY T2.price LIMIT 1
```

Based on the results, the model already has some understanding of how to fetch certain data (such as the minimum price or start date); however, it still exhibits schema hallucination.

### SQL Syntax Validation

Link: <https://sqlglot.com/sqlglot.html>

To check the validity of the generated SQL syntax, we use the `sqlglot` library.

As a baseline, we can use a pretrained model for comparison. Based on the result:

Model	Result
-------	--------

Model	Result
Baseline Qwen/Qwen2.5-Coder-1.5B-Instruct	0.19% valid SQL generation
Finetuned Qwen/Qwen2.5-Coder-1.5B-Instruct	89.02% valid SQL generation

Based on the syntax, the finetuned model has significantly improved the correctness of the syntax.

## Future Work

Several directions can be explored to further improve model performance and robustness in Text2SQL generation:

- **Model Exploration:** **StarCoder2-3B** is a model alternative compared to Qwen2.5-Coder-1.5B-Instruct. Theoretically, the larger model should have a better capacity for learning. See <https://huggingface.co/bigcode/starcoder2-3b>.
- **Evaluation Metrics Enhancement**
  - **AST-based syntax equivalence:** Instead of string matching, abstract syntax tree (AST) comparisons can be used to account for semantically equivalent SQL queries
  - **Execution-based Metrics:**
    - **Execution Accuracy (EX)** to measure whether generated queries return the same results as the ground-truth SQL.
    - **VES-style metrics** to better quantify semantic correctness and schema grounding, particularly in cases of schema hallucination.
    - See <https://arxiv.org/pdf/2305.03111>
- **Quantization Strategy Exploration:** Future experiments could explore **FP4 quantization** to assess whether alternative low-precision formats offer improved numerical stability or generation quality.
- **LoRA Capacity Scaling:** Increasing the LoRA rank ( `r` ) may allow the adapters to capture more task-specific information, potentially improving complex SQL generation.
- **Training Dynamics:** Increasing epochs could further improve the quality of the fine-tuned model. We have tried adding epochs on Qwen and the validity of SQL syntax increases to 96.52% (See [DOKU\\_Text2SQL\\_Falcon.ipynb](#) ). Due to time limitation, we cannot implement the same for Falcon 3 model.
- **Expand Target Modules:** Including `k_proj` could further refine the attention mechanism of the model.

# Conclusion

Despite limited resources, QLoRA enables us to fine-tune the LLM effectively on a single 6GB RAM GPU. The Python script shows that the finetuning process of Qwen significantly improves the correctness of SQL syntax.