Using LoRA to Fine Tune Gemma Models for Text to SQL

Code

DSAN 5800 Final Project Report

AUTHORS
CJ Jones
Tianyu Zhao

AFFILIATION
Georgetown University
Georgetown University

1 Abstract

A core competency of recent Large Language Models is an increasingly strong performance on structured translation tasks. The ability to convert natural language text into formats understandable by computers, such as SQL, offers new opportunities to transform unstructured data requests into precise, repeatable queries. This process, known as text-to-SQL generation, involves translating written text into SQL queries that can be executed on running databases. Although straightforward, this task presents several challenging obstacles. SQL queries need to be generated from variable database schemas, SQL has strict syntax requirements, and small differences in natural language prompting can make executable outputs very fragile to small perturbations in token arrangement. Since it is computationally expensive and practically infeasible to train a full-scale large language model for any non-enterprise application, this project aims to investigate whether a recently released low parameter LLM, Google's Gemma-3 4B model, can be fine-tuned to perform a variety of basic text-to-SQL tasks. Instead of updating all of the model's weights, fine-tuning is accomplished using a parameter-efficient method called Low-Rank Adaptation (LoRA), which introduces small trainable adapter modules into the otherwise frozen model, making the process computationally efficient and more feasible for limited resources. This project constructs a supervised fine-tuning pipeline that keeps the pre-trained backbone network fixed and uses a rank thirty-two LoRA adapter applied to all major projection matrices and quantized in four-bit NF4 precision. A large text-to-SQL corpus, WikiSQL, provides general SQL generation competency, while evaluation occurs on a held-out set of bifurcated WikiSQL tables to measure generalization.

Based on the evaluation results, the LoRA-fine-tuned Gemma model exhibits a substantial performance improvement over the baseline across all measured dimensions. While the baseline model fails to produce a single executable SQL query, the LoRA-adapted model achieves correct execution on over half of evaluated examples, with more than 99 percent of generated queries being syntactically valid. Text-based similarity metrics show corresponding gains, with BLEU increasing from 12.6 to 95.2, ROUGE-L from 0.67 to 0.97, and token-level F1 from 0.69 to 0.99. These results demonstrate that parameter-efficient fine-tuning enables a general-purpose language model to reliably acquire structured query generation capabilities, yielding large accuracy improvements without modifying or retraining the underlying pretrained backbone.

2 Introduction

Text to SQL generation addresses the problem of converting natural language questions into executable database queries. While modern large language models exhibit strong general language understanding and can often produce SQL-like text in zero shot or few shot settings, they struggle with the precision required for reliable query generation. SQL is unforgiving to small mistakes. A single incorrect column name, misplaced operator, or missing clause can cause a query to fail or return an incorrect result. In addition, SQL queries are always grounded in a specific database schema. A model must correctly identify which columns are available, determine how they relate to the question being asked, and assemble a query that is both syntactically valid and semantically correct with respect to the underlying table.

Adapting a large language model to meet these requirements poses a challenge. Fully fine tuning all model parameters is computationally expensive and often impractical outside of large scale training environments. Parameter efficient fine tuning methods offer an alternative by introducing a small number of trainable parameters into an otherwise frozen model. Low Rank Adaptation is one of the most effective of these techniques, allowing task specific updates to be learned through low dimensional projections while preserving the pretrained backbone. This approach dramatically reduces memory and compute requirements while still enabling meaningful behavioral adaptation. In this project, LoRA is applied to the Gemma 3 model with approximately four billion parameters in order to specialize it for structured SQL generation without modifying the full weight matrices.

The project is organized around three tightly connected stages. The first stage focuses on data preparation and prompt construction using the WikiSQL dataset, transforming each example into a structured prompt and completion format that aligns with causal language modeling. The second stage involves fine tuning the Gemma model using LoRA adapters under a supervised training objective that encourages accurate SQL continuation while keeping the backbone frozen and quantized. The final stage evaluates both the baseline and fine tuned models under identical conditions using an execution based pipeline that measures syntactic validity, semantic correctness, and textual similarity. Together, these stages provide a complete end to end examination of how parameter efficient fine tuning can adapt a general purpose language model to the text to SQL task.

3 Literature Review

The design of this project draws from several key areas in the literature. Parameter-efficient fine-tuning methods, including LoRA, have shown that full-model updates are unnecessary for many downstream tasks. The LoRA formulation introduced by Hu et al. (2021) demonstrates that weight updates in large transformer models often lie near low-rank subspaces. A frozen weight matrix W_0 is therefore augmented with a rank-r update expressed as

$$W = W_0 + \Delta W \qquad \text{where} \qquad \Delta W = BA$$

where $A \in \mathbb{R}^{k\times r}$ and $B \in \mathbb{R}^{r\times d}$ are trainable matrices with $r \ll \min(d, k)$. This formulation considerably reduces the number of trainable parameters while enabling expressive adaptation.

Dettmers et al. (2023) introduced QLoRA, which combines LoRA updates with four-bit quantization of the frozen base model. QLoRA demonstrated that powerful models can be fine-tuned on single-GPU hardware, laying the foundation for the training strategy used in this project. By quantizing the Gemma-3 4B model into NF4 format and training only LoRA adapters, this project follows the practical recommendations of QLoRA to maximize efficiency and stability.

The foundational computational structures of transformer models, including self-attention, residual pathways, and feed-forward projections, follow directly from the course materials in DSAN 5800. Larson (2025a) reviews sequence modeling fundamentals and motivates the use of attention mechanisms. Larson (2025b) expands this foundation into full transformer architectures, describing multi-head self-attention, positional encoding, and the stacking of transformer blocks. These theoretical components directly inform the deeper architectural review provided in this report.

Finally, the WikiSQL dataset has played a central role in text-to-SQL research since its introduction by Zhong et al. It is notable for its constrained SQL grammar and the large variety of tables, which makes it suitable for evaluating generalization across schema-specific tasks. Existing research demonstrates that models must integrate schema information and natural language semantics to perform well on WikiSQL. This project adopts the standard execution-based evaluation criterion, aligning with prior work.

Together, these strands of research motivate the methodological design of this project: use LoRA to specialize a mid-sized LLM for SQL generation, leverage four-bit quantization to control memory costs, and evaluate rigorously using execution correctness.

4 Dataset

This project relies exclusively on the WikiSQL dataset for both supervised fine-tuning and evaluation. The approach taken here uses only real WikiSQL examples so that the fine-tuned model learns directly from the human-generated question–SQL pairs and table schemas. This constraint has the large benefit that it simplifies the experimental setting and ensures that evaluation reflects performance on the same structural distribution seen during training.

4.1 Dataset Characteristics

WikiSQL is particularly well suited for studying foundational text-to-SQL behavior because its structure isolates the core challenge of translating natural language into relational queries without introducing unnecessary complexity. Each query operates over a single table and uses a limited set of operations, which keeps the focus on learning how questions map onto selections, filters, and simple aggregations. By excluding multi table joins and deeply nested logic, the dataset avoids confounding factors and makes it easier to attribute model errors to failures in schema understanding or language grounding rather than to advanced SQL mechanics.

At the same time, WikiSQL offers substantial diversity across schemas and tasks. The dataset contains tens of thousands of unique tables, each with different column names, orders, and value distributions. Because every question is tied to its own table, models cannot rely on memorized templates or familiar column patterns. Instead, they must learn how to interpret a question in the context of an arbitrary schema and determine which fields are relevant based solely on their names and structure. This forces the model to develop a more flexible form of schema alignment rather than exploiting superficial regularities. As a result, WikiSQL provides a strong setting for evaluating whether a model can generalize text to SQL behavior beyond previously seen tables.

Another defining feature of WikiSQL is the presence of a canonical SQL query for every natural language question. Each example includes a reference query that represents the correct interpretation of the question with respect to the table. These reference queries serve as the ground truth signal for supervised fine tuning and make it possible to evaluate predictions through direct execution. Both the model generated query and the reference query can be run against the same table to determine whether they return identical results, providing an unambiguous measure of semantic correctness. This combination of paired questions, schemas, and executable reference queries is a key reason the dataset remains widely used in research.

The dataset is also carefully partitioned to support meaningful evaluation. Tables are split across training, validation, and test sets so that no table appears in more than one split. This design prevents models from memorizing schema specific patterns and ensures that performance on the test set reflects genuine generalization to unseen tables. Together, these properties make WikiSQL a well controlled and effective benchmark for studying how language models learn to generate SQL from natural language. A summary of the dataset sizes used in this project is shown below:

Split	Rows	Used For
Train	56,355	Training set
Dev	8,421	Validation & monitoring
Test	15,878	Test Set

These counts correspond to the examples that remained after filtering malformed SQL programs, invalid schemas, or tables that could not be serialized cleanly. Each surviving row is represented using four structured features

1. A question in ordinary English.
2. The table schema, specifying all column names and data types.
3. The target SQL query, represented as both a logical tuple and executable SQL text.
4. Example table rows for reconstructing the corresponding SQLite table.

Example WikiSQL Data Item

QUESTION:

```
Tell me what the notes are for South Australia
```

TABLE HEADERS:

```
- State/territory
- Text/background colour
- Format
- Current slogan
- Current series
- Notes
```

EXAMPLE ROWS:

```
 | State/territory              | Text/background colour | Format      | Current slogan                        | Current
                               series | Notes                                     |
|-----------------------------|-----------------------|-------------|--------------------------------------|----------
                               ------|-------------------------------------------|
 | Australian Capital Territory| blue/white             | Yaa·nna     | ACT · CELEBRATION OF A CENTURY 2013  | YIL·00A
                                   | Slogan screenprinted on plate             |
 | New South Wales             | black/yellow           | aa·nn·aa    | NEW SOUTH WALES                       | BX·99·HI
                                   | No slogan on current series               |
 | New South Wales              | black/white            | aaa·nna     | NSW                                   | CPX·12A
                                   | Optional white slimline series            |
 | Northern Territory          | ochre/white            | Ca·nn·aa    | NT · OUTBACK AUSTRALIA                | CB·06·ZZ
                                   | New series began in June 2011             |
 | Queensland                   | maroon/white           | nnn·aaa     | QUEENSLAND · SUNSHINE STATE           | 999·TLG
                                   | Slogan embossed on plate                  |
 | South Australia             | black/white            | Snnn·aaa    | SOUTH AUSTRALIA                       | S000·AZD
                                   | No slogan on current series               |
 | Victoria                     | blue/white             | aaa·nnn     | VICTORIA - THE PLACE TO BE            | ZZZ·562
                                   | Current series will be exhausted this year |
```

LOGICAL SQL:

```
  sel  = 5
  agg  = 0
conds = [[3, 0, "SOUTH AUSTRALIA"]]
```

EXECUTABLE SQL:

```
SELECT "Notes"
FROM data
WHERE "Current slogan" = 'SOUTH AUSTRALIA'
```

Example 1. WikiSQL Illustration – Table preview, schema, question, and corresponding executable SQL

4.2 Preprocessing and Prompt Construction

To convert WikiSQL examples into training-ready sequences, each example is reformatted into a prompt–completion pair using a strict, instruction-driven structure. This design aligns with the constraints of a causal decoder-only model such as Gemma, which generates output exclusively through next-token prediction. **NEED TO HUMANIZE**[By placing the SQL query in the completion rather than embedding it within the prompt, the supervision signal directly mirrors the model's inference-time behavior.

The prompt itself is organized into clearly delineated sections - `<INSTRUCTIONS>`, `<SCHEMA>`, `<QUESTION>`, and `<SQL_Query>` - each represented by explicit opening and closing tags that appear verbatim in the input text. These tags function as unambiguous structural boundaries, ensuring that the model can reliably distinguish between natural-language instructions, schema metadata, question content, and the region where SQL generation begins.] Such explicit segmentation has been shown to stabilize text-to-SQL training by reducing format drift, preventing the model from mixing explanation and code, and framing SQL prediction as a discrete subtask rather than a free-form generation problem. The resulting prompt format is therefore both human-interpretable and optimized for predictable model behavior.

Importantly, in the training pipeline all row level table data and the logical SQL encodings (sel/agg/conds) are intentionally excluded from the LLM's input. Only the schema representation and the natural language question are included in the prompt, and only the executable SQL string appears in the completion. This design mirrors practical deployment settings, where exposing full table contents would be infeasible due to size, privacy, or API constraints. It also forces the model to infer column relevance, aggregation type, and filtering conditions purely from the question–schema alignment rather than memorizing example values. The exclusion of table rows prevents shortcut learning, such as exploiting statistical correlations between specific cell values and question patterns. By contrast, the serialized schema gives the model exactly the minimal structural information required to produce syntactically valid and semantically grounded SQL queries. The strict tag structure and constrained input-output interface jointly act as a curriculum and, along with the consistent instructions field, they work to guide the model toward producing clean, well-formed SQL while minimizing opportunities for hallucination or format drift.

The schema is serialized into the following uniform text structure:

PROMPT FIELD:

```
<INSTRUCTIONS>
You are a precise text-to-SQL generator. Using the known schema of the sql
database you must output only a valid SQL query and nothing else.
</INSTRUCTIONS>

<SCHEMA>
- State/territory (TEXT)
- Text/background colour (TEXT)
- Format (TEXT)
- Current slogan (TEXT)
- Current series (TEXT)
- Notes (TEXT)
</SCHEMA>

<QUESTION>
Tell me what the notes are for South Australia
</QUESTION>

<SQL_Query>
```

COMPLETION FIELD:

```
SELECT "Notes" FROM data
WHERE "Current slogan" = 'SOUTH AUSTRALIA'
</SQL_Query>
```

Example 2. Prompt and completion structure for text-to-SQL training – prompt with explicit instruction, schema, and question fields (top); model completion containing only executable SQL (bottom). This format mirrors real-world inference and enforces strict input–output separation, ensuring the model learns to generate SQL solely from the provided schema and question.

4.3 SQLite Reconstruction and Executable SQL

For evaluation, each WikiSQL table is materialized as a lightweight, in-memory SQLite database so that both gold and model-generated queries can be executed under a consistent, realistic SQL semantics. SQLite is a natural choice in this setting because it supports the core subset of SQL operators used in WikiSQL (selection, aggregation, and simple conjunctions), imposes strict syntactic requirements that expose malformed queries, and can be constructed directly from the table headers and rows provided in the dataset. Concretely, each table object is first converted into a Pandas DataFrame using the original column headers and row values.

This DataFrame is then written into an ephemeral SQLite database under a fixed table name (`data`) as shown:

```
import sqlite3

# df is the DataFrame representation of the WikiSQL table
conn = sqlite3.connect(":memory:")  # create an in-memory SQLite database
df.to_sql("data", conn, index=False, if_exists="replace")
```

The logical SQL encoding provided by WikiSQL (`sel`, `agg`, `conds`) is mapped into an executable SQL string using the `logical_to_sql()` helper, which expands the selected column index, aggregation operator, and condition tuples into a syntactically valid `SELECT ... FROM data WHERE ...` statement.

At evaluation time, both the gold SQL and the model's predicted SQL are executed against the same in-memory database using a shared `execute_sql()` wrapper:

```
def execute_sql(conn, sql):
    try:
        cursor = conn.execute(sql)
        return cursor.fetchall()
    except Exception as e:
        return f"Error: {e}"
```

This design serves multiple purposes: it enforces that the model produce not just syntactically valid SQL but also semantically executable queries; it decouples the modeling code from any particular storage backend; and it enables execution-based metrics (e.g., whether the predicted query returns exactly the same result set as the gold query) that directly reflect end-to-end task success.

5 Methods

5.1 Gemma-3 Model Architecture

Gemma-3 4B is a mid-sized transformer-based language model designed by Google with efficiency, adaptability, and fine-tuning stability as primary objectives. At approximately four billion parameters, it occupies a practical middle ground: large enough to support high-quality text generation and structured reasoning tasks, yet compact enough to be fine-tuned on a single modern GPU using parameter-efficient methods such as LoRA. The model follows the general decoder-only transformer paradigm common to contemporary large language models, but introduces several refinements—including hybrid attention patterns, large vocabulary capacity, and optimized projection structures—to improve performance on long-context, instruction-following, and code-generation tasks.

At a high level, Gemma-3 4B consists of 34 transformer blocks, each operating over a hidden dimensionality of 2560 units. The architecture uses rotary positional embeddings to encode word order and maintain relative positional invariances, which improves generalization across sequence lengths. Attention is implemented via multi-head self-attention with 8 heads, and select layers employ sliding-window attention to reduce computational cost on long sequences. These local attention blocks are interspersed with globally attentive layers to periodically refresh holistic contextual representations, a strategy that allows Gemma-3 to scale to long contexts without incurring the quadratic cost of full attention at every layer.

A summary of the most relevant architectural hyperparameters is presented below:

Table 1. Gemma-3 Architecture Hyperparameters

Feature	Value / Description
Total Trainable Parameters	1,342,504,960 (~23%)
Total Non-trainable Parameters	5,708,161,392
Number of Layers	34 Transformer blocks
Hidden Size	2560
Attention Heads	8 (4 key/value heads)
MLP Intermediate Size	10,240
Activation Function	GELU-Tanh
Vocabulary Size	262,208 tokens
Positional Encoding	RoPE with linear scaling (×8), effective context window: 131k tokens
Attention Type	Sliding-window (1024 tokens) + global full attention every 6 blocks

Each transformer block in Gemma-3 follows the established pattern of attention → feed-forward → residual connection, but with careful engineering to improve numerical stability and training efficiency:

1. Multi-Head Self-Attention (MHSA). The model computes learned projections for queries, keys, and values using matrices W_Q, W_K, and W_V. These are grouped into multiple heads, which allows the model to attend to different types of dependencies in parallel. Attention is computed via the scaled dot-product formulation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

The outputs of all heads are concatenated and passed through an output projection W_O before being added to the residual pathway.

2. Feed-Forward Network (FFN). Following attention, each block applies a feed-forward MLP of the form

$$\mathrm{FFN}(x) = W_{\mathrm{down}}\, f\big(W_{\mathrm{up}}\, x\big)$$

   where f is a smooth nonlinearity such as SiLU. The upward projection expands the dimensionality, allowing for richer intermediate representations, while the downward projection compresses the result back to the hidden dimension.

3. Normalization and Residual Structure. Each sublayer includes normalization layers to regulate activation magnitudes and improve gradient flow. The residual pathways serve as shortcuts that mitigate vanishing-gradient issues and stabilize both pretraining and fine-tuning.

5.2 Low-Rank Adaptation (LoRA) Configuration

LoRA is a parameter-efficient adaptation method that tunes large language models by inserting small, trainable matrices—called adapters—into selected linear projections within a frozen base transformer. For a given base weight matrix W_0, LoRA augments the weights with a low-rank update as follows:

$$W = W_0 + \frac{\alpha}{r} AB$$

where A and B are trainable matrices of size (d, r) and (r, k), respectively (r denoting the low-rank dimension), and α is a scaling factor to stabilize and scale the update. Only the small A and B matrices are modified during training; the full weight matrix W_0 remains fixed. This setup dramatically reduces the number of tunable parameters, as the count now grows linearly with r instead of with the size of W_0.

In our configuration, the LoRA adapters use rank $r = 32$ and scaling factor $\alpha = 16$, providing a strong balance between flexibility and stability on moderate hardware. Adapters are inserted into the most key linear projections: the query, key, value, and output weights in attention, as well as the gate, up, and down projections in the feed-forward sublayers. This placement enables the model to efficiently adapt its mechanisms for parsing schema and generating SQL from natural language.

Importantly, LoRA provides memory efficiency not just during training, but also when saving and deploying models. Because only the adapter (LoRA) weights are updated, these small matrices—rather than the entire set of base model weights—can be saved after fine-tuning. This means that storage and sharing of task-specific models becomes highly efficient: just the adapter file needs to be retained, and later it can be “attached” or merged with any compatible base model for easy reuse or deployment, without duplicating the large pretrained model files.

Our broader training approach follows the QLoRA framework: the base Gemma weights are held fixed and stored in quantized (NF4) precision for substantial GPU memory savings, while the smaller LoRA adapters are kept at higher precision for effective optimization. This decoupling lets us train and deploy strong models even on limited hardware, and further maximizes storage and sharing efficiency by siloing the small, learnable adapter parameters from the large, frozen base model.

The Colab training environment configures LoRA using the following settings

```
peft_config = LoraConfig(
    r=32,
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
    bias="none",
    task_type="CAUSAL_LM",
    modules_to_save=["lm_head", "embed_tokens"]
)
```

This configuration directs LoRA to focus its adaptation capacity on the projections most critical for SQL reasoning. It also preserves the token embedding and final language modeling head in full precision which helps maintain stable output distributions during training. The result is an efficient and effective fine tuning mechanism that improves task specific behavior without modifying the billions of pretrained parameters in the Gemma model.

5.3 Training Plan

The goal of the training pipeline is to adapt the Gemma 3 model, with roughly four billion parameters, to the text-to-SQL task using LoRA adapters while keeping the pretrained backbone frozen and quantized. The method follows a causal language modeling objective, which means that the model is trained to predict the next token in a sequence that contains both the prompt and the gold SQL completion. Because SQL prediction in deployment is also a continuation-based process, the training objective and inference behavior remain aligned, reducing the risk of format drift and encouraging the model to internalize SQL generation as a structured next-token prediction problem. Only the LoRA parameters are updated during training while the transformer backbone stays fixed in 4-bit NF4 precision. This separation provides a stable optimization surface and keeps the computational and memory demands well within the limits of a single GPU environment.

The WikiSQL dataset, once preprocessed, is fed into the model as a series of prompt–completion pairs. Each prompt contains an instruction block, a serialized table schema, and the user question, all wrapped in explicit XML-like tags that allow the tokenizer and model to reliably partition the semantic components of each example. The completion field then contains the corresponding SQL query ending with a closing tag. These examples are loaded into Hugging Face Datasets objects, which provide efficient memory mapping and automatic integration with the tokenization pipeline. During training, the SFTTrainer prepares each example by concatenating the prompt and SQL completion into a single sequence that reflects the exact text the model will see during inference. The tokenizer then applies Gemma's byte-pair encoding rules to convert text into token IDs, and the trainer assembles these token sequences into batches. Because packing=True is enabled, multiple short examples are combined into the same 512-token window, reducing padding waste and ensuring that GPU memory is used efficiently.

Once tokenized, each sequence is copied into a parallel label sequence that is shifted by one position. This means that for token position t, the model attempts to predict token t+1. The resulting cross-entropy loss is computed only over the completion region of each sequence, so the model is not penalized for regenerating instructions or schema information. This selective masking ensures that the optimization pressure focuses on SQL formation rather than reproducing prompt structure. The causal objective therefore becomes a direct mechanism for teaching the model how to form valid SQL: each incorrect token prediction increases the loss, and the LoRA adapters update in a direction that improves the syntactic and semantic correctness of future predictions.

The mechanics of loss reduction follow the standard structure of a decoder-only transformer. For every token position, Gemma produces a probability distribution over the entire vocabulary. The training signal compares this distribution to the true next token through a cross-entropy objective. When the predicted distribution assigns too little mass to the correct SQL token, the associated gradient flows backward through the LoRA adapter matrices and updates them to improve future predictions. The frozen 4-bit backbone never changes, but it continues to supply rich contextual and semantic embeddings. The LoRA matrices essentially bend these pretrained features toward SQL behavior without disturbing the foundational linguistic knowledge encoded in Gemma. Gradient accumulation is employed to expand the effective batch size: although the nominal batch size per device is one, gradients from four successive forward–backward passes are aggregated before the optimizer performs a weight update. This method stabilizes training while avoiding memory overflows.

The training configuration, defined through SFTConfig, governs all aspects of optimization and resource management. The 512-token sequence length reflects the short nature of WikiSQL examples and avoids unnecessary cost. Packed sequences improve throughput, which is particularly important for quantized models whose performance often depends on efficient batching. Three epochs provide adequate exposure to the training distribution without encouraging memorization of table-specific patterns, especially because WikiSQL partitions tables across splits to prevent leakage. The learning rate is intentionally conservative, following QLoRA recommendations, because adapter tuning on a quantized backbone can become unstable at higher rates. A warmup period ensures that the optimizer does not immediately apply large updates, which helps prevent divergence in the early stages of training. Mixed precision, either FP16 or BF16 depending on hardware capabilities, reduces memory usage and speeds up computation. The use of gradient checkpointing further lowers memory pressure by recomputing intermediate activations during the backward pass. The constant learning rate scheduler maintains a steady optimization environment once warmup is complete. Each of these settings contributes to an efficient and stable fine-tuning procedure suitable for limited hardware.

During training, the SFTTrainer logs several metrics that provide insight into learning dynamics. The primary signal is the training loss, which shows how effectively the model is predicting SQL tokens as training progresses. Additional logs include the active learning rate, the number of tokens processed per second, and internal trainer statistics.

The training pipeline relies on an integrated stack of Python libraries. Hugging Face Transformers provides the Gemma architecture along with quantization hooks that allow NF4 loading through BitsAndBytes. The TRL library supplies the SFTTrainer, which automates dataset packing, label masking, forward passes, and loss computation for causal LM objectives. The PEFT library manages LoRA initialization, weight injection, and adapter merging. Hugging Face Datasets provides the backbone for efficient preprocessing and dataset streaming. SQLite and Pandas support evaluation, enabling real SQL execution through a lightweight, in-memory database. TensorBoard records scalar metrics that allow visual inspection of the loss curves throughout training. This entire stack is orchestrated inside a Google Colab environment, with persistent storage managed through Google Drive and experiment metadata captured through timestamped directories.

From end to end, training proceeds through a clear sequence of steps. WikiSQL examples are loaded, validated, and converted into prompt–completion structures. These examples are tokenized, packed, and fed into the model during supervised tuning. The Gemma backbone remains fixed while LoRA adapters gradually learn task-specific behavior. The trainer monitors loss and writes checkpoints at the end of each epoch. After fine-tuning is complete, the adapter weights and tokenizer are saved, and the model is evaluated through SQL execution on reconstructed SQLite tables.

5.4 Text to SQL Experimental Setup

The evaluation procedure is designed to compare two models under identical conditions. The first model is the unmodified Gemma 3 4B pretrained checkpoint, which serves as the baseline. The second is the fine-tuned LoRA-augmented Gemma 3 4B model produced by the training pipeline described earlier. Both models are evaluated using the same prompt generation process, the same SQL cleaning and extraction logic, and the same SQLite execution environment. This design ensures that measured performance differences derive from learned SQL generation behavior rather than from changes in preprocessing or evaluation tooling.

Evaluation begins by converting each WikiSQL test example into the same structured prompt format used during supervised training. The prompt consists of an instruction block, a serialized representation of the table schema, and the natural-language question. These components are wrapped in explicit XML-like tags, which allow the model to identify the point where SQL generation begins. The model receives no information about table rows during inference; it must infer the correct aggregation, selection, and filtering operations entirely from the schema and question. The prompt is then fed into the model, which generates a continuation consisting of a predicted SQL query. The query is extracted and cleaned through a standardized parsing step that identifies the first valid SQL-like region inside the `<SQL_Query>` block and removes any trailing explanation or stray text the model may have produced. This procedure, applied uniformly to both models, isolates the predicted SQL in a consistent and deterministic manner.

The predicted SQL is evaluated in several ways. First, the query is executed against a reconstructed SQLite database created from the table rows provided in the WikiSQL test set. Because each table in WikiSQL includes fully specified column names and row data, it is straightforward to materialize a corresponding SQLite table inside an in-memory database and then execute both the gold query and the predicted query within that environment. Execution results are compared directly. If the predicted SQL runs without a syntax error and returns the same result rows as the gold query, it is marked as correct. If the query is executable but returns an incorrect result set, it is counted as a wrong-result case. If the SQL engine raises a syntax or runtime error, the prediction is categorized as invalid. This execution-based evaluation is the most stringent test available because it verifies both syntactic correctness and semantic alignment with the true relational operation described in the question.

Beyond execution correctness, the system performs a suite of similarity measurements. These include Jaccard token similarity, normalized Levenshtein similarity, and structural alignment metrics that compare selected columns, referenced columns in the WHERE clause, and operator usage. These structural scores are important because SQL queries can differ superficially while still expressing the same computation, and in other cases, they can appear similar while differing in a subtle but meaningful way. Text-based metrics such as BLEU and ROUGE are computed as well, along with exact-match comparisons and token-level F1 scores. The combination of execution and similarity metrics provides a multidimensional picture of the model's predictive behavior and helps identify whether errors arise from structural misunderstandings, lexical drift, or outright semantic failures.

Inference with Gemma 3 4B is not instantaneous. Each query requires prompt construction, tokenization, autoregressive generation, SQL cleaning, schema materialization, and SQLite execution. The average end-to-end time for a single evaluation example is approximately fifteen seconds in the Colab environment used for this project. Running the full WikiSQL test set, which contains over fifteen thousand examples, would therefore require several hours of continuous inference, far exceeding the time available under practical constraints. To address this constraint while preserving statistical integrity, the evaluation operates on a random ten percent sample of the full test set. A reproducible random seed selects this subsample, allowing the experiment to run within an acceptable time window while maintaining representative coverage of the underlying data distribution. Each sampled example goes through the full evaluation sequence, and the aggregated results constitute the reported performance metrics for both the baseline and trained models.

The final evaluation step involves computing summary statistics over all predicted queries. These summaries include the proportion of syntactically valid SQL queries, the proportion that execute correctly, averaged similarity metrics, and aggregate BLEU, ROUGE, EM, and F1 values. Row-level logs are saved in JSONL format, preserving each question, predicted SQL, gold SQL, execution result, and error type, while a separate summary file records the global metrics that form the basis of the quantitative results in the report. All evaluations are performed twice, once for the baseline model and once for the LoRA-fine-tuned model, enabling a direct and controlled comparison of the effect of LoRA adaptation on SQL generation behavior.

6 Results

This section presents the empirical results of training and evaluating the LoRA-fine-tuned Gemma 3 model on the WikiSQL task. Results are organized to first establish training stability, then compare baseline and fine-tuned models under identical evaluation conditions. All reported metrics are computed on the same randomly sampled ten percent of the WikiSQL test set using identical prompt construction, SQL extraction, and SQLite execution logic.

6.1 Training Results

Figure 1 reports mean token-level accuracy over training steps for the LoRA-fine-tuned Gemma model. Token accuracy rises sharply during the initial phase of training. It stabilizes near the end, indicating that the adapter parameters rapidly learn the structure of SQL generation as they converge to a stable solution. Occasional downward spikes occur throughout training, consistent with the presence of more difficult batches containing longer or structurally diverse queries. Significantly, the model recovers quickly after these drops, suggesting that optimization remains well-behaved and that adapter training does not introduce catastrophic instability.

Figure 2 shows the gradient norm over training steps. Early training is characterized by higher and more variable gradient magnitudes, reflecting the initial adaptation of LoRA parameters to the SQL task. As training progresses, the gradient norm decreases and stabilizes, indicating convergence and diminishing parameter updates. The absence of sustained growth or runaway spikes confirms that training does not suffer from gradient explosion despite operating on a quantized backbone.

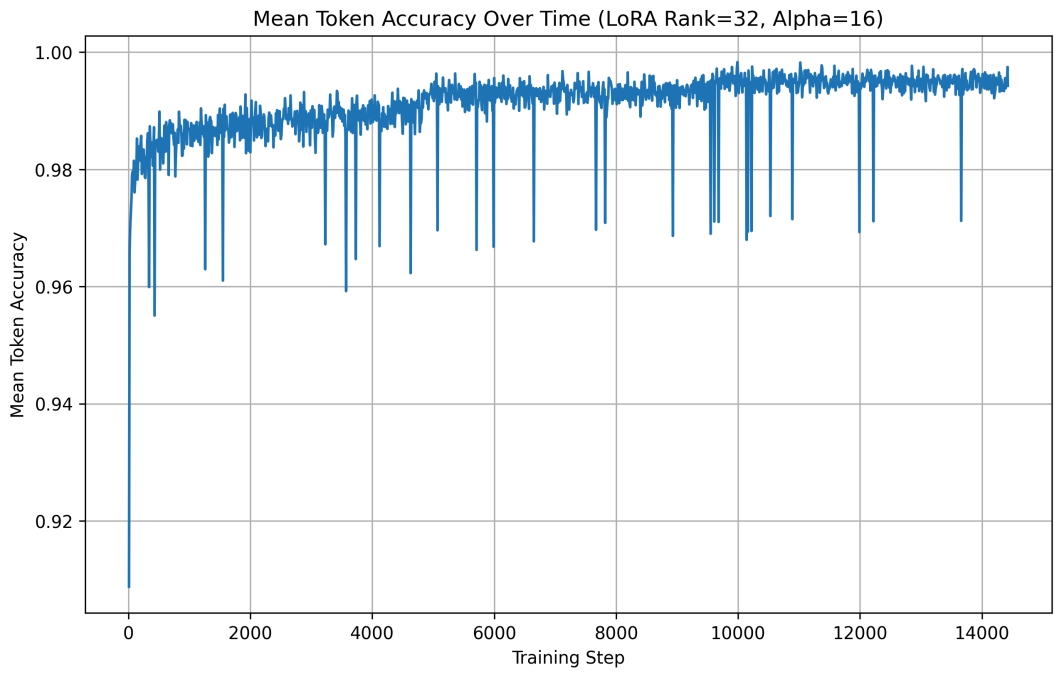

Fig. 1: Mean token-level accuracy during LoRA fine-tuning, showing rapid convergence and stable training.

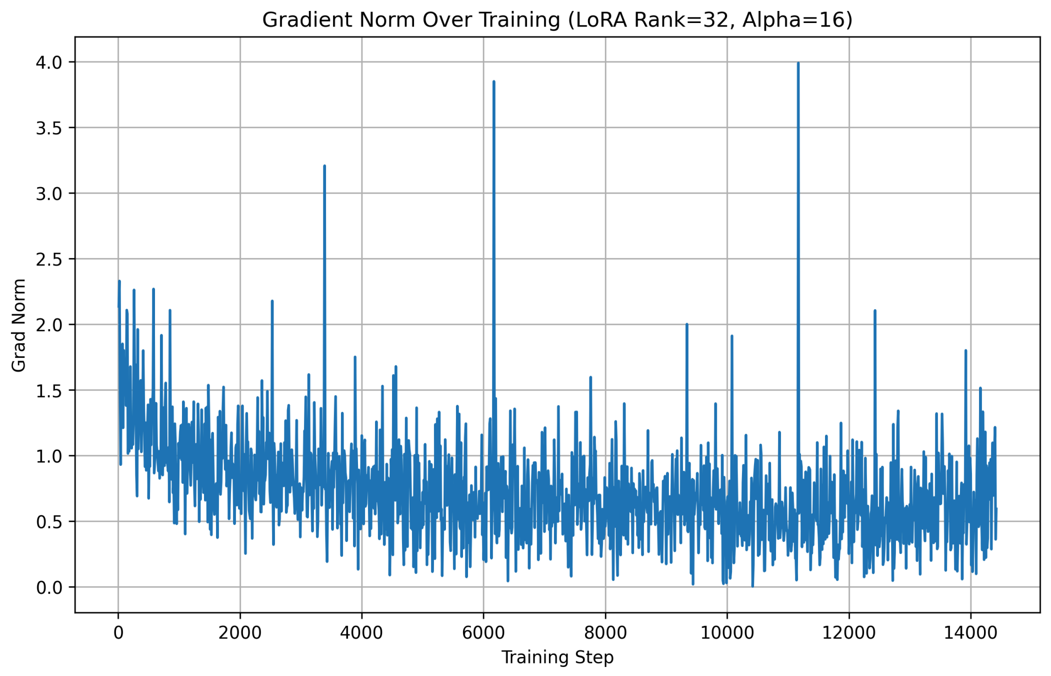

Fig. 2: Gradient norm during LoRA fine-tuning, illustrating convergence and stable parameter updates.

Finally, and most importantly for the training cycle, Figure 3 displays the training loss curve. Loss decreases rapidly at the beginning of training, followed by a slower but consistent downward trend as the model refines its predictions. The overall shape of the curve indicates effective learning with no signs of divergence. Together, these three plots demonstrate that LoRA fine-tuning is stable, efficient, and well-suited to adapting Gemma for structured SQL generation.

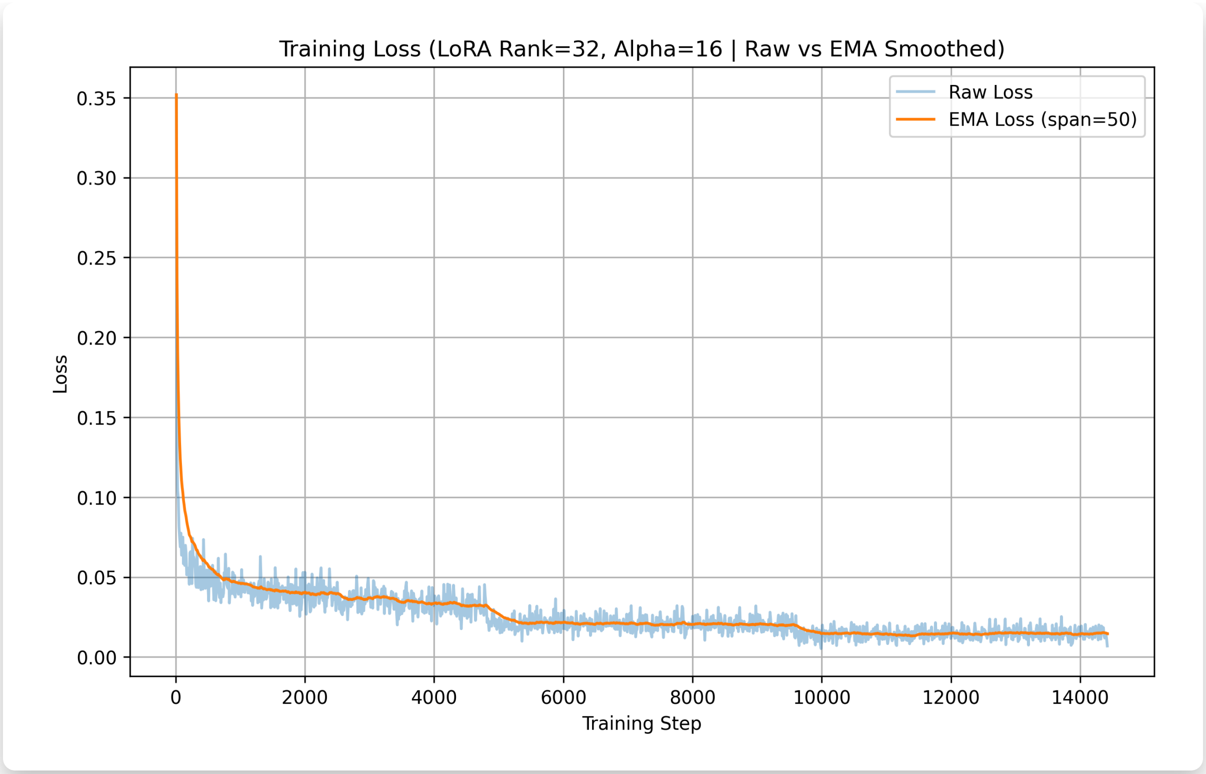

Fig. 3: Training loss over epochs, showing rapid initial decline and continued improvement as fine-tuning progresses.

6.2 Text to SQL Execution Evaluation

Execution accuracy is the most demanding evaluation criterion in text-to-SQL tasks because it simultaneously tests syntactic correctness and semantic alignment with the underlying database. A query must not only conform to valid SQL syntax, but also be valid. However, it must also encode the correct relational logic so that, when executed, it returns the same result as the reference query. For this reason, execution-based evaluation provides a stronger signal than surface-level similarity metrics. It serves as the primary indicator of whether a model can be deployed in a real database setting.

Figure 4 presents the breakdown of the SQL prediction error for the baseline Gemma model without LoRA adaptation. All evaluated predictions fall into the SQL error category, meaning that none of the generated queries execute successfully against the reconstructed SQLite tables. In practice, these failures arise from a combination of malformed SQL syntax, invalid column references, incorrect quoting, and structurally incomplete queries. This outcome underscores a central challenge in text-to-SQL generation. Although large pretrained language models possess strong general linguistic capabilities, they do not reliably internalize the rigid structural constraints required for executable SQL without explicit task-specific supervision. The baseline results, therefore, establish a clear lower bound and underscore the need for targeted adaptation.

Figure 5 shows the breakdown of execution errors for the LoRA-fine-tuned Gemma model. In sharp contrast to the baseline, more than half of the generated queries now execute correctly and return results that exactly match the gold query output. A substantial fraction of the remaining predictions produce syntactically valid SQL that executes successfully but yields an incorrect result set. These cases typically reflect partial semantic mismatches such as selecting the wrong column, applying an incorrect filter condition, or omitting a required aggregation. Importantly, only a very small proportion of predictions result in execution failures due to syntax or runtime errors.

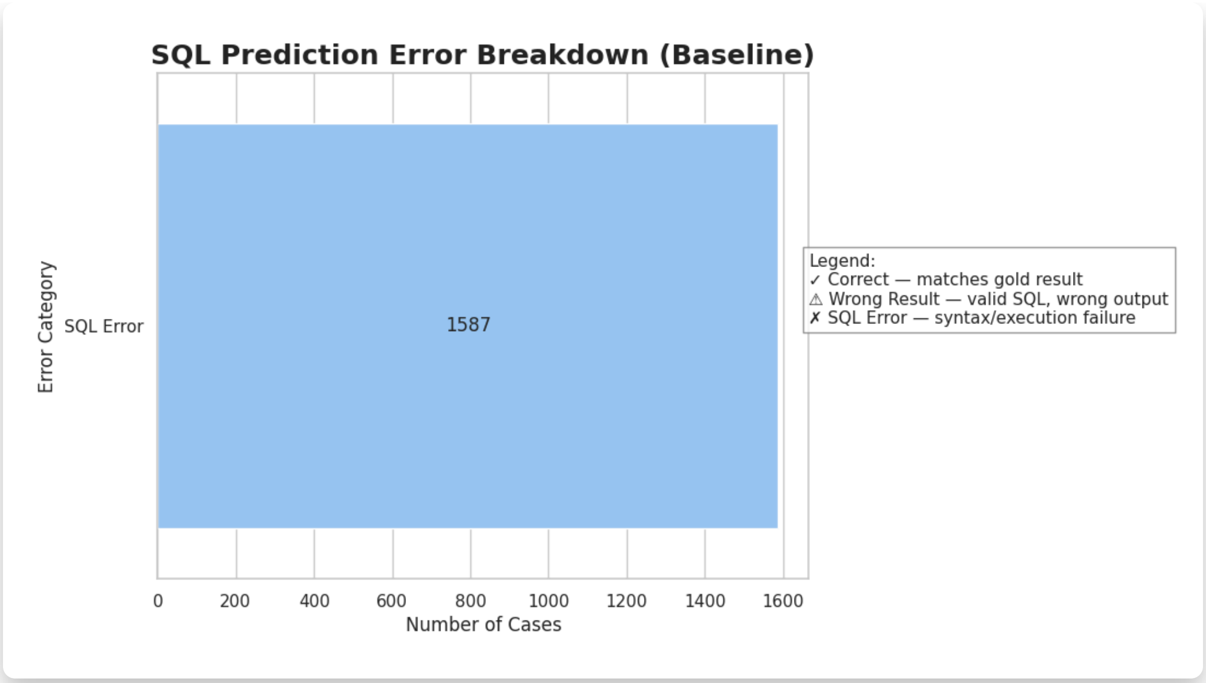

Fig. 4: Error breakdown for baseline Gemma model predictions. All generated queries fail to execute, with errors stemming from syntax and schema issues.

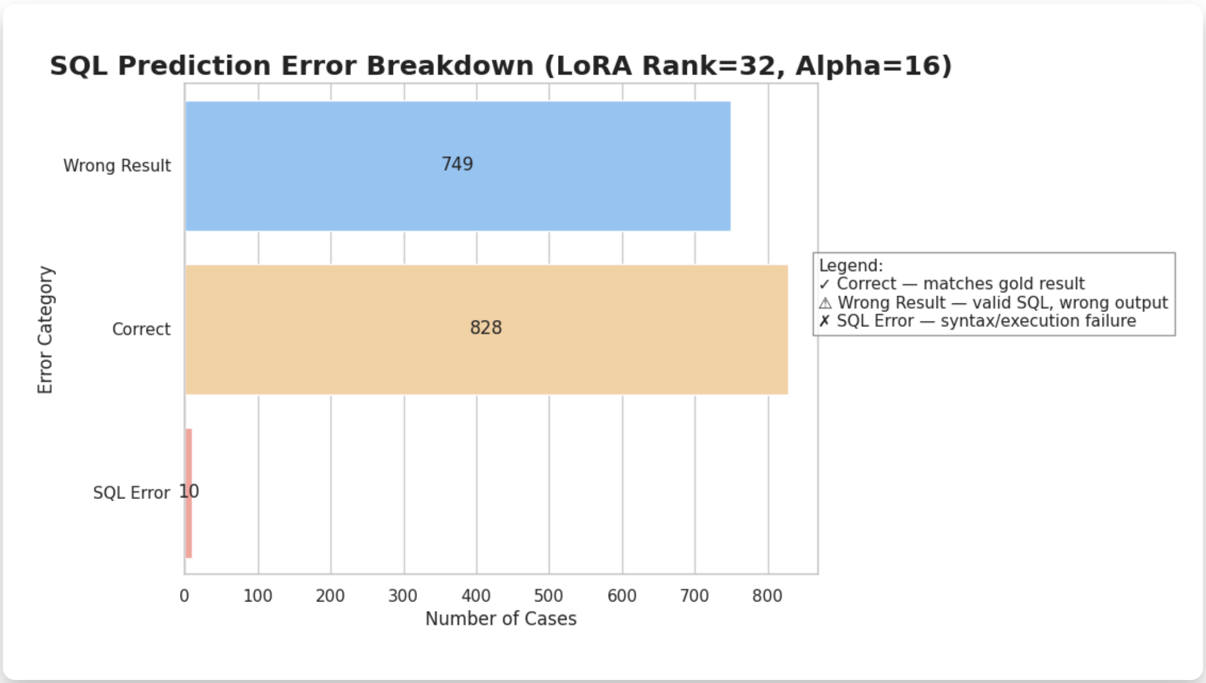

Fig. 5: Error breakdown for LoRA-fine-tuned Gemma model predictions. More than half of queries now execute correctly, with most remaining errors due to semantic mismatches rather than syntax or runtime issues.

This shift in the error distribution is significant for several reasons. First, it demonstrates that LoRA fine-tuning is sufficient to teach a large pretrained model the structural rules of SQL, even when the backbone parameters remain frozen and quantized. The dramatic reduction in execution errors indicates that the model has learned to produce well-formed SQL statements that respect schema constraints consistently. Second, the presence of a sizable valid-but-wrong category suggests that the remaining errors are primarily semantic rather than syntactic. This distinction is important because semantic errors are generally easier to address through additional training data, improved schema encoding, or more refined prompt design. In contrast, syntactic failures often indicate a more fundamental inability to represent the target language. Finally, the emergence of a majority correct category confirms that the fine-tuned model has crossed a qualitative threshold from non-functional to operational, achieving a level of reliability that supports meaningful downstream evaluation and comparison.

Taken together, the execution results provide strong evidence that parameter-efficient fine-tuning fundamentally alters the model's behavior. LoRA adaptation transforms the baseline Gemma model from a system that consistently fails to generate executable SQL into one that can reliably translate natural-language questions into correct database queries, all while maintaining a modest computational footprint.

6.3 Text to SQL Text Based Metrics

Text-based evaluation metrics quantify how closely a generated SQL query matches the reference SQL at the sequence and token levels, without requiring database execution. These metrics capture complementary aspects of similarity. BLEU measures n-gram overlap and emphasizes local token ordering. ROUGE-L measures the longest common subsequence overlap and reflects global structural alignment. Exact match reports the proportion of predictions that are identical to the gold SQL after normalization. Token-level F1 measures overlap at the level of individual SQL tokens and is less sensitive to ordering or minor syntactic differences. Together, these metrics provide a detailed view of how well the model reproduces the surface form and structure of SQL queries.

Figure 8 reports these metrics for the baseline Gemma 3 model without LoRA adaptation. The baseline achieves moderate token-level F1, indicating that it often generates correct SQL tokens such as column names, aggregation functions, or operators. However, exact-match accuracy is zero, and BLEU remains low. This gap reveals that while the baseline model recognizes SQL-relevant vocabulary, it fails to assemble these tokens into complete, correct queries consistently. ROUGE-L further confirms partial structural overlap without full sequence alignment. These results show that surface-level similarity alone is insufficient for correct SQL generation when the model lacks task-specific training.

Figure 9 presents the same metrics for the LoRA-fine-tuned model. All scores increase substantially. Exact match accuracy exceeds seventy-six percent, demonstrating that a large majority of predictions are entirely identical to the gold SQL. Token-level F1 scores approach one, indicating near-perfect token selection. BLEU and ROUGE-L scores also rise sharply, reflecting strong alignment in both local token sequences and overall query structure. Unlike the baseline model, the fine-tuned model consistently produces SQL that matches not only the semantics but also the precise syntactic form of the reference queries.

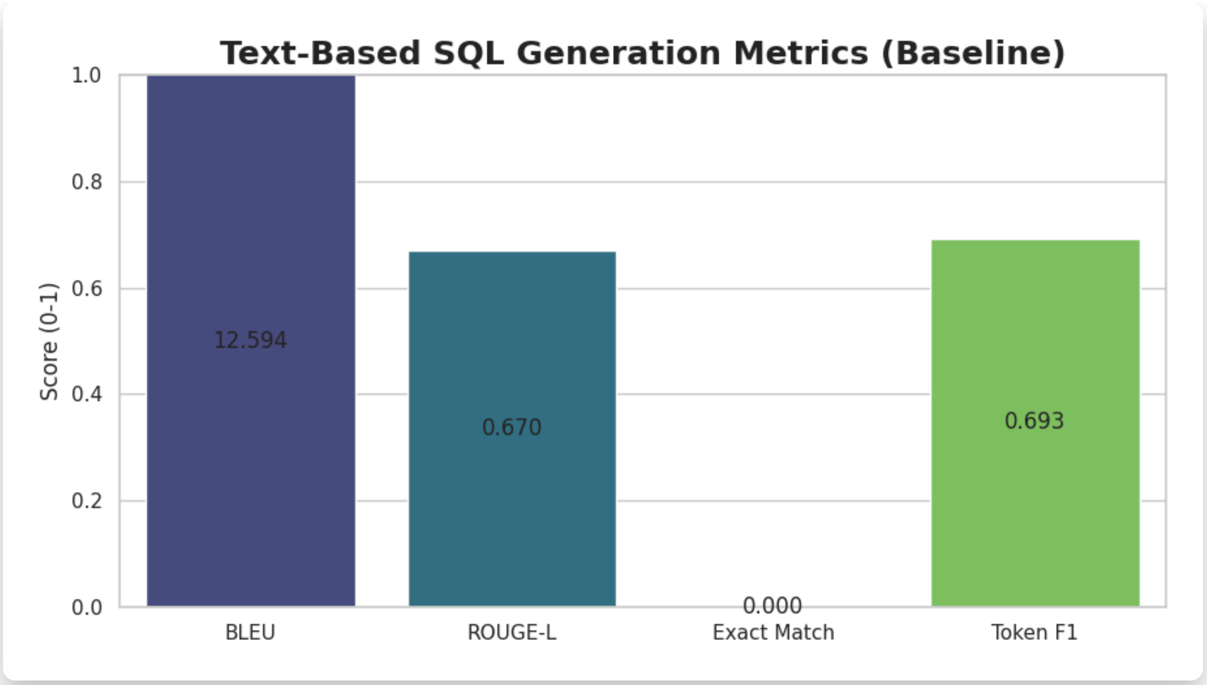

Fig. 6: Baseline – Text-based Evaluation Metrics

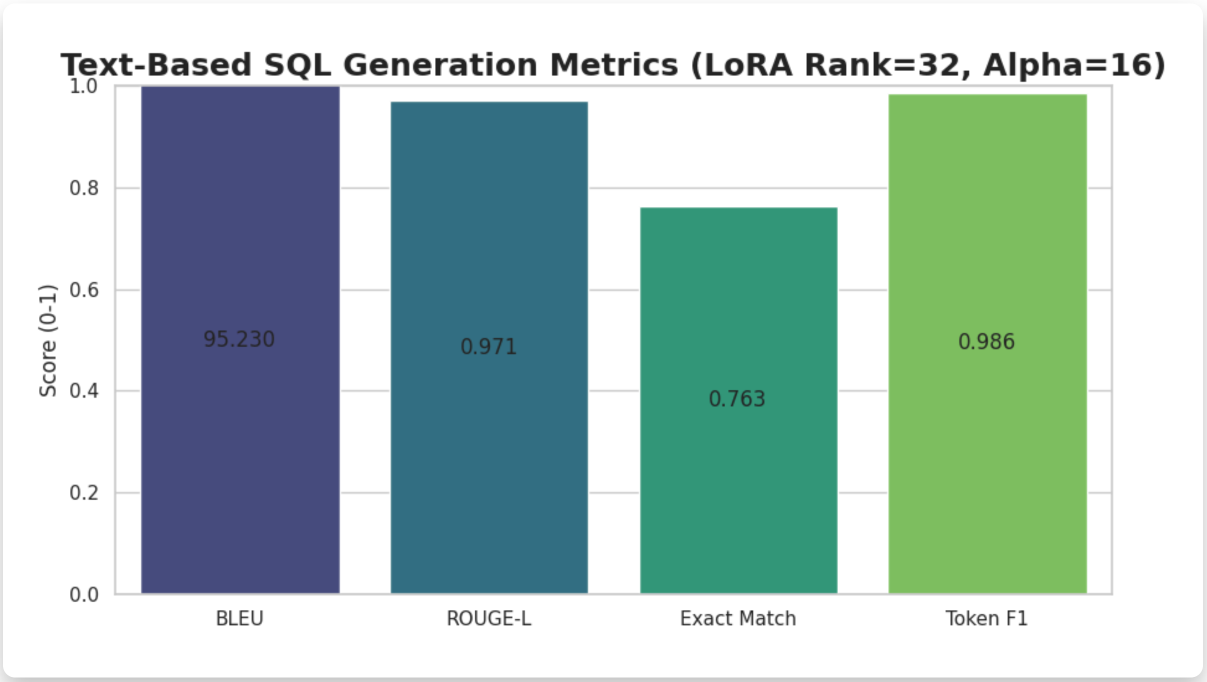

Fig. 7: LoRA – Text-based Evaluation Metrics

The contrast between Figures 8 and 9 clearly illustrates the effect of LoRA adaptation. Fine-tuning transforms the model from one that produces loosely SQL-like text into one that reliably generates structured, reference-aligned queries. These improvements demonstrate that LoRA fine-tuning not only increases execution success but also enforces strong textual and structural fidelity. This alignment is particularly important for downstream systems that rely on deterministic SQL generation, where even minor deviations in syntax or ordering can lead to incorrect results or execution failures.

6.4 Text to SQL Similarity Evaluation

Figure 6 and Figure 7 present the distributions of Jaccard and normalized Levenshtein similarity between gold SQL queries and model predictions for the baseline and LoRA-fine-tuned models, respectively. These metrics capture token overlap and edit distance, providing a view of structural and lexical similarity independent of execution.

For the baseline model in Figure 6, similarity scores are broadly distributed. Jaccard overlap is moderate, and Levenshtein similarity varies widely, indicating that while some predicted queries share tokens with the gold SQL, many differ substantially in structure or ordering. This pattern reflects the baseline model's tendency to produce SQL-like fragments without reliably assembling them into correct or complete queries.

In contrast, Figure 7 shows that the LoRA-fine-tuned model's similarity scores are tightly concentrated around one. Both Jaccard and Levenshtein similarity cluster at high values, indicating that fine-tuning enables the model to consistently generate SQL queries that closely match the gold references at both the token and sequence level. This shift confirms that LoRA adaptation substantially improves structural fidelity in SQL generation.

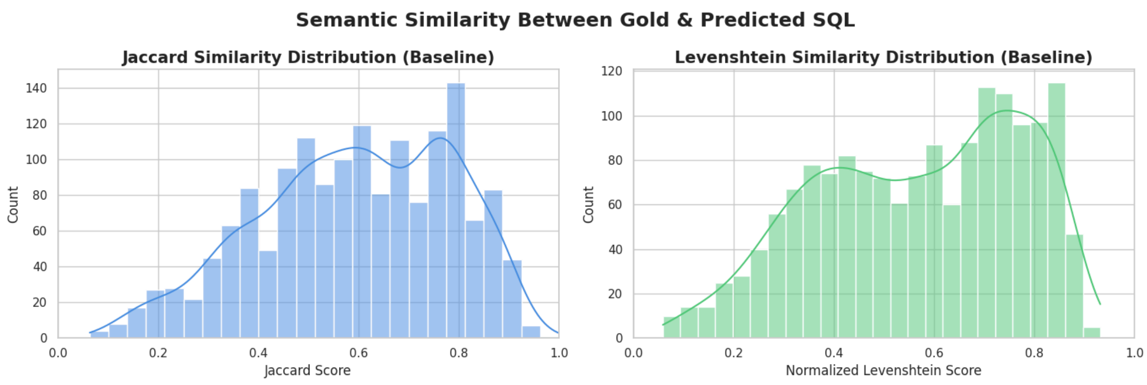

Fig. 8: Baseline model – Jaccard and Levenshtein similarity distributions between predicted and gold SQL. Score distributions are broad, indicating low and variable structural fidelity.

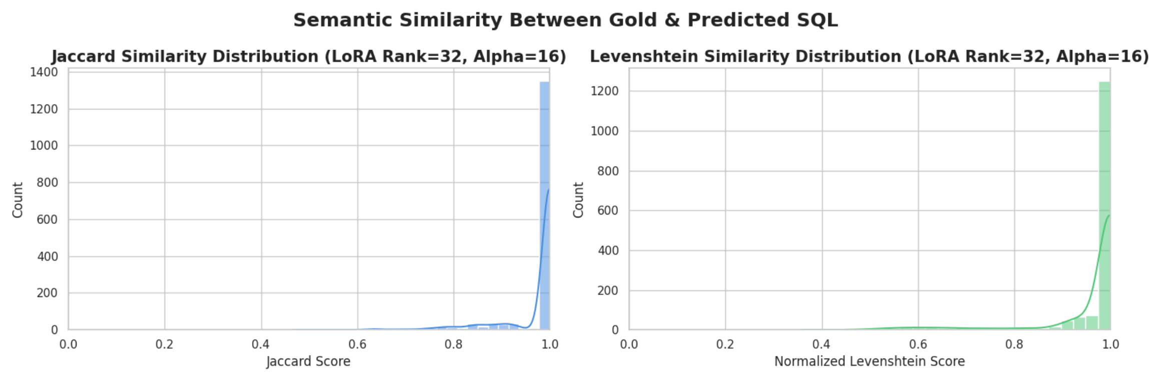

Fig. 9: LoRA-fine-tuned model – Jaccard and Levenshtein similarity distributions between predicted and gold SQL. Scores are tightly clustered at high values, showing consistent structural similarity after fine-tuning.

6.5 Text to SQL Similarity and Text Based Metric Correlations

The similarity–correctness plots in Figure 10 and Figure 11 reveal an initially counterintuitive result. While the baseline model shows no correlation between similarity scores and execution correctness for expected reasons, the LoRA-fine-tuned model also fails to exhibit a strong monotonic relationship between textual similarity and correct execution. This outcome is surprising because the fine-tuned model achieves very high BLEU, ROUGE-L, and token-level F1 scores, suggesting near-perfect textual alignment with the gold SQL.

For the baseline model, the lack of correlation is straightforward to explain. Because none of the generated queries execute correctly, similarity metrics vary independently of correctness. Token overlap and edit distance reflect superficial resemblance to SQL syntax but provide no indication of semantic validity. This confirms that similarity-based metrics are not meaningful indicators of correctness when the model has not learned the underlying relational structure of SQL.

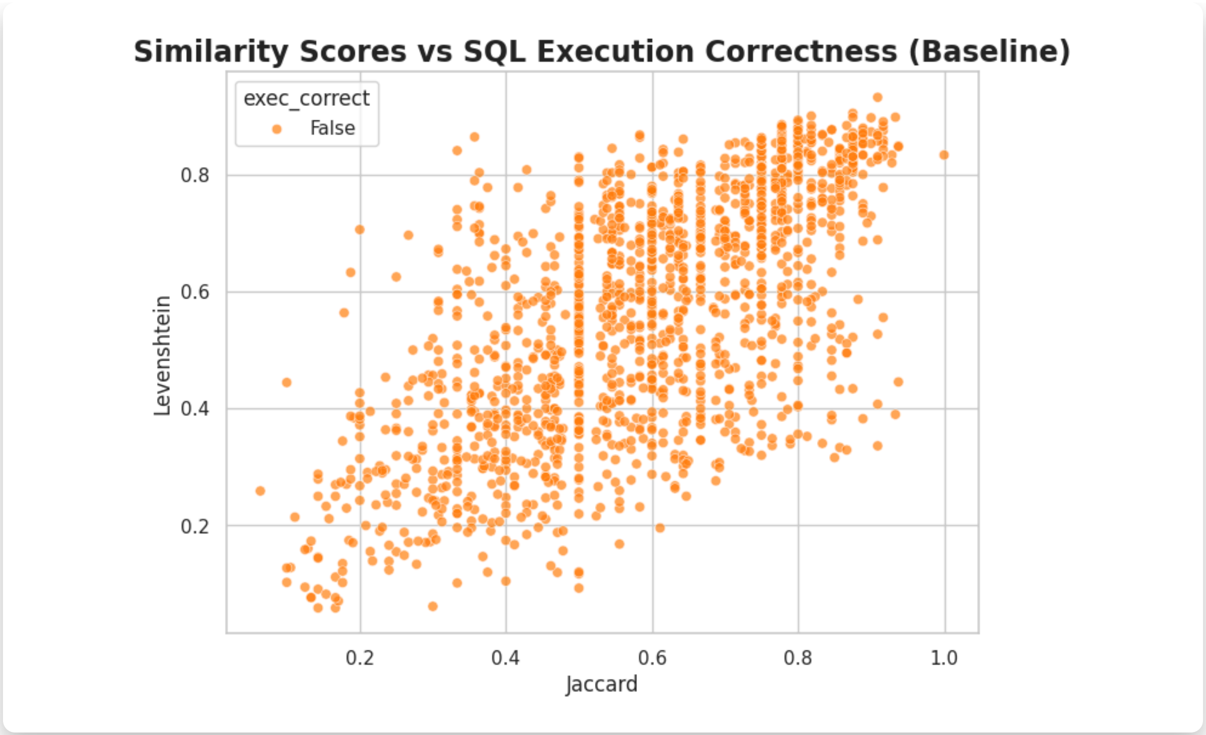

Fig. 10: Baseline – Similarity vs Correctness.

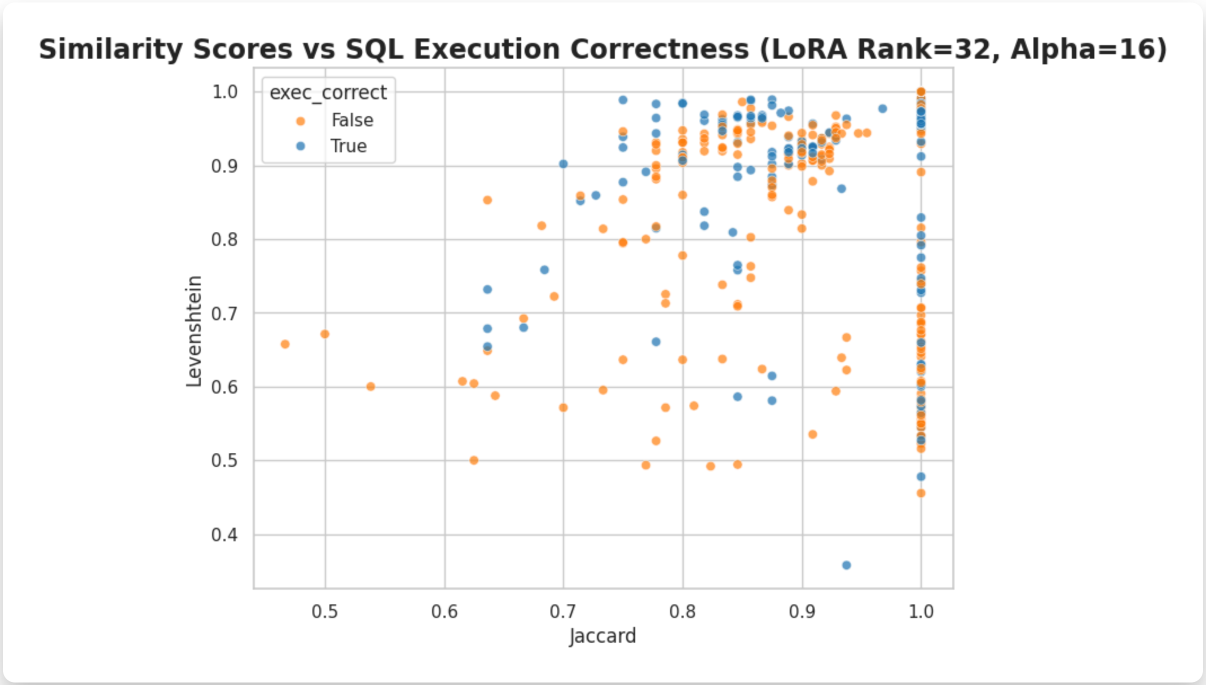

Fig. 11: LoRA – Similarity vs Correctness.

The LoRA-fine-tuned model, however, presents a more subtle and informative pattern. Although correct executions tend to cluster at high similarity values, a large number of incorrect executions also occur in the same high-similarity region. In other words, even when the predicted SQL is nearly identical to the gold query at the token and sequence level, it may still return an incorrect result. This weak correlation suggests that small semantic errors dominate the remaining failure modes. Minor deviations—such as an incorrect comparison operator, a swapped column in the WHERE clause, or a missing aggregation—can preserve most of the query's surface form while altering its meaning enough to affect execution.

This behavior highlights a fundamental limitation of text-based similarity metrics for evaluating structured prediction tasks. SQL semantics are brittle: a single-token difference can completely alter query behavior, yet have minimal impact on BLEU, ROUGE, Jaccard, or Levenshtein scores. As a result, high textual similarity does not guarantee semantic equivalence, even for a well-trained model. The LoRA-fine-tuned model has clearly learned the syntax and canonical structure of SQL, but its remaining errors reflect logical misalignment rather than representational noise.

From a modeling perspective, this result suggests that the LoRA adapters have successfully constrained the output space to well-formed SQL. However, the training objective does not explicitly penalize semantically incorrect but syntactically plausible queries. The causal language modeling objective optimizes next-token likelihood rather than execution behavior. Consequently, the model learns to reproduce the most probable SQL patterns observed during training, even when those patterns encode slightly incorrect logic for a specific question.

6.6 Example Prompt Comparison

The following example illustrates a representative failure mode of the baseline model and highlights how LoRA fine-tuning corrects it by enforcing schema awareness and an executable SQL structure. For the prompt *"What's the title of the episode that Rob Heyland wrote?"*, both models attempt to construct a simple selection query conditioned on the `Writer` column. However, the baseline model produces SQL that does not correctly align with the database schema, while the LoRA-fine-tuned model generates an exact match to the gold query.

The baseline model's query, `SELECT Title FROM Total WHERE Writer = 'Rob Heyland'`, fails at execution time because it incorrectly treats a column name as a table identifier. Although the table contains a column labeled `Total#`, the actual table name in the SQLite reconstruction is `data`. This error demonstrates a common baseline failure pattern observed throughout the evaluation. The model captures the general intent of the query and selects the appropriate column and condition. However, it fails to ground those tokens in the concrete schema representation provided in the prompt. As a result, the query appears SQL-like yet remains non-executable. This behavior aligns with the broader finding that the baseline model often produces high-similarity but semantically invalid SQL.

By contrast, the LoRA-fine-tuned model produces ` SELECT "Title" FROM data WHERE "Writer" = 'Rob Heyland', which exactly matches the gold SQL query. This output correctly references the table name, quotes column identifiers appropriately, and encodes the correct filtering logic. The successful execution and matching result confirm that fine-tuning enables the model to reliably map natural-language questions to schema-grounded SQL expressions rather than relying on superficial token patterns.

Overall, this example encapsulates the broader effect of LoRA fine-tuning observed across the evaluation. The baseline model demonstrates approximate pattern matching without accurate schema understanding, while the fine-tuned model consistently produces executable, schema-aligned SQL. The transition from non-executable yet SQL-like outputs to exact, correct queries highlights the value of structured prompt–completion training and parameter-efficient adaptation for text-to-SQL generation tasks.

```
                              EXAMPLE TABLE PREVIEW:

   | Total# | Series# | Title                        | Writer              | Director          |
                             Original air date            |
|--------|---------|----------------------------|-------------------|-----------------|-------
----------------------|
  | 14     | 1       | " Sister Hood "             | Dominic Minghella | Ciaran Donnelly |
                    6October2007, 7.30–8.15pm   |
  | 15     | 2       | " The Booby and the Beast " | Simon Ashford     | Ciaran Donnelly |
                    13October2007, 7.30–8.15pm  |
  | 16     | 3       | " Childhood "               | Jason Sutton      | Ciaran Donnelly |
                    20October2007, 7.15–8.00pm  |
  | 17     | 4       | " The Angel of Death "      | Julian Unthank    | Matthew Evans   |
                    27October2007, 7.15–8.00pm  |
  | 18     | 5       | " Ducking and Diving "      | Debbie Oates      | Matthew Evans   |
                    3November2007, 7.15–8.00pm  |

SCHEMA USED:
- Total# (TEXT)
- Series# (TEXT)
- Title (TEXT)
- Writer (TEXT)
- Director (TEXT)
- Original air date (TEXT)

QUESTION:
What's the title of the episode that Rob Heyland wrote?

-----------------------------------------------------------------------------------------------
--------------------

BASELINE MODEL GENERATED SQL:
SELECT Title FROM Total WHERE Writer = 'Rob Heyland'

TRAINED MODEL GENERATED SQL:
SELECT "Title" FROM data WHERE "Writer" = 'Rob Heyland'

GOLD SQL QUERY:
SELECT "Title" FROM data WHERE "Writer" = 'Rob Heyland'

-----------------------------------------------------------------------------------------------
--------------------

EXECUTION SUMMARY:

| Item               | Value                        |
|--------------------|------------------------------|
| Baseline Result    | ERROR: no such table: Total  |
| Trained LoRa Result| [('" For England…! "',)]     |
| Gold Result        | [('" For England…! "',)]     |

SIMILARITY METRICS:

| Metric       | Baseline Value  | Trained LoRa Value|
|--------------|-----------------|-------------------|
| Exact Match  | 0              | 1                  |
| Jaccard      | 0.5            | 1                  |
| Levenshtein  | 0.873          | 1                  |
| Token F1     | 0.667          | 1                  |
```

Example 3. Output comparison for prompt, baseline SQL, trained SQL, and gold SQL — illustrating the progression from model output to execution and similarity evaluation; demonstrates the benefit of structured prompt–completion training and schema alignment in improving SQL generation and execution accuracy.

7 Discussion

7.1 Achievements

The primary outcome of this experiment is that LoRA-based supervised fine-tuning fundamentally alters how Gemma 3 4B approaches the text-to-SQL task. In its baseline form, the model frequently produces outputs that resemble SQL syntactically but fail to execute due to schema grounding errors or subtle structural mistakes. After fine-tuning, the same model consistently generates well-formed, schema-aligned queries and achieves correct execution on a substantial portion of the evaluation set. This shift demonstrates that lightweight, parameter-efficient adaptation is sufficient to transform a general-purpose language model into a functional text-to-SQL system within the scope of the WikiSQL task.

The improvements observed across execution accuracy and text-based similarity metrics reinforce this conclusion. Exact match, BLEU, ROUGE-L, and token-level F1 all increase dramatically after fine-tuning, indicating that the adapted model learns not only to produce executable SQL but also to match the canonical structure and wording of reference queries closely. Importantly, these gains are achieved without modifying the frozen backbone, highlighting the efficiency of the LoRA approach and validating the design choice to focus adaptation capacity on attention and feed-forward projections.

Equally important is the evaluation pipeline itself. By holding prompts, decoding logic, SQL extraction, and execution environments constant across models, the experiment isolates model behavior as the primary driver of performance differences. The resulting row-level logs and error categorizations provide a clear window into how and why predictions succeed or fail, enabling meaningful interpretation beyond aggregate scores.

7.2 Findings and Interpretation

A key finding is that text-based similarity metrics, while useful, do not fully explain execution behavior. For the baseline model, moderate similarity scores coexist with universal execution failure, confirming that surface resemblance to SQL does not imply semantic correctness. For the LoRA-fine-tuned model, similarity scores are uniformly high, yet execution correctness still varies. This is initially surprising, but it reflects the brittle nature of SQL semantics. Small deviations in operators, column selection, or conditions can preserve most of the query text while changing the result entirely. As a consequence, similarity metrics saturate before execution accuracy does.

This pattern suggests that LoRA fine-tuning is especially effective at teaching canonical SQL form and schema-aware syntax, but less directly optimized for resolving fine-grained semantic decisions. The training objective rewards reproducing the gold query string rather than verifying that the query computes the correct denotation. As a result, the remaining errors tend to be subtle and logical rather than structural, indicating that the model has moved beyond basic formatting failures into a regime where errors are harder to detect using text-based metrics alone.

7.3 Limitations

The characteristics of the WikiSQL dataset constrain the scope of the conclusions. Queries are limited to single tables and relatively simple compositions, which means the results do not establish robustness to joins, nested queries, or more complex relational reasoning. In addition, schemas are provided as serialized text rather than structured objects, encouraging the model to learn correlations between language patterns and column names instead of deeper relational representations.

The reliance on string-level supervision also introduces limitations. Because training optimizes next-token prediction against a single reference query, the model is not explicitly penalized for producing semantically incorrect but syntactically plausible SQL. This explains why high similarity does not always imply correct execution, even after fine-tuning. Finally, practical constraints require evaluating only a subset of the test set, which introduces variance and may underrepresent rare but important failure modes.

7.4 Future Work

The results point to several directions for improvement. The most direct extension is to incorporate execution-aware signals into training or inference, allowing the model to receive feedback based on query results rather than solely on surface form. More structured schema representations and constrained decoding could further reduce grounding errors by limiting the space of valid outputs. Additional analysis that stratifies errors by query type or semantic operation would help isolate the decisions the model still struggles to make.

Taken together, these findings suggest that LoRA fine-tuning is a powerful first step for text-to-SQL adaptation, effectively teaching syntax and schema alignment at low computational cost. Addressing the remaining errors will likely require objectives and constraints that operate at the semantic level rather than relying solely on textual similarity.

8 Conclusion

This project demonstrates that parameter-efficient fine-tuning with LoRA can effectively adapt a general-purpose language model to the text-to-SQL task while maintaining a modest computational footprint. Using Gemma 3 4B as a baseline, the experiments show that supervised fine-tuning on WikiSQL transforms the model from one that produces largely non-executable queries into a system capable of generating schema-aligned and frequently correct SQL. Execution-based evaluation reveals substantial gains that are not fully captured by text similarity metrics alone, highlighting the importance of end-to-end evaluation with a real SQL engine. At the same time, the analysis exposes clear limitations in semantic robustness, where high textual similarity does not always guarantee correct execution. Overall, the results suggest that LoRA fine-tuning provides a strong foundation for practical text-to-SQL systems, and that future improvements will depend on integrating execution-aware objectives and richer schema reasoning rather than increasing model size or training cost.

9 References

- Dettmers et al. (2023) “QLoRA: Efficient Finetuning of Quantized LLMs”
- Hu et al. (2021) “LoRA: Low-Rank Adaptation of Large Language Models”
- Larson (2025) “Lectures 7: Sequence Models and Attention Mechanisms” (DSAN 5800)
- Larson (2025) “Lecture 8: Transformers” (DSAN 5800)
- Google (2024) “Fine-tuning Gemma Models with QLoRA using Hugging Face.” Gemma Documentation. https://ai.google.dev/gemma/docs/core/huggingface_text_finetune_qlora
- OpenAI (2025) ChatGPT (GPT-5.2) — used for summarization and language refinement

---

References

Dettmers, Tim, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. “QLoRA: Efficient Finetuning of Quantized LLMs.” *arXiv Preprint arXiv:2305.14314*.

Hu, Edward, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Wei Chen. 2021. “LoRA: Low-Rank Adaptation of Large Language Models.” *arXiv Preprint arXiv:2106.09685*.

Larson, Chris. 2025a. “Lecture 7: Sequence Models and Attention Mechanisms.” DSAN 5800: Advanced NLP, Georgetown University.

———. 2025b. “Lecture 8: Transformers.” DSAN 5800: Advanced NLP, Georgetown University.