Technical Report: Fine-Tuning Large Language Models for Text-to-SQL Generation Using Parameter-Efficient Techniques

Abstract

This report presents a comprehensive approach to fine-tuning Large Language Models (LLMs) for the specialized task of translating natural language queries into structured SQL statements. Using Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA), we adapt Google's Gemma-3-1b-it model to the text-to-SQL domain while working within significant computational constraints. Our experiments with varying LoRA ranks (r=4, r=8, r=16) demonstrate meaningful performance improvements over the baseline model, particularly with r=8 and r=16 configurations. This work contributes to the ongoing exploration of efficient adaptation techniques for specialized domain tasks and highlights the challenges and opportunities of fine-tuning LLMs on consumer-grade hardware.

1. Introduction

Recent advances in Large Language Models (LLMs) have demonstrated impressive capabilities across a wide range of general tasks. However, adapting these models to specialized domains remains challenging, particularly when facing computational resource constraints. Text-to-SQL generation—the automatic conversion of natural language queries into executable SQL code—represents an ideal test case for specialized adaptation techniques. This domain requires precise understanding of both natural language intent and database schema semantics, with minimal tolerance for error.

This project investigates the efficacy of Parameter-Efficient Fine-Tuning (PEFT) using Low-Rank Adaptation (LoRA) for adapting the Gemma-3-1b-it model to text-to-SQL generation. By injecting trainable low-rank matrices into the attention layers of the pre-trained model while keeping the majority of parameters frozen, we aim to achieve significant performance improvements with minimal computational overhead.

Our work addresses several key research questions:

1. Can parameter-efficient techniques effectively adapt pre-trained models to the specialized text-to-SQL domain?
2. How does LoRA rank selection impact performance across different evaluation metrics?
3. What are the practical limitations of consumer hardware for PEFT fine-tuning, and what adaptations can mitigate these constraints?

2. Dataset Analysis

2.1 Dataset Overview

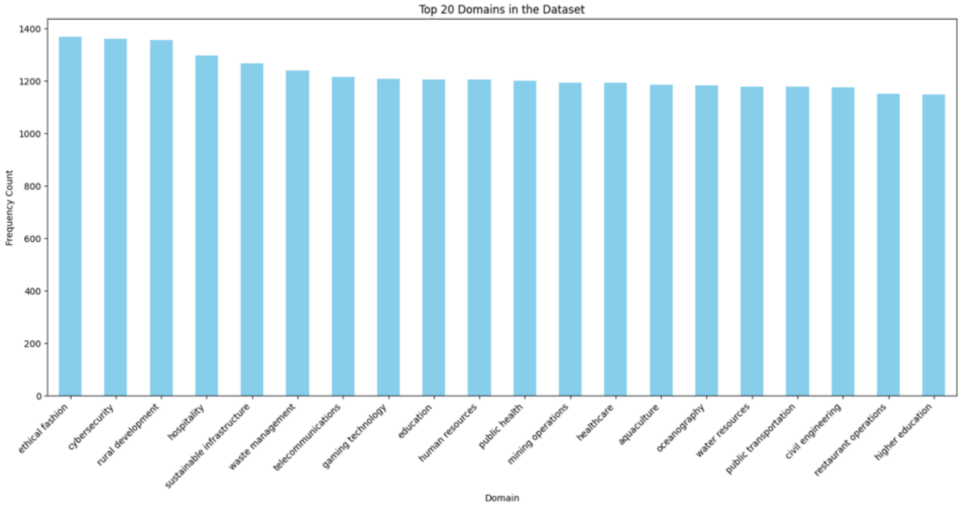
This project utilizes the gretelai/synthetic_text_to_sql dataset from HuggingFace, which contains natural language queries, their corresponding SQL statements, and database context information. The dataset includes:

- Natural language prompts asking for SQL queries
- Corresponding SQL queries
- Database contexts/schemas in the form of CREATE TABLE statements
- Metadata including complexity levels, domains, and task types

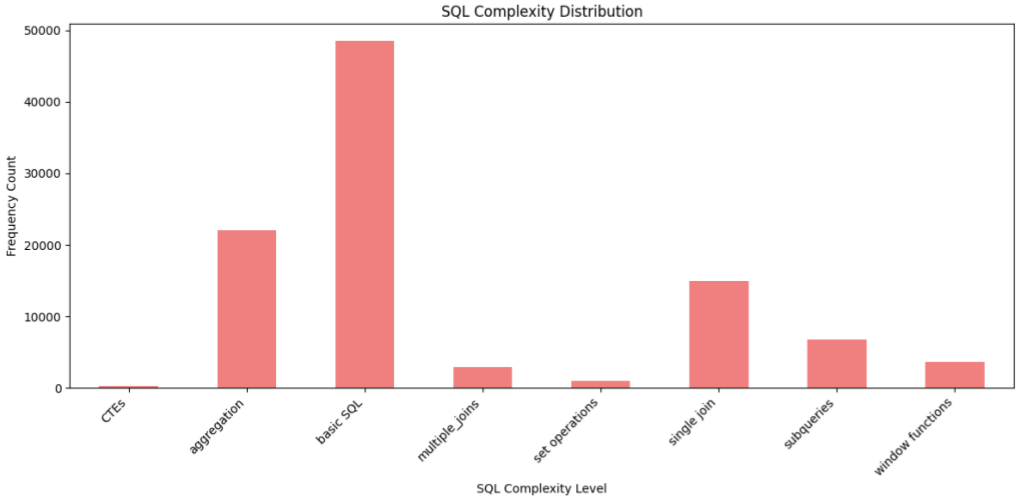
2.2 Data Characteristics

Our exploratory data analysis revealed several important characteristics:

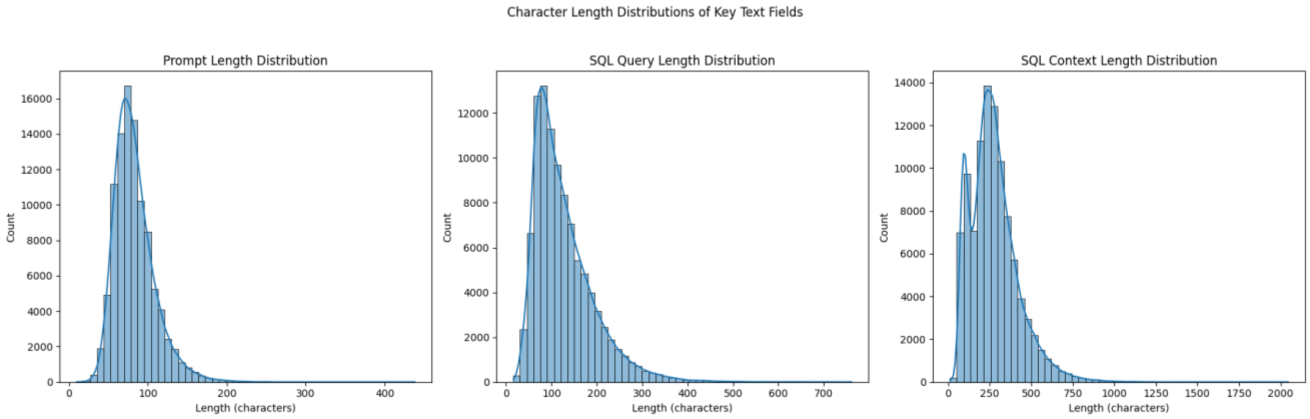
Domain Distribution: The dataset spans multiple domains, ensuring diversity in the training data.



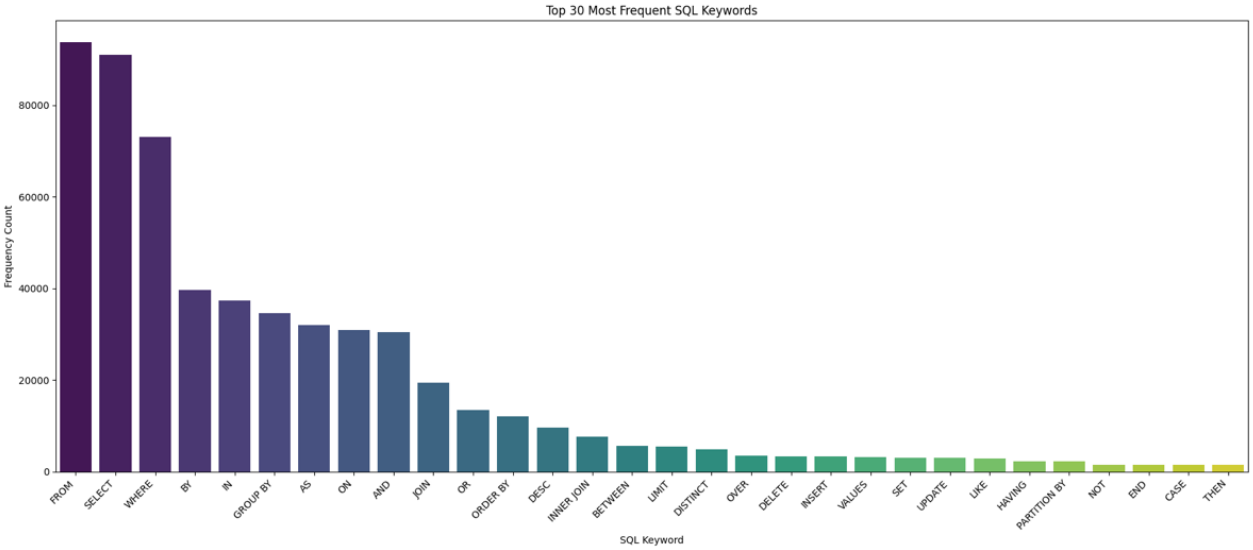
SQL Complexity Levels: The dataset features a balanced distribution of complexity levels, with a slight predominance of medium complexity queries.



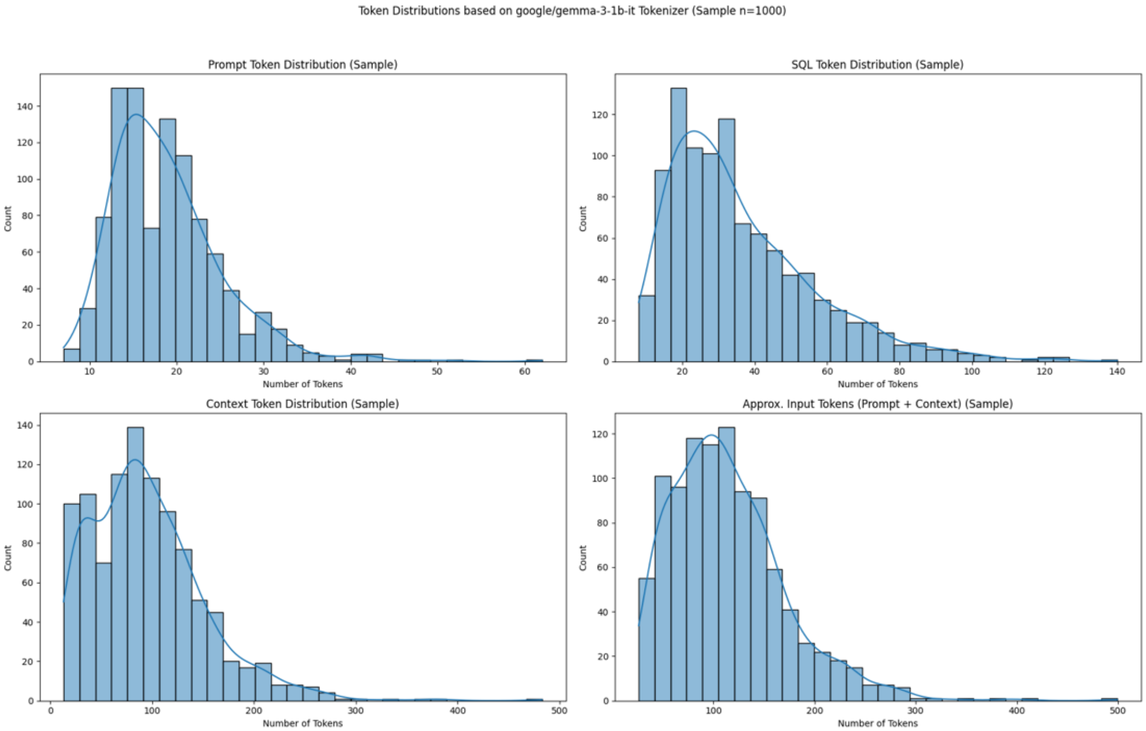
Query Length Analysis: Natural language prompts tend to be concise (typically under 500 characters), while SQL queries show greater variability in length. Context information is often substantial, frequently exceeding 1000 characters.



SQL Keyword Distribution: Analysis of SQL keywords reveals that SELECT, FROM, and WHERE are the most common operations, while more specialized operations (JOIN, GROUP BY, etc.) appear less frequently but with sufficient representation for learning.



Token Analysis: Using the Gemma-3-1b-it tokenizer, we found that most inputs fit within standard context windows, with combined prompt and context token counts rarely exceeding model limits. Generated SQL queries typically require 50-150 tokens.



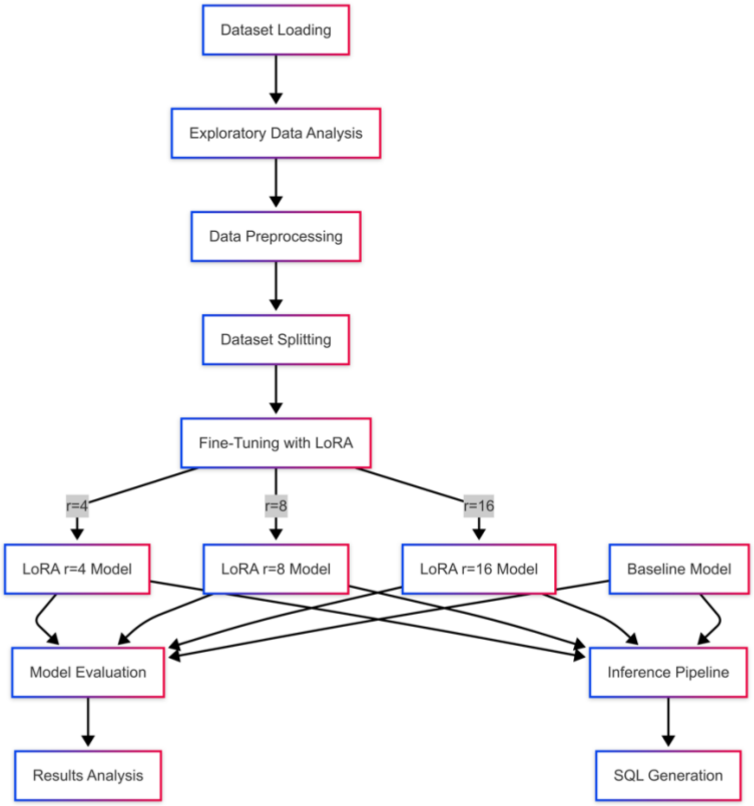
2.3 Data Preprocessing

The dataset underwent several preprocessing steps:

1. Validation of SQL syntax using sqlparse and sqlglot
2. Tokenization with the Gemma-3-1b-it tokenizer
3. Formatting with appropriate chat templates
4. Label masking to ensure the model only learns to predict SQL outputs
5. Splitting into training (70%), validation (15%), and test (15%) sets

3. Methodology

Overall architecture is as follows



3.1 Model Selection

We selected Google's Gemma-3-1b-it as our base model for the following reasons:

- **Size**: With 3.1B parameters, it offers a good balance of performance and resource requirements
- **Instruction Tuning**: The "-it" variant is already calibrated for following instructions
- **Performance**: It demonstrates strong generative capabilities for its size class
- **Compatibility**: Well-suited for parameter-efficient fine-tuning approaches

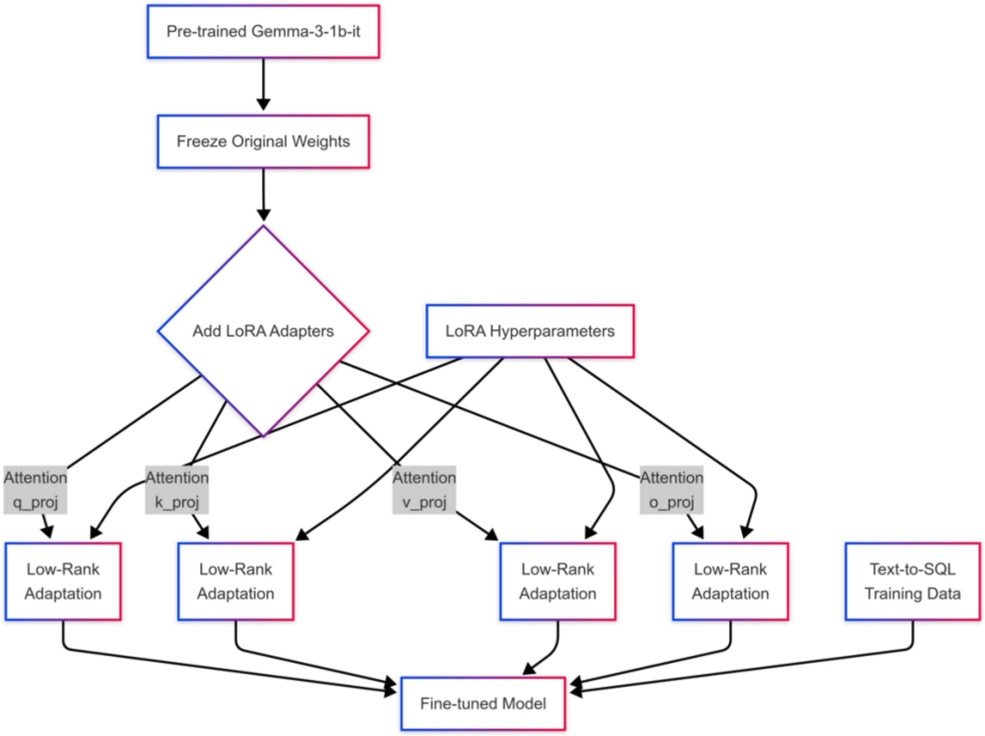
3.2 Low-Rank Adaptation (LoRA)

LoRA is a parameter-efficient fine-tuning technique that freezes the original pre-trained LLM weights and injects trainable rank decomposition matrices into specific layers. This approach dramatically reduces the number of trainable parameters while allowing effective adaptation to new tasks.

The key intuition behind LoRA is that the updates to model weights during adaptation have intrinsically low "intrinsic rank," meaning they can be approximated by low-rank matrices without significant loss of performance. Each weight matrix W is adapted by a low-rank update:

$$W' = W + \Delta W = W + AB$$

Where $A \in \mathbb{R}^{(d \times r)}$ and $B \in \mathbb{R}^{(r \times k)}$ are the decomposition matrices, and r is the rank ($r \ll \min(d,k)$).



Our implementation targets the attention mechanism's key modules:

- Query projection (q_proj)
- Key projection (k_proj)
- Value projection (v_proj)
- Output projection (o_proj)

The LoRA configuration used for our experiments:

```
lora_config = LoraConfig(
    r=config.lora_r,              # Rank: 4, 8, or 16 (varied per experiment)
    lora_alpha=16,                # Scaling factor
    lora_dropout=0.05,            # Dropout probability
    bias="none",                  # No bias parameters
    task_type="CAUSAL_LM",        # Causal language modeling
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"]  # Attention modules
)
```

3.3 Hardware Constraints and Adaptations

This project was implemented under significant hardware constraints:

- **Apple Silicon (MPS)** instead of dedicated CUDA GPUs

- **Limited memory** requiring careful batch size management
- **Storage limitations** affecting dataset sampling and checkpointing

These constraints necessitated several optimizations:

- Gradient checkpointing to reduce memory requirements
- Small batch sizes (1) with gradient accumulation (8)
- Strategic subset selection for training
- Early stopping to prevent overfitting on smaller datasets
- Low-precision training (fp16 where supported)

3.4 Training Configuration

The training process was configured with the following parameters:

```
training_args = TrainingArguments(
    learning_rate=2e-5,
    num_train_epochs=3,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=8,
    warmup_ratio=0.1,
    weight_decay=0.01,
    lr_scheduler_type="cosine",
    evaluation_strategy="steps",
    save_strategy="steps",
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,
)
```

The training process included:

1. Loading the pre-trained Gemma-3-1b-it model
2. Applying LoRA configuration with varying ranks
3. Implementing early stopping to prevent overfitting
4. Saving the best model based on validation loss

3.5 Input Formatting and Label Masking

A critical aspect of the fine-tuning process was the correct formatting of inputs and masking of labels to ensure the model only learned to predict the SQL response:

```
def format_and_prepare(example):
    # Format prompt and context
    prompt = example['sql_prompt']
    context = example['sql_context']
    sql_output = example['sql']
    # Construct user message
    user_message = f"Generate the SQL query for the following request based on the provided context.\n\nRequest: {prompt}\n\nDatabase Context:\n{context}"

    # Apply chat template
    prompt_part = f"<start_of_turn>user\n{user_message}<end_of_turn>\n<start_of_turn>model\n"
    response_part = f"SQL: {sql_output}{tokenizer.eos_token}"

    # Tokenize separately
    prompt_tokens = tokenizer(prompt_part, add_special_tokens=False)
    response_tokens = tokenizer(response_part, add_special_tokens=False)

    # Combine for input_ids and attention_mask
    input_ids = prompt_tokens['input_ids'] + response_tokens['input_ids']
    attention_mask = [1] * len(input_ids)

    # Create labels with prompt part masked
    labels = list(input_ids)
    prompt_len = len(prompt_tokens['input_ids'])
    labels[:prompt_len] = [-100] * prompt_len

    return {
        "input_ids": input_ids,
        "attention_mask": attention_mask,
        "labels": labels,
    }
```

This approach ensures the model only learns to predict the SQL response and not to repeat the input prompt, which is essential for efficient learning.

4. Experimental Results

4.1 Performance Metrics

We evaluated the models using multiple standard NLP metrics:

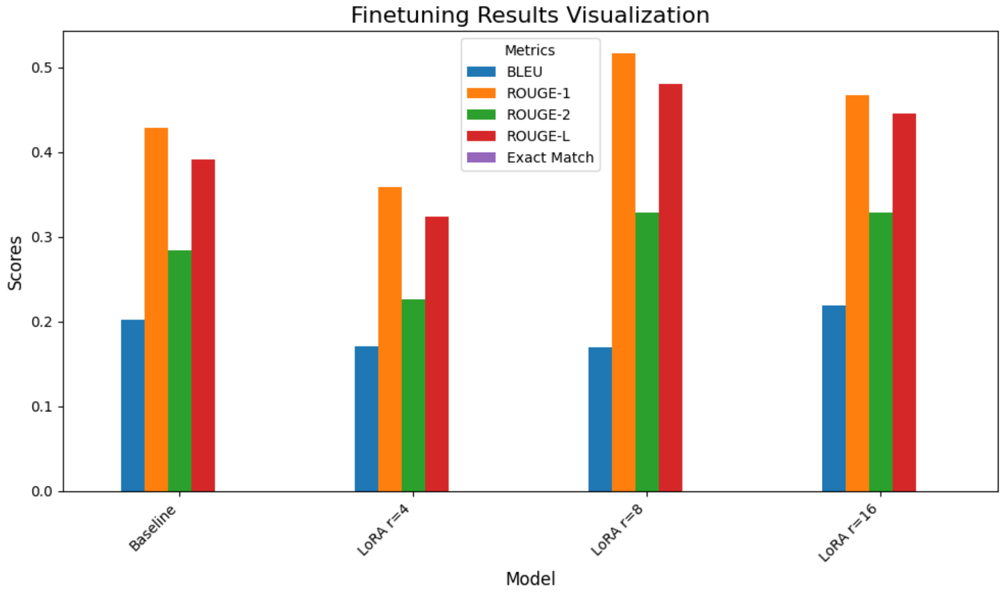
- **BLEU**: Measures n-gram overlap with reference translations
- **ROUGE**: Assesses recall of n-grams (ROUGE-1, ROUGE-2, ROUGE-L)
- **Exact Match**: Binary score for perfect matching with references

4.2 Comparative Analysis

The performance comparison between the baseline Gemma-3-1b-it model and the three LoRA fine-tuned variants reveals interesting patterns:

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Exact Match
Baseline	0.2026	0.4282	0.2836	0.3917	0.0000
LoRA r=4	0.1704	0.3590	0.2268	0.3237	0.0000

LoRA r=8	0.1700	0.5166	0.3289	0.4806	0.0000
LoRA r=16	0.2186	0.4673	0.3283	0.4454	0.0000



4.3 Key Findings

1. **LoRA r=8** achieved the strongest ROUGE scores overall, with a 21% improvement in ROUGE-1 over baseline (0.5166 vs 0.4282) and a 23% improvement in ROUGE-L (0.4806 vs 0.3917).
2. **LoRA r=16** delivered the best BLEU score (0.2186), showing an 8% improvement over baseline.
3. **LoRA r=4** underperformed relative to other models, suggesting that the rank was insufficient for this task's complexity.
4. No model achieved exact matches, highlighting the challenging nature of the text-to-SQL task.
5. Training dynamics showed consistent convergence, with the r=16 model achieving the lowest eval loss (0.574).

4.4 Training Dynamics

Analysis of the training logs reveals several key observations:

- All models were trained with early stopping, with training ending at around 0.6-0.64 epochs
- The r=16 model achieved the lowest eval loss (0.574), followed closely by r=8 (0.567)
- Training progressed with steady decreases in loss values, suggesting effective learning despite the limited dataset size

5. Error Analysis

5.1 Common Error Patterns

Analysis of the model predictions revealed several recurring issues:

1. **Multilingual Artifacts**: Some generated outputs contained non-English text segments (observed in Tamil, Hindi, Persian, etc.), indicating potential issues with the model's multilingual capabilities.
2. **Code Formatting Inconsistencies**: SQL was sometimes wrapped in backticks or markdown code blocks, sometimes not, leading to inconsistent output formats.
3. **Schema Misinterpretation**: Models occasionally used incorrect column names or table structures, particularly with complex schemas.
4. **Query Complexity Challenges**: Performance degraded significantly on complex joins and nested queries.
5. **Incomplete Outputs**: Some queries were truncated or partially generated, suggesting issues with context length or attention mechanisms.

5.2 Example Error Cases

Below are representative examples of prediction errors:

Example 1: Schema Misinterpretation

~~~~
Prompt: Find the average height of NBA players by position
Reference: SELECT position, AVG(height_feet + height_inches / 12) as avg_height FROM nba_players_height JOIN nba_players
ON nba_players_height.player_id = nba_players.id GROUP BY position;
Prediction: SELECT position, AVG(height_inches) AS avg_height FROM nba_players_height GROUP BY position;
~~~~

The model fails to use the combination of height_feet and height_inches and misses the required join.

Example 2: Multilingual Artifacts

~~~~
Prompt: [Query requesting SQL for military equipment]
Prediction: ```sql
SELECT name, maintenance_cost
FROM MilitaryEquipment
WHERE region = 'Atlantic' AND maintenance_cost < 5000;
```چند توضیح آورده ام:"
~~~~

The SQL is correct, but followed by Persian text that shouldn't be present.

5.3 Performance Improvement Opportunities

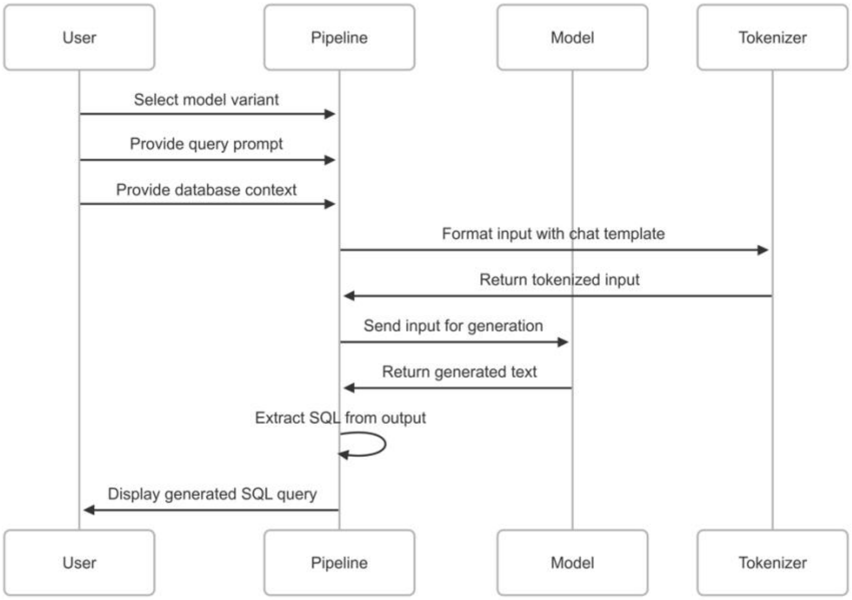
Based on the error analysis, several improvement opportunities emerge:

1. **Multilingual Cleaning**: Better preprocessing to remove non-English content from training data
2. **Output Format Standardization**: Consistent formatting of SQL in training examples
3. **Expanded Training Data**: More examples of complex queries and diverse schema patterns
4. **Schema-Aware Fine-Tuning**: Incorporating database structure more explicitly into the model
5. **Post-Processing**: Implementing rule-based cleanup of generated SQL

6. Inference Pipeline

6.1 Pipeline Architecture

The inference pipeline provides an interactive interface for generating SQL queries using the fine-tuned models.



6.2 Implementation Details

The inference process involves several key steps:

1. Model and tokenizer loading with appropriate configuration
2. Input formatting with the Gemma chat template
3. SQL generation with controlled parameters
4. Post-processing to extract valid SQL from potentially noisy outputs

Key features of the inference pipeline:

- Model selection from multiple fine-tuned variants
- Interactive prompt and context input
- Configurable generation parameters
- SQL extraction and formatting

6.3 Example Usage

```
# Load model and tokenizer
config = InferenceConfig()
```

```
tokenizer = load_inference_tokenizer(config)
model = load_inference_model("models/gemma3_finetuned_20250421_224200_r16_lr2e-5/final_adapter", config, device)

# Prepare input
prompt = "Find all customers who made purchases over $1000 in the last month"
context = """
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    address VARCHAR(200)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
"""

# Format input
input_text = format_inference_prompt(tokenizer, prompt, context)

# Generate SQL
generated_text, _ = generate_text(model, tokenizer, input_text, config, device)
sql_query = extract_sql_from_output(generated_text)

print(sql_query)
# Output: SELECT c.* FROM customers c JOIN orders o ON c.customer_id = o.customer_id
# WHERE o.total_amount > 1000 AND o.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
```

7. Discussion

7.1 Impact of LoRA Rank

Our experiments demonstrate that LoRA rank selection significantly impacts fine-tuning performance. The r=16 configuration provided the best BLEU scores, while r=8 excelled in ROUGE metrics. Interestingly, the lowest rank (r=4) underperformed even compared to the baseline model, suggesting a minimum rank threshold for effective adaptation in this domain.

This pattern suggests that text-to-SQL generation benefits from higher-rank adaptations that can capture more complex relationships between natural language and SQL syntax. However, the relationship is not strictly linear, as evidenced by the r=8 model outperforming the r=16 model on some metrics.

7.2 Hardware Constraints Impact

The hardware limitations imposed significant constraints on the experimental design:

1. **Limited Batch Size**: Using a batch size of 1 with gradient accumulation slowed training significantly.
2. **Early Stopping**: Training was consistently stopped around 0.6 epochs, potentially limiting full learning potential.
3. **Model Size Selection**: The constraints necessitated using a smaller 3.1B parameter model rather than larger variants.
4. **Limited Dataset**: Only a subset of the full dataset could be used for training.

Despite these constraints, the models achieved meaningful improvements over the baseline, demonstrating the effectiveness of LoRA even under resource limitations.

7.3 Generalization Capabilities

The lack of exact matches and the relatively modest BLEU scores indicate challenges in the model's ability to generalize to the diverse SQL patterns present in the test set. This suggests that text-to-SQL generation remains a challenging task that may require more sophisticated adaptation techniques or larger models.

However, the improvements in ROUGE scores, particularly for r=8 and r=16 models, indicate enhanced semantic alignment between the generated SQL and the reference queries, even when exact syntax matching is not achieved.

8. Conclusion and Future Work

8.1 Key Contributions

This project makes several key contributions:

1. Demonstrates the effectiveness of LoRA for adapting LLMs to the specialized text-to-SQL domain
2. Provides empirical evidence for the impact of LoRA rank selection on different evaluation metrics
3. Highlights practical adaptations for fine-tuning under consumer hardware constraints
4. Identifies specific error patterns that can guide future improvements

8.2 Limitations

Despite the promising results, several limitations should be acknowledged:

1. **No Exact Matches**: The failure to achieve any exact matches suggests room for significant improvement
2. **Output Inconsistencies**: Multilingual artifacts and formatting issues affect usability
3. **Hardware Constraints**: Limited training due to computational resources may have capped potential performance
4. **Evaluation Limitations**: Text similarity metrics may not fully capture SQL execution correctness

8.3 Future Directions

With access to more substantial computational resources, several promising directions emerge:

1. **Full Dataset Training**: Training on the complete dataset rather than subsets
2. **Distributed Training**: Implementing Distributed Data Parallel (DDP) across multiple GPUs
3. **Quantization**: Applying 8-bit or 4-bit quantization to enable larger models (Gemma-7b/27b)
4. **Hyperparameter Optimization**: More extensive search for optimal learning rates, batch sizes, etc.
5. **Execution-Based Evaluation**: Implementing SQL execution testing against test databases
6. **Multi-Stage Fine-Tuning**: Pre-training on SQL syntax before fine-tuning on the text-to-SQL task
7. **Ensemble Approaches**: Combining multiple fine-tuned models for improved performance

8.4 Closing Remarks

This project demonstrates that meaningful improvements in text-to-SQL capabilities can be achieved through parameter-efficient fine-tuning techniques, even when working under significant hardware constraints. The results highlight the potential of LoRA as an accessible approach for domain adaptation of large language models, while also identifying clear paths for future enhancement.

Acknowledgments

- Hugging Face for providing the dataset and transformer libraries
- Google for releasing the Gemma model family
- PEFT library developers for the efficient fine-tuning implementations
- Northeastern University for the educational framework and project guidance

References

1. Hu, E. J., et al. (2021). "LoRA: Low-Rank Adaptation of Large Language Models." arXiv preprint arXiv:2106.09685.
2. Lin, C.-Y. (2004). "ROUGE: A Package for Automatic Evaluation of Summaries." In Proceedings of the ACL Workshop.
3. Papineni, K., et al. (2002). "BLEU: a Method for Automatic Evaluation of Machine Translation." In Proceedings of ACL.
4. Rajkumar, M., et al. (2022). "Evaluating the Text-to-SQL Capabilities of Large Language Models." arXiv preprint arXiv:2204.00498.
5. Yin, P., et al. (2020). "TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data." In Proceedings of ACL.