

# CSYE7380

Midterm Project Presentation Deck

By - Kaushik Malikireddy

# Problem

1. Fine-tune two LLMs on Gretel's synthetic text-to-sql dataset.
2. Generate SQL along with an explanation given the schema and a natural language query in the context.
3. Compare the performances of the two fine-tuned models before and after fine-tuning.

# Background

## Challenges with Base LLMs for SQL Generation:

- ❑ General-purpose LLMs lack domain-specific SQL knowledge
- ❑ Often generate syntactically incorrect or inefficient queries
- ❑ May hallucinate incorrect schema elements
- ❑ Don't consistently follow SQL best practices

# Approach & Models

- ❑ Base Models:
  - ❑ Llama 3.2-1B-Instruct (25GB RAM requirement)
  - ❑ Qwen2-0.5B-Instruct (10GB RAM requirement)
- ❑ Fine-tuning Method:
  - ❑ Low-Rank Adaptation (LoRA) for parameter-efficient tuning
  - ❑ Focused on attention mechanism (q\_proj, v\_proj matrices)
- ❑ Dataset: Wildlife conservation SQL examples from `gretelai/synthetic_text_to_sql`
- ❑ GPU: A100 and T4 on Google Collab Pro

# LoRA Implementation Details

python

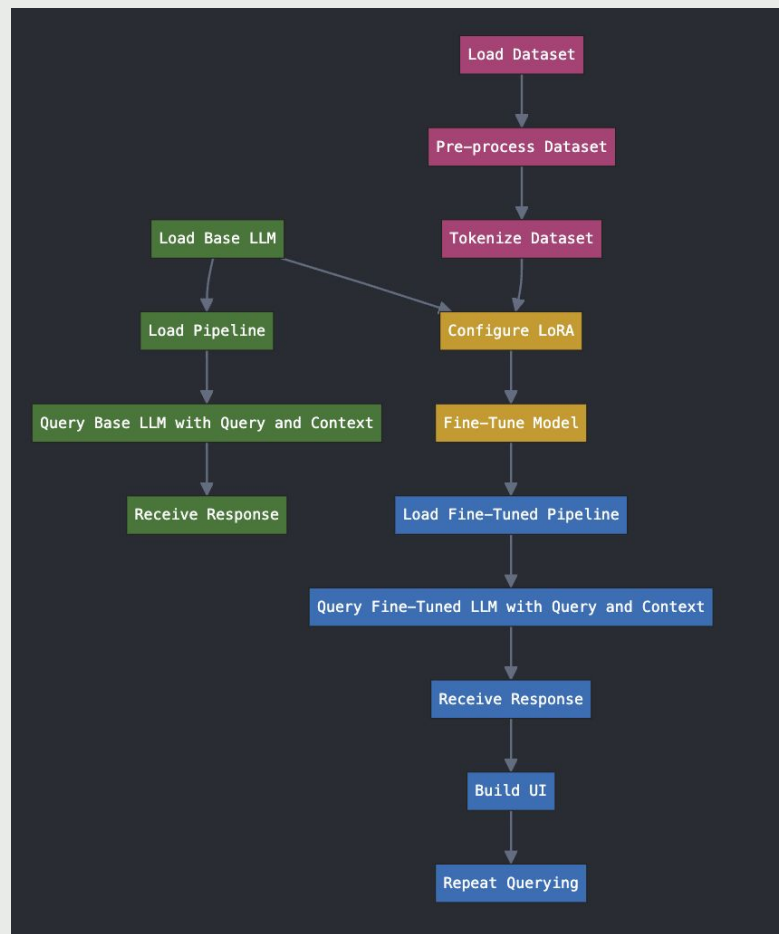
```
lora_config = LoraConfig(  
    r=8,                # Rank of the LoRA matrices  
    lora_alpha=16,       # Scaling factor  
    lora_dropout=0.1,    # Dropout probability  
    target_modules=["q_proj", "v_proj"],  
    bias="none",  
    task_type="CAUSAL_LM"  
)
```

## Training Parameters:

- ❑ 8 epochs
- ❑ Learning rate:  $2e-5$
- ❑ Batch size: 8
- ❑ FP16: True

# Workflow Diagram

- ❑ Load Base LLM
- ❑ Load Pipeline and Query Base LLM with Query and Context
- ❑ Receive Response
- ❑ Load dataset
- ❑ Pre process dataset
- ❑ Tokenzize Dataset
- ❑ Configure Lora
- ❑ Fine Tune Model
- ❑ Load Fine Tuned Pipeline and Queue
- ❑ Fine tuned LLM with Query and context
- ❑ Receive Response
- ❑ Build UI and repeat Querying



# Before Fine-tuning - Llama

Query: What is the total number of animals adopted by each community?

Output:

```
SELECT Community, SUM(AnimalsAdopted) AS TotalAnimalsAdopted
FROM CommunityEducation
WHERE Community NOT IN ('CommunityA', 'CommunityB', 'CommunityC')
GROUP BY Community;
```

Issues:

- ❑ Incorrect WHERE clause (excludes all communities)
- ❑ Misunderstands the basic query intent
- ❑ Query would return empty results

# After Fine-tuning - Llama

Copy

Query: What is the total number of animals adopted by each community?

Output:

```
SELECT Community, SUM(AnimalsAdopted) AS TotalAdopted
FROM CommunityEducation
GROUP BY Community;
```

## Improvements:

- ❑ Correct GROUP BY implementation
- ❑ No unnecessary WHERE clause
- ❑ Appropriate column aliasing
- ❑ Follows SQL best practices



# Key Observations - Llama

## Comparison of Base vs. Fine-tuned Models:

- ❑ **Query Correctness:** Dramatic improvement in SQL syntax and structure
- ❑ **Schema Understanding:** Better utilization of provided schema information
- ❑ **SQL Best Practices:** More consistent use of appropriate GROUP BY clauses
- ❑ **Hallucination Reduction:** Less invention of non-existent constraints

# Before Fine-tuning - Qwen

```
sql Copy  
  
SELECT Community, SUM(AnimalsAdopted) AS TotalAnimalsAdopted  
FROM CommunityEducation  
GROUP BY Community;
```

## Pre-Tuning Observations

- ❑ Concise, syntactically correct SQL query
- ❑ Proper structure with appropriate SELECT, FROM, and GROUP BY clauses
- ❑ Straightforward column naming aligned with database schema

# After Fine-tuning - Qwen

```
SELECT Community,  
       CASE WHEN EXISTS(SELECT * FROM CommunityEducation WHERE Community = Community)  
            THEN Community ELSE NULL END AS No_Animals_Applied,  
       COALESCE(SUM(COALESCE(Abilities_adopted, 0)), 0) AS Total_Animals_Admitted  
FROM CommunityEducation;
```

## Post-Tuning Observations

- ❑ Added verbose explanations before presenting actual SQL code
- ❑ Unnecessarily complex query with nested functions adding no value
- ❑ Missing critical components (GROUP BY) while introducing logical errors

# Key Observations - Qwen

## Major Reasons for Degradation

- ❑ Poor training data quality with complex or incorrect SQL examples
- ❑ Overfitting to explanations at the expense of functional code
- ❑ Loss of fundamental SQL syntax understanding during fine-tuning

# Interactive UI Implementation

THIS SHARE LINK EXPIRES IN 72 HOURS. FOR FREE permanent hosting and 0.00 upgrades, run gradio dep

**Natural Language Query**

Finding the Total Number of Animals Adopted by Each Community

**Database Context**

```
CREATE TABLE CommunityEducation (
  Community VARCHAR(20),
  AnimalsAdopted INT
);

INSERT INTO CommunityEducation VALUES
('CommunityA', 35),
('CommunityB', 28),
('CommunityC', 42);
```

Generate SQL

Translate the natural language query into SQL. Return the SQL code in a separate code block with an explanation.

Context: CREATE TABLE CommunityEducation ( Community VARCHAR(20), AnimalsAdopted INT );

INSERT INTO CommunityEducation VALUES ('CommunityA', 35), ('CommunityB', 28), ('CommunityC', 42);

User Query: Finding the Total Number of Animals Adopted by Each Community, Where community SELECT COUNT (NAME) AS "SELECT ANIMALS\_ADOPTED" FROM communities WHERE community = 'communityName' ;

Return SQL Code

## Gradio for SQL Generation:

- ❑ Simple, intuitive interface built with Gradio
- ❑ Input fields for natural language query and database context
- ❑ Runs directly in Google Colab
- ❑ Customizable styling for improved user experience
- ❑ Real-time SQL generation with explanations

# Images of Testing - Llama

Qwen-F x Llama-F x Gradio x Weights x from tra x Deployin x Meet Ka x LLM fine x Untitled x +

ec30d593c093e13b15.gradio.live

Jobs IT\_App Northeastern Internships Imp Docs Learn more YouTube Outlook NU Yahoo Mail Gmail All Bookmarks

## SQL Query Generator

Enter your natural language query and database context to generate SQL.

Natural Language Query

Finding the Average Hours Worked by Volunteers at Each Wildlife Rescue Center

Database Context

```
CREATE TABLE Volunteers (  
  VolunteerID INT PRIMARY KEY,  
  CenterID INT,  
  VolunteerName VARCHAR(50),  
  HoursWorked INT,  
  FOREIGN KEY (CenterID) REFERENCES WildlifeRescueCenter(CenterID)  
);  
  
INSERT INTO Volunteers VALUES  
(201, 1, 'John Doe', 120),  
(202, 1, 'Jane Smith', 150),  
(203, 2, 'Alice Johnson', 90),  
(204, 3, 'Bob Brown', 200);
```

Generate SQL

Translate the natural language query into SQL. Return the SQL code in a separate code block with an explanation.

Context: CREATE TABLE Volunteers ( VolunteerID INT PRIMARY KEY, CenterID INT, VolunteerName VARCHAR(50), HoursWorked INT, FOREIGN KEY (CenterID) REFERENCES WildlifeRescueCenter(CenterID) );  
  
INSERT INTO Volunteers VALUES (201, 1, 'John Doe', 120), (202, 1, 'Jane Smith', 150), (203, 2, 'Alice Johnson', 90), (204, 3, 'Bob Brown', 200);

User Query: Finding the Average Hours Worked by Volunteers at Each Wildlife Rescue Center

```
SELECT  
  C.CenterName AS CenterName,  
  AVG(V.VolunteerHours) AS AverageHoursWorkedPerVolunteer  
FROM  
  Volunteers V  
JOIN  
  WildlifeRescueCenter C ON V.CenterID = C.CenterID  
GROUP BY  
  C.CenterName;
```

Explanation:  
  
The original user query is finding the average hours worked per volunteer for each wildlife rescue center. The AVG function calculates the mean of values in column VolunteerHours, which represents the total number of working hours performed by all volunteers under their care. To achieve this result using Natural Language Processing and SQL, we can create a new table to store information about each

Kaushik Malikireddy

# Images of Testing - Llama

Generate SQL

The original user query is finding the average hours worked per volunteer for each wildlife rescue center. The `AVG` function calculates the mean of values in column `VolunteerHours`, which represents the total number of working hours performed by all volunteers under their care. To achieve this result using Natural Language Processing and SQL, we can create a new table to store information about each Wildlife Rescue Center, including its name and any relevant data such as the population size or other metrics that might be useful for calculating averages. Let's assume we have additional columns like "PopulationSize" and "AverageHours", where these are calculated based on historical records.

Here is how you could structure your database schema:

```
-- Create table for centers

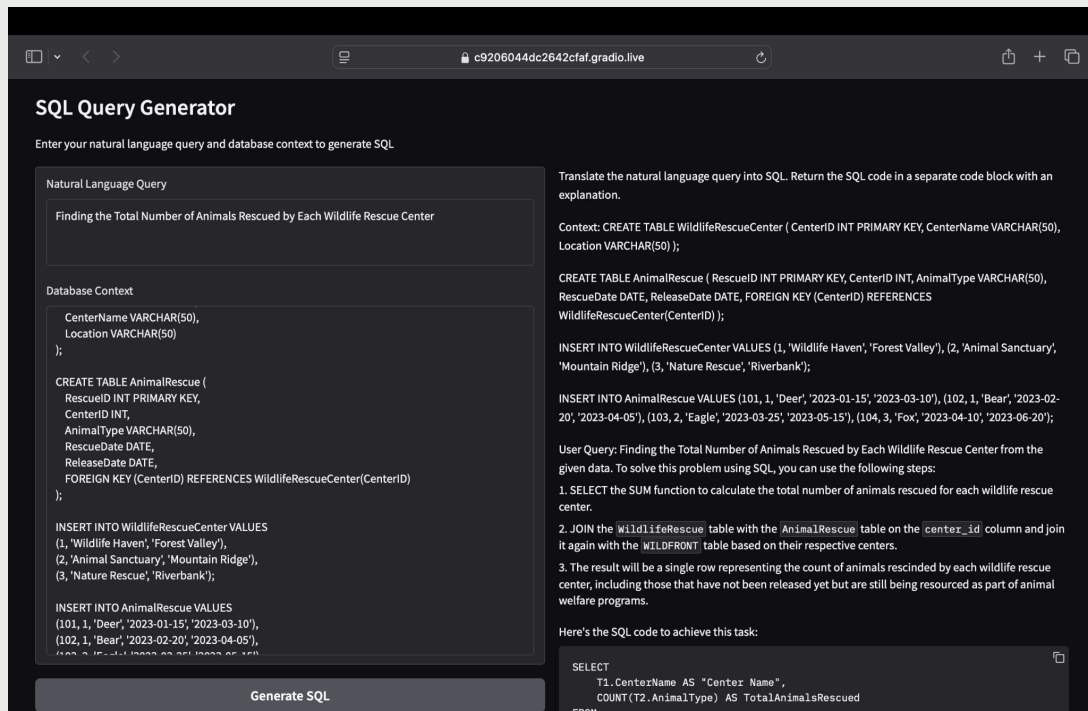
CREATE TABLE Centers (
  CenterID INT PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  PopulationSize DECIMAL(10,2), -- Assuming decimal precision here!
  AverageHours DECIMAL(5,2)-- Calculated from previous step!
);
```

Next let us write our SQL statement:

```
SELECT
  c.Name AS CenterName,
  SUM(v.AverageHours) / COUNT(DISTINCT v.CenterID) AS AverageHoursWorkedPerVolu
FROM
  Volunteers v
INNER JOIN
  Centers c ON v.CenterID = c.CenterID;
```

This will return the desired results!

# Images of Testing - Qwen





# Images of Testing - Qwen

CenterID INT,  
AnimalType VARCHAR(50),  
RescueDate DATE,  
ReleaseDate DATE,  
FOREIGN KEY (CenterID) REFERENCES WildlifeRescueCenter(CenterID)  
);  
  
INSERT INTO WildlifeRescueCenter VALUES  
(1, 'Wildlife Haven', 'Forest Valley'),  
(2, 'Animal Sanctuary', 'Mountain Ridge'),  
(3, 'Nature Rescue', 'Riverbank');  
  
INSERT INTO AnimalRescue VALUES  
(101, 1, 'Deer', '2023-01-15', '2023-03-10'),  
(102, 1, 'Bear', '2023-02-20', '2023-04-05'),  
(103, 2, 'Eagle', '2023-03-25', '2023-05-15'),  
(104, 3, 'Fox', '2023-04-10', '2023-06-20');

Generate SQL

User Query: Finding the Total Number of Animals Rescued by Each Wildlife Rescue Center from the given data. To solve this problem using SQL, you can use the following steps:

1. SELECT the SUM function to calculate the total number of animals rescued for each wildlife rescue center.
2. JOIN the `WildlifeRescue` table with the `AnimalRescue` table on the `center_id` column and join it again with the `WILDFRONT` table based on their respective centers.
3. The result will be a single row representing the count of animals rescinded by each wildlife rescue center, including those that have not been released yet but are still being resourced as part of animal welfare programs.

Here's the SQL code to achieve this task:

```
SELECT  
  T1.CenterName AS "Center Name",  
  COUNT(T2.AnimalType) AS TotalAnimalsRescued  
FROM  
  WildlifeRescueCenter T1  
JOIN  
  AnimalRescue T2 ON T1.CentreID = T2.CenterID  
LEFT JOIN  
  WildFront F ON T1.CenterID = F.WildFrontId  
GROUP BY  
  T1.CenterName;
```

Explanation:

The above SQL statement selects the name of the wildlife center and counts the total number of animals saved at each location. It uses LEFT JOIN between the two tables (`WildFront`) to link animals rescued before they were removed or released back into nature after their assigned care periods. Then, GROUP BY T1.CenterName is used to sort the results according to the total number of animals saved per wildlife center. This way, we get the sum of all animals saved at each center along with its corresponding center name.

Use via API • Built with Gradio • Settings

# Thank you!

Why is Fine  
Tuning  
Important?

