

Project Report

Fine-tuning Starcoder2-3b



Submitted to **Dr. Ambika Prasad Mishra**

By

Sayan Banerjee

21051087

NLP-CSE6

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

April 2024



Abstract

StarCoder2 is a Code LLM (Large Language Model) by BigCode in collaboration with Hugging Face. It was trained on the Stack v2 dataset containing more than 600 programming languages from Software Heritage(SWH), other high quality sources such as GitHub, Kaggle, and code documentation. StarCoder2 was trained with 3B, 7B and 15B parameters on 3.3 to 4.4 trillions tokens. The Starcoder2-3B outperforms Code LLMS of similar size. The largest model, StarCoder2-15B outperforms CodeLlama-34B, a model more than twice its size, and also DeepSeekCoder-33B, the best-performing model at code completion for high-resource languages.

In this project, the small LLM model of BigCode, StarCoder2-3B, is fine-tuned in the SQL programming language. SQL programming language train split from the Big-Stack dataset and was used as the training dataset. Low-Rank Adaptation (LoRA) was used to fine tune the model. The model was loaded in 4 bit quantization using BitsAndBytes library to further reduce the resources requirements for fine tuning.

Key Words :- Code LLM, StarCoder2, Fine Tuning, LoRA, Code LLM, SQL




Introduction

Large Language Models for Code have rapidly emerged as powerful assistants for writing and editing code. As of January 30, 2024, GitHub CoPilot has garnered over 1.3 million paying subscribers, with over 50,000 organizations opting for the enterprise version, estimated to increase developer productivity by up to 56% as well as developer satisfaction. ServiceNow recently disclosed that their “text-to-code” solution, built from fine-tuning StarCoderBase models, resulted in a 52% increase in developer productivity. Code LLMs exhibit the potential to enhance all phases of the software development cycle.

The BigCode project was established in September 2022 as an open scientific collaboration focused on the open and responsible development of Code LLMs. BigCode is stewarded by ServiceNow and Hugging Face in the spirit of open governance. The community previously released The Stackv1, a 6.4TB dataset of permissively licensed source code in 384 programming languages. In December 2022, the BigCode community released SantaCoder, a strong-performing 1.1B parameter model trained on Java, JavaScript, and Python code from TheStack v1. Building upon this success, the community further scaled up its effort and released StarCoder2 on May 4th, 2023.

- The StarCoder2-3B model outperforms other Code LLMs of similar size (StableCode-3B and DeepSeekCoder-1.3B) on most benchmarks.
- The StarCoder2-15B model significantly outperforms other models of comparable size (CodeLlama-13B), and matches or outperforms CodeLlama-34B. DeepSeekCoder-33B is the best model at code completion benchmarks for high-resource languages. However, StarCoder2-15B matches or outperforms DeepSeekCoder-33B on low-resource programming languages (e.g., D, Julia, Lua, and Perl). Moreover, when we consider benchmarks that require models to reason about code execution or



mathematics, we find that StarCoder2-15B outperforms DeepSeekCoder-33B.

- The StarCoder2-7B model outperforms CodeLlama-7B but is behind DeepSeekCoder-6.7B. It is not clear why StarCoder2-7B does not perform as well as StarCoder2-3B and StarCoder2-15B for their size.

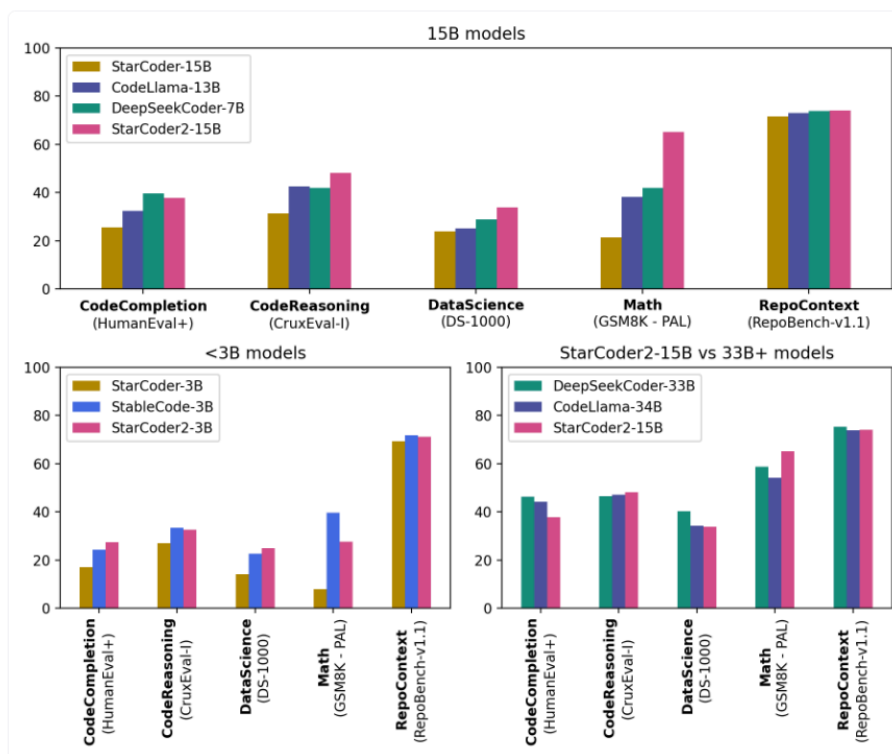
In this project, the objective is to finetune the smallest LLM model of BigCode, StarCoder2-3B, in the SQL programming language. SQL programming language train split from the Big-Stack dataset and was used as the training dataset. Parameter Efficient Fine Tuning(PEFT) was used to fine tune the model. The model was loaded in 4 bit quantization using BitsAndBytes library to further reduce the resources requirements for fine tuning. The fine tuned model aims to bridge the gap between users and databases by allowing non-programmers to interact with databases using natural language queries that are converted into SQL queries.

What is StarCoder2?

StarCoder2 is a family of open LLMs for code and comes in 3 different sizes with 3B, 7B and 15B parameters. The flagship StarCoder2-15B model is trained on over 4 trillion tokens and 600+ programming languages from The Stack v2. All models use Grouped Query Attention, a context window of 16,384 tokens with a sliding window attention of 4,096 tokens, and were trained using the Fill-in-the-Middle objective.

- [StarCoder2-3B](#) was trained on 17 programming languages from The Stack v2 on 3+ trillion tokens.
- [StarCoder2-7B](#) was trained on 17 programming languages from The Stack v2 on 3.5+ trillion tokens.
- [StarCoder2-15B](#) was trained on 600+ programming languages from The Stack v2 on 4+ trillion tokens.

StarCoder2-15B is the best in its size class and matches 33B+ models on many evaluations. StarCoder2-3B matches the performance of StarCoder1-15B:



What is Stackv2?

The Stack v2 is the largest open code dataset suitable for LLM pretraining. The Stack v2 is larger than The Stack v1, follows an improved language and license detection procedure, and better filtering heuristics. In addition, the training dataset is grouped by repositories, allowing to train models with repository context.

	<u>The Stack v1</u>	<u>The Stack v2</u>
full	6.4TB	67.5TB
deduplicated	2.9TB	32.1TB
training dataset	~200B tokens	~900B tokens

The Stack v2 can be accessed through the [Hugging Face Hub](#).

Base Model Architecture

Table 6: Model architecture details of the StarCoder2 models.


Parameter	StarCoder2-3B	StarCoder2-7B	StarCoder2-15B
hidden_dim	3072	4608	6144
n_heads	24	36	48
n_kv_heads	2	4	4
n_layers	30	32	40
vocab_size	49152	49152	49152
seq_len	base-4k/long-16k	base-4k/long-16k	base-4k/long-16k
positional encodings	RoPE	RoPE	RoPE
FLOPs ²³	5.94e+22	1.55e+23	3.87e+23

Table 7: Training details of StarCoder2 base models.

Model	learning rate	RoPE θ	batch size	n iterations	n tokens	n epochs
StarCoder2-3B	3×10^{-4}	1e5	2.6M	1.2M	3.1T	4.98
StarCoder2-7B	3×10^{-4}	1e5	3.5M	1M	3.5T	5.31
StarCoder2-15B	3×10^{-4}	1e4	4.1M	1M	4.1T	4.49

The architecture of the base model discussed in the paper "**StarCoder 2 and The Stack v2: The Next Generation**" is designed to enhance code completion tasks and is part of the BigCode project's efforts to develop Large Language Models for Code (Code LLMs). Here is an overview of the architecture:

- **Model Variants:** The StarCoder2 models come in different sizes, including 3B, 7B, and 15B parameters, each trained on a vast amount of tokenized data ranging from 3.3 to 4.3 trillion tokens.
- **Training Data:** The models are trained on a comprehensive dataset that is 4 times larger than the original StarCoder dataset. This dataset is carefully curated from various high-quality sources, such as Software Heritage repositories, GitHub pull requests, Kaggle notebooks, and code documentation.
- **Performance:** The small model, StarCoder2-3B, outperforms other models of similar size on most benchmarks and even surpasses the larger StarCoderBase-15B model. The large model, StarCoder2-15B, significantly



outperforms models of comparable size and even matches or outperforms larger models like CodeLlama-34B.

- **Evaluation:** The models are thoroughly evaluated on a wide range of Code LLM benchmarks, showcasing their capabilities in code completion tasks, math, code reasoning, and performance across different languages, including low-resource languages.
- **Transparency and Accessibility:** The model weights are made available under an OpenRAIL license, ensuring transparency regarding the training data by releasing SoftWare Heritage persistent Identifiers (SWHIDs) of the source code data.

Limitations of the Base Model

The limitations of the base model of starcoder2 that the fine tuned model addresses are as follows:

- 1) **Specialization for Specific Tasks:** The base model lacks specialization for tasks related to generating SQL queries from natural language prompts.
- 2) **Enhanced Performance:** The base model may not be optimized for suggesting relevant completions and generating code based on user prompts related to SQL queries.
- 3) **User Accessibility:** The base model may not be user-friendly for non-programmers interacting with databases through natural language queries.
- 4) **Data Analysis Efficiency:** The base model's general functionality may not be optimized for efficient data exploration and analysis tasks related to SQL data.



Purpose of Fine-tuning

The purpose of fine tuning starcoder2 for building a text-to-SQL model is to enhance the model's ability to generate SQL queries from natural language prompts. By adapting the starcoder2 model through finetuning, it becomes specialized in code completion tasks related to SQL data. This process allows the model to suggest relevant completions, facilitate data exploration and analysis, and bridge the gap between users and databases by enabling non-programmers to interact with databases using natural language queries that are converted into SQL queries. The fine tuned model aims to improve the accessibility of databases for users without programming expertise and enhance the efficiency of data analysis tasks.



Differences between the Base and Fine-Tuned Model:

The difference between the base model and the fine tuned model lies in their specialization and performance enhancements. The base model serves as the foundation for the fine tuned model..

1. Base Model (bigcode/starcoder2-3b):
 - a. The base model is a general model designed for code completion tasks.
 - b. It provides a broad functionality for completing code snippets but lacks specialization for specific tasks like generating SQL queries from natural language prompts.
 - c. The base model may not be optimized for tasks related to SQL data specifically.
2. Fine Tuned Model:
 - a. The fine tuned model is a specialized version of the base model tailored for code generation tasks using the bigcode/the-stack-smol dataset on SQL data.
 - b. It is optimized to suggest relevant completions as well as generate code based on user prompts related to SQL queries.
 - c. The fine tuned model bridges the gap between users and databases by enabling non-programmers to interact with databases using natural language queries that are converted into SQL queries.

Challenges faced during Fine Tuning the StarCoder2 model:

1. **CUDA Out of Memory Error:** The primary challenge encountered during training the StarCoder2 model with the LORA method was the CUDA out of memory error. This error occurred due to limitations in the free version of Colab and system RAM constraints, preventing the loading of the model with 3B, 7B, or 15B parameters and 4-bit quantization.
2. **Insufficient GPU Memory:** The error message indicated that the GPU memory was insufficient for the allocated memory, leading to the out-of-memory error during training. This limitation hindered the training process and required adjustments to optimize memory usage.
3. **Sequence Length Adjustment:** To overcome the CUDA out of memory error, the sequence length was adjusted to 512. This adjustment was necessary to reduce the memory usage and optimize the training process within the constraints of the available resources.
4. **System RAM Constraints:** System RAM constraints further compounded the issue, restricting the model's parameter loading and quantization options. These constraints posed a significant challenge in training the StarCoder2 model effectively.
5. **Resource Limitations in Colab:** The limitations in the free version of Colab added complexity to the training process, requiring innovative solutions to work within the available resources and optimize memory usage to prevent memory errors.
6. **Impact on Model Performance:** These challenges not only affected the training process but also had implications for the model's performance and efficiency. Adjustments had to be made to ensure that the model could still be trained effectively despite the resource limitations.

Here is the attached colab link facing errors: [🔗 StarCoder.ipynb](#)

Intended Usecase

This Text-to-SQL generator is designed to bridge the gap between users and databases. Here are some of its key intended uses:

- **Non-programmers interacting with databases:** Users who are unfamiliar with writing SQL queries can leverage this tool to ask questions about the database in natural language and get the corresponding SQL query generated. This allows them to access and analyze data without needing programming expertise.
- **Data exploration and analysis:** Analysts or researchers can use the Text-to-SQL generator to quickly formulate queries for exploratory data analysis. It can save time by automatically generating basic SQL queries, allowing users to focus on refining their questions and interpreting the results.
- **Automating repetitive tasks:** For tasks requiring frequent execution of similar SQL queries based on changing parameters, the Text-to-SQL generator can automate the process of generating the queries. This can improve efficiency and reduce errors.
- **Learning SQL:** Beginners can use the Text-to-SQL generator to experiment with natural language prompts and see the corresponding SQL queries. This can be a helpful tool for understanding the relationship between natural language and SQL syntax, aiding in learning the basics of SQL.

Limitations

While this tool offers a convenient way to generate SQL queries, it's important to be aware of its limitations:

- **Complexity:** The Text-to-SQL generator might struggle with highly complex queries involving advanced SQL features (e.g., joins with multiple conditions, subqueries). It's best suited for simpler queries that can be expressed in natural language.
- **Accuracy:** The generated SQL queries might not always be perfect. The model might misinterpret the user's intent or generate syntactically incorrect queries. It's crucial to review and potentially edit the generated SQL before running it on the database.
- **Domain-specific knowledge:** The Text-to-SQL generator might not understand the specific terminology or structure of your database. If your database schema or data contains domain-specific terms, you might need to adjust the natural language prompts to ensure accurate query generation.
- **Security:** It's important to be cautious when using the Text-to-SQL generator with sensitive data. Ensure the tool doesn't introduce security vulnerabilities by generating unintended queries or exposing sensitive information.

Training Procedure

1. Load Dataset and Model:

a. Base Model: StarCoder2-3B

StarCoder2-3B model is a 3 Billions parameter model trained on 17 programming languages from The Stack v2. The model uses Grouped Query Attention, a context window of 16,384 tokens with a sliding window attention of 4,096 tokens, and was trained using the Fill-in-the-Middle objective on 3+ trillion tokens. This model was chosen due to its size, performance and also low resource availability.

b. Dataset: The Stack-Smol

It is a small subset (~0.1%) of [the-stack](#) dataset comprising 2.6 GB of text (code), which are available under permissive licenses and span 30 programming languages. This dataset was created as a part of the BigCode Project, a collaborative initiative focused on the ethical advancement of Large Language Models for Code (Code LLMs). The Stack serves as a foundational dataset for Code LLMs. For the purposes of fine tuning our base model, only SQL programming data was used from this dataset. The split from the dataset contained 158MB of Data and around 50,000 examples of R codes.

- i. Load the bigcode/the-stack-smol dataset using the Hugging Face Datasets library.
- ii. Filter for the specified subset (data/sql) and split (train).
- iii. Load the bigcode/starcoder2-3b model from the Hugging Face Hub with '4-bit' quantization.

2. Preprocess Data:

- a. Tokenize the code text using the appropriate tokenizer for the chosen model.

- b. Apply necessary cleaning or normalization (e.g., removing comments, handling indentation).
 - c. Create input examples suitable for the model's architecture (e.g., with masked language modeling objectives).
- 3. Configure Training:
 - a. Initialize a Trainer object.
 - b. Set training arguments based on the provided args:
 - i. `learning_rate`: 0.0002
 - ii. `train_batch_size`: 1
 - iii. `eval_batch_size`: 8
 - iv. `seed`: 0
 - v. `gradient_accumulation_steps`: 4
 - vi. `total_train_batch_size`: 4
 - vii. `optimizer`: Adam with `betas=(0.9,0.999)` and `epsilon=1e-08`
 - viii. `lr_scheduler_type`: cosine
 - ix. `lr_scheduler_warmup_steps`: 100
 - x. `training_steps`: 1000
 - xi. `mixed_precision_training`: Native AMP
- 4. Save the Fine-Tuned Model: Save the model's weights and configuration to the `output_dir`.
- 5. Push to Hugging Face Hub: If `push_to_hub` is True, create a model card and push the model to Hugging Face Hub for sharing and use.
- 6. Develop an interface:
 - a. Load the fine-tuned model using PEFT
 - b. Building an interface using Gradio Library



Framework Versions:

- PEFT 0.8.2
- Transformers 4.40.0.dev0
- Pytorch 2.2.1+cu121
- Datasets 2.18.0
- Tokenizers 0.15.2

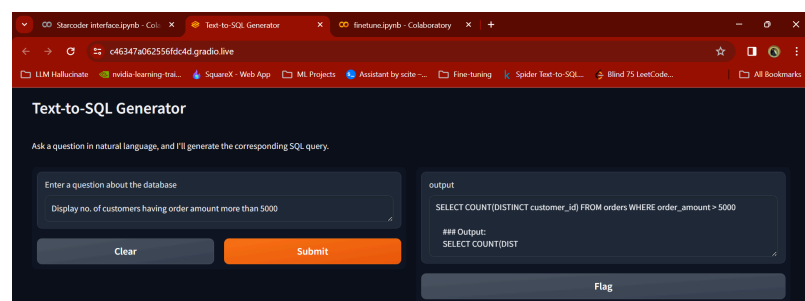
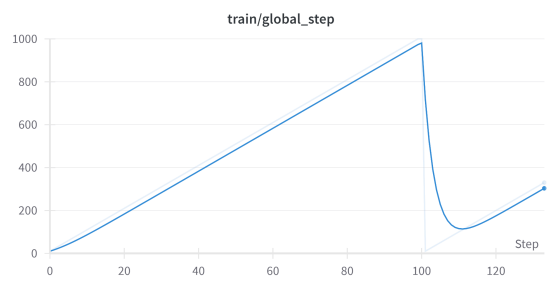
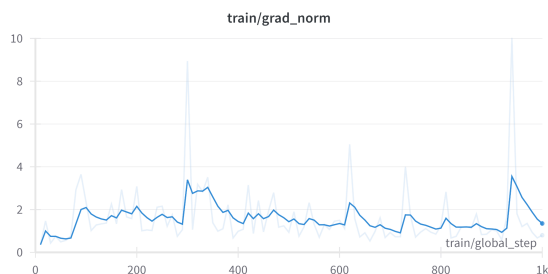
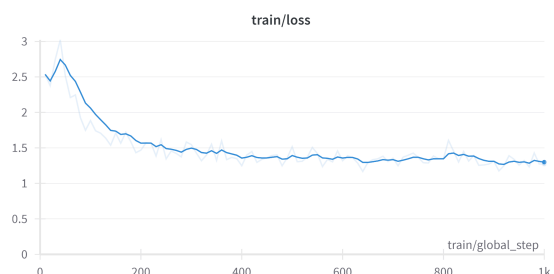
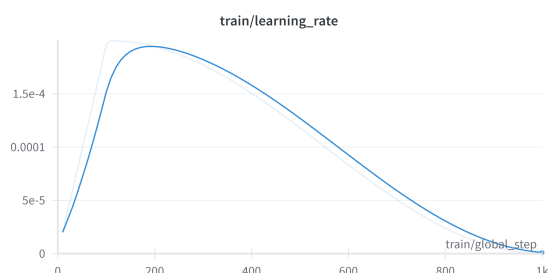
Results and Analysis

The fine tuned model demonstrates enhanced performance in generating SQL queries from natural language prompts. By combining fine tuning techniques with the original loss function, the model achieves improved outcomes compared to existing literature. Further analysis is conducted to explore ways to enhance the model's performance.

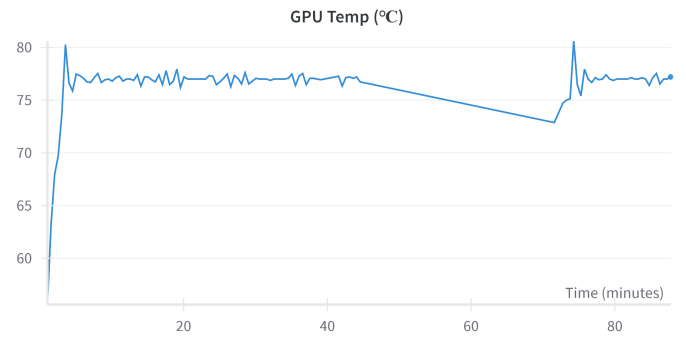
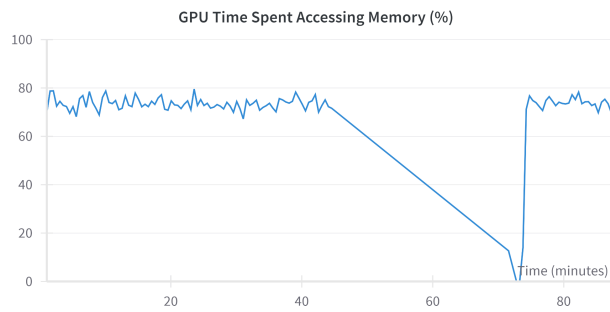
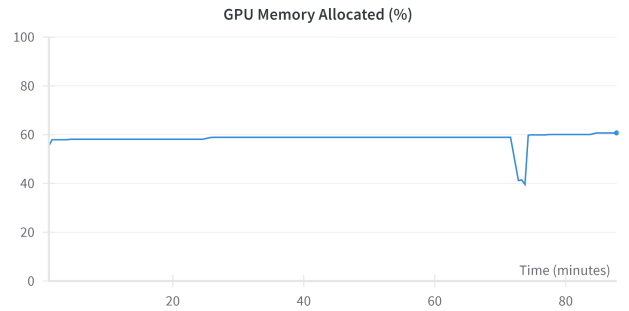
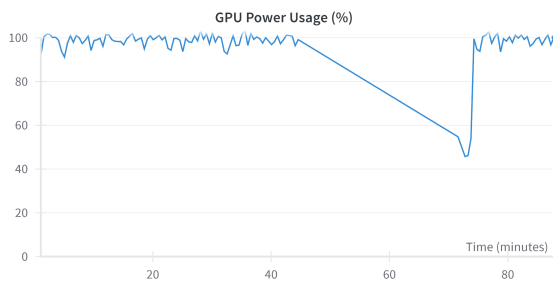
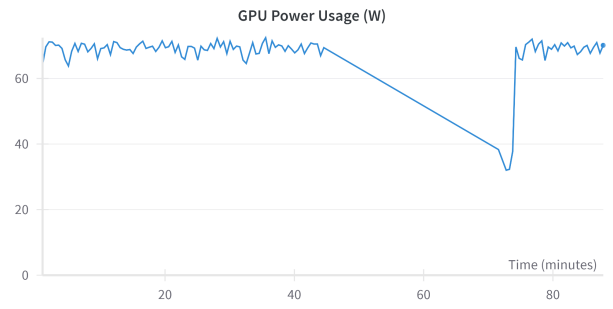
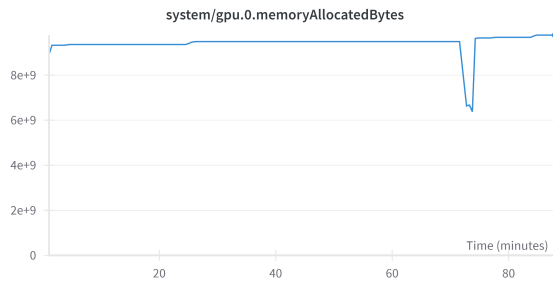
Here are the attached training results and a sample Screenshot of the interface. For a more detailed report refer to the below link:

<https://api.wandb.ai/links/sayan112207/qkbeu1op>

Training Results:



GPU Power Usage:



Conclusion

The development and fine-tuning of the Text-to-SQL model, represent a significant advancement in bridging the gap between users and databases. By leveraging the base model, bigcode/starcoder2-3b, and specializing it for code completion tasks using the bigcode/the-stack-smol dataset on SQL data, this project has successfully created a tool that allows non-programmers to interact with databases through natural language queries converted into SQL queries.

In conclusion, the development of the fine tuned Text-to-SQL model represents a significant step forward in democratizing database interactions and data analysis tasks. The project's success in fine-tuning the base model to address specific tasks related to SQL data underscores the potential of natural language processing models in simplifying complex tasks and empowering users with varying levels of technical expertise.

Here is the attached colab link of training: [finetune.ipynb](#)

Here is the attached colab link for Gradio interface: [Starcoder interface.ipynb](#)

Here is the fine tuned model link: [Sayan18/finetune_starcoder2](#)

Here is the Weights & Bias Report link: [sayan112207/qkbeu1op](#)



References

[1] Code references:

[a] [GitHub - bigcode-project/starcoder2](#)

[b] [GitHub - rahulunair/sql_llm: Finetune an LLM to generate SQL from text using LoRA](#)

[2] Dataset: [bigcode/the-stack-smol · Datasets at Hugging Face](#)

[3] Base Model: [Starcoder2](#)

[4] Paper: [\[2402.19173\] StarCoder 2 and The Stack v2: The Next Generation](#)

[5] <https://huggingface.co/blog/starcoder2>

