Advancing the Landscape of NL2SQL: A Comprehensive Evaluation of Fine-tuned Phi-2 and DeFog Models Approaches

DISSERTATION

Submitted in partial fulfilment of the requirements of the
MTech Data Science and Engineering Degree programme

By

B Venkata Narahara Sesha Sai Pavan Kumar
2021SC04115

Under the supervision of

Ganesh Belde, Manager - Data Science
Amnet Digital

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA

January 2024

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

Introduction

NL2SQL has been a crucial advancement in the NLP space ever since. The ability of a model to write code which is generated from a natural language input is groundbreaking as it will automate development without the help of a developer. Large language models have increased the ability and agility of NL2SQL development.

The industry leaders such as Defog have achieved maximum performance with their state-of-the-art model SQLCoder 34b which surpasses the capability of GPT-4. The performance of this model is very good and it has produced consistent, accurate SQL queries for complex database schemas. But on the other end of this, developers who can leverage models like these daily for better work-life, were unable to run it on their machines as most of the LLMs require a lot of VRAM in order to infer from.

On 12th Dec 2023, Microsoft came up with Phi-2, a new LLM with a smaller size, by calling it a small language model which is pre-trained on a specific technique called "Textbooks are all you need", which stands on the concept of "Content In - Content Out, Garbage In - Garbage Out". This essentially means that the data on which the models are trained should be processed and cleaned like a textbook for the model to learn from.

This model has outperformed most of the state-of-the-art base models such as Llama 7B, 13B, 70B (Coding), and even Mistral 7B. Mistral 7B is the base model on top of which the SQLCoder is built through fine-tuning. If the Phi-2, a base model is better than Mistral 7b, a base model, there is a possibility that fine-tuned Phi-2 can be better than SQLCoder 7b, a fine-tuned model. This is crucial as this provides a model which can be used daily by many developers for consistent SQL generation with smaller computes.

Through this fine-tuning and comparative analysis, we anticipate contributing to the ongoing dialogue on the intersection of accuracy and deployability in NL2SQL, aiming to advance the field towards more feasible and even CPU-friendly solutions.

Literature Review

LLMs and Fine-tuning:

Large Language Models (LLMs) are an innovation in natural language processing (NLP) that has caught the attention of researchers, developers, and the public. These huge models are trained on trillions of words of text data, allowing them to learn complex relations between words, generate natural language text, answer questions based on context, and perform other creative tasks. Over time, the usage and demand for better models have significantly increased, along with innovation and releases.

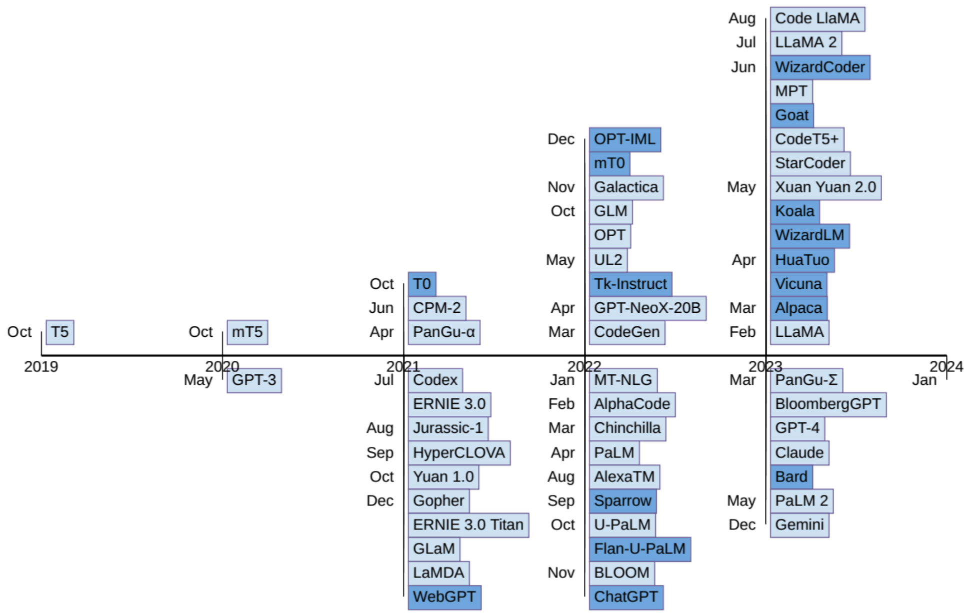


Figure 1: Transformers releases over time - Comprehensive study on LLMs

Architecture:

Most of the LLMs are based on the powerful neural network architectures known as Transformers. These models work on attention mechanisms to process input text and allow them to compare the relationships between words within a sentence and across the entire corpus. Think of it as reading and rereading the text with a spotlight, focusing on different words and their relations with other words, to understand the overall meaning.

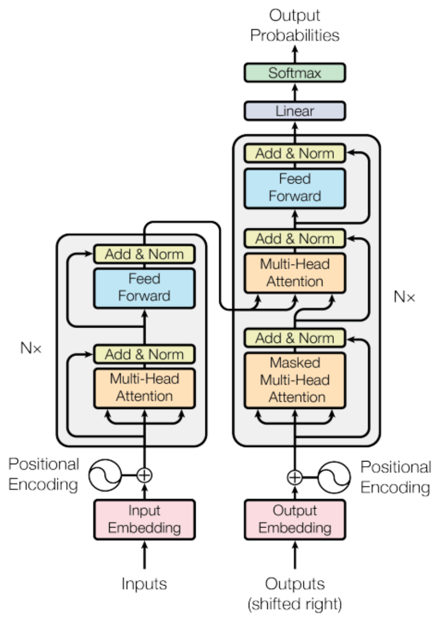


Figure 2: Transformer Architecture - Attention is all you need

Training:

LLMs are trained on huge datasets of text and code, often containing billions of words. This training involves various kinds of supervised and unsupervised training such as

- Masked Language Modelling (MLM): The LLM predicts the missing words in masked sentences.
- Span Corruption: The LLM reconstructs corrupted text back to its original form
- Casual Language Modelling (CLM): The LLM predicts the upcoming word by looking at the words before it.

By going through these tasks multiple times with multiple samples, the LLM learns the basics of the language and builds a representation of it. Often, the pre-training is done by organisations with access to heavy GPU power as the training usually requires a huge VRAM for a long period.

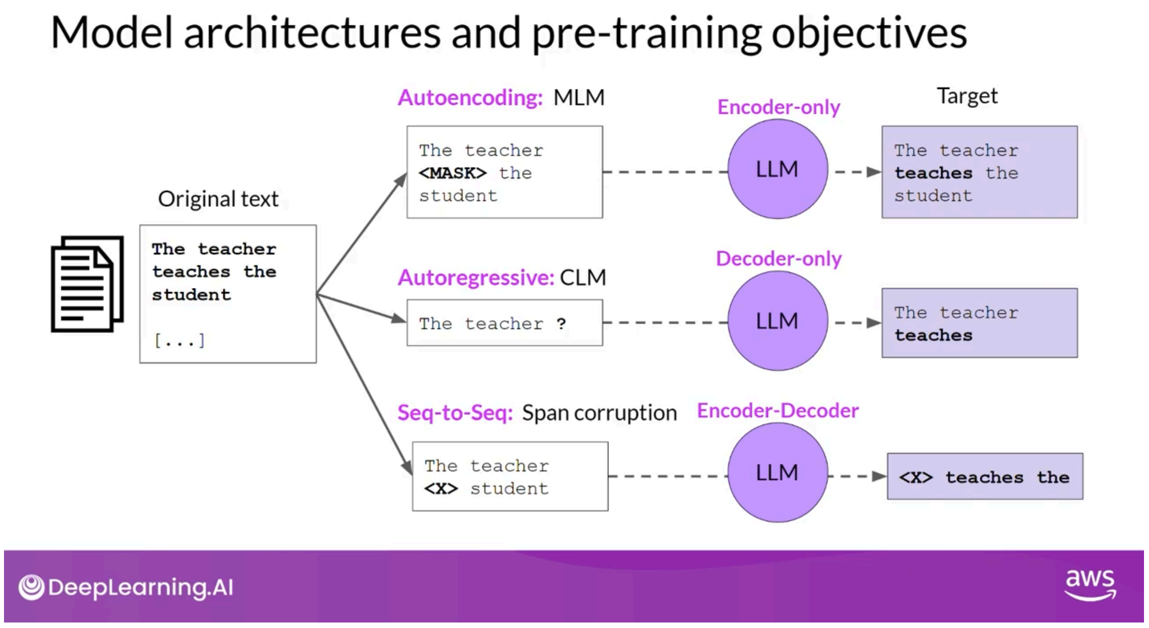


Figure 3: Pre-training Methods of LLMs

Capabilities:

LLMs boast a wide range of capabilities, far exceeding basic language understanding. They can:

- Text Generation: From content creation to code generation, LLMs can produce natural language text that often resembles human writing.
- Machine Translation: LLMs can accurately translate between natural languages, understanding the ambiguities and preserving the original meaning.
- Question Answering: LLMs can process and answer questions based on the context provided along with the question or by drawing on their knowledge from pre-training.
- Text Summarisation: LLMs can reduce lengthy documents into concentrated summaries, capturing the key details and discarding irrelevant information.
- Others: LLMs are also being explored in music composition, speech-to-text generation and even image generation.

Fine-tuning LLMs:

Fine-tuning is a technique for adapting pre-trained LLMs with capabilities like those described above, to perform niche tasks. We can fine-tune an LLM to convert a specific input text of questions into a respective output text. This is a complex task that requires the LLM to understand both natural language and fine-tune specific requirements.

Here's how fine-tuning works:

- Select a dataset: We choose a dataset containing pairs of input text and their corresponding output text for a niche use case. This data will guide the LLM towards the specific task of fine-tuning.
- Freeze most parameters: Instead of retraining the entire LLM, we "freeze" most of its parameters, preserving the general language knowledge from its pre-training.
- Fine-tune specific layers: We add or adjust a few additional layers specifically designed for the fine-tuning use case. These layers are then trained on the chosen dataset, enabling the LLM to learn the mapping between input and output.
- Evaluation and refinement: We evaluate the performance of the fine-tuned LLM on unseen data and make adjustments to the additional layers or training process as needed.

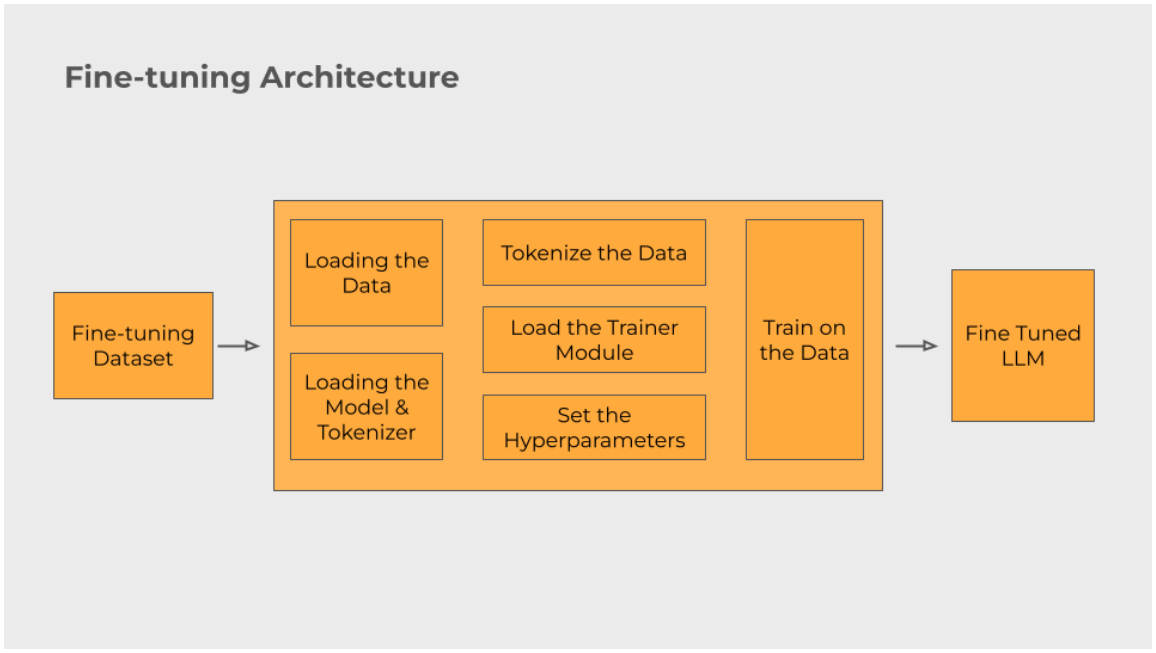


Figure 4: Flow diagram of Fine-tuning of LLM

By fine-tuning, we leverage the general language understanding of the LLM while tailoring it to the fine tune specific requirements. This can significantly improve the model's performance compared to the non-finetuned model. Overall, LLMs represent a remarkable advancement in NLP, and fine-tuning unlocks their potential for specialized tasks. We can expect further progress in both LLM design and fine-tuning techniques, enabling these powerful models to solve even more complex challenges.

NL2SQL:

NL2SQL, or Natural Language to SQL, is a concept that focuses on translating natural language queries into executable SQL statements. NL2SQL allows non-technical users to interact with databases using natural language, without having to write complex queries.

Applications of NL2SQL:

- Business intelligence: NL2SQL empowers analysts to explore data and generate reports with natural language queries.
- Customer service: Chatbots can understand and respond to customer queries by accessing relevant information from databases.
- Education: Students can learn data analysis by formulating queries in natural language.

Common Challenges in NL2SQL:

- Complexity of natural language: Understanding the construction of human language, including ambiguity, synonyms, and complex sentence structures, can be difficult for machines.
- Variety of database schemas: Training a model for different database structures and types of SQL presents a significant challenge.
- Handling complex queries: Translating queries involves aggregates, joins, and subqueries that require reasoning and understanding of database relationships.
- Bias and fairness: NL2SQL models trained on biased data can replicate those biases in their outputs.

Existing Approaches to NL2SQL:

- Rule-based systems: These systems rely on handwritten rules to map natural language keywords to specific SQL clauses. They are often underperforming and require significant manual effort for inference.
- Machine learning models: Techniques like neural networks and tree-based methods can be trained on the mapping from natural language to SQL. These models offer more performance than rule-based systems but can require large amounts of data for real-time use.
- LLM-based techniques: Large Language Models (LLMs) pre-trained on huge text datasets can be fine-tuned for NL2SQL tasks. Their advantage lies in their ability to capture complex relationships between words and adapt to new data quickly.

Fine-tuning LLMs for NL2SQL Use Cases:

- Reduced computation: Fine-tuning only a small portion of the LLM, requires a significantly smaller compute, addressing the concern about accessibility for developers.
- Improved performance: LLMs offer the capability for better accuracy in handling complex queries compared to traditional models.
- Faster development: Fine-tuning pre-trained LLMs requires less training data and can be much easier to implement than training new models from scratch.

Phi-2 and Defog SQLCoder LLMs:

Defog SQLCoder 7b is a powerful model tackling the NL2SQL challenge, but it requires a big compute due to its size. Whereas Phi-2 is a smaller model which can be run on smaller computes. Let's delve into their architecture, training, strengths, and weaknesses:

Phi-2:

- Architecture:
  - Small Language model: Built with fewer parameters than other LLMs, making it lighter and faster to finetune and run.
  - Dense Attention Network (DAN): Focuses on interactions between words. This helps in understanding long-range dependencies between words which in turn helps in understanding complex sentences.
  - Encoder-Decoder structure: Encodes the input text into embeddings and decodes it back into output text.
- Training:
  - Supervised learning: Trained on a large mixture of synthetic and web datasets of NLP and coding.
  - Parameters: Contains a total of 2.7 Billion Parameters which have not undergone alignment techniques such as RLHF.
- Strengths:
  - Smaller size: More accessible for deployment and faster inference compared to larger LLMs.
  - Strong language understanding: Scored more than 50% in benchmarks such as HellaSwag, BoolIQ, SQuADv2, and MMLU.
  - Good performance in Math and Coding: Scored more than 40% in benchmarks such as MBPP, HumanEval, and GSM8K.

- Weaknesses:
  - Parameter Count: LLMs with more parameters tend to remember more information and perform better.
  - Susceptibility to biases: Potential for inheriting biases from the pre-training data.
- Required Hardware:
  - Pretraining of Phi-2 took 14 days on 96 A-100 GPUs.
  - Calculation of required VRAM for Inference:
    - float32 quantization:
      2.7 Billion * 4 bytes (32 bits) = 10.8GB VRAM
    - Bfloat16 quantization:
      2.7 Billion * 2 bytes (16 bits) = 5.4GB VRAM
    - Int8 quantization:
      2.7 Billion * 1 byte (8 bits) = 2.7GB VRAM

Defog SQLCoder:

- Architecture:
  - SQLCoder 7b is fine-tuned on the Mistral 7b LLM, therefore sharing its architecture.
  - Mistral 7b uses the transformer architecture with sliding window attention.
  - This is continued in SQLCoder as it has Mistral as the base model.
- Training:
  - Question Classification: The dataset was classified into easy, medium, and hard using a rubric utilized by the Spider dataset.
  - Finetuning: First fine-tuned on easy and medium. Then the model is further fine-tuned on hard SQL queries.
- Strengths:
  - Stronger performance on complex queries: Performs joins, aggregations, and subqueries with accuracy more than GPT3.5 Turbo.
  - Less susceptible to biases: Multi-stage training with synthetic data helps reduce bias issues.
- Weaknesses:
  - Larger model size: Requires more computational resources for fine-tuning and deployment compared to Phi-2.
  - Slower inference: Due to its large size, it is either slow or computationally much more expensive.
- Required Hardware:
  - Calculation of required VRAM for Inference:
    - float32 quantization:
      7 Billion * 4 bytes (32 bits) = 28GB VRAM

- Bfloat16 quantization:
  7 Billion * 2 bytes (16 bits) = 14GB VRAM
- Int8 quantization:
  7 Billion * 1 byte (8 bits) = 7GB VRAM

Model	Size	BBH	Commonsense Reasoning	Language Understanding	Math	Coding
Llama-2	7B	40.0	62.2	56.7	16.5	21.0
	13B	47.8	65.0	61.9	34.2	25.4
	70B	66.5	69.2	67.6	64.1	38.3
Mistral	7B	57.2	66.4	63.7	46.4	39.4
Phi-2	2.7B	59.2	68.8	62.0	61.1	53.7

Table 1: Comparison of Phi-2 with other SOTA LLMs

Challenges working with Big LLMs:

There are multiple challenges to face while working with huge language models. Till 2024, the largest model to exist is Google PaLM with 540 Billion, which is extremely big when compared to the average LLM size. The LLMs which are marked as “small” start from 7 billion parameters which are significantly smaller yet not small enough to be computationally cheap. Below are some of the calculations made for a 7 billion LLM, to project how expensive inferring or fine-tuning can become as parameters increase.

Inference Memory:

The memory required as VRAM for a model to be loaded and inferred is Inference Memory. This is the minimum memory required to use a large language model. This includes the memory occupied by parameters.

For a model of 7 Billion parameters:

- 7 Billion = 7,000,000,000 parameters
- 1 parameter = 32 bits
- 1 byte = 8 bits

Total VRAM required in Bytes = 7,000,000,000 * 32 / 8 = 28,000,000,000
Total VRAM required in GB = 28 GB

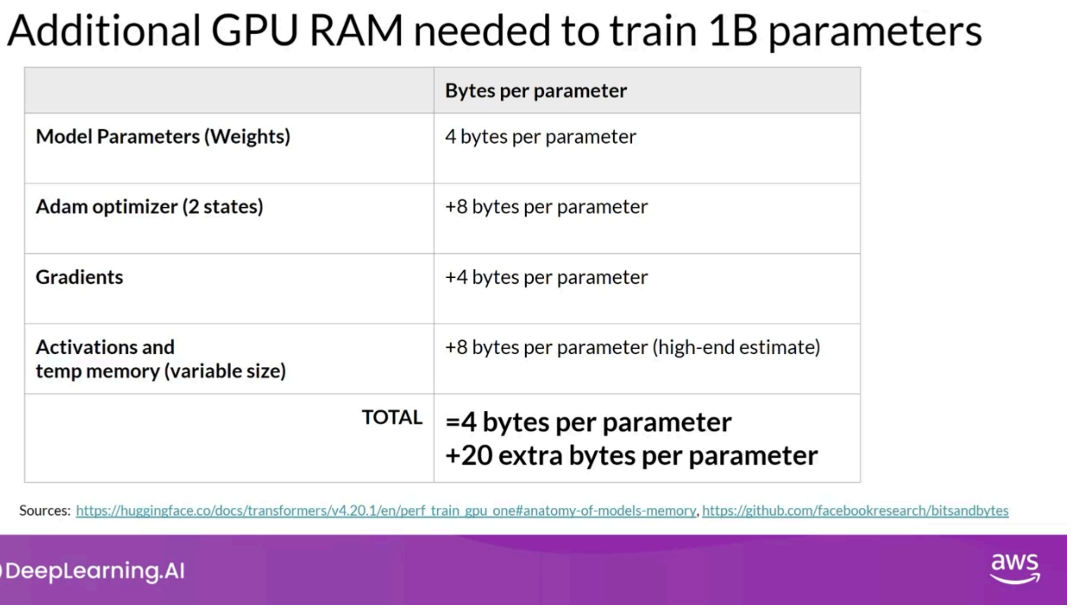
Training Memory:

The memory required as VRAM for a model to be loaded, inferred, and fine-tuned is Training Memory. This is the minimum memory required to train a large language model on 1 data sample. This includes the memory occupied by parameters, optimizer states, gradients, and activations.

For a model of 7 Billion parameters:

- 7 Billion = 7,000,000,000 parameters
- 2 optimizer states = 8 bytes/param
- Gradients = 4 bytes/param
- Activations = 8 bytes/param
- 1 parameter = 4 bytes
- Total bytes per parameter = 8+4+8+8 = 24 Bytes

Total VRAM required in Bytes = 7,000,000,000 * 24 = 168,000,000,000
Total VRAM required in GB = 168 GB



Additional GPU RAM needed to train 1B parameters

	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter
Adam optimizer (2 states)	+8 bytes per parameter
Gradients	+4 bytes per parameter
Activations and temp memory (variable size)	+8 bytes per parameter (high-end estimate)
TOTAL	**=4 bytes per parameter +20 extra bytes per parameter**

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, https://github.com/facebookresearch/bitsandbytes

DeepLearning.AI aws

Figure 5: Number of bytes required per parameter to fine-tune an LLM

Computation Cost:

The computational cost is different for inference and fine-tuning due to their VRAM requirements. The inference cost is always less than fine-tuning as the required parameters to be loaded are different.

GPU Cloud	GPU	VRAM (GB)	CPUs	RAM (GB)	$ per hr
AWS EC2	A100	80	12	144	5.12
Microsoft Azure	A100	80	12	238	4.1
Lambda Labs	RTX A6000	48	14	100	1.45
AWS EC2	A100	40	12	144	4.1
Google Cloud	A100	40	12	85	3.65
Microsoft Azure	A100	40	12	112	3.4
AWS EC2	V100	32	12	96	3.9

Table 2: Comparison of Price of the GPUs between Cloud providers

The above table is a reference from paperspace.com/gpu-cloud-comparison, which has an exhaustive list of all the cloud providers with their respective GPUs and their pricing.

Handling the challenges:

Quantization in LLMs (Large Language Models) is one of the techniques used to make huge LLMs more friendly for deployment and inference. It involves reducing the size of the LLM without sacrificing too much accuracy, by reducing the precision level of the parameters.

Imagine each parameter in the LLM is like a high-resolution photo. Quantization is shrinking the photo by compressing the pixels and reducing the resolution. This leads to reduced clarity, or even, reduced precision. Similarly, it converts the parameters (often stored as 32-bit floating points) to lower-precision formats like 8-bit integers. This means representing each parameter with fewer bits, significantly shrinking the model size.

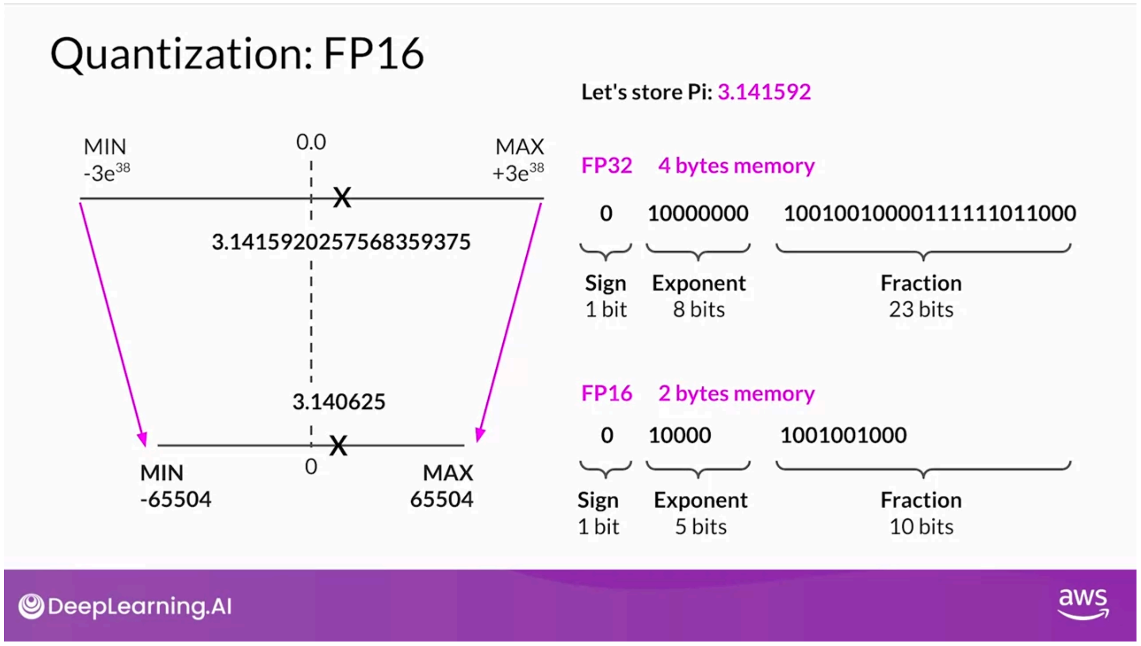


Figure 6: Quantization of Floating Point 32 to Floating Point 16

3. Methodology

- **Datasets & Pre-processing:**
  - The dataset being used here is the Hugging Face's b-mc2/sql-create-context. This combines both the Spider and WikiSQL datasets with additional context through the database schema-related information.
  - The dataset has columns: question, answer, and context. This provides so much feasibility for the developers to create their context-based prompts based on the pre-trained models.
  - While preprocessing, these columns can be used to understand the difficulty of the query that is being generated.
  - The usage of this model for training helps during the inference. The users just need to provide the relevant db schema to the model along with the question which can generate SQL.

- **Fine-tuning Phi-2:**
  - By splitting the dataset of 78k rows into Train (74.1k), Validation (2.34k), and Test (1.56k) datasets, we'll prepare the dataset.
  - With the dataset columns, we will prepare an instruct prompt for the Phi-2 which will be used for fine-tuning.
  - Configuring BitsAndBytes for quantization is a crucial step and leads to easier and more efficient fine-tuning.
  - Configuring Training Arguments / Hyperparameters with relevant values.
  - Configuring Lora parameters is also a crucial step, as it will ensure parameter-efficient fine-tuning.
  - After this, we can instantiate the Trainer Module from transformers, and start fine-tuning.
  - Must consider the possibility of mismatched versions or GPU-related issues during fine-tuning and resolve them.
  - At each step, we should fine-tune over the validation dataset, to understand the state of configurations.

- **Evaluation Metrics**
  - Once the CSV file is created, we load the scoring data frame.
  - From the columns of the scoring data frame, we will fetch the generated SQL and actual SQL and compare them.
  - The inference times, GPU usage per inference, overall GPU usage, CPU usage per inference, and overall CPU usage will also be considered to establish a comparison between Phi-2 and Defog SQLCoder.

- **Comparison with DeFoG SQLCoder**
    - Once fine-tuning is completed, we should store the model in local or hf_hub.
    - In a new kernel, using the test dataset, we shall infer on the fine-tuned phi-2 model, and store various factors of the result.
    - In another new kernel, using the test dataset, we shall infer on the defog SQLCoder-7b model, and store various factors of result.
    - Storing the scores of Phi-2 and Defog SQLCoder and saving them in another CSV file helps in evaluation.

- **Analysis and Findings**
    - The findings shall be documented and a deck shall be prepared to showcase the detailed analysis.
    - The comparison will also showcase if the fine-tuned Phi-2 is better than the Defog SQLCoder.

4. References

[1] Revanth, T. J., Sai, (2021, December 1). NL2SQL: Natural Language to SQL Query Translator.
https://doi.org/10.1007/978-981-16-1342-5_21

[2] Vaswani, A. (2017, June 12). Attention Is All You Need. arXiv.org.
https://arxiv.org/abs/1706.03762

[3] Naveed, H. (2023, July 12). A Comprehensive Overview of Large Language Models. arXiv.org.
https://arxiv.org/abs/2307.06435

[4] Alekh Jindal, (2024, January 14). Turning Databases Into Generative AI Machines.
https://www.cidrdb.org/cidr2024/papers/p81-jindal.pdf

[5] Tony, K., Shaji, K. S., Noble, N., Devasia, R. J., & Chandrasekhar, N. (2023, January 1). NL2SQL: Rule-Based Model for Natural Language to SQL. Algorithms for Intelligent Systems.
https://doi.org/10.1007/978-981-99-4626-6_66

[6] Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance. (2022, April 4).
https://blog.research.google/2022/04/pathways-language-model-palm-scaling-to.html

[7] Large Language Models: A New Moore's Law? (n.d.).
https://huggingface.co/blog/large-language-models

[8] Malta, E. M., Rodamilans, C., Avila, S., & Borin, E. (2019, April 12). A cost-benefit analysis of GPU-based EC2 instances for a deep learning algorithm.
https://doi.org/10.5753/eradsp.2019.13588

[9] 2023 GPU Pricing Comparison: AWS, GCP, Azure & More | Paperspace. (n.d.).
https://www.paperspace.com/gpu-cloud-comparison

[10] Han, F., Liu, C., Wu, B., Li, F., Tan, J., & Sun, J. (2023, February 1). CatSQL : Towards Real World Natural Language to SQL Applications. Proceedings of the VLDB Endowment.
https://doi.org/10.14778/3583140.3583165

[11] Gunasekar, S. (2024, January 22). Textbooks Are All You Need - Microsoft Research. Microsoft Research.
https://www.microsoft.com/en-us/research/publication/textbooks-are-all-you-need/

[12] Hughes, A. (2023, December 16). Phi-2: The surprising power of small language models - Microsoft Research. Microsoft Research.
https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/

[13] Ma, G. (2023, August 16). Pre-training with Large Language Model-based Document Expansion for Dense Passage Retrieval. arXiv.org.
https://arxiv.org/abs/2308.08285

[14] Open-sourcing SQLCoder2-15b and SQLCoder-7b. (n.d.).
https://defog.ai/blog/open-sourcing-sqlcoder2-7b/

[15] Barth, Antje. "Scaling Laws and Compute-optimal Models." *Coursera*, www.coursera.org/learn/generative-ai-with-llms/lecture/SmRNp/scaling-laws-and-compute-optimal-models.

[16] Barth, Antje. "Pre-training Large Language Models." *Coursera*, www.coursera.org/learn/generative-ai-with-llms/lecture/2T3Au/pre-training-large-language-models.

[17] Barth, Antje. "Computational Challenges of Training LLMs." *Coursera*, www.coursera.org/learn/generative-ai-with-llms/lecture/gZArr/computational-challenges-of-training-llms.