**LLM FINE TUNING:**

LLM FINE TUNING involves taking a pre trained LLM and training it on our own private dataset in order to make the LLM more personalized and responsive. Fine tuning helps improve the performance of the model in a particular task thus allowing for more detailed and tailored responses. Fine tuning is more beneficial and viable as compared to building a LLM model from scratch as it uses the knowledge and expertise which are gained by the LLM model from its pre training and harnesses them for a specific use case or task.

Steps involved in fine tuning:

* Selecting a LLM model

Selecting a suitable LLM model from all the available pre trained models based on specific use case and requirements.

* Preparation of task specific dataset

Preparing a labelled and task specific dataset in order to train the LLM on it.

* Fine-tuning process

The pretrained model is now trained on the custom dataset thus adjusting its internal parameters thus minimizing the gap between the prediction and ground truth.

* Evaluation

Evaluate the fine-tuned model's performance using appropriate metrics and iterate on the process if needed**.**

**1] Selecting a LLM model**Amongst the many LLM models, choosing the right model based on our specific needs and task requirements is a necessity.

This link lists all the open source LLMs which are available for use including fine tuning: [open source LLM's](https://github.com/eugeneyan/open-llms)

In our case, we will be fine tuning our LLM model on company data for querying the database based on user prompts and accurately fetching the necessary data for generation of graphs. Below are some LLM models which can be appropriate for our particular use case.

1. Llama 2: Developed by Meta AI and Microsoft, Llama 2 is trained on publicly available online data and is a strong contender for fine-tuning.
2. Mistral: Mistral AI offers several models, including Mistral 7B and Mistral 8x7B, which are known for their strong performance and efficiency.
3. Gemma: Google's Gemma models, available in different sizes, are another good option for fine-tuning, offering a balance of performance and resource requirements.

Common techniques for fine tuning:

* **Supervised Fine-Tuning:**

This is the most common approach, where the pre-trained model is trained on a labelled dataset specific to the target task. The model's weights are adjusted based on the task-specific data.

* **Instruction Fine-Tuning:**

A specific type of supervised fine-tuning that focuses on training the model on instruction-response pairs, allowing it to better understand and follow human instructions.

* **Parameter-Efficient Fine-Tuning (PEFT):**

PEFT methods focus on fine-tuning only a small subset of the model's parameters, such as adding small trainable layers (adapters) to the pre-trained model, while keeping the majority of the pre-trained parameters frozen. This significantly reduces computational cost and memory usage.

* **Transfer Learning:**

Fine-tuning is a form of transfer learning, where knowledge from a source task (the pre-trained model) is applied to a related target task.

* **Multi-task Fine-Tuning:**

Fine-tuning the model on multiple related tasks simultaneously, which can lead to better generalization and performance on each individual task.

* **Few-shot Learning:**

Fine-tuning with a very small amount of labelled data, often leveraging techniques like data augmentation to improve performance.

**2] Preparation of task specific dataset:**

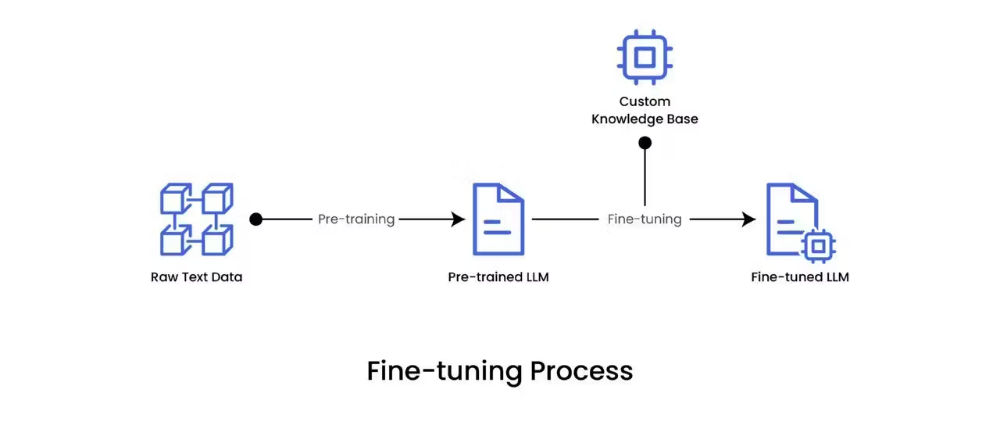
Preparing a fine-tuning dataset involves selecting, cleaning, and formatting data to train a model for a specific task. The process typically includes collecting relevant data, ensuring its quality and diversity, and formatting it in a way that the chosen model can understand**.**

Create Fine-tuning Datasets Step-by-Step Guide

* **1. Define the Goals:** Start by clearly defining what you want to achieve with your fine-tuned model. Whether it’s enhancing accuracy, speeding up response times, saving costs, or customizing interactions, setting clear objectives guides the entire dataset creation process.
* **2. Collect and Organize Data:** Gather relevant and diverse examples that mirror the real-world applications your model will encounter. Quality over quantity, so ensure the data is highly relevant and closely matches the contexts your model needs to excel in. Organize this data in a format that supports efficient tuning, as shown in the example above.
* **3. Deploy, Evaluate, and Iterate:** Once the model is trained on the dataset, regularly assess the performance of your fine-tuned model against your objectives. Use insights from this evaluation to refine your dataset further, making sure your model remains effective and responsive to edge cases.

Example of dataset which can be used for fine tuning:   
Link: [**HelpSteer**](https://huggingface.co/datasets/nvidia/HelpSteer)

* The NVIDIA HelpSteer dataset is a collection of 1.4 million human-written instructions for self-driving cars. It covers a wide range of scenarios and includes detailed, step-by-step instructions. This dataset can be valuable for fine-tuning LLMs to generate clear and concise instructions for autonomous vehicles. This is particularly important as clarity and precision in instructions are vital for the safety and reliability of self-driving cars. By training LLMs with the HelpSteer dataset, it’s possible to enhance the communication interface between the vehicle and its control systems, thereby improving the car’s ability to make informed and accurate decisions in real-time. This integration of LLMs with comprehensive driving data sets marks a significant step towards more intelligent, aware, and responsive autonomous vehicles, potentially revolutionizing the future of transportation.

**3]** **Fine-tuning process:  
**

Fine-tuning is the process of adjusting the parameters of a pre-trained large language model to a specific task or domain. Although pre-trained language models like GPT possess vast language knowledge, they lack specialization in specific areas. LLM fine-tuning addresses this limitation by allowing the model to learn from domain-specific data to make it more accurate and effective for targeted applications.

By exposing the model to task-specific examples during fine-tuning, the model can acquire a deeper understanding of the nuances of the domain. This bridges the gap between a general-purpose language model and a specialized one, unlocking the full potential of LLMs in specific domains or applications.

**Different types of LLM fine tuning**: Fine-tuning involves adjusting LLM parameters, and the scale of this adjustment depends on the specific task that you want to fulfil. Broadly, there are two fundamental approaches to fine-tuning LLMs: feature extraction and full fine-tuning. Let’s explore each option in brief.

1. **Feature extraction (repurposing)**

Feature extraction, also known as repurposing, is a primary approach to fine-tuning LLMs. In this method, the pre-trained LLM is treated as a fixed feature extractor. The model, having been trained on a vast dataset, has already learned significant language features that can be repurposed for the specific task at hand.

The final layers of the model are then trained on the task-specific data while the rest of the model remains frozen. This approach leverages the rich representations learned by the LLM and adapts them to the specific task, offering a cost-effective and efficient way to fine-tune LLMs.

1. **Full fine-tuning**

Full fine-tuning is another primary approach to fine-tuning LLMs for specific purposes. Unlike feature extraction, where only the final layers are adjusted, full fine-tuning involves training the entire model on the task-specific data. This means all the model layers are adjusted during the training process.

This approach is particularly beneficial when the task-specific dataset is large and significantly different from the pre-training data. By allowing the whole model to learn from the task-specific data, full fine-tuning can lead to a more profound adaptation of the model to the specific task, potentially resulting in superior performance. It is worth noting that full fine-tuning requires more computational resources and time compared to feature extraction.

**Different Methods of LLM fine tuning:**

There are several fine-tuning methods and techniques used to adjust the model parameters to a given requirement. Broadly, we can classify these methods in two categories: supervised fine-tuning and reinforcement learning from human feedback (RLHF).

**a. Supervised fine-tuning**

In this method, the model is trained on a task-specific labelled dataset, where each input data point is associated with a correct answer or label. The model learns to adjust its parameters to predict these labels as accurately as possible. This process guides the model to apply its pre-existing knowledge, gained from pre-training on a large dataset, to the specific task at hand. Supervised fine-tuning can significantly improve the model's performance on the task, making it an effective and efficient method for customizing LLMs.

The most common supervised fine-tuning techniques are:

**1. Basic hyperparameter tuning**

Basic hyperparameter tuning is a simple approach that involves manually adjusting the model hyperparameters, such as the learning rate, batch size, and the number of epochs, until you achieve the desired performance.

The goal is to find the set of hyperparameters that allows the model to learn most effectively from the data, balancing the trade-off between learning speed and the risk of overfitting. Optimal hyperparameters can significantly enhance the model's performance on the specific task.

**2. Transfer learning**

Transfer learning is a powerful technique that’s particularly beneficial when dealing with limited task-specific data. In this approach, a model pre-trained on a large, general dataset is used as a starting point.

The model is then fine-tuned on the task-specific data, allowing it to adapt its pre-existing knowledge to the new task. This process significantly reduces the amount of data and training time required and often leads to superior performance compared to training a model from scratch.

**3. Multi-task learning**

In multi-task learning, the model is fine-tuned on multiple related tasks simultaneously. The idea is to leverage the commonalities and differences across these tasks to improve the model's performance. The model can develop a more robust and generalized understanding of the data by learning to perform multiple tasks simultaneously.

**4. Few-shot learning**

Few-shot learning enables a model to adapt to a new task with little task-specific data. The idea is to leverage the vast knowledge model has already gained from pre-training to learn effectively from just a few examples of the new task. This approach is beneficial when the task-specific labeled data is scarce or expensive.

In this technique, the model is given a few examples or "shots” during inference time to learn a new task. The idea behind few-shot learning is to guide the model's predictions by providing context and examples directly in the prompt.

**5. Task-specific fine-tuning**

This method allows the model to adapt its parameters to the nuances and requirements of the targeted task, thereby enhancing its performance and relevance to that particular domain. Task-specific fine-tuning is particularly valuable when you want to optimize the model's performance for a single, well-defined task, ensuring that the model excels in generating task-specific content with precision and accuracy.

**b. Reinforcement learning from human feedback (RLHF)**

Reinforcement learning from human feedback (RLHF) is an innovative approach that involves training language models through interactions with human feedback. By incorporating human feedback into the learning process, RLHF facilitates the continuous enhancement of language models so they produce more accurate and contextually appropriate responses. This approach not only leverages the expertise of human evaluators but also enables the model to adapt and evolve based on real-world feedback, ultimately leading to more effective and refined capabilities.

The most common RLHF techniques are:

**1. Reward modelling**

In this technique, the model generates several possible outputs or actions, and human evaluators rank or rate these outputs based on their quality. The model then learns to predict these human-provided rewards and adjusts its behaviour to maximize the predicted rewards.

Reward modelling provides a practical way to incorporate human judgment into the learning process, allowing the model to learn complex tasks that are difficult to define with a simple function. This method enables the model to learn and adapt based on human-provided incentives, ultimately enhancing its capabilities.

**2. Proximal policy optimization**

Proximal policy optimization (PPO) is an iterative algorithm that updates the language model's policy to maximize the expected reward. The core idea of PPO is to take actions that improve the policy while ensuring the changes are not too drastic from the previous policy. This balance is achieved by introducing a constraint on the policy update that prevents harmful large updates while still allowing beneficial small updates.

This constraint is enforced by introducing a surrogate objective function with a clipped probability ratio that serves as a constraint. This approach makes the algorithm more stable and efficient compared to other reinforcement learning methods.

**3. Comparative ranking**

Comparative ranking is similar to reward modelling, but in comparative ranking, the model learns from relative rankings of multiple outputs provided by human evaluators, focusing more on the comparison between different outputs. gtIn this approach, the model generates multiple outputs or actions, and human evaluators rank these outputs based on their quality or appropriateness. The model then learns to adjust its behaviour to produce outputs that are ranked higher by the evaluators.

By comparing and ranking multiple outputs rather than evaluating each output in isolation, comparative ranking provides more nuanced and relative feedback to the model. This method helps the model understand the task subtleties better, leading to improved results.

**4. Preference learning (reinforcement learning with preference feedback)**

Preference learning, also known as reinforcement learning with preference feedback, focuses on training models to learn from human feedback in the form of preferences between states, actions, or trajectories. In this approach, the model generates multiple outputs, and human evaluators indicate their preference between pairs of outputs.

The model then learns to adjust its behaviour to produce outputs that align with the human evaluators' preferences. This method is useful when it is difficult to quantify the output quality with a numerical reward but easier to express a preference between two outputs. Preference learning allows the model to learn complex tasks based on nuanced human judgment, making it an effective technique for fine-tuning the model on real-life applications.

**5. Parameter efficient fine-tuning**

Parameter-efficient fine-tuning (PEFT) is a technique used to improve the performance of pre-trained LLMs on specific downstream tasks while minimizing the number of trainable parameters. It offers a more efficient approach by updating only a minor fraction of the model parameters during fine-tuning.

PEFT selectively modifies only a small subset of the LLM's parameters, typically by adding new layers or modifying existing ones in a task-specific manner. This approach significantly reduces the computational and storage requirements while maintaining comparable performance to full fine-tuning.