Fine tuning :

What is Fine tuning ?

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.** 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.

Example of Fine tune model :

chat → chat gpt

gpt4 — github copilot

pcp — cardiologist

base model - fine tune → accurate answer

Benefit :

1. stop hallucinations : random answer stop
2. increase consistency. :
3. reduce unwanted information

Where:

pytorch

hugging face

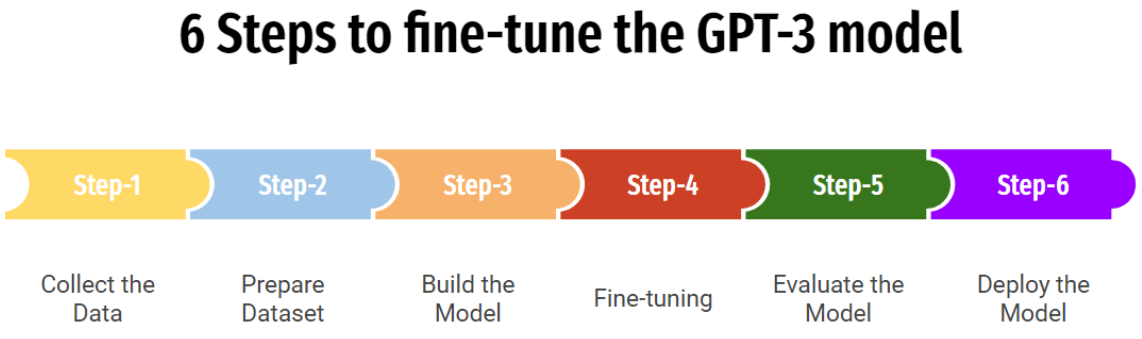
Llama library(Lamini)

unsloth

How to Fine-Tune a Model

Here is an overview of the fine-tuning process:

1. Start with a pre-trained model like GPT-3.
2. Gather a dataset specific to your task. This is known as the “fine-tuning set.”
3. Pass **examples** from the dataset to the model and collect its outputs.
4. Calculate the loss between the model’s outputs and the expected outputs.
5. Update the model parameters to reduce the loss using gradient descent and backpropagation.
6. Repeat steps 3–5 for multiple epochs until the model converges.
7. The fine-tuned model can now be deployed for inference on new data.



Step 1: Choose a pre-trained model and a dataset

Step 2: Load the data to use

Step 3: Tokenizer

Step 4: Initialize our base model

Step 5: Evaluate method

Step 6: Fine-tune using the Trainer Method

Use Cases for Fine-Tuned Models :

- **Customer support:** Fine-tune a model on **customer support** tickets to generate automated responses.
- **Content generation:** Fine-tune on a **company's documentation** to automatically generate content adhering to their voice and style.
- **Translation:** Fine-tune translation models on industry-specific data like medical or legal documents.
- **Information retrieval:** Fine-tune a QA model to answer domain-specific questions.
- **Sentiment analysis:** Fine-tune a classifier to identify sentiment in social media related to your products.

When to Fine-Tune a Model

Here are some examples of when fine-tuning can be beneficial:

- **Adapting to a new domain or genre:** Fine-tune a general model on technical documents to specialize in that field.
- **Improving performance on a specific task:** Fine-tune a model to generate better poetry or translate between two languages.
- **Customizing output characteristics:** Fine-tune a model to adjust its tone, personality or level of detail.
- **Adapting to new data:** If your data distribution changes over time, fine-tune the model to keep u
- **Customization**

  Every domain or task has its own unique language patterns, terminologies, and contextual nuances. By fine-tuning a pre-trained LLM, you can

customize it to better understand these unique aspects and generate content specific to your domain.

Limited labeled data

In many real-world scenarios, obtaining large quantities of labeled data for a specific task or domain can be challenging and costly. Fine-tuning allows organizations to leverage pre-**existing labeled data more effectivel**

Disadvatages:

1. overfitting
2. **loss of original model**
3. underfitting
4. Lack of tranferiablity : medical → psychology .
5. Loss of Generalisation

When Not to Fine-Tune

While fine-tuning is powerful, it isn't always the best approach. Here are some cases where it may not be beneficial:

- **Your dataset is very small.** Fine-tuning requires hundreds to thousands of quality examples.
- Your task is **extremely dissimilar from the original model's training data.** The model may struggle to connect its existing knowledge to this new domain.
- You need to **frequently update or modify the model.** Retraining from scratch allows for more flexibility.
- Your problem can be **solved with simpler methods**. Fine-tuning large models can be **overkill.**

Primary fine-tuning approaches :

Two fundamental approaches to fine-tuning LLMs: **feature extraction** and **full fine-tuning**.

Feature extraction (repurposing)

Feature extractio**n, 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.

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.

Parameter Efficient Fine-Tuning (PEFT):

It is a form of instruction fine-tuning that is much more efficient than full fine-tuning. Training a language model, especially for full LLM fine-tuning,

demands significant computational resources. Memory allocation is not only required for storing the model but also for essential parameters during training, presenting a challenge for simple hardware. PEFT addresses this by updating only a subset of parameters, effectively “freezing” the rest.

Why PEFT ?

PEFT empowers parameter-efficient models with impressive performance, revolutionizing the landscape of NLP. Here are a few reasons why we use PEFT.

- **Reduced Computational Costs:** PEFT requires fewer **GPUs and GPU time, making it more accessible and cost-effective** for training large language models.
- **Faster Training Times:** With PEFT, models finish training faster, enabling rapid iterations and quicker deployment in real-world applications.
- **Lower Hardware Requirements:** PEFT works efficiently with smaller GPUs and requires less memory, making it feasible for resource-constrained environments.
- **Improved Modeling Performance:** PEFT produces more robust and accurate models for diverse tasks by reducing overfitting.
- **Space-Efficient Storage:** With shared weights across tasks, PEFT minimizes storage requirements, optimizing model deployment and management.

There are various ways of **achieving Parameter efficient fine-tunin**g. Low-Rank Adaptation **LoRA** & **QLoRA** are the most widely used and effective.

What is LoRa?

LoRA is an improved finetuning method where instead of finetuning all the weights that constitute the weight matrix of the pre-trained large language model, two smaller matrices that approximate this larger matrix are fine-tuned. These matrices constitute the **LoRA adapter.** This fine-tuned adapter is then loaded into the pre-trained model and used for inference.

After LoRA fine-tuning for a specific task or use case, the outcome is an unchanged original LLM and the emergence of a considerably smaller “LoRA adapter,” often representing a single-digit percentage of the original LLM size (in MBs rather than GBs).

During inference, the LoRA adapter must be combined with its original LLM. The advantage lies in the ability of many LoRA adapters to reuse the original LLM, thereby **reducing overall memory requirements when handling multiple tasks and use cases**

What is Quantized LoRA (QLoRA)?

QLoRA represents a **more memory-efficient iteration of LoRA**. **QLoRA** takes **LoRA** a step further by also **quantizing the weights of the LoRA adapters** (smaller matrices) to lower precision (**e.g., 4-bit instead of 8-bit)**. This further reduces the memory footprint and storage requirements. In QLoRA, the pre-trained model is loaded into GPU memory with quantized 4-bit weights, in contrast to the 8-bit used in LoRA. Despite this reduction in bit precision, QLoRA maintains a comparable level of effectiveness to LoRA.

Both LoRA (Low-Rank Adaptation) and QLoRA (Quantized Low-Rank Adaptation) are techniques used to fine-tune large language models (LLMs) more efficiently.

However, they have some key differences:

Feature	LoRA	QLoRA
Parameter reduction	Low-rank approximation of ΔW	Quantization of LoRA adapter weights
Memory footprint	Reduced	Further reduced
Fine-tuning speed	Fast	Slightly slower than LoRA
Performance	Close to traditional fine-tuning	Similar to LoRA

LoRA: Low Rank Adaptation

- **Reduces memory footprint:** LoRA achieves this by applying a low-rank approximation to the weight update matrix (ΔW). This means it represents ΔW as the product of two smaller matrices, significantly reducing the number of parameters needed to store ΔW.
- **Fast fine-tuning:** LoRA offers fast training times compared to traditional fine-tuning methods due to its reduced parameter footprint.
- **Maintains performance:** LoRA has been shown to maintain performance close to traditional fine-tuning methods in several tasks.
- process:
    1. instead of updating : we track the weight
    2. multiply that matrix of same size for updating .
    3. rank 1 = 5 ∗5 = 25 matrices
    4. Rank 2 = 10∗ 10= 100
    5. exmaple : 512 only 1.22% parameter need to update

QLoRA:

- **Enhances parameter efficiency:** QLoRA takes LoRA a step further by also quantizing the weights of the LoRA adapters (smaller matrices) to lower precision (e.g., 4-bit instead of 8-bit). This further reduces the memory footprint and storage requirements.

- **More memory efficient:** QLoRA is even more memory efficient than LoRA, making it ideal for resource-constrained environments.
- **Similar effectiveness:** QLoRA has been shown to maintain similar effectiveness to LoRA in terms of performance, while offering significant memory advantages.

Choosing between LoRA and QLoRA:

The best choice between LoRA and QLoRA depends on your specific needs:

- **If memory footprint is the primary concern:** QLoRA is the better choice due to its even greater memory efficiency.
- **If fine-tuning speed is crucial:** LoRA may be preferable due to its slightly faster training times.
- **If both memory and speed are important:** QLoRA offers a good balance between both.

1. Loading dataset
2. Create Bitsandbytes configuration
3. Loading the Pre-Trained model
4. Tokenization
5. Test the Model with Zero Shot Inferencing
6. Pre-processing dataset
7. Preparing the model for QLoRA
8. Setup PEFT for Fine-Tuning
9. Train PEFT Adapter
10. Evaluate the Model Qualitatively (Human Evaluation)

11. Evaluate the Model Quantitatively (with ROUGE Metric)

Prominent fine-tuning method:

supervised fine-tuning and **reinforcement learning from human feedback (RLHF)**.

the model is trained on a task-specific labeled 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.

This approach leads to improved performance, especially when the tasks it will perform are closely related or when there is limited data for individual tasks. Multi-task learning requires a labeled dataset for each task, making it an inherent component of supervised fine-tuning.

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.

Few-shot learning can also be integrated into the reinforcement learning from human feedback (RLHF) approach if the small amount of task-specific data includes human feedback that guides the model's learning process.

5. Task-specific fine-tuning

This m**ethod 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.

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.

The most common RLHF techniques are:

1. Reward modeling

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 behavior to maximize the predicted rewards.

Reward modeling 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) i**s 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 modeling, 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.

In 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 behavior 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 behavior 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.

DAY : 2

What is Instruction Fine tuning?

Instruction fine-tuning is a specialized technique to tailor large language models to perform specific tasks based on explicit instructions. While traditional fine-tuning involves training a model on task-specific data, instruction fine-tuning goes further by incorporating high-level instructions or demonstrations to guide the model's behavior.

Methods to improve performance for specific use cases

- In context learning, identifying instructions in a prompt. Small LLM may fail to carry out tasks.
- Giving one or more examples can be enough to identify and carry out a task. Smaller models don't always work as expected with instructions
-

Steps :

Step 1: Load the Pre-trained Language Model and Tokenizer

Step 2: Prepare the Instruction Data and Sentiment Analysis Dataset

Step 3: Customize the Model Architecture with Instructions

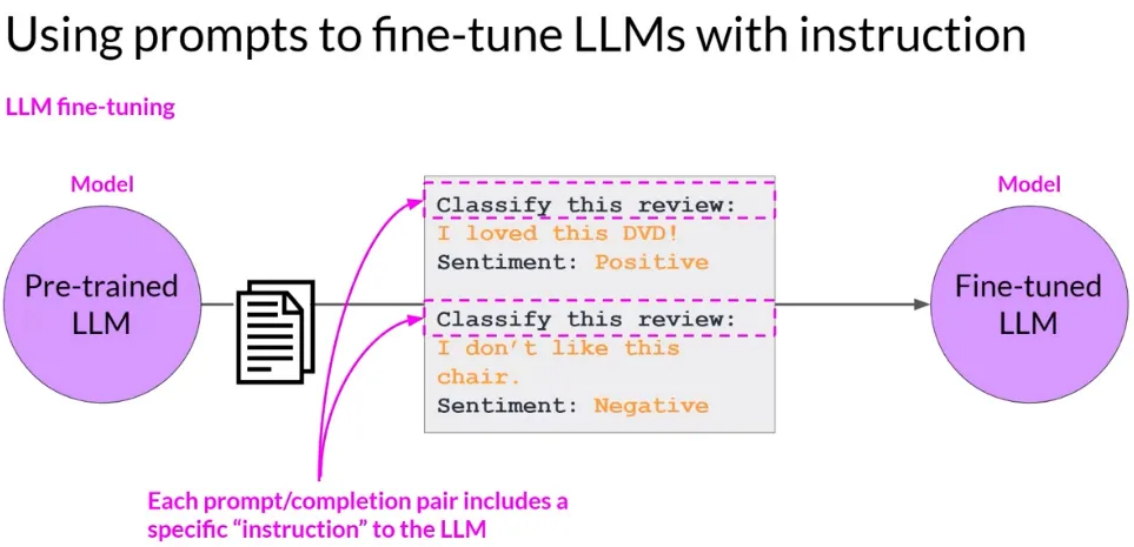
Step 4: Fine-Tune the Model with Instructions .

- Instruction data is ready
- Training, Validation, and Tests split
- Response to training data

Instruction Tuning and Its Significance

While pre-training LLMs on vast amounts of data allows them to learn about the world, they often struggle to respond to specific prompts or instructions. **This is**

where instruction fine-tuning comes into play. By training the model to change its behavior and respond more effectively to instructions, we can enhance its performance and usefulness



Unlike pre-training, where models learn to predict the next word based on general text, fine-tuning allows us to train the model on a smaller dataset specifically tailored to following instructions. This fine-tuning process enables the model to adapt and excel at specific tasks.

Sample instructions like :

1. Classfication /sentiment analysis
2. Text generation
3. Text summarization

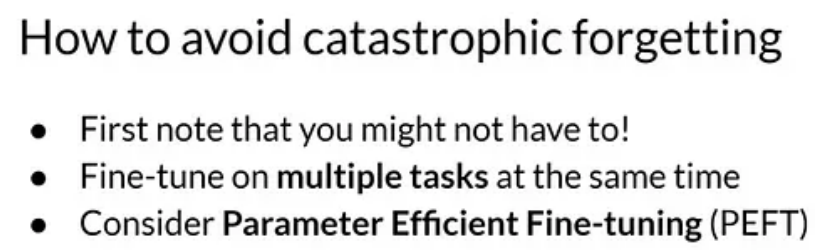
Finetune process :

instruction dataset → pretrained model — > Labelling → loss measure

Fine tune on single task :

model —> single task — > model

500-1000 example to trains for single task



catastrophic forgetting :

1. Good for single and forget the other task : sentiment anslysis but forget other
2. reduce the ability for other tasks

How to handle:

How to avoid catastrophic forgetting?

- Fine-tune on Multiple tasks at a time
- Parameter efficient fine-tuning(PEFT — big area of research)

Multitask finetuning :

summarize

rate the review

translate into python code

indentify the places

Multitask fine-tuning, an extension of single task fine-tuning, has emerged as a powerful technique to enhance the performance of language models across various tasks simultaneously. By training the model on a diverse dataset comprising examples from multiple tasks, including summarization, review rating, code translation, and entity recognition, catastrophic forgetting can be avoided.

issues : lots of data need for example 50 t0 100 thousand

Multi Task Fine Tuning —

- Extension of a single task output instruction
- Single input instructions and multiple output formats carry out variety of tasks
-

Text Summarisation , code trainslation and entity recognition, catastrophic forgteting how to be good at multiple tasks to resulting model

Update wieghts of model in

It required a lot of data 50–100 thoudsandas examples to finetune, good performance and many taska re desirable

Model trainined on Multitasks

Dataset tasks used during tuning — data sets are chosen from variety of sources of research papaers, and other datasets

FLAN — Fine tune Language Net — last step of trianing proces s — General purpose instruct model.

FLAN T5 — T5 foundation model, general purpose instruct model

FLAN PaLM —Fine tuned with PaLM as foundation model

Introducing the FLAN Family of Models:

The FLAN (fine-tuned language net) family of models exemplifies the success of multitask instruction fine-tuning. Models such as FLAN-T5, derived from the T5 foundation model, and FLAN-PALM, based on the palm foundation model, have been trained on a staggering 473 datasets across 146 task categories.

Customizing FLAN-T5 for Domain-Specific Dialogue Summarization

While FLAN-T5 demonstrates competence across multiple tasks, specific use cases may demand further improvement. Suppose you are a data scientist developing an application to support a customer service team

Padding / Truncation

Padding and truncation are preprocessing techniques used in transformers to ensure that all **input sequences have the same length.**

Padding refers to the process of adding extra tokens (usually a special token such as `[PAD]`) to the end of short sequences so that they all have the same length. This is done so that the model can process all the sequences in a batch simultaneously. The padded tokens do not carry any semantic meaning and are just used to fill up the extra space in the shorter sequences.

Truncation, on the other hand, refers to the process of cutting off the end of longer sequences so that they are all the same length. This is done to ensure that the model is not overwhelmed by very long sequences and to reduce the computational overhead of processing large sequences.

Sequence 1: "The cat sat on the mat"

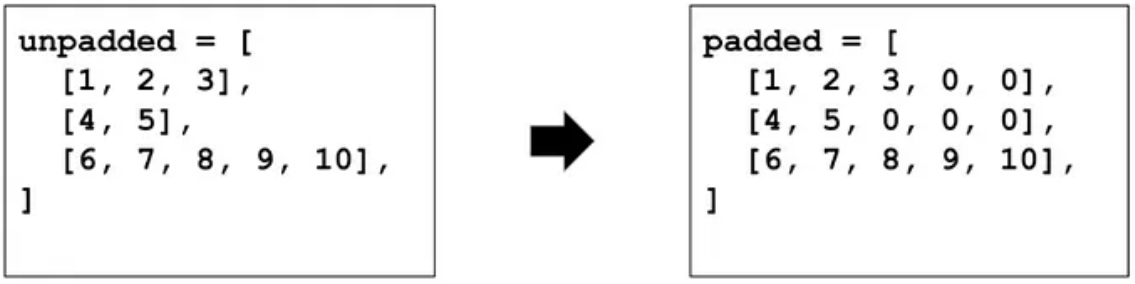
Sequence 2: "The dog chased the cat"

Sequence 3: "The mouse ran away from the cat and the dog"

Sequence 1: "The cat sat on the mat [PAD] [PAD] [PAD] [PAD]"

Sequence 2: "The dog chased the cat [PAD] [PAD] [PAD] [PAD] [PAD]"

Sequence 3: "The mouse ran away from the cat and the dog"



Evaluating Fine-Tuned Models:

Model Evaluation Metrics:

LLM evaluation challenges

Accuracy = Correct prediction / Actual predictions- for small model these values are deterministic

For LLM it is nondeterministic and evaluation is much more challenging hence, we use automated ways to evaluate these models using Rouge or BLEU Score with

ROUGE — Recall Oriented Under study Gesting of Evaluation : To assess quality of automatically generated summaries to that of human generated summaries

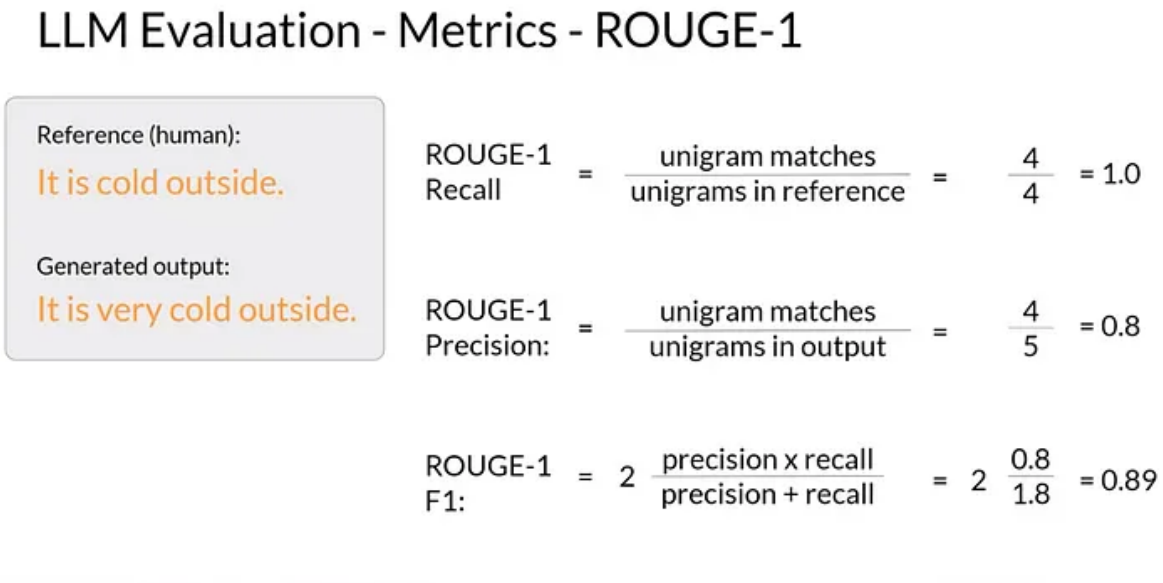
BLEU Score — **Bilignual Evalution Understudy** to quality of machine translated text to compare with human-generated translation. It is a French word for BLEU

Recall, Precision , F1 Score

using Unigrams, Bigrams or n-grams

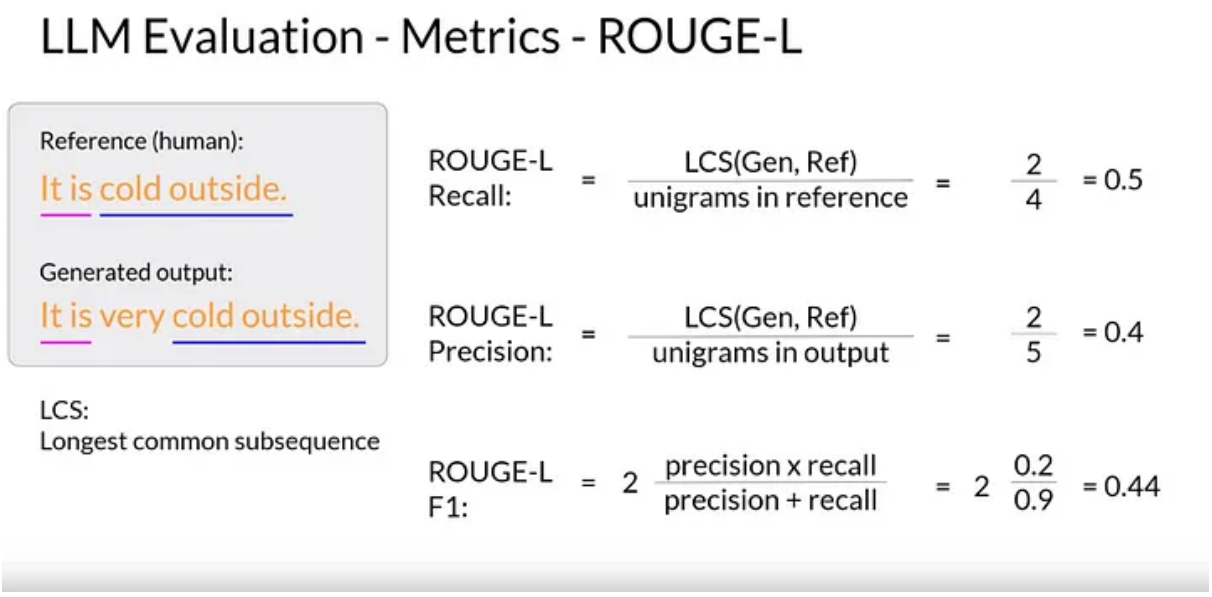
ROUGE-L — Longest Common Sequence (LCS)

ROUGE scores for different tasks



ROUGE scores alone may not capture the complete context and ordering of words, leading to potentially deceptive results. To overcome this **limitation, the**

ROUGE-L score calculates the longest common subsequence between the reference and generated outputs, giving a more comprehensive evaluation. It takes into account the ordering of words and provides a more accurate assessment.



BLEU: Bilingual Evaluation Understudy

BLEU, originally designed for evaluating machine-translated text, measures the quality of translations by comparing n-gram matches between the machine-generated translation and the reference translation. BLEU scores are calculated for a range of n-gram sizes, and the average precision across these sizes is used to determine the BLEU score. It provides a measure of how closely the generated output matches the reference translation.

Bleu Score

Finally, to calculate the Bleu Score, we multiply the Brevity Penalty with the Geometric Average of the Precision Scores.

$$Bleu\ (N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

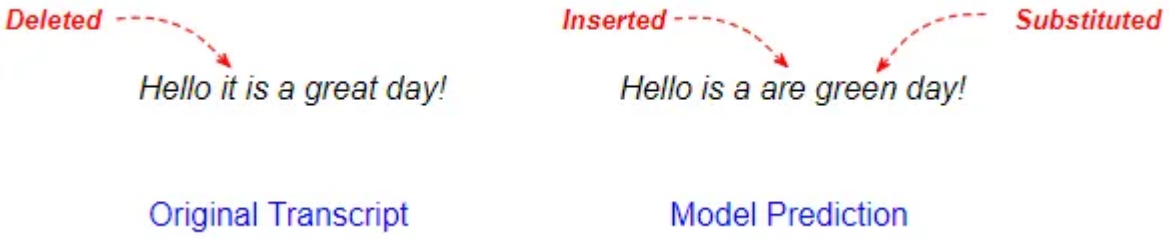
Bleu Score can be computed for different values of N. Typically, we use N = 4.

- BLEU-1 uses the unigram Precision score
- BLEU-2 uses the geometric average of unigram and bigram precision
- BLEU-3 uses the geometric average of unigram, bigram, and trigram precision

Strengths of Bleu Score

The reason that Bleu Score is so popular is that it has several strengths:

- It is quick to calculate and easy to understand.
- It corresponds with the way a human would evaluate the same text.
- Importantly, it is language-independent making it straightforward to apply to your NLP models.
- It can be used when you have more than one ground truth sentence.
- It is used very widely, which makes it easier to compare your results with other work



ROUGE and BLEU metrics serve as valuable tools for evaluating the performance of large language models in tasks like summarization and translation. These metrics provide an automated and structured way to measure the quality and similarity of generated outputs compared to human

references. However, for a more comprehensive evaluation, it is essential to consider task-specific evaluation benchmarks developed by researchers in the field.

Benchmarking Language Models :

1.
GLUE and SuperGLUE: GLUE (General Language Understanding Evaluation) introduced in 2018, comprises diverse natural language tasks like sentiment analysis and question-answering. SuperGLUE, introduced in 2019, addresses GLUE's limitations and features more challenging tasks, including multi-sentence reasoning and reading comprehension. Both benchmarks provide leaderboards to compare model performance and track progress.

GLUE — used in evaluation of following tasks —

- Sentiment Analysis
- Question Answering

SuperGLUE —

- Multi sentence reasoning
- Teading comprehension

The Holistic Evaluation of Language Models (HELM): HELM framework aims to enhance model transparency and guide performance assessment. It employs a

multimetric approach, measuring seven metrics across 16 core scenarios, beyond basic accuracy measures. HELM also includes metrics for fairness, bias, and toxicity, crucial as LLMs become more capable of human-like generation and potential harm.

MMLU- Massive Multitask Language Understanding

— Specifically for Modern LLMs

- extensive world knowledge
- problem-solving ability way beyond basic language understanding.