Lightricks

Fine-Tuning Large Language Models (LLMs)

Oren Sultan – AI / NLP Researcher

Computer Science PhD candidate @HebrewU

Researcher – Student @Lightricks (DS Group, LTX Studio)

Lightricks

Agenda

Session 1 (23.9.24)

- **Fine-tuning overview** 📖
- **Coffee break** ☕
- **Fine-tuning code example** 💻

Session 2 (30.9.24)

- Real use-case from Lightricks
  - Deployed in VideoLeap
  - Resulting in a research paper titled: “Visual Editing with LLM-based Tool Chaining: An Efficient Distillation Approach for Real-Time Applications”

Lightricks

What You'll Learn Today

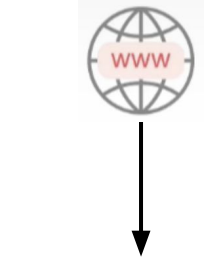
- **Fine-tuning overview**
  - What is Fine-tuning?
  - When to use Prompt Engineering vs. RAG vs. Fine-tuning?
  - How to Fine-tune an LLM?

- **Fine-tuning code example**
  - How to implement Supervised Fine-tuning (SFT) in code?
  - How to evaluate an LLM?

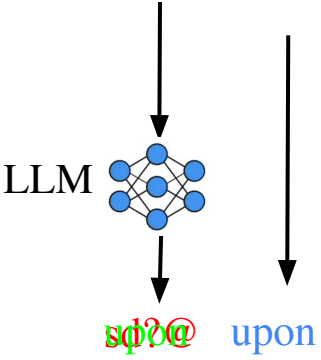
Lightricks

Part I: Fine-Tuning Overview 📖

Pre-training

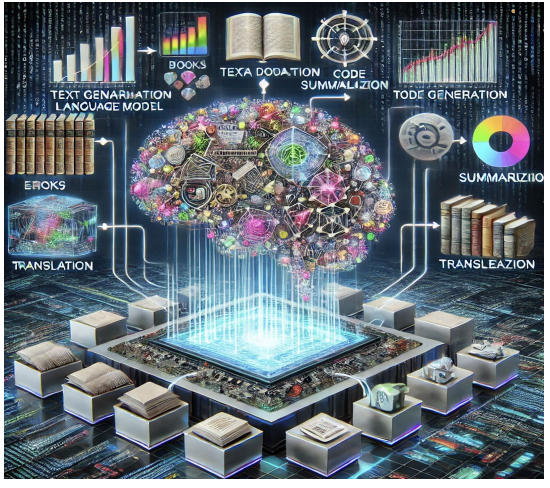


Once upon a midnight dreary while I pondered.



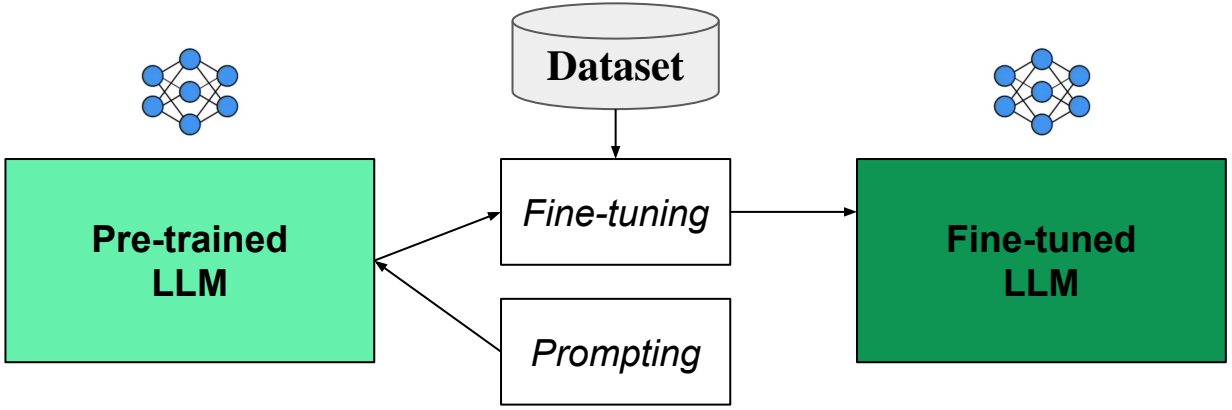
- Model at the start
  - Zero knowledge about the world
  - Can't form English words
- **Task:** Next token prediction
- **Data:** Giant corpus of text data from the web: "unlabeled"
- **Training:** Self-supervised learning

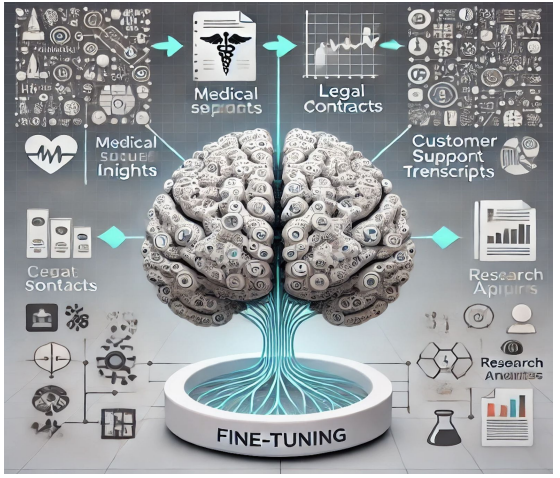
- After training
  - Learns language
  - Learns Knowledge



What is Fine-tuning?

- **Fine-tuning** is taking a **pre-trained** (general purpose) model and train some of its weights.
  - **A general-purpose base model → specialized model** for a **particular use case**.
- **Fine-tuning vs. prompt engineering:**
  - Gets the model to learn the data (**adjusting model's weights**), rather than just get access to it.
  - Let you put more data into the model than what fits in the prompt.
- **Fine-tuning vs. pre-training:**
  - Requires significantly less data and computational resources.





Base model vs. Fine-tuned model Completions – Example I

Base model: GPT-3

Prompt: How to fine-tune a model?

Completion:
How can I control the complexity of a model?
How do I know when my model is done?
How do I test a model?
…

Fine-tuned model: GPT-3.5-turbo (ChatGPT)

Prompt: How to fine-tune a model?

Completion:
Fine-tuning a model typically involves taking a pre-trained model and further training it on a new dataset specific to your task. Here are the general steps to fine-tune a model:
1. Choose a pre-trained model: Start by selecting a pre-trained model that is relevant to your task.…
2. Prepare your new dataset: Collect or create a dataset that is specific to your task…
3. Modify the top layers: Remove the final layers of the pre-trained model…
4. Fine-tune the model: Train the modified model on your new dataset…
5. Evaluate the model: Once training is complete, evaluate the fine-tuned model on the validation set to assess its performance…
…

Change behavior (instruction-based):
More desirable, practical and helpful completions!

Base model vs. Fine-tuned model Completions – Example II

Skin irritation
Redness
Itching

Base Model LLM

Probably acne.

Primary Care Physician (PCP)

Cardiologist

Dermatologist

Skin irritation
Redness
Itching

Fine-tuned LLM

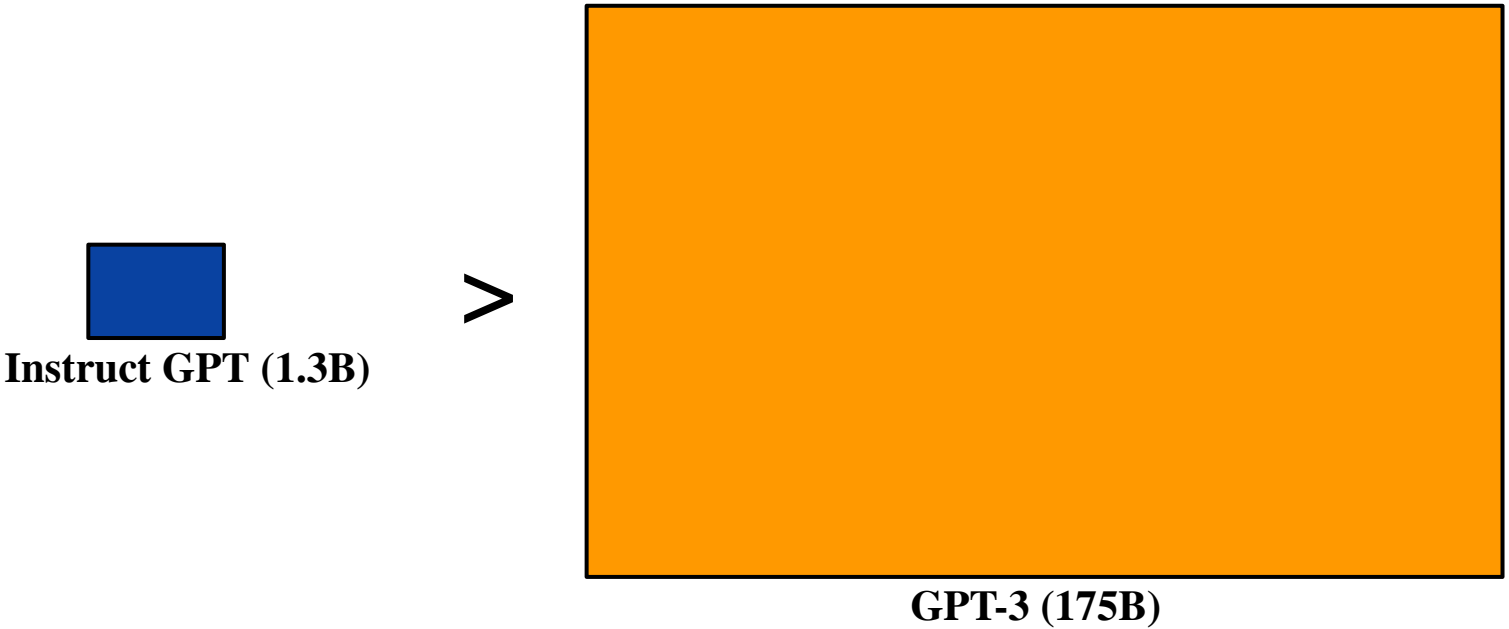
Dermatology data.

You have a mix of non-inflammatory comedonal acne and inflammatory papulopustular acne.

Change knowledge

Why to Fine-tune?

A smaller (fine-tuned) model can often outperform larger (more expensive) models on the set of tasks on which it was fine-tuned!



OpenAI paper: "Training language models to follow instructions with human feedback"

Prompt Engineering vs. Fine-tuning

Prompting

+ No data (very few) to get started
+ Smaller upfront cost
+ No technical knowledge needed

- Much less data fits
- Forgets data
- Hallucinations

Fine-tuning

- More high-quality data
- Upfront compute cost
- Needs some technical knowledge

+ Nearly unlimited data fits
+ Learn from your data (user's behavioral signals)
+ Less cost afterwards if smaller model

Generic, side projects, prototypes

Domain-specific, enterprise, production usage

Benefits of Fine-tuning your own LLM

1. **Performance**
   - Decrease Hallucinations
   - Increase consistency
2. **Privacy**
   - On premise
   - Prevent leakage

3. **Cost**
   - Lower cost per request
   - Increased transparency
   - Greater control
4. **Reliability**
   - Control uptime
   - Lower latency
   - Moderation

Retrieval-Augmented Generation (RAG) vs. Fine-tuning

RAG

+ Better in integrating new (dynamic) knowledge
+ No training, no retraining
+ More accurate responses – reducing hallucinations

- Slower: two-step process
- Lower performance in specific tasks with high-quality data
- Not suitable for changing the behavior of the responses

Fine-tuning

- Not suitable for learning evolving knowledge
- More expensive (high-quality data, re-training)
- Still prone to hallucinations

+ Optimizes LLM performance for specific tasks
+ Precise control over the training data
  - Adjusting tone / style of a language
  - Aligning with user preferences
  - Controlling output format
+ Improve efficiency (latency & cost)

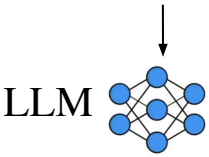
Generic, context-heavy tasks, dynamic knowledge injection

Improve performance for specific task

3 Ways to Fine-tune

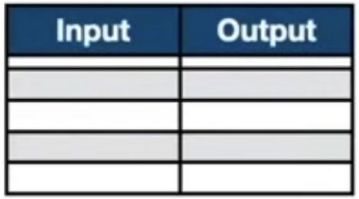
1) Self-supervised

Houston, we have a



problem.

2) Supervised



Input: Who was the first President of the USA?
Output: George Washington

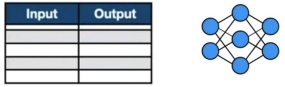
"""Please answer the following question.

Q: {Question}

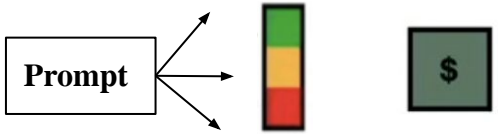
A: {Answer}"""

3) Reinforcement Learning

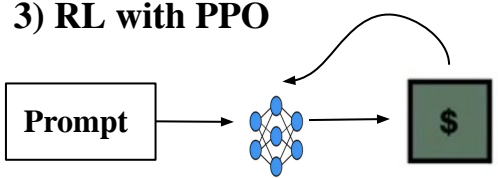
1) Supervised Fine-tuning



2) Train Reward model

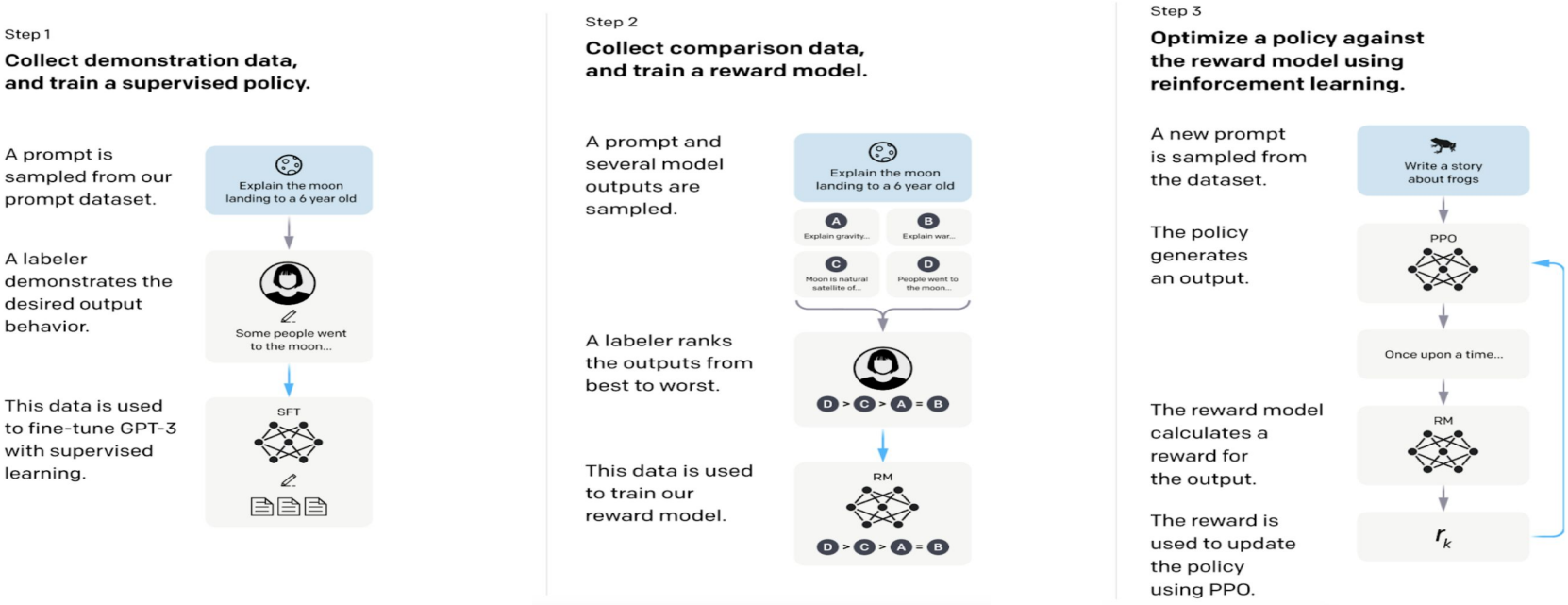


3) RL with PPO



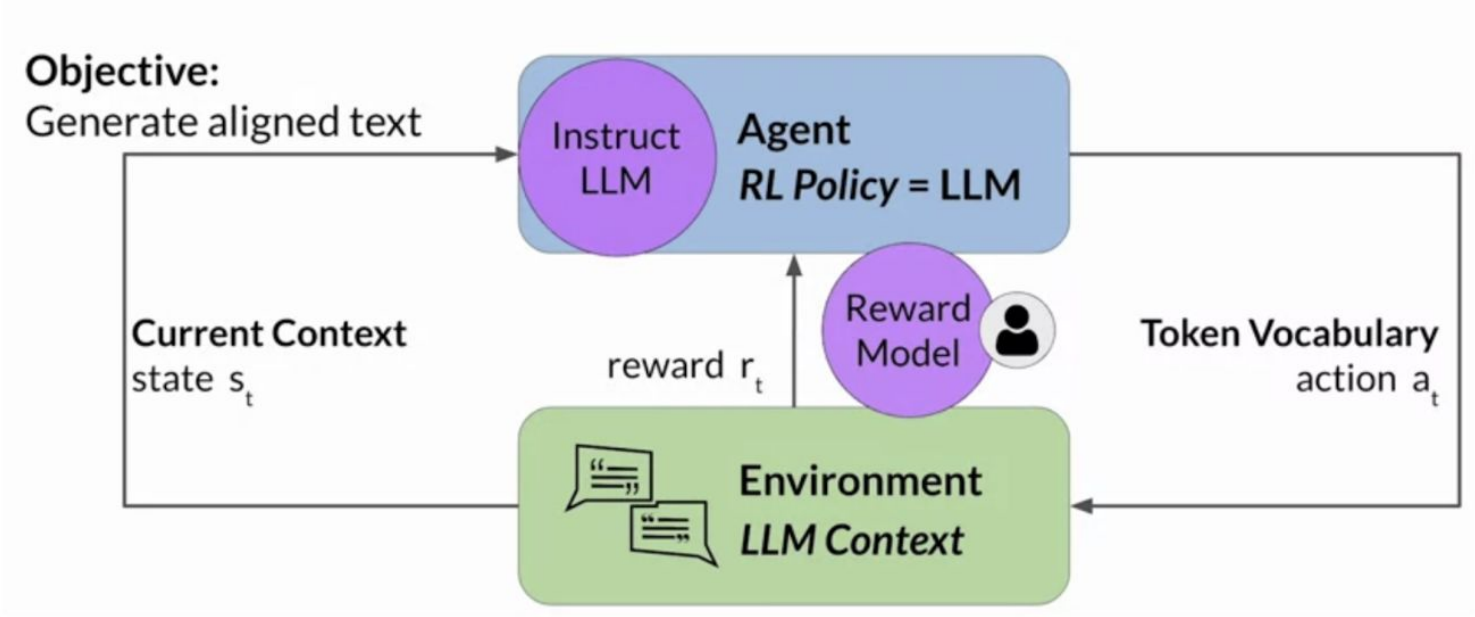
Reinforcement Learning from Human Feedback (RLHF)

Goal: to improve the alignment, performance, and safety of LLMs by incorporating human judgments into their training process.



OpenAI paper: "Training language models to follow instructions with human feedback"

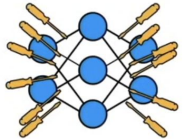
Reinforcement Learning from Human Feedback (RLHF)



3 Ways to Parameter Training

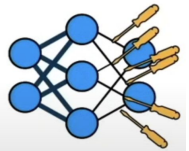
1 Retrain all parameters (Full)

- Computationally expensive
- Phenomenon of **"catastrophic forgetting"**



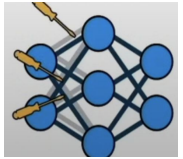
2 Transfer Learning (TL)

- Freeze most of the parameters
- Fine-tune the last few layers (head)



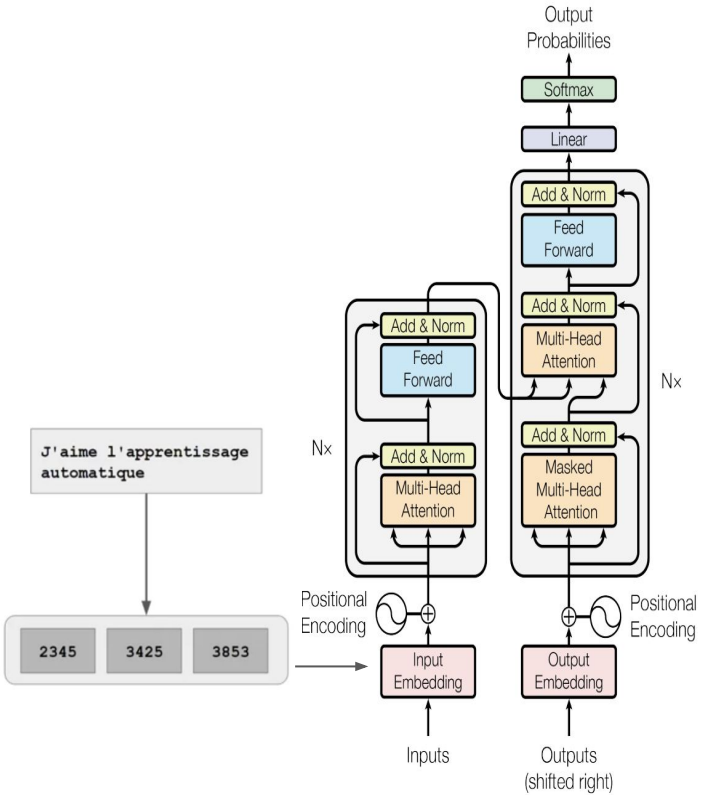
3 Parameter Efficient Fine-tuning (PEFT)

- Freeze all of the parameters
- Augmenting a relatively small number of trainable parameters.



LLMs are based on the Transformers Architecture

"Attention Is All You Need"
(Vaswani et al.)



<u>Encoder only models (Auto encoder):</u>

Training objective: Pre-trained using MLM (Mask Language Modeling). The training objective is to predict the masked token (denoising objective).

Models: BERT, ROBERTA. **Tasks**: sentiment analysis, NER.

<u>Decoder only (Auto-regressive):</u>

Training objective: to predict the next token.

They can see only previous tokens.

Models: GPT, LLaMA. **Tasks:** Text generation.

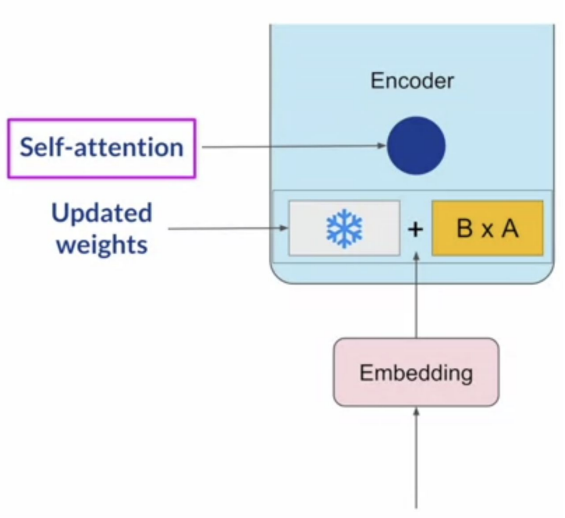
<u>Encoder - Decoder (sequence to sequence):</u>

Training objective: in T5 we train the encoder by **span corruption** (mask few adjacent tokens), then replaced by <x> token (which is not part of the dictionary). Then, the decoder's goal is to reconstruct it.

Models: T5, BART. **Tasks:** Translation, Summarization, and QA.

Parameter Efficient Fine-tuning (PEFT) – LoRA: Low Rank Adaptation of LLMs

Motivation: to make fine-tuning more feasible by significantly reducing the computational, storage, and resource requirements associated with traditional full fine-tuning.



Training:

1) Freeze the original LLM weights.
2) Inject 2 rank decomposition matrices: B, A.
3) Train the weights of the smaller matrices.

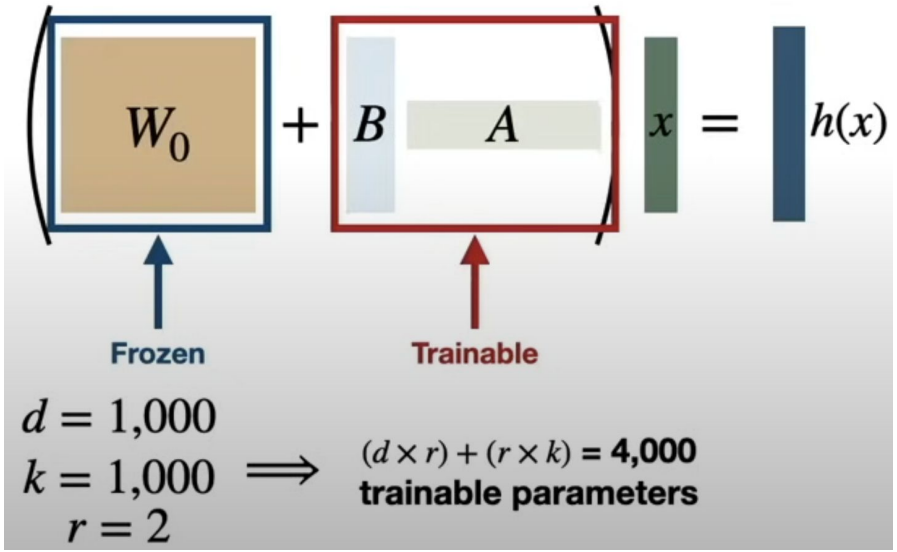
Inference:

1) Matrix multiply the low rank matrices

2) Add to original weights

LoRA Example

$$h(x) = W_0 x + \Delta W x \qquad \Delta W = BA$$
$$= W_0 x + BAx$$

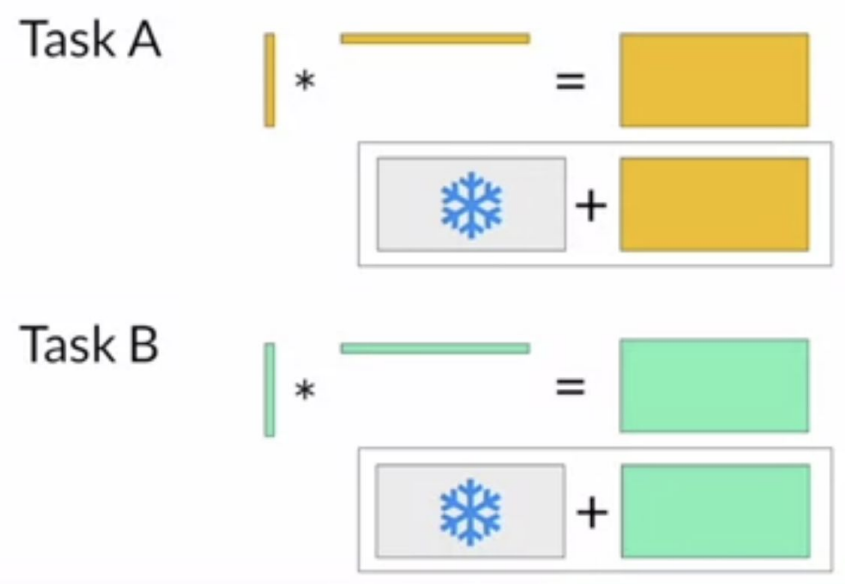


$$W_0, \Delta W \in R^{d \times k}$$
$$B \in R^{d \times r}$$
$$A \in R^{r \times k}$$
$$h(x) \in R^{d \times 1}$$

➔ If r is too low, we will lose information by deleting linearly independent rows.

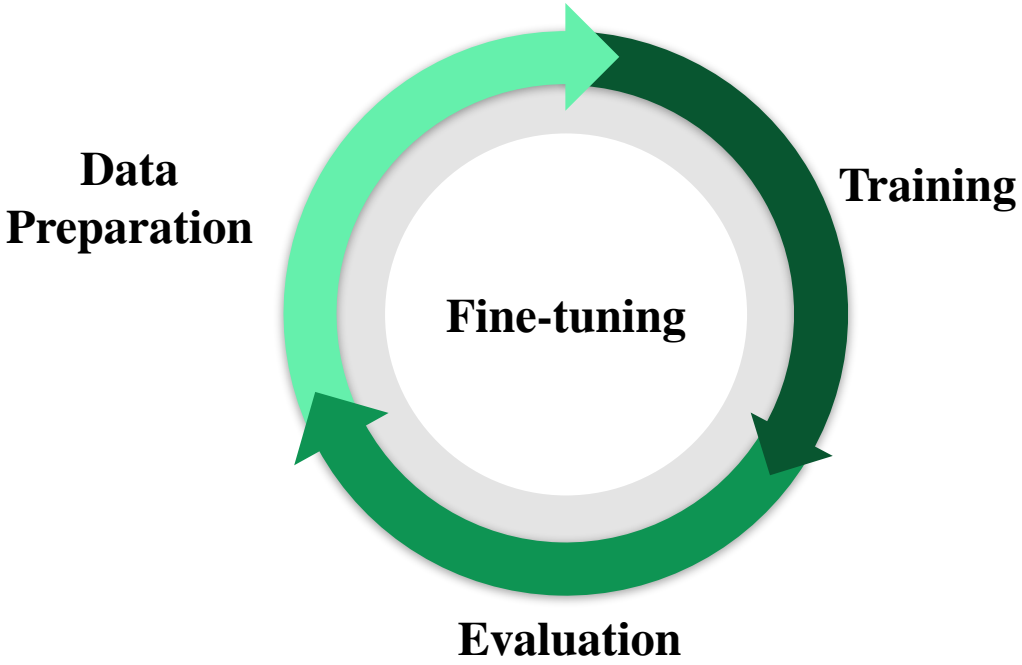
➔ If r is too high, we will have linearly dependent rows and will not significantly reduce parameters.

LoRA for different tasks



1) Train different rank decomposition matrices for different tasks
2) Update weights before inference

Fine-tuning Iterations Process



Lightricks

Part II: Fine-Tuning Code Example

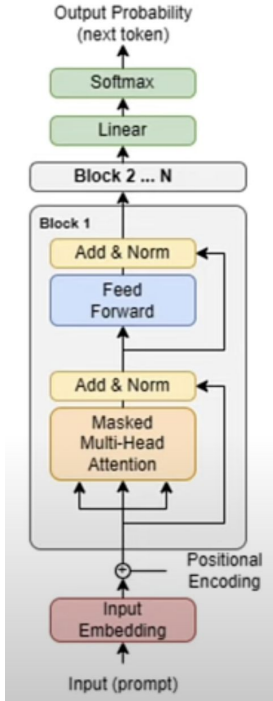
Model: LLaMA-2-7B-chat

Dataset: Patient/doctor interaction

Model: LLaMA-2

Meta

- **LLaMA-2** – pre-trained Language Models (7B, 13B, 70B) by MetaAI
  - Auto-regressive (decoder-only)
  - Stacking of decoder blocks
- **LLaMA-2-chat**
  - Fine-tuned LLaMA-2 as a chatbot (Q&A)
  - RLHF – align with human preferences, safe and helpful

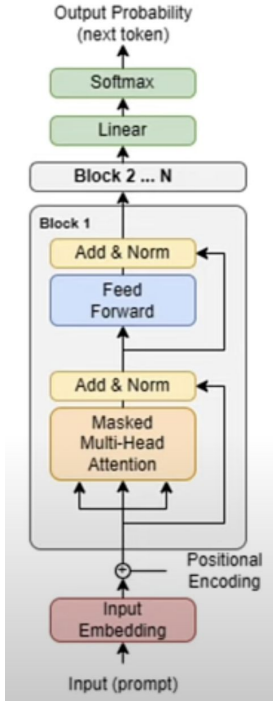


Base model	**Fine-tuned (chatbot)**
GPT-3	GPT-3.5-Turbo (ChatGPT)
LLaMA-2	LLaMA-2-chat

LLaMA-2 vs. GPT-3

Meta

- Open source
- Public data
- Smaller architecture, more training data
  - Comparable, sometimes better performance than GPT-3
- Fast inference



Preliminaries

Installation and Setup

```
!pip install -q accelerate==0.21.0
peft==0.4.0 bitsandbytes==0.40.2
transformers==4.31.0 trl==0.4.7
tensorboard huggingface_hub[cli] xformers
```

Hugging Face login

```
!huggingface-cli login
```

Importing Dependencies

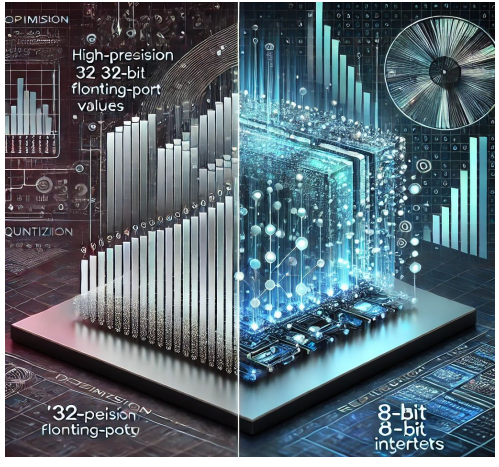
```
import os
import torch
import transformers
from datasets import load_dataset
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    HfArgumentParser,
    TrainingArguments,
    pipeline,
    logging
)
from peft import LoraConfig
from trl import SFTTrainer
```

Loading the model – Quantization

- **Motivation:** How can you leverage advanced models without being hindered by memory constraints?

- **Quantization:** is a technique where the **precision of the model's weights is reduced** to make the model smaller and more efficient, often at the cost of a slight reduction in accuracy.
  - A model with 7B parameters in fp64 (8 bytes) precision would need 56 GB of memory.

- **A Solution from HuggingFace: BitsAndBytes**
  - dynamically adjust the precision used when loading a model into memory, irrespective of the precision utilized during training.



Quantization Config

- **Inference**
  - Parameters

- **Training (3-4 times memory)**
  - Parameters
  - Gradients
  - Optimizer states

```
model_name = "meta-llama/Llama-2-7b-chat-hf"

device_map = {"": 0}

use_4bit = True

bnb_4bit_compute_dtype = "float16"

bnb_4bit_quant_type = "nf4"
```

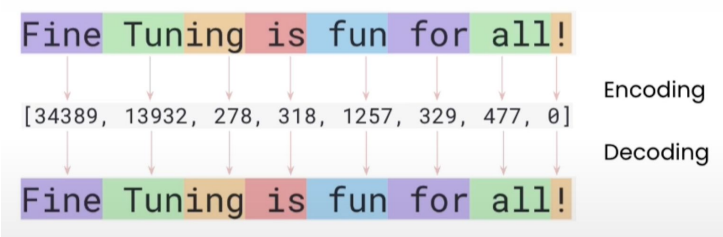
Quantization Config

```
compute_dtype = getattr(torch, bnb_4bit_compute_dtype)
bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant
)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map=device_map,
    quantization_config=bnb_config,
)
```

Tokenizer

- **Tokenization**: converts the text into a sequence of integers, each representing a specific word, subword, or character.
- Tokenizing the input data in the exact way the model was trained is crucial!




```
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
```

- Neural models, especially transformer-based ones, expect input data in fixed-sized batches.
- **Padding**: ensures consistent tensor dimensions across different inputs.

```
tokenizer.pad_token = "<PAD>"
tokenizer.padding_side = "right"
```

Inference

```
def display_response(prompt, generated_response):
    instruction_start = prompt.find("[INST]") + len("[INST]")
    instruction_end = prompt.find("[/INST]")
    instruction = prompt[instruction_start:instruction_end].strip()
    prefix = "As a medical doctor, respond to this patient query: Patient: "
    if instruction.startswith(prefix):
        instruction = instruction[len(prefix):].strip()

    response_text = generated_response[0]['generated_text']
    doctor_response_start = response_text.find("[/INST]") + len("[/INST]")
    doctor_response = response_text[doctor_response_start:].strip()

    print("Human:\n" + instruction + "\n" + "Assistance:\n" + doctor_response)
```

```
prompt = """<s>[INST] Hi, Are you there? How are you?  [/INST] """
generator = pipeline(task="text-generation", model=model, tokenizer=tokenizer)
display_response(prompt, generator(prompt, max_new_tokens=100))
```

Inference

Human:
Hi, Are you there? How are you?

Assistance:
Hello! I'm just an AI, I don't have feelings or emotions, so I can't experience emotions like humans do. I'm here to help answer your questions and provide information to the best of my ability.
How can I assist you today?

Data Preparation

Dataset source.

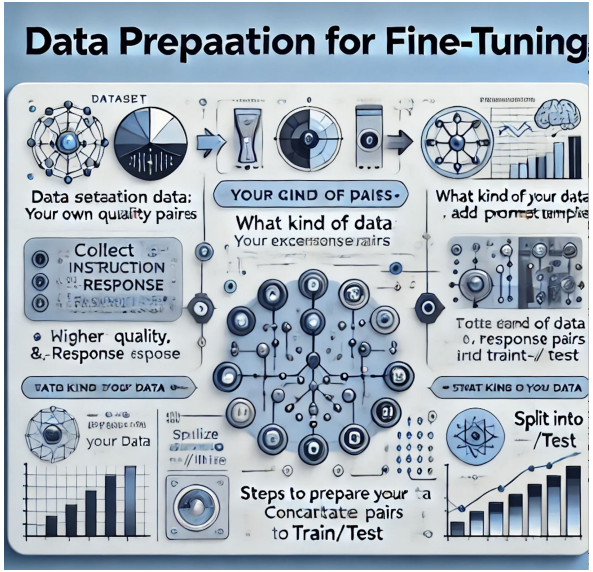
- Your own dataset
- Existing dataset (e.g., from Hugging Face)

What kind of data.

- Higher quality
- Diversity
- Real
- More

Steps to prepare your data.

- Collect instruction-response pairs
- Concatenate pairs (add prompt template)
- Tokenize: pad, truncate
- Split into train/test



Loading and Preprocessing the Dataset

Dataset Formatting: pairs of prompts and expected outputs.

```
<s>[INST] {user_message} [/INST] {response}
```

```
def template_dataset(sample):
    cleaned_response = sample['Doctor'].replace('<start>', '').replace('<end>', '').strip()
    instruction = f"<s>[INST] As a medical doctor, respond to this patient query: Patient: {sample['Patient']} [/INST]"
    response = f"Doctor: {cleaned_response}"
    sample["text"] = instruction + response + tokenizer.eos_token
    return sample

dataset_name = 'sid6i7/patient-doctor'
dataset = load_dataset(dataset_name, split="train")
dataset_sample = 3000
if dataset_sample > 0:
  dataset_shuffled = dataset.shuffle(seed=1234)
  dataset = dataset_shuffled.select(range(dataset_sample))

dataset = dataset.map(template_dataset, remove_columns=[f for f in dataset.features if not f == 'text'])
new_model = 'Llama-7b-medical-assistance'
```

Example from the Dataset

Patient:

My 5 year old daughter is 5 days post tonsillectomy. After the surgery she seemed to have a lot of nasal mucus and developed a cough that is very wet. I can hear the mucus in her chest...
Is this normal after a tonsillectomy?

Assistance:

Hello there! As a medical doctor, I understand your concern about your 5-year-old daughter's post-tonsillectomy symptoms. It's common for children to experience nasal mucus and coughing after a tonsillectomy, and in most cases, these symptoms are a normal part of the recovery process.
After a tonsillectomy, the body needs time to heal and remove any remaining tissue or debris from the tonsils. This can cause swelling and inflammation in the nasal passages, leading to increased mucus production and coughing. The mucus in the chest is likely a result of the nasal mucus draining down the back of the throat and into the lungs.

Training – Supervised Fine-tuning (SFT)

LoRA Configuration

```
lora_r = 64
lora_alpha = 64
lora_dropout = 0.1
peft_config = LoraConfig(r=lora_r, lora_alpha=lora_alpha,
                         lora_dropout=lora_dropout,
    inference_mode=False, bias="none", task_type="CAUSAL_LM")
```

Save training results

```
output_dir = "./results"
final_checkpoint_dir = os.path.join(output_dir, "final_checkpoint")
```

lora_alpha

$$W' = W + \frac{\alpha}{r} \cdot (A \cdot B)$$

Training – Supervised Fine-tuning (SFT)

Training Arguments

```
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=1, max_steps=-1,
    fp16=True, bf16=False,
    per_device_train_batch_size=4, gradient_accumulation_steps=1,
    max_grad_norm = 0.3,
    learning_rate=4e-5, lr_scheduler_type = "constant"
    optim="paged_adamw_32bit", weight_decay = 0.001,
    group_by_length = True,
    save_steps=50, logging_steps=10,
    report_to="tensorboard")


max_seq_length = None
packing = False
```

Training – Supervised Fine-tuning (SFT)

```
trainer = SFTTrainer(model=model, train_dataset=dataset,
                     dataset_text_field="text", peft_config=peft_config,
                     tokenizer=tokenizer, args=training_arguments,
                     max_seq_length=max_seq_length, packing=packing)

resume_checkpoint = None
transformers.logging.set_verbosity_info()

trainer.train(resume_checkpoint)
```

**** Running training *****
Num examples = **3,000**
Num Epochs = 1
Instantaneous batch size per device = 4
Total train batch size (w. parallel, distributed & accumulation) = **4**
Gradient Accumulation steps = 1
Total optimization steps = **750**
Number of trainable parameters = **33,554,432**
[750/750 49:59, Epoch 1/1]

Saving, Loading and Exporting the Model

Saving

```
trainer.save_model(final_checkpoint_dir)
```

Saving model checkpoint to ./results/results_sep23/final_checkpoint
tokenizer config file saved in ./results/results_sep23/final_checkpoint/tokenizer_config.json
Special tokens file saved in ./results/results_sep23/final_checkpoint/special_tokens_map.json

Loading

```
output_dir = "./results/results-sep23"
final_checkpoint_dir = os.path.join(output_dir, "final_checkpoint")
device_map = {"": 0}
reloaded_model = AutoPeftModelForCausalLM.from_pretrained(final_checkpoint_dir,
    low_cpu_mem_usage=True, return_dict=True, torch_dtype=torch.float16,
                                                          device_map=device_map)
reloaded_tokenizer = AutoTokenizer.from_pretrained(final_checkpoint_dir)
merged_model = reloaded_model.merge_and_unload()
reloaded_generator = pipeline(task="text-generation", model=merged_model,
                              tokenizer=reloaded_tokenizer)
```

Exporting

```
hf_repo = "llama-2-7b-chat-hf-instruct-medical-assistance"
merged_model.push_to_hub(hf_repo, max_shard_size="4GB")
```

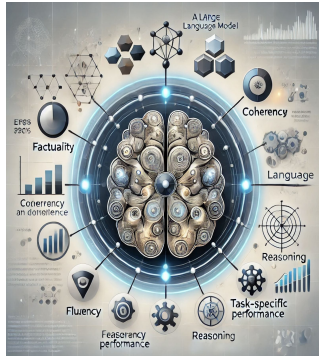
Evaluation

Evaluating generative models is a **challenging** task.

- It is not necessarily exact match anymore (e.g., in classification – sentiment analysis, NER, etc.).
- There is no one correct answer, there are endless possibilities to generate a good answer.

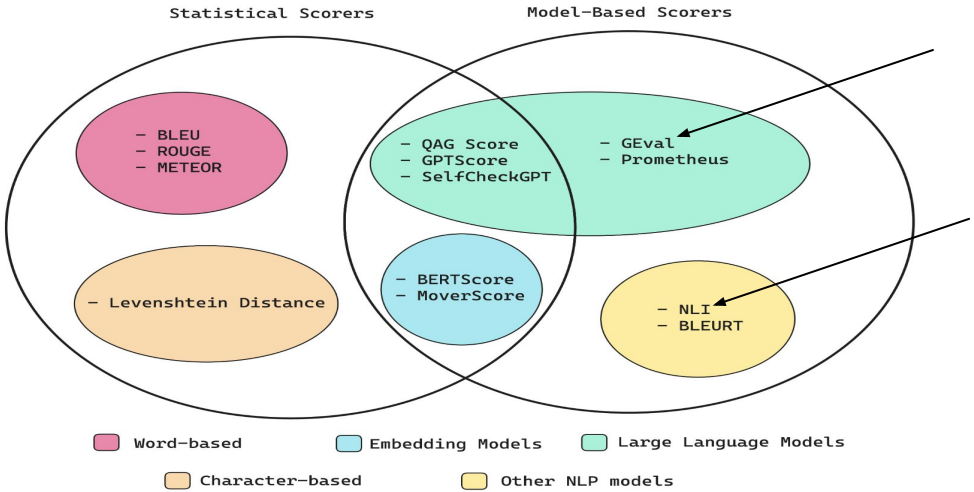
Important evaluation aspects before LLM deployment.

1	**Answer Relevancy**	Determines whether an LLM output is able to address the given input in an informative and concise manner.
2	**Correctness**	Determines whether an LLM output is factually correct based on some ground truth.
3	**Hallucination**	Determines whether an LLM output contains fake or made-up information.
4	**Responsibility**	Determines whether an LLM output contains harmful and offensive content.
5	**Task-specific Metrics**	Tailored criteria based on the use case.

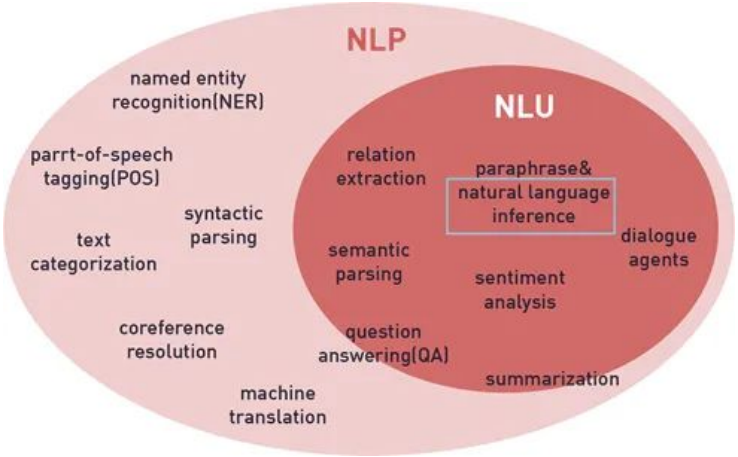


Types of Offline Evaluation Automatic Metrics/Scorers

- **Human Evaluation** – reliable, accurate, but expensive
- **Automatic Evaluation**
  - **Traditional Statistical scorers** – reliable, less accurate
  - **Model-based scorers** – less reliable (probabilistic), more accurate (take semantic into account)
- This shouldn't be a surprise but, scorers that are not LLM-based perform worse than LLM-Evals.



Natural Language Inference (NLI)



The task of determining whether the given “hypothesis” and “premise” logically:
1) follow – **entailment**
2) unfollow – **contradiction**
3) undetermined – **neutral**

Example:
Premise: "Female players are playing the game"
Hypothesis: "The game is played by only males" → **contradiction**

NLI Model for Evaluating Correctness and Hallucinations

- Choosing a fine-tuned NLI model
- For every sample from our test-set (for example in the patient-doctor interaction dataset)
  - **Premise**: the ground-truth (the response of a doctor)
  - **Hypothesis**: the generated response of our LLM for the input patient text

premise:
"...It's common for children to experience nasal mucus and coughing after a tonsillectomy, and in most cases, these symptoms are a normal part of the recovery process…"

hypothesis:
"...Your daughter's symptoms, such as nasal mucus and coughing, are not normal after a tonsillectomy. These symptoms could indicate a serious complication, and you should seek immediate medical attention…"

contradiction

G Eval – Using LLM as an Evaluator for a Task-Specific Metric

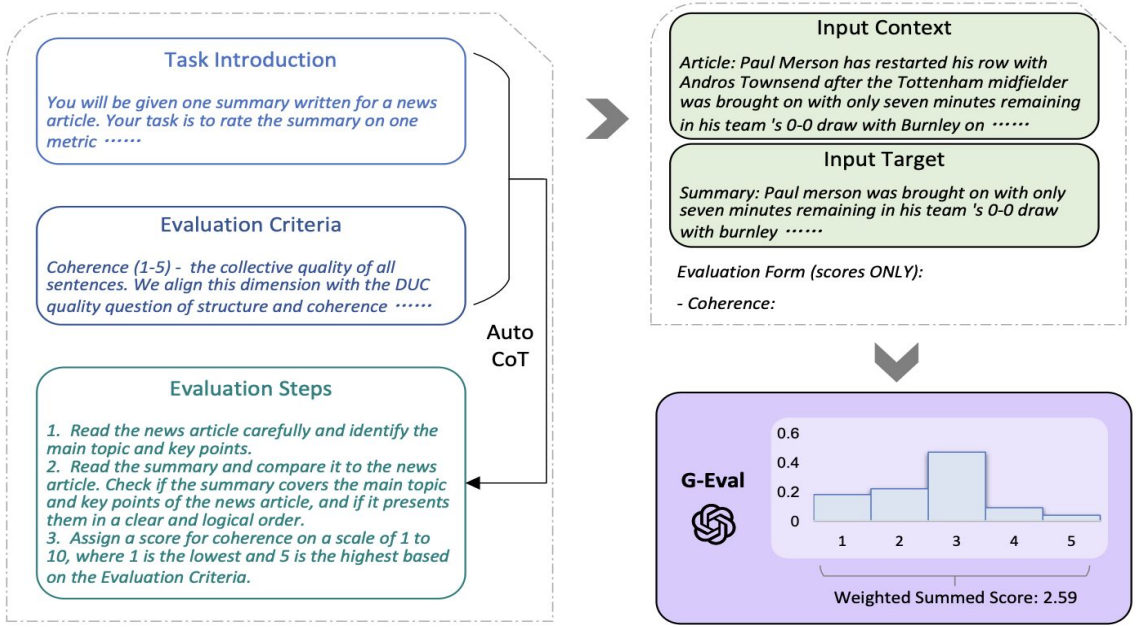


Figure 1: The overall framework of G-EVAL. We first input Task Introduction and Evaluation Criteria to the LLM, and ask it to generate a CoT of detailed Evaluation Steps. Then we use the prompt along with the generated CoT to evaluate the NLG outputs in a form-filling paradigm. Finally, we use the probability-weighted summation of the output scores as the final score.

Microsoft Paper: "NLG Evaluation using GPT-4 with Better Human Alignment"

Lightricks

Questions?

Session 1 (23.9.24)

- **Fine-tuning overview** 📖
- **Coffee break** ☕
- **Fine-tuning code example** 💻

Session 2 (30.9.24)

- Real use-case from Lightricks
  - Deployed in VideoLeap
  - Resulting in a research paper titled: “Visual Editing with LLM-based Tool Chaining: An Efficient Distillation Approach for Real-Time Applications”