LLM Assignment-2 :: *Debnath Kundu (MT22026)*

Part 1: LLM Exploration Loading and Inference (20%)

1. Choose publicly available LLMs of different sizes.
   NousResearch/Llama-2-7b-chat-hf · Hugging Face
   CallComply/Starling-LM-11B-alpha · Hugging Face
   microsoft/phi-2 · Hugging Face

2. Explore Generation with LLMs (https://huggingface.co/docs/transformers/en/llm_tutorial) to load these LLMs.

```
# setting the model generation configuration
generation_config = GenerationConfig(
   do_sample = True,
   temperature = 0.8,
   repetition_penalty = 1.5,
   max_new_tokens = 256)
```

3. Design and implement functions to perform inference on these LLMs using different prompts.

```
## Prompt 1: Simple Q/A
query = "What do you know about the sun?"

## Prompt 2: Multi-Step Reasoning Question
query = "Prepare a plan how to make a cake, considering budget limitations, ingredient preference. Generate a step-by-step answer."

## Prompt 3: Real-Time Interaction Simulation
query = "Imagine you're a senior developer helping a junior developer fix a bug in a web app. Explain how to find and fix the problem step by step, like checking the code and trying out different solutions."
```

4. Evaluate the inference time for each LLM on a fixed prompt and discuss the trade-off between model size and speed.

	Prompt-1	Prompt-2	Prompt-3	Average
Llama 2 (7B)	6.9	8.04	14.6	9.84
Starling LM (11B)	19.03	19.28	20.19	19.5
Microsoft Phi (2.7B)	0.49	10.66	6.63	5.92

Prompt Engineering (20%)

5. Select above public dataset relevant to a specific task (e.g., question answering, summarization). super_glue · Datasets at Hugging Face [axb]
6. Design different prompts [https://huggingface.co/docs/transformers/en/tasks/prompting] for the chosen LLMs to perform the chosen task on the dataset.

```python
def check_implication_zero_shot(sentence1, sentence2):
    prompt = f"""
    Does the hypothesis entail the premise? Please return answer as 0 for entailment, 1 for no entailment.

    Hypothesis: {sentence1}
    Premise: {sentence2}

    The answer is :
"""

    # # Assuming model.predict returns the model's response
    # response = model.predict(prompt)
    # return response
    return prompt

# Example usage:
# model = load_your_model()  # Load your NLI-capable model
# result = check_implication("All birds can fly.", "Penguins can fly.", model)
# print("Implication: ", result)
print(check_implication_zero_shot("All birds can fly.", "Penguins can fly."))
```

Success

✓ 0.0s

```
Does the hypothesis entail the premise? Please return answer as 0 for entailment, 1 for no entailment.

Hypothesis: All birds can fly.
Premise: Penguins can fly.

The answer is :
```

```python
def check_implication_few_shot(sentence1, sentence2):
    # Example data for few-shot learning
    examples = [
        ("Writing Java is not too different from programming with handcuffs.", "Writing Java
        ("Some birds can fly.", "All birds can fly.", "not_entailment"),
        ("She sells seashells.", "She sells seashells on the seashore.", "not_entailment")
    ]

    # Constructing the prompt
    prompt = ""
    for ex in examples:
        prompt += f"Hypothesis: {ex[0]}\nPremise: {ex[1]}\nAnswer: {ex[2]}\n\n"

    # Add the new sentences to be evaluated
    prompt += f"Hypothesis: {sentence1}\nPremise: {sentence2}\nThe answer is :"

    return prompt

print(check_implication_few_shot("All birds can fly.", "Penguins can fly."))
```

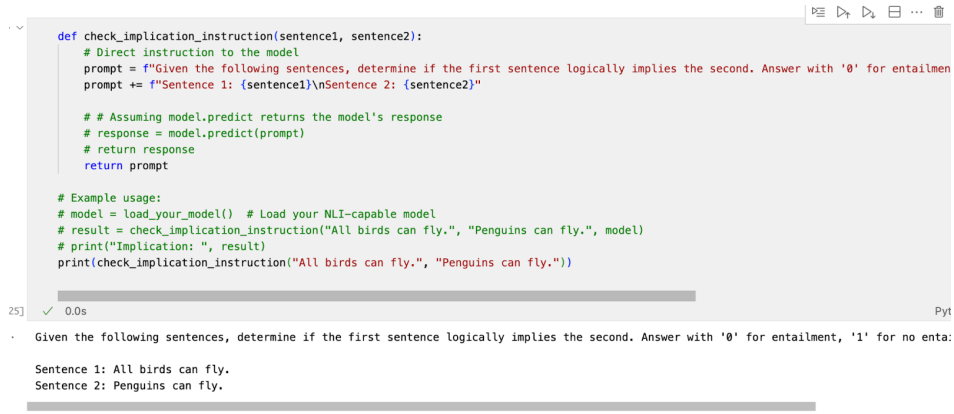
✓ 0.0s

```
Hypothesis: Writing Java is not too different from programming with handcuffs.
Premise: Writing Java is similar to programming with handcuffs.
Answer: entailment

Hypothesis: Some birds can fly.
Premise: All birds can fly.
Answer: not_entailment

Hypothesis: She sells seashells.
Premise: She sells seashells on the seashore.
Answer: not_entailment

Hypothesis: All birds can fly.
Premise: Penguins can fly.
The answer is :
```



```
def check_implication_instruction(sentence1, sentence2):
    # Direct instruction to the model
    prompt = f"Given the following sentences, determine if the first sentence logically implies the second. Answer with '0' for entailmen
    prompt += f"Sentence 1: {sentence1}\nSentence 2: {sentence2}"

    # # Assuming model.predict returns the model's response
    # response = model.predict(prompt)
    # return response
    return prompt

# Example usage:
# model = load_your_model()  # Load your NLI-capable model
# result = check_implication_instruction("All birds can fly.", "Penguins can fly.", model)
# print("Implication: ", result)
print(check_implication_instruction("All birds can fly.", "Penguins can fly."))
```

25] ✓ 0.0s Pyt

```
Given the following sentences, determine if the first sentence logically implies the second. Answer with '0' for entailment, '1' for no enta:

Sentence 1: All birds can fly.
Sentence 2: Penguins can fly.
```

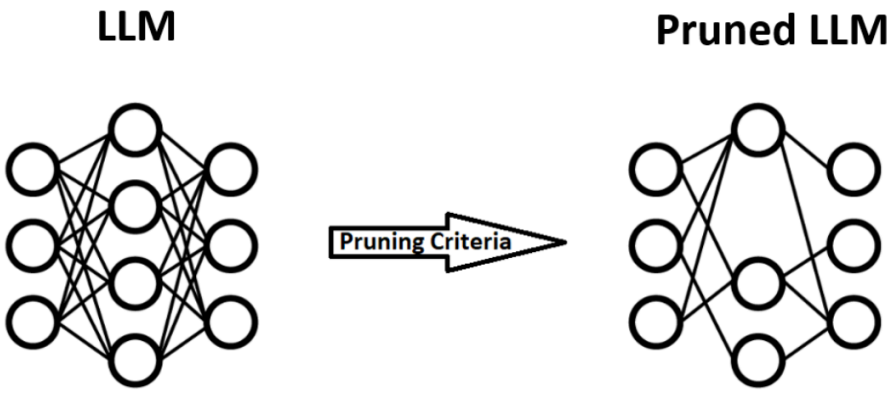
7. Experiment with different prompt variations (e.g., adding instructions, examples) and compare the quality of the outputs from the LLMs.

	Zero-Shot	Few-Shot	Instruction Prompt	Average
Llama 2 (7B)	0.42(42.6s)	0.58(20s)	0.56(70.2s)	0.52(s)
Starling LM (11B)	0.50(637.5s)	0.54(855.9s)	0.64(649.1s)	0.5(s)
Microsoft Phi (2.7B)	0.50(328.7s)	0.54(137.4s)	0.46(270.2s)	0.5(s)

Part 2: LLM Optimization and Fine-tuning, Optimization Techniques (15%)

8. Research and explain LLM optimisation techniques [https://huggingface.co/docs/transformers/en/peft]: Pruning, Erasing, Fission, and Transposition (PEFT) and Low Rank Adaptation (LoRA).

Pruning: Pruning is a technique that aims to reduce the model size of LLM by removing the weights that contribute minimally to the output. This is based on the observation that not all parameters used in LLM are equally important for making predictions. By identifying and eliminating these low-impact parameters, pruning reduces the model's size and the number of computations required during inference, leading to faster and more efficient performance. The following diagram represents the pruned LLM after some weights have been removed.



With the benefit of reduced model size, increased efficiency, and overall good generalisation, the limitations can be performance trade-offs, irreversibility and inconsistency across tasks.

Erasing: Erasing refers to a technique aimed at improving model performance by selectively removing data or learned features that are less important or detrimental to the model's accuracy and efficiency. Unlike *pruning*, which focuses on reducing the number of parameters by eliminating weights in the neural network, erasing targets the data or activations directly, leading to a more focused and potentially less noisy model. Some techniques are:

- Data Erasing
- Feature Erasing
- Attention Masking

The benefits are reduced overfitting and flexible implementation; the limitations are the risk of losing important information, the potential for bias and dependency on domain knowledge.

Fission: It is an optimisation technique for LLMs, which involves splitting the model into smaller, more manageable components that can be trained or executed more efficiently. This technique is particularly useful when dealing with large models such as those in the GPT or BERT series, which may not fit entirely in the memory of a single machine or could benefit from parallel processing architectures. Some key aspects of fission are:

- Model Splitting
- Parallel Processing
- Specialised Optimization

- Scalability and Flexibility

Transposition (PEFT): Parameter Efficient Fine-Tuning (PEFT) is an instruction fine-tuning method that is more efficient than full fine-tuning. Full LLM fine-tuning demands substantial computational resources, posing a challenge for memory allocation. PEFT updates only a subset of parameters, effectively "freezing" the rest, and reducing trainable parameters. This makes memory requirements more manageable, preventing catastrophic forgetting. Unlike full fine-tuning, PEFT maintains the original LLM weights, preserving previously learned information. This proves advantageous for storage issues when fine-tuning for multiple tasks.

Various methods, such as Low-Rank Adaptation (LoRA) and QLoRA, are widely used and effective for achieving parameter-efficient fine-tuning. By optimising task scheduling, PEFT maximises the use of available computational resources, reducing idle times and improving overall system throughput. Effective task scheduling can also significantly decrease the time required to complete tasks.

The main advantage of PEFT is scalability and flexibility; the disadvantage is the potential for sub-optimal scheduling and complexity in implementation.

Low-Rank Adaptation (LoRA): LoRA (Low-Rank Adaptation) is a highly efficient method of LLM fine-tuning, which puts LLM development into the hands of smaller organisations and individual developers. LoRA makes it possible to run a specialised LLM model on a single machine, opening major opportunities for LLM development in the broader data science community.

LoRA modifies the fine-tuning process by freezing the original model weights and applying changes to a separate set of weights, which are then added to the original parameters. LoRA transforms the model parameters into a lower-rank dimension, reducing the number of parameters that need training, thus speeding up the process and lowering costs. This method is particularly useful in scenarios where multiple clients need fine-tuned models for different applications, as it allows for creating a set of weights for each specific use case without the need for separate models.

The main benefits of LORA are parameter efficiency and preservation of pre-trained weights. Limitations can be the dependence on base model quality & potential for underfitting.

9. Implement and Discuss the potential benefits and limitations of each technique for reducing LLM size and computational cost.

```
tokenizer.padding_side = "right" # Fix weird overflow

# Load LoRA configuration
peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM",
)

# Set training parameters
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch
    gradient_accumulation_steps=gradient_accumulation_
    optim=optim,
    save_steps=save_steps,
    logging_steps=logging_steps,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
```

```
################################################################################
# QLoRA parameters
################################################################################

# LoRA attention dimension
lora_r = 64

# Alpha parameter for LoRA scaling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1

################################################################################
# bitsandbytes parameters
################################################################################

# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = False
```

Adaptor Tuning (10%)

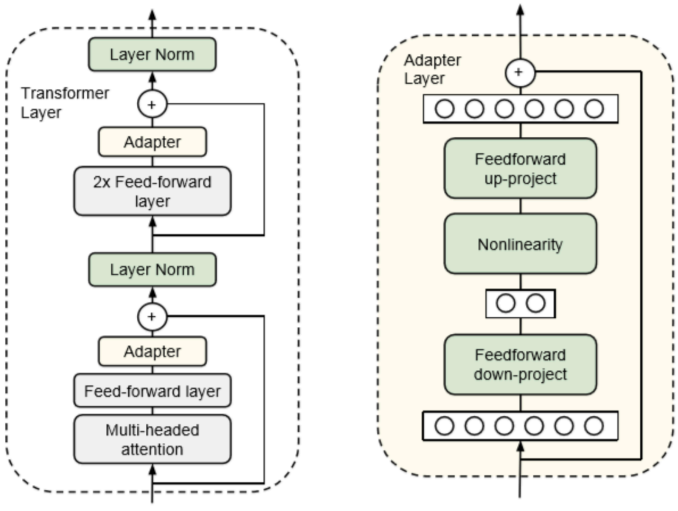
10. Explain the concept of Adaptor Tuning for LLMs.

Adaptor tuning is an efficient technique for fine-tuning large language models (LLMs) by inserting small trainable modules, known as *adaptors*, between the layers of a pre-trained model. This approach maintains the original model's parameters unchanged while only the adaptors are trained, significantly reducing the computational resources required. Adaptor tuning preserves the extensive knowledge embedded in the pre-trained model, minimising the risk of catastrophic forgetting and allowing rapid adaptation to specific tasks with minimal data. Despite its efficiency and flexibility, the technique may have limitations in fully capturing complex task-specific nuances due to the limited scope of adaptation and reliance on the underlying quality of the base model. Overall, adaptor tuning is a resource-efficient method that facilitates quick deployment and customisation of LLMs across varied applications.

11. Describe and perform how Adaptor Tuning can be used to improve LLM performance on specific tasks without requiring full fine-tuning.

The proposed adapter model adds new modules between layers of a pre-trained network called adapters. This means that parameters are copied over from pre-training (meaning they remain fixed), and only a few additional task-specific parameters are added for each new task, all without affecting previous ones. The innovation here is in the strategy that is used to design the adapter module to achieve parameter efficiency with one single model while not compromising performance. In fact, a simple model was compared with a fully fine-tuned BERT model on several text classification tasks. The findings show that only 3% of task-specific parameters are needed almost to match the results of the 100% task-specific parameters used by the fully fine-tuned model.

The traditional way of fine-tuning involves: 1) adding a new layer to fit the targets specified in the downstream task and 2) co-training the new layer with the original weights. In contrast, the adapter tuning strategy injects new layers (randomly initialised) into the original network. Parameter sharing between tasks is supported since the original network's parameters are frozen. The following diagram explains the process:



Part 3: Reinforcement Learning from Human Feedback (RLHF) (20%)

12. Explain the concept of LLM alignment and why it's important.

LLM alignment refers to the process of ensuring that the outputs of an LLM are in accordance with human intentions, values, and ethical standards. This involves tuning and training the models so that their responses not only provide useful and accurate information but also align with societal norms and individual user expectations. The importance of LLM alignment is multi-faceted, impacting both practical and ethical dimensions of technology use.

LLM alignment is required for several reasons including safety and reliability, ethical considerations, user trust and adoption and regulatory compliance.

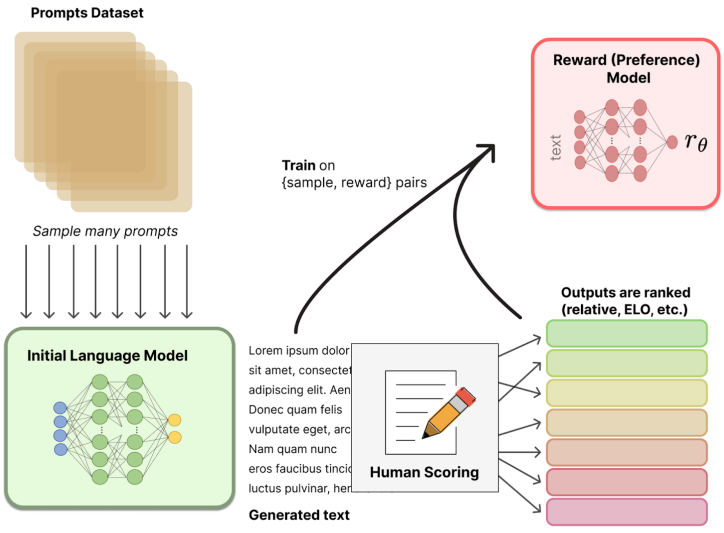
13. Discuss the potential risks of misaligned LLMs, such as generating harmful or misleading content.

Misaligned large language models (LLMs) pose significant risks, primarily due to their capacity to generate content that is harmful, misleading, or otherwise inappropriate. Some risks are:

- Generation of Harmful Content: *offensive or discriminatory language, hate speech and toxicity*
- Spreading Misinformation: *Legal liability, ethical dilemmas*
- Misuse and Malicious Use: *Manipulation and adversarial attacks, Weaponization of AI*

14. Research and explain the concept of RLHF for LLM alignment.

RLHF is an approach in artificial intelligence and machine learning that combines reinforcement learning techniques with human guidance to improve the learning process. It involves training an agent or model to make decisions and take action in an environment while receiving feedback from human experts. The input humans can be in the form of rewards, preferences, or demonstrations, which helps guide the model's learning process. RLHF enables the agent to adapt and learn from the expertise of humans, allowing for more efficient and effective learning in complex and dynamic environments.

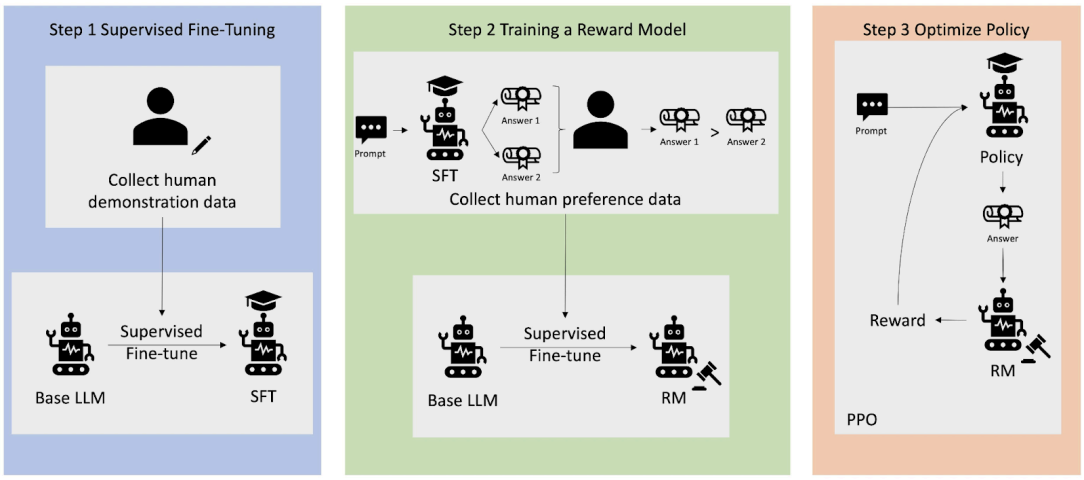


The following image shows an overview of the RLHF learning process:

15. Describe and implement how RLHF works by providing human feedback to guide the LLM towards desired outputs.

RLHF is performed in four core steps:

- *Pretraining a language model (LM):* The model is initially trained on a large dataset to understand a broad language. This is typically done using unsupervised or self-supervised learning methods where the model learns to predict parts of the text.
- *Gathering data:* Human feedback is collected in the form of demonstrations or corrections. For example, humans might demonstrate how to perform a task or correct the model's responses to better align with desired outcomes.
- *Reward Modeling:* A reward model is trained to predict the quality of the model's outputs based on human feedback. Essentially, this model learns to estimate how well a given response aligns with human preferences.
- *Policy Training:* Using the reward model as a guide, the policy (the main model) is fine-tuned through reinforcement learning. The model interacts with a training environment, receives synthetic rewards from the reward model, and updates its parameters to maximise these rewards.



16. Analyze the challenges associated with implementing RLHF, such as defining appropriate reward signals and gathering high-quality human feedback.

Some key challenges that can impact the effectiveness and efficiency of the training process in RLHF are:

- *Defining Appropriate Reward Signals:*
  - Subjectivity: Human feedback is inherently subjective.
  - Sparse Feedback: High-quality human feedback can be costly and time-consuming to gather, which may result in sparse feedback.
  - Reward Shaping: Designing reward functions that accurately capture human preferences without introducing unintended biases or behaviours can be difficult.

- *Gathering High-Quality Human Feedback:*
  - Consistency and Reliability: Ensuring that feedback is consistent across different evaluators and over time is challenging.
  - Scalability: Collecting feedback at scale while maintaining quality is a significant challenge. As the need for data increases, ensuring that all human annotators provide consistent quality feedback becomes more difficult.

- Training and Calibration: Human raters must be properly trained and calibrated to ensure their evaluations align with desired outcomes.