# WEEK 1

### **Introduction to Generative AI and LLMs**

* **What are LLMs?**   
  Large Language Models (LLMs) are AI systems trained on massive datasets to generate human-like text and solve diverse language tasks.

### **Understanding How LLMs Learn: Training and Fine-Tuning**

* **Model Training / Pre-training:**   
  LLMs learn from vast datasets using extensive computational power, capturing language patterns.
* **Fine-Tuning for Customization:**   
  Pre-trained models can be adapted to specific tasks by training them on specialized datasets.
* **Instruction Tuning:** A Fine-tuning technique  
  Enhances model performance by refining its understanding of specific prompts and instructions.  
  Performs better even on unseen tasks
* **Reinforcement Learning from Human Feedback (RLHF):**   
  Ensures AI aligns with human values by rewarding useful responses and discouraging harmful outputs.

### **Transformer Architecture**

* **Why Transformers?** Introduced in 2017, transformers replaced RNNs, enabling efficient large-scale language processing. (Enabled Parallel execution by removing dependency on sequential models like RNN, LSTM…)
* **Self-Attention Mechanism:**   
  Helps the model focus on relevant words in a sentence, improving language understanding.
* **Multi-Head Attention:**   
  Captures multiple relationships between words simultaneously, enhancing contextual accuracy.
* **Encoder-Decoder vs. Decoder-Only Models:**   
  Encoder-decoder models are suited for translation, while decoder-only models like GPT specialize in text generation. Encoder only models for classification, entity recognition tasks

### **Optimizing Model Responses with Prompt Engineering and In-Context Learning**

* **Prompt Engineering:** The process of crafting effective input text to guide the model’s response.
* **Zero-Shot Learning:** The model performs a task without prior examples in the prompt.
* **One-Shot & Few-Shot Learning:** Providing one or multiple examples in the prompt improves the model’s accuracy.

### **Controlling AI Output: Generation Parameters and Sampling Techniques**

* **Max Tokens:** Limits the length of generated text.
* **Greedy Decoding:** Always selects the most probable word but may lead to repetitive outputs.
* **Top-k Sampling:** Restricts choices to the top-k probable words, introducing **controlled randomness**
* **Top-p (Nucleus) Sampling:** Selects from words whose cumulative probability reaches a set threshold, allowing dynamic selection.
* **Temperature Adjustment:** A higher value increases randomness and creativity, while a lower value produces more deterministic responses.

### **Building AI Applications: The Generative AI Project Lifecycle**

1. **Problem Definition:** Clearly define the task and scope of the AI model’s role in the application.
2. **Model Selection:** Choose between pre-trained, fine-tuned, or custom-trained models based on the use case.
3. **Prompt Engineering:** Optimize how prompts are structured to improve model responses.
4. **Fine-tuning & Adaptation:** Train the model further using domain-specific data if needed.
5. **Evaluation & Benchmarking:** Assess performance using key metrics to ensure accuracy and efficiency.
6. **Deployment & Optimization:** Optimize the model for cost, speed, and reliability before integrating it into real-world applications.
7. **Enhancements & External Data Integration:** Connect the model to APIs and external data sources to overcome its limitations.

### **Choosing the Right Model for Your Generative AI Application**

* **Pre-trained vs. Custom Models:** Most applications use existing foundation models, but custom training may be necessary in specialized domains.
* **Model Hubs:** Platforms like Hugging Face and PyTorch offer curated repositories of models with detailed descriptions (model cards).
* **Transformer Variants:** Different architectures (encoder-only, decoder-only, encoder-decoder) are suited for different tasks.

### **Understanding How LLMs are Trained: Pre-Training and Model Objectives**

* **Pre-Training Phase:** LLMs learn statistical patterns from massive datasets (terabytes to petabytes of text) using self-supervised learning.
* **Encoder-Only Models (Autoencoding Models):** Use masked language modeling (MLM) to predict missing words, making them useful for classification tasks like sentiment analysis.
* **Decoder-Only Models (Autoregressive Models):** Use causal language modeling (CLM) to predict the next token in a sequence, making them ideal for text generation (e.g., GPT).
* **Encoder-Decoder Models (Seq2Seq Models):** Use span corruption and autoregressive decoding, making them effective for tasks like translation and summarization (e.g., T5, BART).

### **Quantization: Reducing Model Size Without Major Performance Loss**

* **FP32 vs. FP16 vs. INT8:** Lowering precision from 32-bit floating points to 16-bit or 8-bit integers reduces memory needs significantly.
* **BFLOAT16 (Brain Floating Point Format):** A Google-developed format that retains FP32's range while using only 16 bits, improving training efficiency.
* **Trade-offs in Quantization:** Lower precision reduces memory usage but may lead to minor accuracy losses.

### **Multi-GPU Training: Scaling Beyond a Single Processor**

* **Data-Parallel Training (DDP):** Copies the model across multiple GPUs and distributes data batches for faster processing.
* **Model Sharding (FSDP & ZeRO Optimizer):** Distributes model parameters, gradients, and optimizer states across GPUs to fit larger models into memory.
* **Performance Considerations:** More GPUs reduce training time, but increased inter-GPU communication can slow down performance.

### **Scaling Laws and Compute-Optimal Models**

* **Balancing Model Size, Data, and Compute:** Performance improves with larger datasets, bigger models, and more computing power, but finding the right balance is key.
* **Power-Law Relationships:**
  + **Compute Budget vs. Model Performance:** More compute improves model accuracy but with diminishing returns.
  + **Training Dataset Size vs. Performance:** Larger datasets improve results, but beyond a point, additional data has a limited impact.
  + **Model Size vs. Performance:** More parameters improve results, but excessively large models may be inefficient if under-trained.
* **Chinchilla Scaling Laws:** Suggests many large models (e.g., GPT-3) are over-parameterized and under-trained, and smaller models trained on more data can achieve similar or better performance.

### **Domain-Specific Pretraining: When to Train a Custom LLM**

* **Specialized Vocabulary:** Some fields (e.g., legal, medical, financial) use unique terminology not well-represented in general-purpose LLMs.
* **Example – BloombergGPT:** A finance-focused LLM trained on a mix of financial and general text to optimize for domain-specific accuracy.

# WEEK 2

### **Improving LLM Performance: Instruction Fine-Tuning & Efficient Fine-Tuning Methods**

* **Why Fine-Tune LLMs?** Pre-trained models understand language but may not follow specific instructions effectively. Fine-tuning improves their ability to complete tasks accurately.
* **Instruction Fine-Tuning:** Trains an LLM on instruction-prompt datasets to align responses with human intent.

### **Instruction Fine-Tuning: Process & Considerations**

* **Data Format:** Uses labeled prompt-completion pairs to train the model on specific tasks like sentiment classification, summarization, or translation.
* **Advantages:** Requires far fewer examples (500-1,000) than pretraining but significantly improves performance.
* **Downside:** May degrade model performance on tasks outside the fine-tuned domain (catastrophic forgetting).

### **Multitask Fine-Tuning: Preserving Generalization**

* **What is multitasking fine-tuning?** Instead of training on a single task, LLMs are fine-tuned using datasets covering multiple tasks (e.g., summarization, translation, entity recognition).
* **Benefit:** Reduces catastrophic forgetting.
* **Challenge:** Requires much larger datasets (50,000-100,000 examples) and higher compute resources.
* **Example:** The FLAN family of models - FLAN-T5.

### **Evaluation Metrics for Fine-Tuned Models**

* **Why Evaluate?** LLMs generate text probabilistically, so traditional accuracy metrics are insufficient.
* **Common Metrics:**
  + **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** Recall - Denominator is Human Reference n-grams (ROUGE-1, ROUGE-2, ROUGE-L (LCS - Longest Common Subsequence)). Summarization Tasks
  + **BLEU (Bilingual Evaluation Understudy):** Precision - Denominator is Machine n-grams. Used for translation tasks. Penalizes longer and shorter translations (Disadvantageous?).
* **Challenges:** Basic ROUGE/BLEU scores may not detect subtle errors (e.g., incorrect word order, missing context).
* **Advanced Benchmarks:**
  + **GLUE & SuperGLUE(General Language Understanding Evaluation):** Test model generalization across tasks.
  + **MMLU (Massive Multitask Language Understanding):** Evaluates world knowledge and reasoning with zero-shot and few-shot techniques
  + **HELM (Holistic Evaluation of Language Models):** Assesses accuracy, bias, and fairness.

### **Parameter-Efficient Fine-Tuning (PEFT): Reducing Compute Costs**

* **Why PEFT?** Full fine-tuning is resource-intensive. PEFT updates only a small fraction of model parameters, reducing memory and compute requirements.
* **PEFT Techniques:**
  1. **Selective Methods:** Fine-tune only specific layers of the model.
  2. **Reparameterization Methods (e.g., LoRA):** Modify model weights using low-rank matrices.
  3. **Additive Methods (e.g., Prompt Tuning):** Introduce small trainable components without modifying LLM weights.

### **Low-Rank Adaptation (LoRA): A Memory-Efficient Fine-Tuning Approach**

* **How LoRA Works:**
  + Keeps LLM weights frozen and trains two small low-rank matrices (A & B).
  + At inference, these matrices are multiplied and added to the model’s original weights.
* **Benefits:**
  + Reduces trainable parameters by ~86%, making fine-tuning feasible on a single GPU.
  + LoRA adapters can be swapped in/out for different tasks, avoiding multiple full-sized models.
* **Performance Trade-off:** LoRA achieves nearly the same performance as full fine-tuning but with significantly lower compute costs.

### **Prompt Tuning: Improving Performance Without Changing Model Weights**

* **What is Prompt Tuning?** Instead of modifying model weights, **soft prompts** (trainable tokens) are added to the input text.
* **How It Works:**
  + Soft prompts are learned via supervised training.
  + Unlike human-written prompts, these tokens exist in the model’s continuous embedding space.
* **Advantages:**
  + Requires fewer parameters than LoRA.
  + Enables task specialization while preserving generalization.
* **Limitations:** Works best with larger models (>10B parameters).

### **Comparing Fine-Tuning Strategies: Performance vs. Efficiency**

| **Fine-Tuning Method** | **Trainable Parameters** | **Compute Requirement** | **Performance Impact** |
| --- | --- | --- | --- |
| **Full Fine-Tuning** | 100% | High | Best Performance |
| **LoRA (PEFT)** | ~1-10% | Low | Slightly Lower than Full Fine-Tuning |
| **Prompt Tuning (PEFT)** | <1% | Very Low | Works Best for Large Models |

# WEEK 3

### **Reinforcement Learning from Human Feedback (RLHF): Aligning LLMs with Human Preferences**

* **Why RLHF?** Large Language Models (LLMs) sometimes generate harmful, biased, or incorrect responses due to training on diverse internet data. RLHF helps align models with human values—**helpfulness, honesty, and harmlessness (HHH).**
* **How RLHF Works:** Uses reinforcement learning (RL) with human feedback to train LLMs to improve response quality while minimizing harmful outputs.

### **Training LLMs with RLHF: Reward Models & Human Feedback**

* **Step 1: Collecting Human Feedback**
  + Humans rank multiple responses to a prompt based on helpfulness.
  + Rankings are used to train a **reward model** that mimics human preferences.
* **Step 2: Training a Reward Model**
  + The model is trained using **pairwise comparisons**—preferred vs. less preferred completions.
  + A reward score is assigned to guide future completions.
* **Step 3: Using RL to Update the LLM**
  + The LLM generates a response → Reward model assigns a score → RL algorithm updates LLM weights to maximize high-scoring responses.

### **Proximal Policy Optimization (PPO): A Key Algorithm for RLHF**

* **Why PPO?** Optimizes model updates without drastic changes, ensuring stable training.

### **Avoiding RLHF Pitfalls: Reward Hacking & Over-Optimization**

* **What is Reward Hacking?** The LLM learns to game the reward function rather than improve responses. Example:
  + Instead of removing toxic language, it overuses **"most awesome"** phrases to seem positive.
* **Preventing Reward Hacking:**
  + Use a **reference model** (frozen version of the original LLM) to compare changes.
  + Compute **KL divergence** to penalize excessive deviation from the base model.

### **Optimizing LLMs for Deployment: Distillation, Quantization & Pruning**

* **Model Distillation:**
  + Trains a **smaller student model** to mimic a larger **teacher model** for faster inference.
  + Effective for encoder models like BERT but less so for decoder-based LLMs.
* **Quantization:**
  + Reduces model size by lowering precision (e.g., FP32 → FP16 → INT8).
  + Post-training quantization (PTQ) further optimizes storage and inference speed.
* **Pruning:**
  + Removes redundant weights to reduce memory footprint.
  + Some methods, like LoRA, fine-tune only specific layers.

### **LLM Limitations & Overcoming Knowledge Cutoff Issues**

* **Knowledge Cutoff:** LLMs only "know" information available up to their last training date.
* **Math Limitations:** LLMs **predict** numbers but don't perform real calculations.
* **Hallucinations:** LLMs may generate incorrect facts or completely fabricated answers.

### **Retrieval-Augmented Generation (RAG): Accessing External Knowledge**

* **Why Use RAG?** Instead of retraining LLMs, RAG retrieves external data at inference time.
* **How It Works:**
  + **User Query → Encoded as a search query.**
  + **Retriever searches a database, document corpus, or web source.**
  + **Relevant information is combined with the user query.**
  + **LLM generates a response using both the query and retrieved data.**
* **Example Applications:**
  + Legal research (retrieving case law).
  + Medical AI (accessing recent research papers).

### **LLMs as Reasoning Engines: Enhancing Decision-Making with External Tools**

* **LLMs Can Execute Actions:** Instead of just answering questions, LLMs can trigger APIs, run Python scripts, or retrieve live data.

### **Improving Reasoning: Chain-of-Thought (CoT) Prompting**

* **What is CoT Prompting?** Encourages step-by-step reasoning to solve multi-step problems.
* **Useful for:** Math problems, logical reasoning, physics explanations.

### **PAL (Program-Aided Language Models): Using External Tools for Math & Logic**

* **Why Use PAL?** LLMs struggle with accurate calculations. PAL lets them generate Python scripts to compute correct results.
* **How It Works:**
  1. **User Query** → LLM generates a Python script.
  2. **Script is executed** in a Python interpreter.
  3. **Correct Answer** is appended to the prompt.
  4. **LLM returns an accurate, verified response.**

### **ReAct: Combining Reasoning and Action Execution**

* **What is ReAct?** A framework that allows LLMs to **reason through problems** and **execute actions** like web searches or database queries.
* **How It Works:**
  + **Thought:** Model reasons about a task.
  + **Action:** Model calls an API (e.g., Wikipedia search).
  + **Observation:** Model integrates new knowledge and updates its plan.
  + **Final Answer:** Model provides an accurate response based on retrieved information.