Fine-Tuning, Deployment & Business Integration

03-Hours Seminar @ NUST, ISB
PART 1

Dr. Basharat Hussain
Assistant Professor, NUCES
Islamabad, Pakistan
28th Aug 2025

Introduction to Speaker

- Dr. **Basharat Hussain,**
  Assistant Professor, FAST NUCES, Islamabad
- **Ph.D.** in Computer Science (2025, COMSATS) – Autonomous Vehicles, Intelligent Transportation Systems, Federated Learning, Deep Learning
- **MS** in Computer Science (IIUI, 2016) – Distinction (4.0/4.0 CGPA)
- **MSc** in Computer Science (IIUI, 2001)

Introduction to Speaker

- Teaching: Cloud Computing, Data Mining, Machine Learning, Software Architecture, Computer Programming
- Mobile: (+92) 300 510 6398
- E-mail: basharat@live.com (private)
- LinkedIn: linkedin.com/in/basharathussain
- Website: https://basharathussain.github.io
- Location: Wah Cantt, 44070 Pakistan

List of Publications

- Article (J04) A. A. Khan, B. Hussain*, M. Islam, M. M. A. Dabel and A. K. Bashir, "**Optimizing Content Cache with Vehicular Edge Computing: A Deep Federated Learning based Novel Predictive Study**," IEEE Transactions on Consumer Electronics, vol. 71, no. 2, pp. 6069-6079, May 2025, doi: 10.1109/TCE.2025.3571029.
  Impact Factor: 10.9 (2025 JCR)

- Article (J03) B. Hussain and M. K. Afzal, "**Optimizing Urban Traffic Incident Prediction with Vertical Federated Learning: A Feature Selection based Approach**", IEEE Transactions on Network Science and Engineering, vol. 12, no. 1, pp. 145-155, Jan.-Feb. 2025, doi: 10.1109/TNSE.2024.3487268
  Impact Factor: 6.7 (2024 JCR).

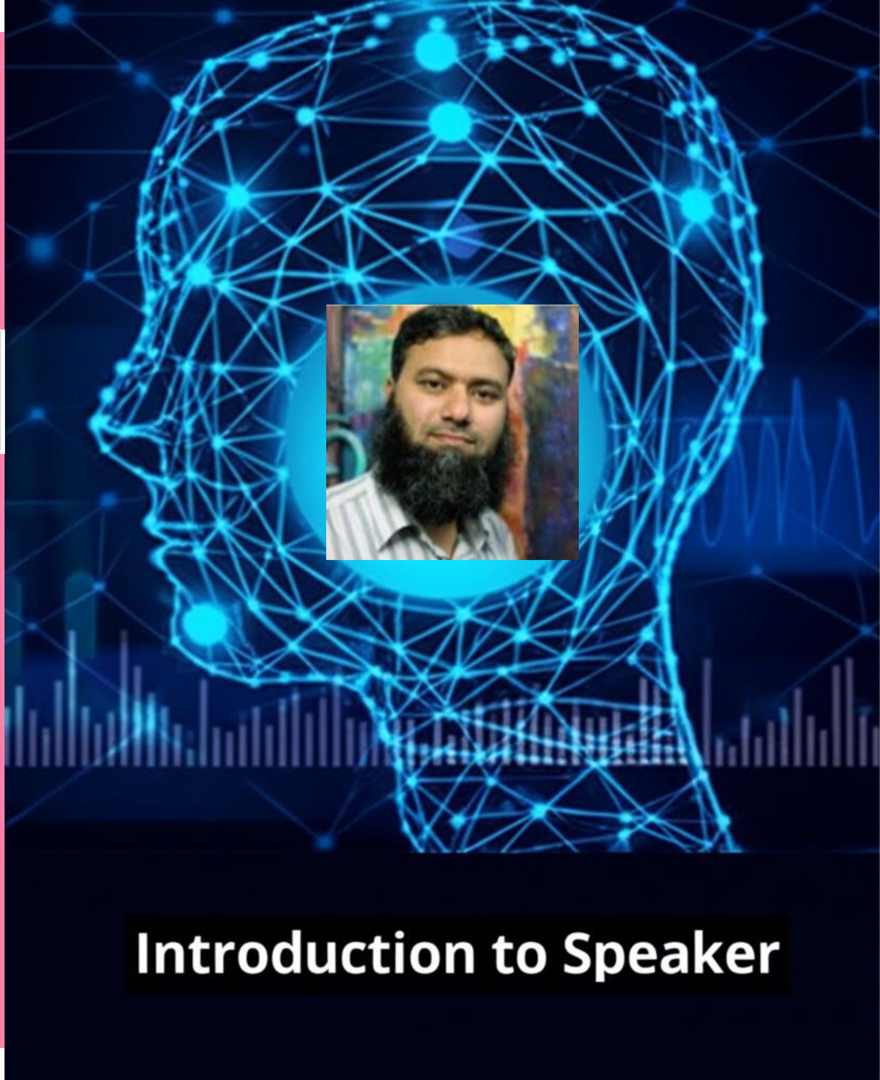
- Article (J02) B. Hussain, M. K. Afzal, S. Anjum, I. Rao and B-S. Kim, "**A Novel Graph Convolutional Gated Recurrent Unit Framework for Network-Based Traffic Prediction**", IEEE Access, vol. 11, pp. 130102-130118, 2023, doi: 10.1109/ACCESS.2023.3333938
  Impact Factor: 3.6 (2023 JCR).

- Article (J01) B. Hussain, M. K. Afzal, S. Ahmad and A. M. Mostafa, "**Intelligent Traffic Flow Prediction Using Optimized GRU Model**", IEEE Access, vol. 9, pp. 100736-100746, 2021, doi: 10.1109/ACCESS.2021.3097141
  Impact Factor: 3.4 (2021 JCR).

Research Focus

GenAI / LLM	Multimodality	RAG	Model Fine-Tuning	AgenticAI	N8N	Software Technical Lead	Solution Architect	ML / DL / DevOps	Cloud Computing

- Traffic Prediction with Deep Neural Networks & Federated Learning
- Pedestrian & Human Behavior Modeling for Autonomous Vehicles
- LLM-Powered Structured Data Extraction & Decision Support



Industry Experience (20+ Years)

- Experience in Software Development
- - AI Consultant (2iQ Financial Services, 2025) → Model fine-tuning LLM pipeline for structured financial data (94% precision), Building business process automation using N8N and Agentic AI.
- - CTO (FindVaccineNow, 2020–2024, USA) → Global COVID-19 vaccine platform (135+ countries, partnered with Apple)
- - Chief Architect (Route Trading, UK) → FinTech AML & compliance systems
- - 20+ years in Software Development (C++, Python, C#, ASP.NET, Java, ML systems)

Fine-Tuning, Deployment & Business Integration

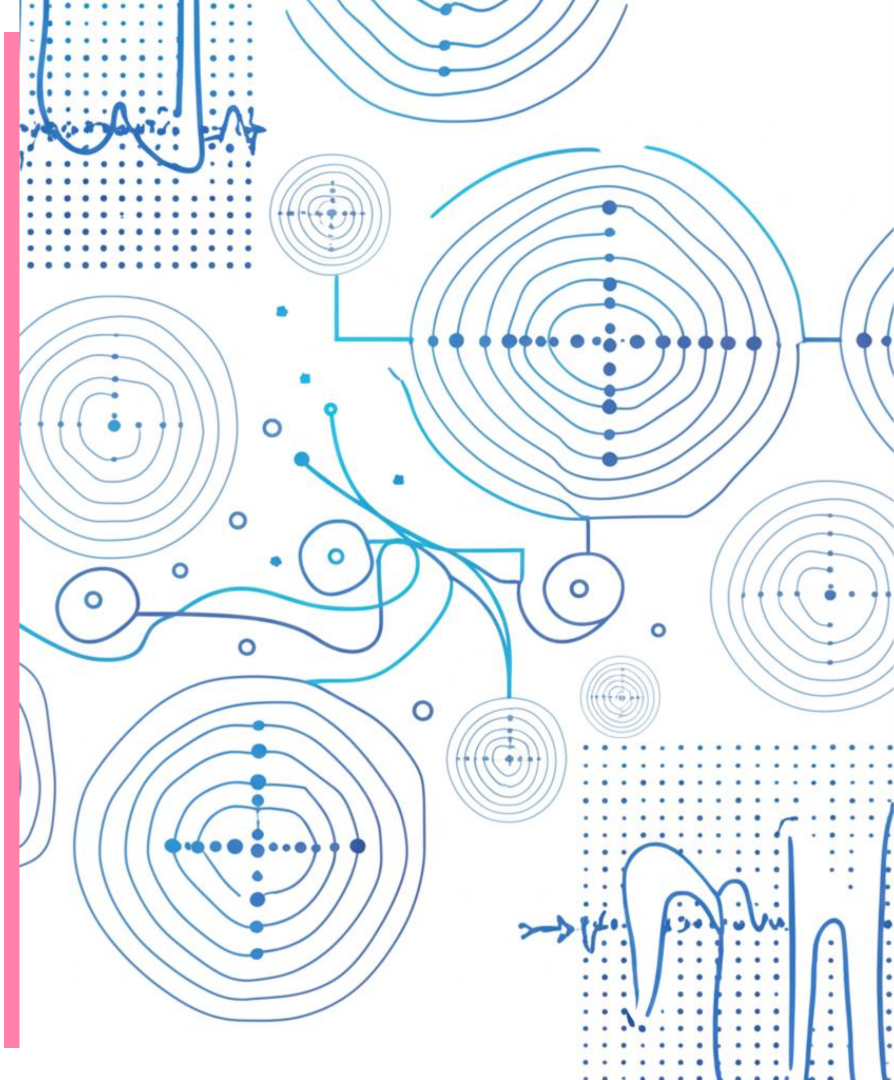
FINE-TUNING

DEPLOYMENT

BUSINESS INTEGRATION

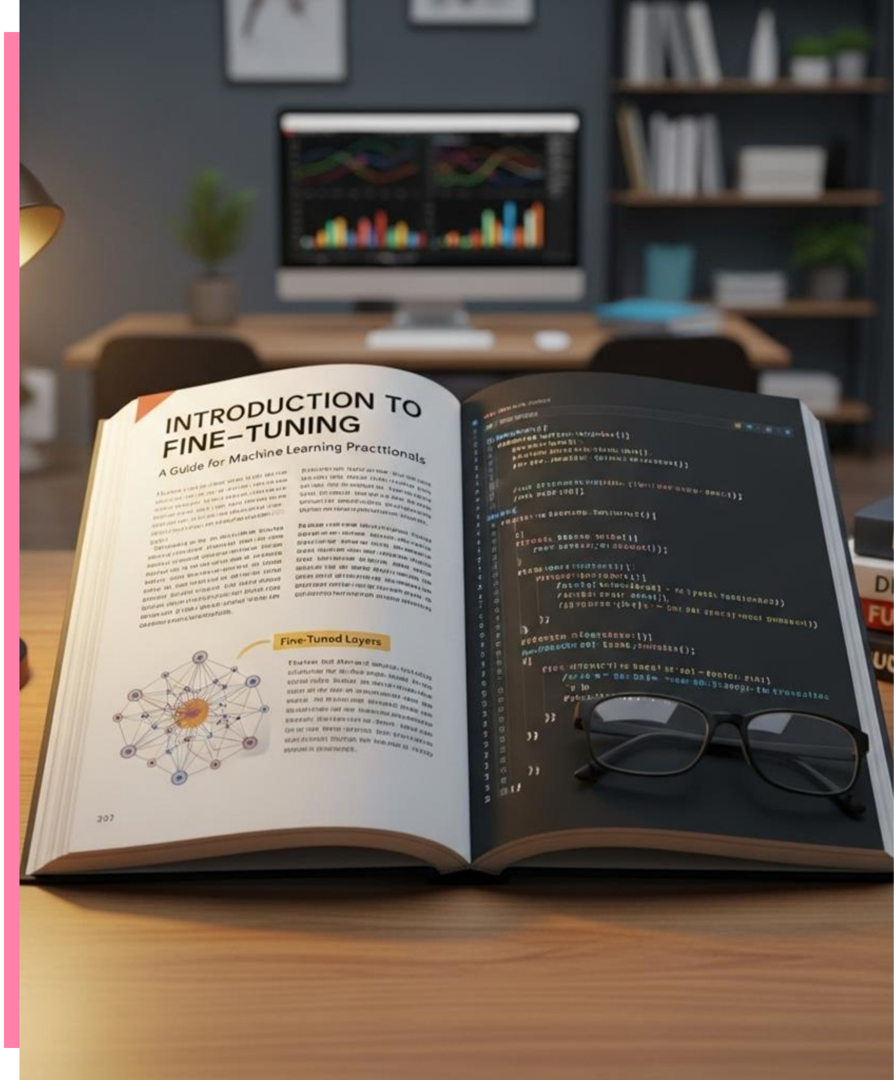
Outlines

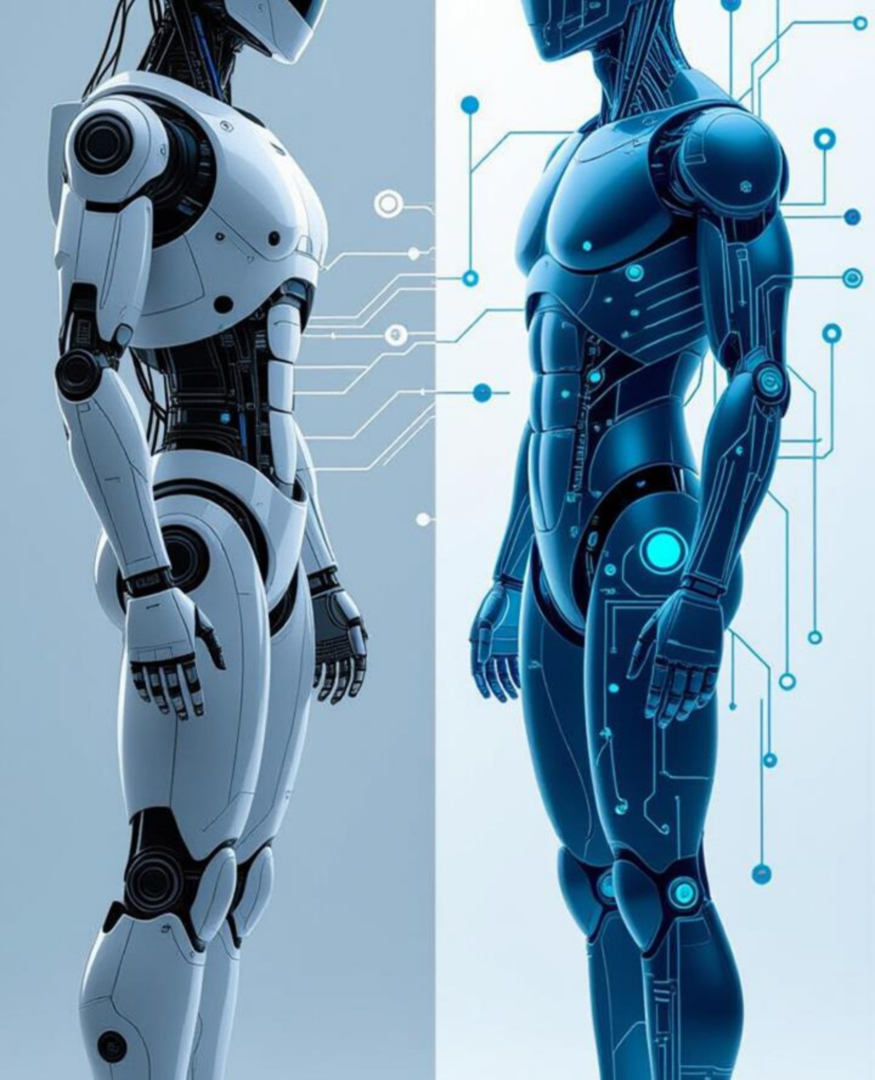
- Key Learning Objectives (KLOs) of the Seminar
- Model Fine-tuning
- Pre-trained Model vs Fine-tuned Model
- Limitation of Pre-trained Base Models
- Advantage of Fine-tuning Your Own LLM
- Instruction Fine-tuning
- Data Preparation
- Approach to Fine-tuning
- PEFT: Parameter Efficient Fine-tuning
- Error Analysis
- Sample Training Code



Key Learning Objectives (KLOs)

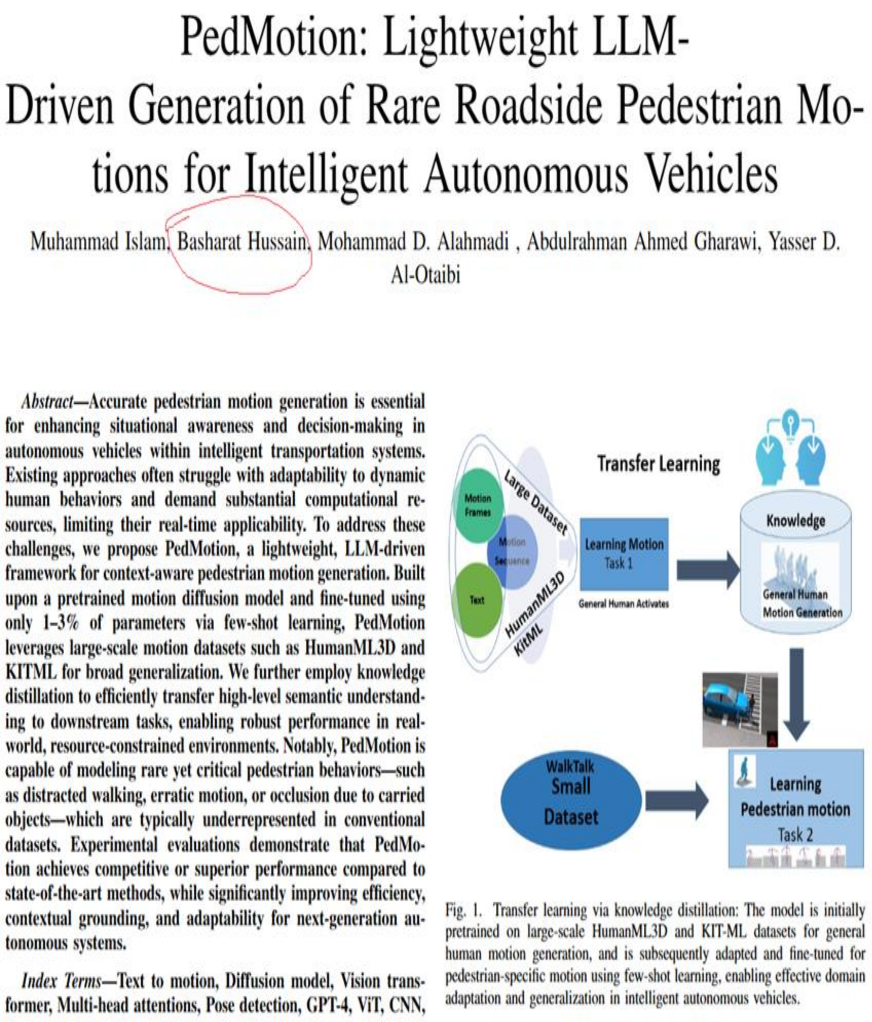
- Understand the concept of fine-tuning
- Differentiate b/w pre-trained & fine-tuned models
- Recognize the role of pre-training in LLMs
- Identify limitations of pre-trained base models
- Explore the advantages of fine-tuning
- Explain instruction fine-tuning
- Prepare high-quality datasets for fine-tuning
- Apply systematic approaches to fine-tuning
- Understand PEFT (Parameter-Efficient Fine-Tuning)
- Gain hands-on experience with training code





Introduction to Fine-tuning

- **Definition**: In deep learning, fine-tuning is an approach to **transfer learning** in which the weights of a pre-trained model are trained on **new data**.
  - To a Specific domain (e.g., medicine) or task (e.g., legal document analysis).
- **How it works**: Start with a large pre-trained model that has general knowledge. Train it further on task-specific data (e.g., sentiment analysis, text generation, document similarity).



PedMotion: Lightweight LLM-Driven Generation of Rare Roadside Pedestrian Motions for Intelligent Autonomous Vehicles

Muhammad Islam, Basharat Hussain, Mohammad D. Alahmadi , Abdulrahman Ahmed Gharawi, Yasser D. Al-Otaibi

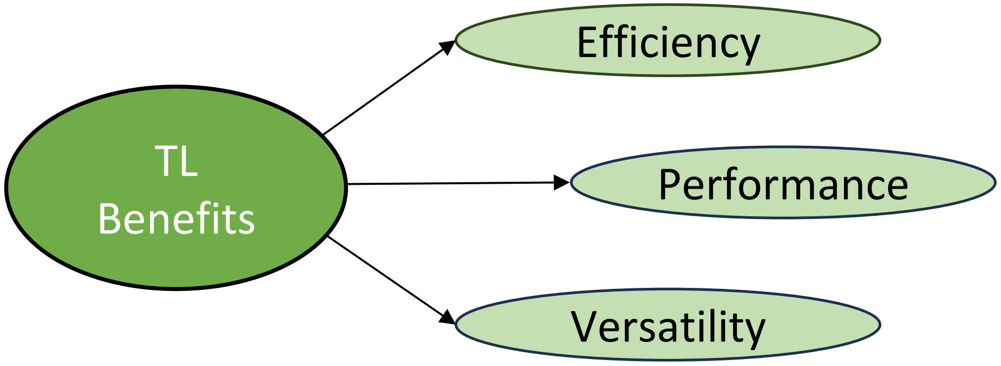
Abstract—Accurate pedestrian motion generation is essential for enhancing situational awareness and decision-making in autonomous vehicles within intelligent transportation systems. Existing approaches often struggle with adaptability to dynamic human behaviors and demand substantial computational resources, limiting their real-time applicability. To address these challenges, we propose PedMotion, a lightweight, LLM-driven framework for context-aware pedestrian motion generation. Built upon a pretrained motion diffusion model and fine-tuned using only 1–3% of parameters via few-shot learning, PedMotion leverages large-scale motion datasets such as HumanML3D and KITML for broad generalization. We further employ knowledge distillation to efficiently transfer high-level semantic understanding to downstream tasks, enabling robust performance in real-world, resource-constrained environments. Notably, PedMotion is capable of modeling rare yet critical pedestrian behaviors—such as distracted walking, erratic motion, or occlusion due to carried objects—which are typically underrepresented in conventional datasets. Experimental evaluations demonstrate that PedMotion achieves competitive or superior performance compared to state-of-the-art methods, while significantly improving efficiency, contextual grounding, and adaptability for next-generation autonomous systems.

Index Terms—Text to motion, Diffusion model, Vision transformer, Multi-head attentions, Pose detection, GPT-4, ViT, CNN,

Fig. 1. Transfer learning via knowledge distillation: The model is initially pretrained on large-scale HumanML3D and KIT-ML datasets for general human motion generation, and is subsequently adapted and fine-tuned for pedestrian-specific motion using few-shot learning, enabling effective domain adaptation and generalization in intelligent autonomous vehicles.

What is transfer learning (TL)?

- A machine learning method where a model developed for one task is reused as the starting point for a model on a different but related task.



When do you need to fine-tune the model?

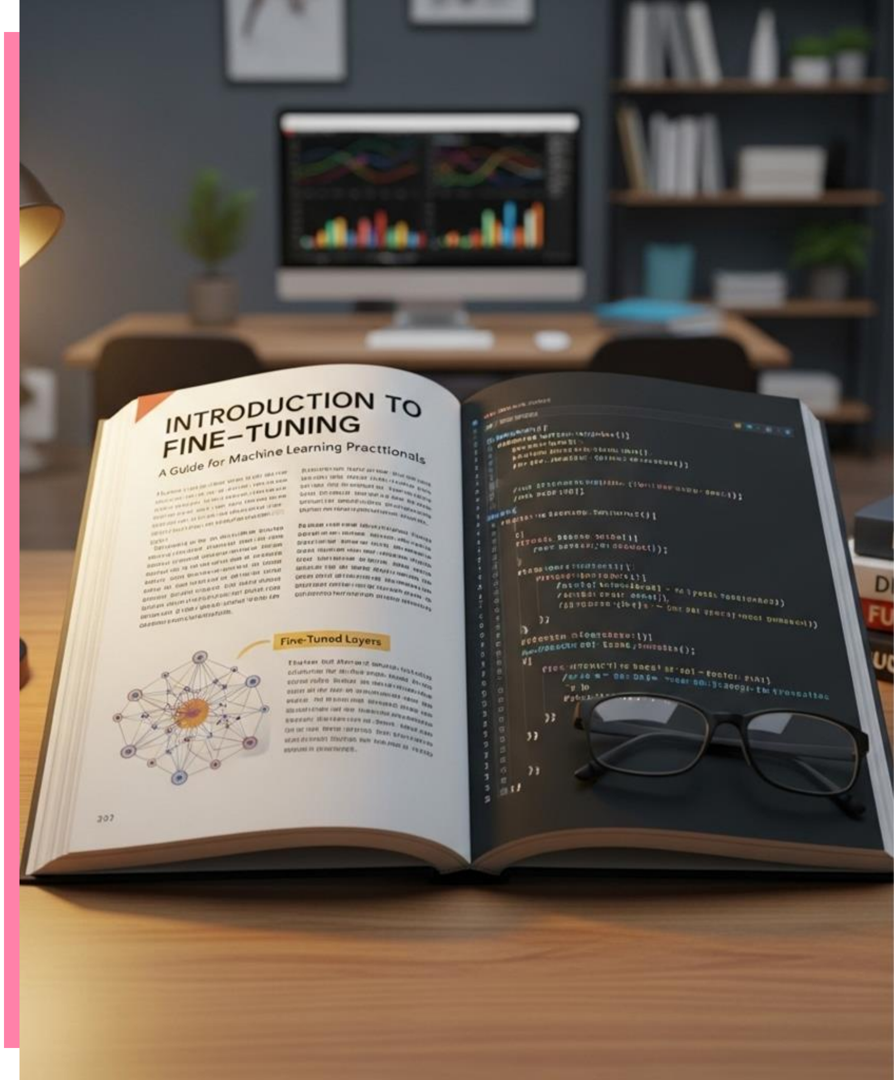
✅ **Prompt engineering did not work out –** If adjusting prompts doesn't achieve the desired results.

✅ **Retrieval augmented generation (RAG) didn't work out –** If augmenting with external knowledge retrieval still fails.

✅ **Highly qualitative data for training is available –** If you have strong domain-specific or labeled data.

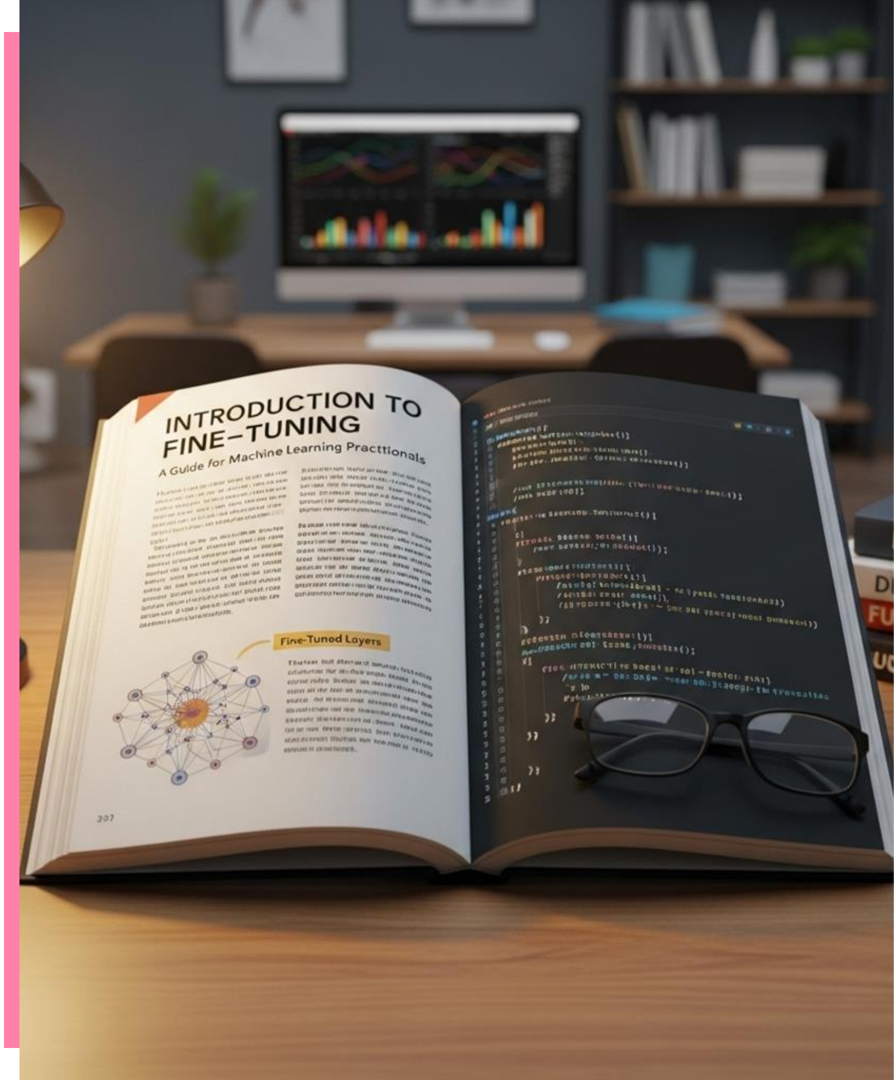
✅ **Cost is not a problem –** Fine-tuning requires resources, so this matters.

✅ **It is clear how to measure the result** – If success metrics are well-defined and measurable.

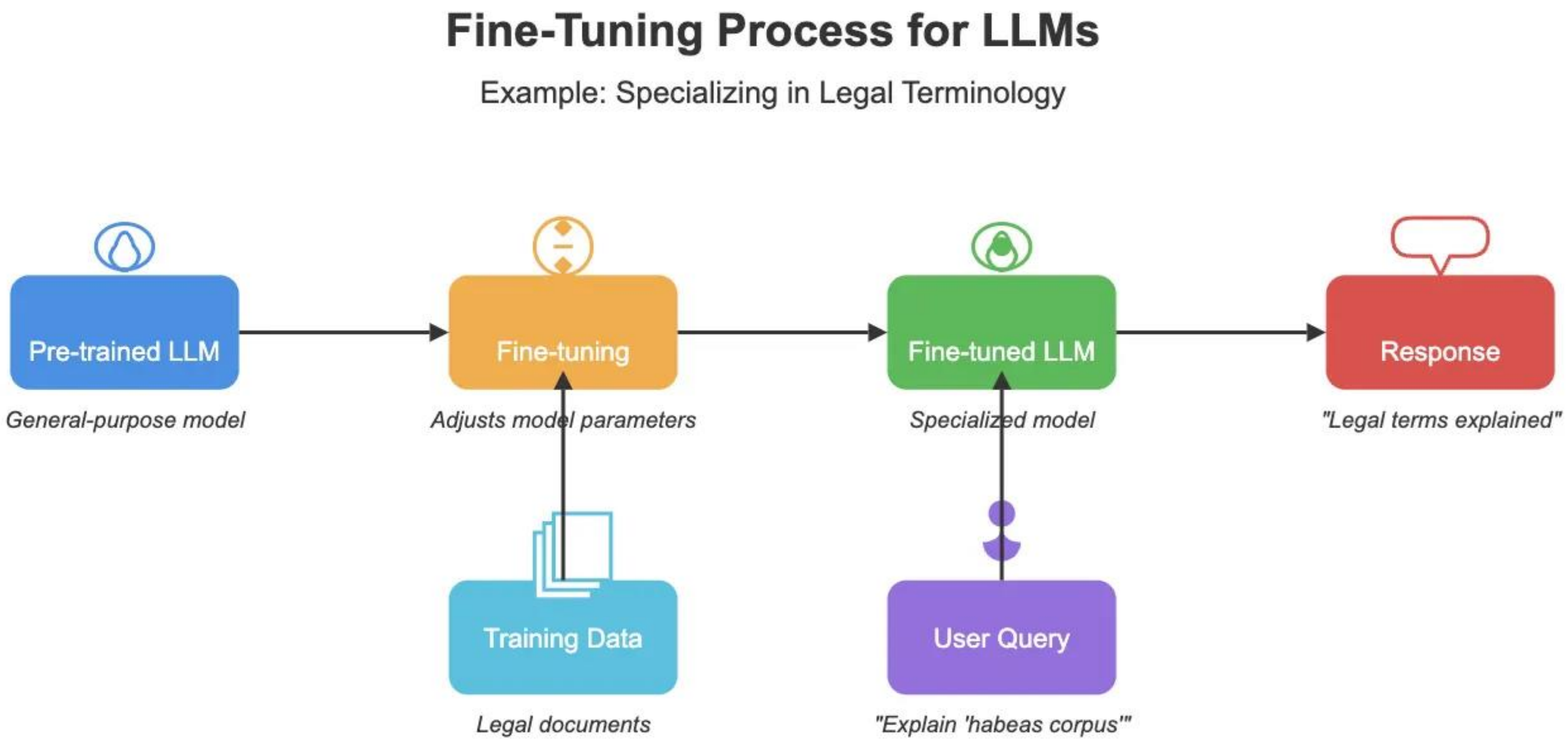


What Fine-tuning Does for the Model?

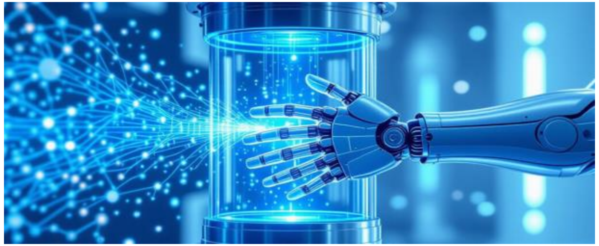
- Enables the model to learn task-specific patterns, not just rely on general knowledge.
- Produces more consistent and reliable outputs.
- Helps reduce hallucinations and irrelevant answers.
- Customizes the model for a domain or use case (e.g., legal, medical, financial).



Understanding Agents: Passive vs. Active



Understanding Pre-trained Model vs Fine-tuned Model



Pre-trained Model

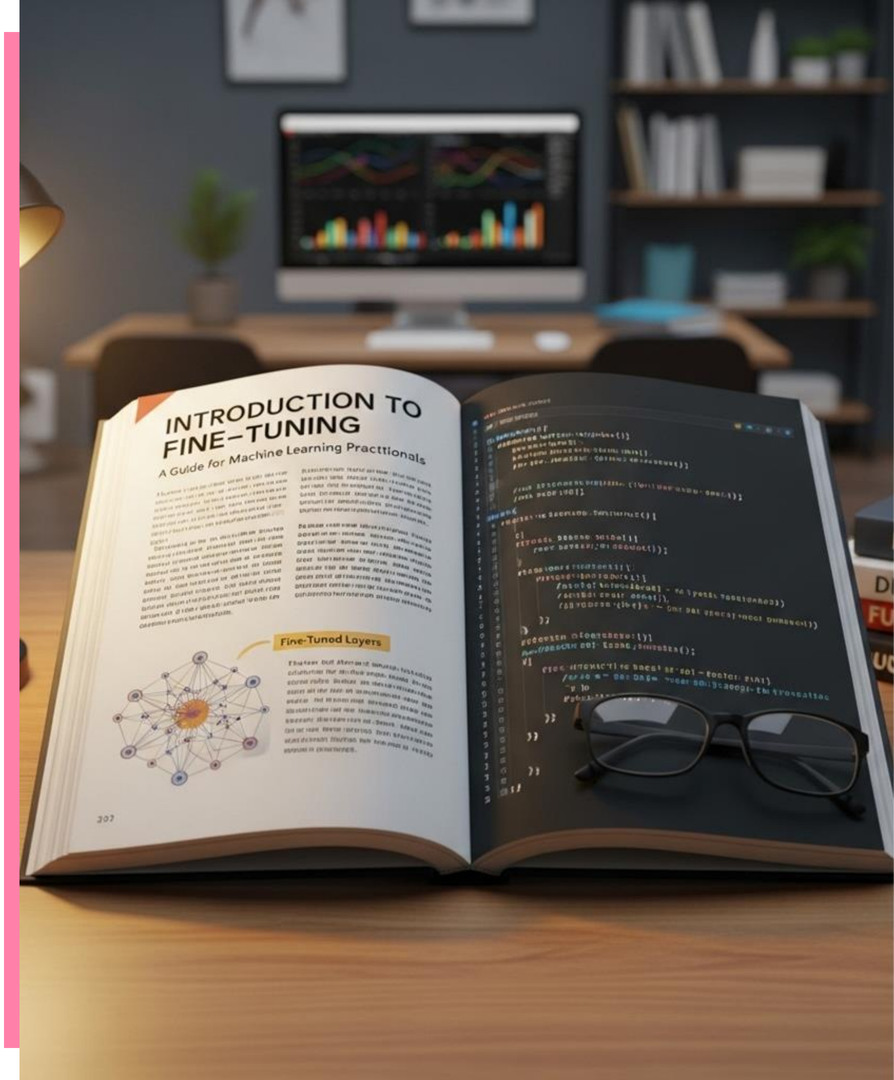
- ✅ No data required to get started
- ✅ Smaller upfront cost
- ✅ No technical expertise needed
- ✅ Can connect external data via Retrieval-Augmented Generation (RAG)
- ⚠️ Produces generic outputs (less specialized)
- ⚠️ May hallucinate
- ⚠️ RAG may miss or retrieve wrong data

Fine-tuned Model

- ⚠️ Requires domain-specific training data
- ⚠️ Higher compute cost upfront
- ⚠️ Needs ML/AI expertise to train
- ✅ Can still use RAG (improved reliability)
- ✅ Learns high-quality, domain-specific and new knowledge
- ✅ Can reduce errors and hallucinations
- ✅ Able to adapt and learn new information
- NOTE: Less cost afterwards, if small model.

Code Demo 1: Fine-tuning LLM

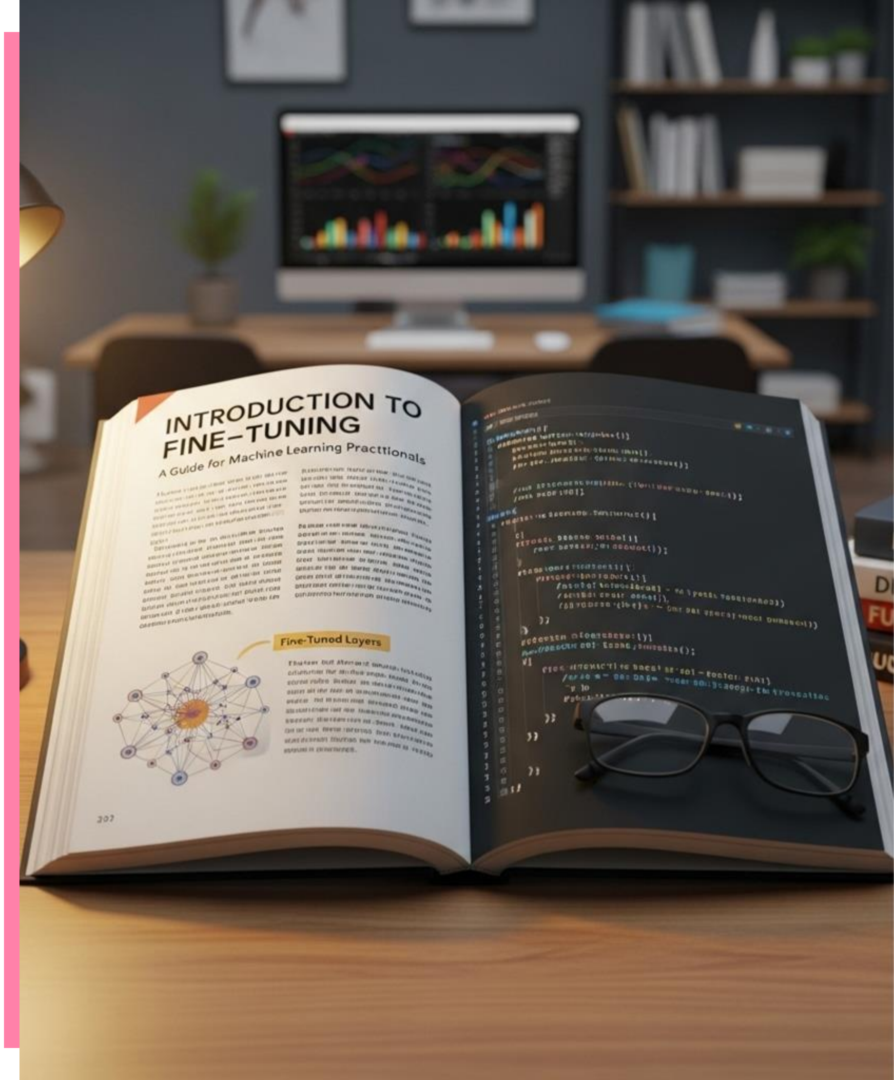
- The notebook demonstrates how to **fine-tune a large language model (Qwen 1.5B Instruct)** using **LoRA (Low-Rank Adaptation)** with **4-bit quantization (QLoRA)** so that it runs efficiently on limited hardware.
- The fine-tuned model is saved locally.
- This model is fine tuned on a small **Dolly dataset** subset, logs results with **wandb**, and saves the fine-tuned model locally and on Hugging Face.



Code Demo 1: Fine-tuning LLM

Main Steps

1. Setup & Installation
2. Model Loading
3. Experiment Tracking
4. Dataset Preparation
5. LoRA Configuration
6. Training
7. Saving the Model



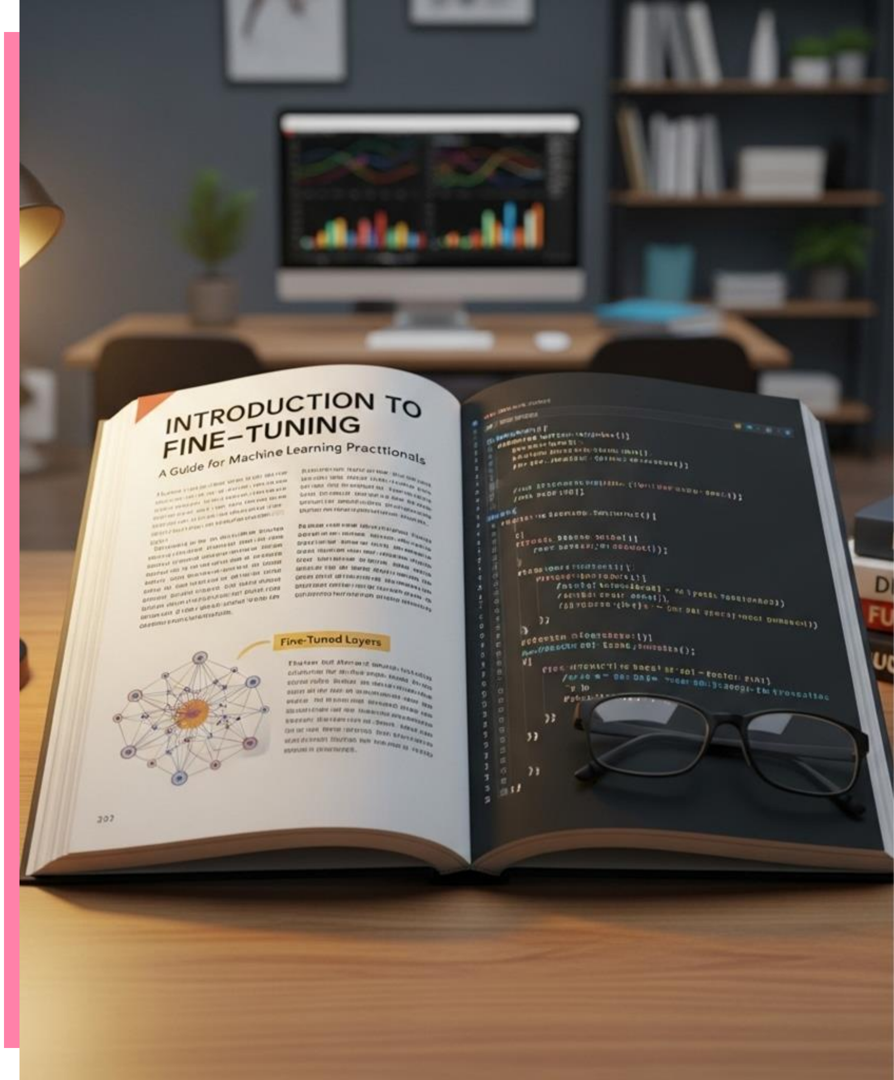
Code Demo 1: Fine-tuning LLM

Why Experiment Tracking?wandb?

When you fine-tune a model, you want to monitor:

- **Loss curve** (does the model keep improving?)
- **Learning rate schedule**
- **Evaluation metrics** (e.g., accuracy, perplexity, etc.)
- **System info** (GPU usage, runtime, etc.)

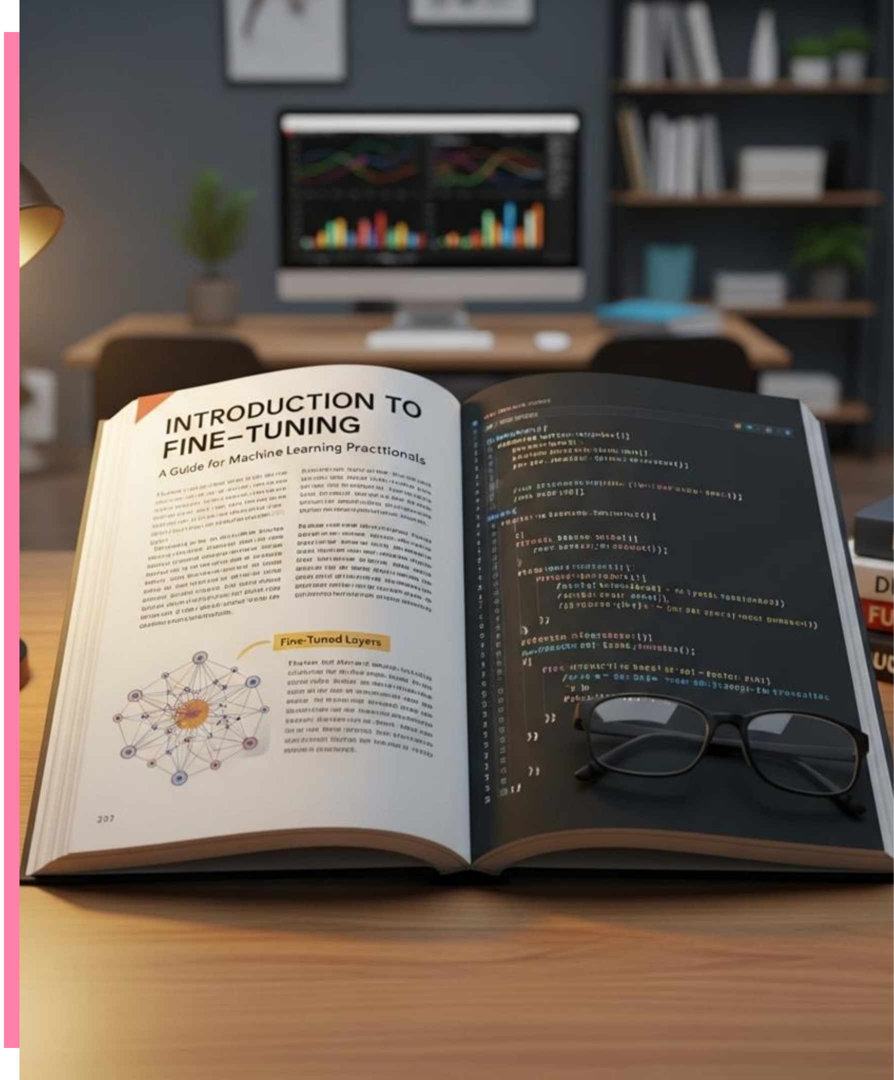
Doing this manually is messy. That’s where **Weights & Biases (wandb)** comes in: it’s a dashboard for logging all training metrics in real time.



Code Demo 1: Fine-tuning LLM

- Source Code Python Notebook: https://drive.google.com/file/d/1ZTG7Or3L2adjH9uzU9w4YiQBRNKqSAld/view?usp=sharing

Perform Code Execution



Training Models to Learn Text Generation (Pre-training)

Pre-training: Training a large language model (LLM) from scratch on a vast amount of unlabeled text.

How it works: Based on Self-Supervised Learning.

- ✓ The model hides/masks a word and learns to predict the next word using context from the preceding words.
- ✓ Over billions of examples, the model develops general language understanding and generation ability.

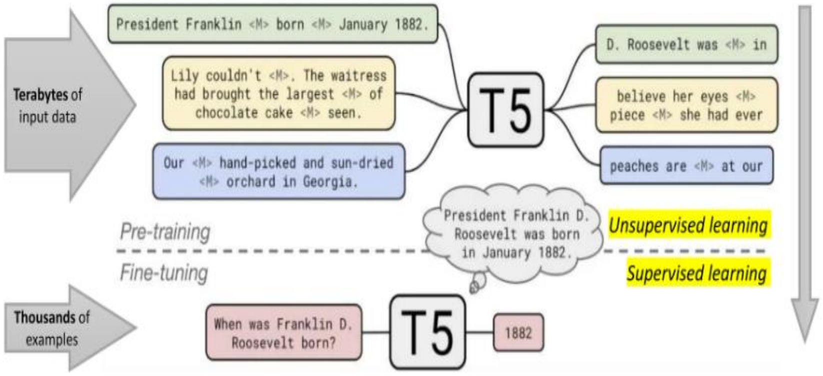
Example: Sentence: "I am a data scientist" Model generates its own labeled pairs:

Text	Label
I	am
I am	a
I am a	data
I am a data	scientist

Limitations of Pre-trained Models

1. **Contextual Understanding**
   - ✓ Struggles to differentiate subtle contexts (e.g., sarcasm, domain-specific meanings).
2. **Generating Misinformation**
   - ✓ May produce incorrect or **misleading information** due to lack of task-specific knowledge.
3. **Lack of Creativity**
   - ✓ Creativity is limited to **pattern imitation** rather than true innovation.
4. **Hallucinations**
   - ✓ Sometimes generates text that **is nonsensical, erroneous, or detached** from reality.

Difference between pretraining and fine-tuning

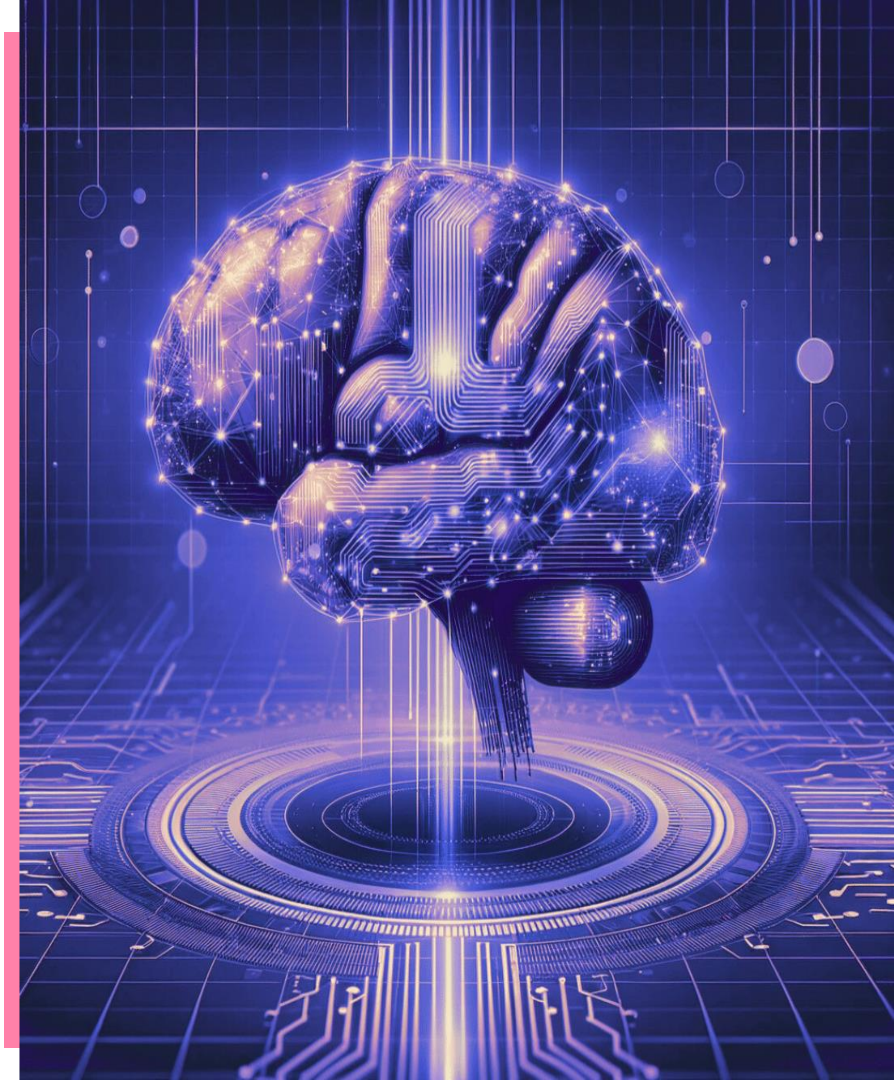


What is Large Language Model (LLM)?

1. A Large Language Model (LLM) is a type of artificial intelligence (AI) program that can **understand, generate, and process** human language.

1. LLMs are a subset of machine learning, specifically deep learning, and are characterized by their massive size, consisting of **billions of parameters.**

1. They function by **predicting the next most likely token** (word or part of a word) in a sequence of text, enabling them to generate coherent and contextually relevant responses.



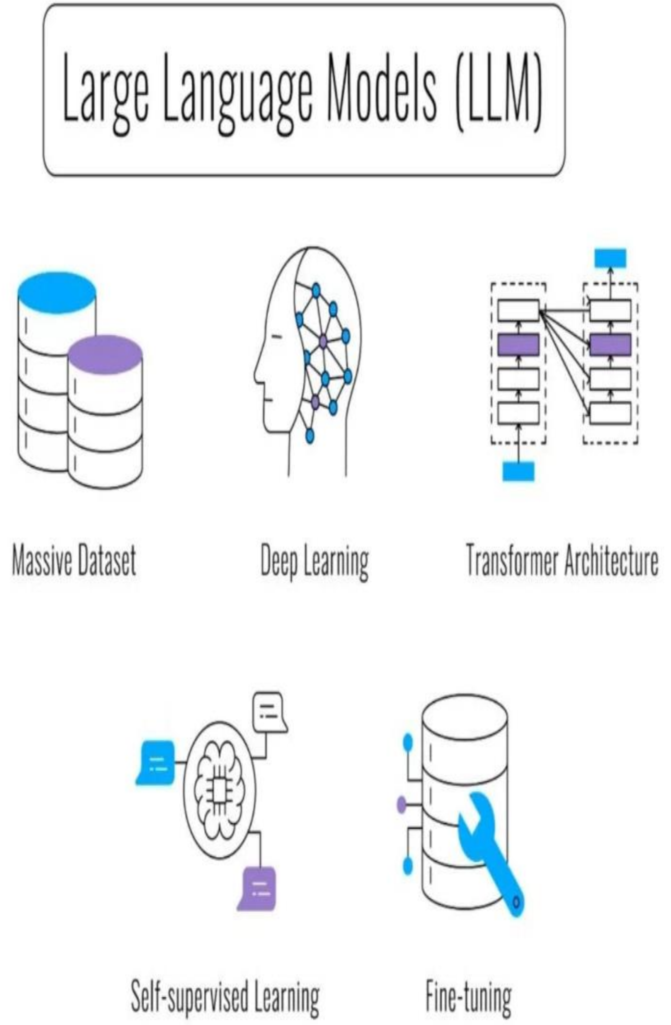
Core Principles of LLMs

LLMs are built upon a foundation of three key concepts:

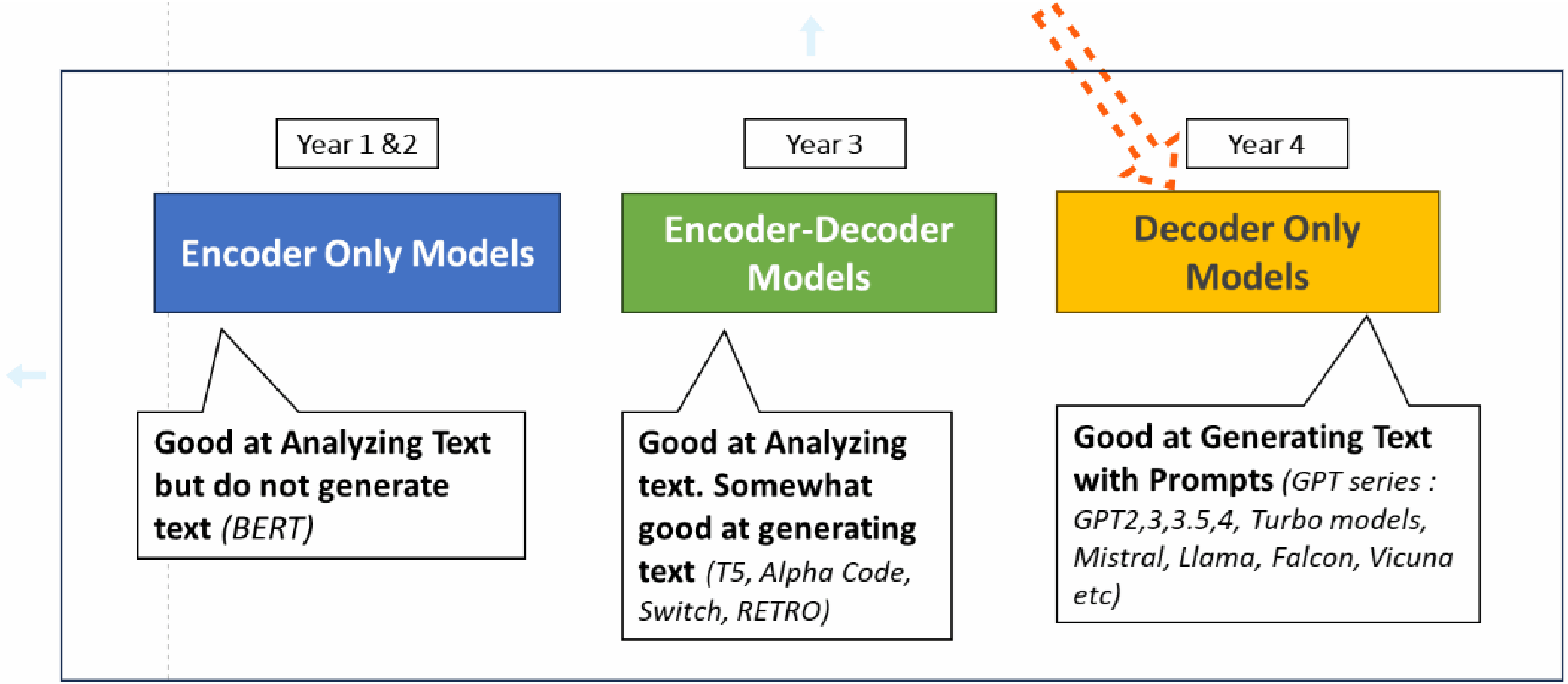
1. **Transformer Architecture:** The Transformer revolutionized NLP with its attention mechanism, allowing LLMs to process entire text sequences in parallel and understand complex relationships between words for coherent output. Attention = context awareness

2. **Scaling Laws:** This principle observes that increasing the number of model parameters, training data, and compute resources predictably improves an LLM's performance, explaining why models have become so massive.

3. **Next Token Prediction:** The core function of an LLM is to predict the most probable next token in a sequence, a process it repeats *autoregressive* to generate a full, contextually relevant response.

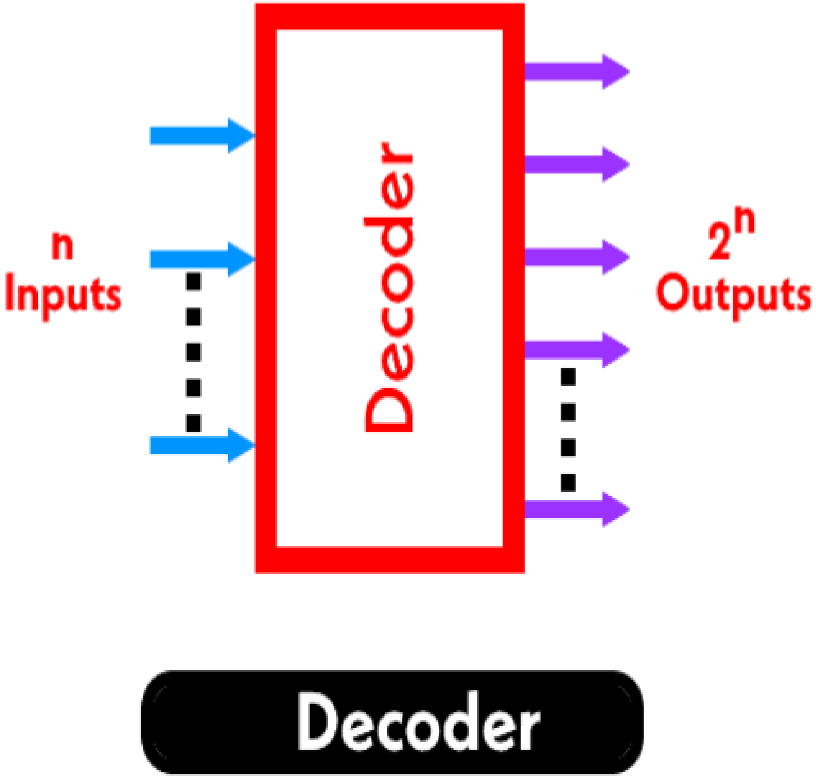


Transformer Architecture Types



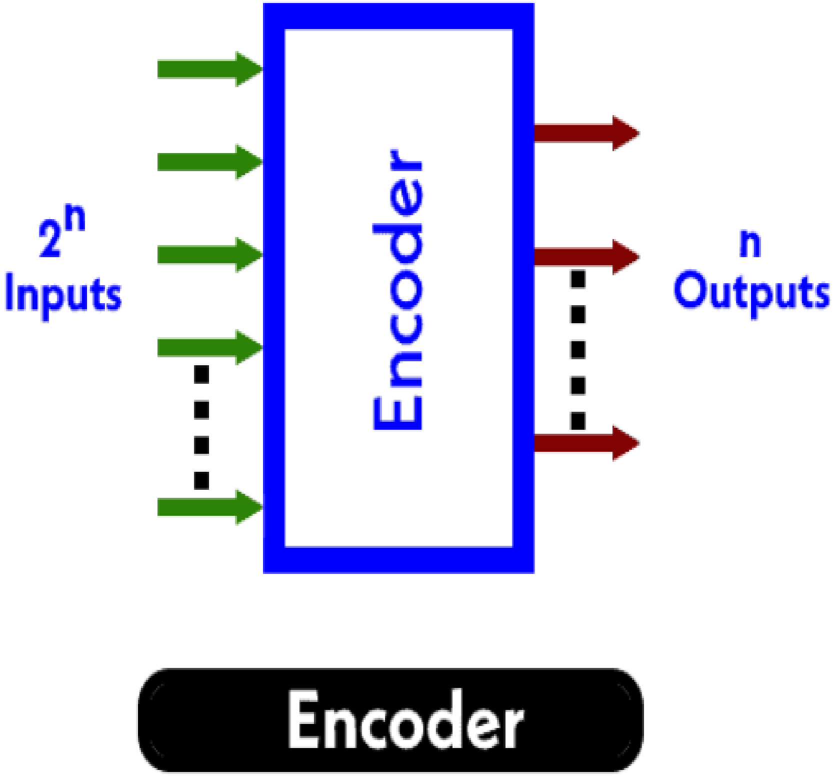
Decoder-Only Models

- Consists solely of a decoder stack of Transformer.
- It processes the input and generates the output in a single, autoregressive stream, predicting one token at a time based on all previous tokens.
-
- Example: ChatGPT, which makes them highly efficient and effective for generative tasks like **text completion, chatbot conversations, and content creation**.
- They are **efficient** models

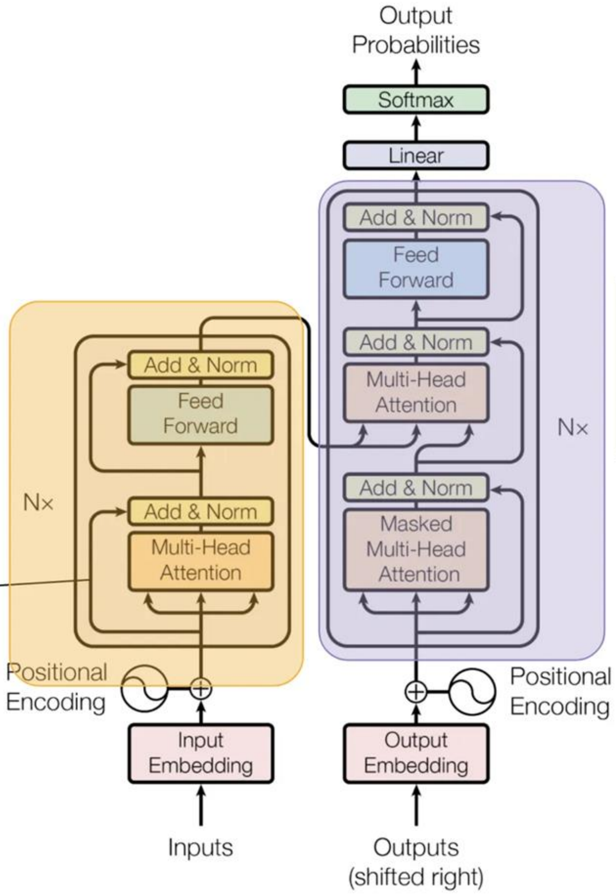


Encoder-Only Models

- Use only the encoder part of the Transformer
-
- Example: like **BERT** (Bidirectional Encoder Representations from Transformers)
- Their primary purpose is not to generate new text, but to deeply understand and create rich contextual representations of the input text.



Encoder-Decoder Models
(The Transformer Architecture)



Encoding layers with Multi-headed Self-Attention

Decoder layers with Masked Multi-headed Self-Attention, and Multi-headed Cross-Attention

Residual Connections, Layer Normalization, & Dropout

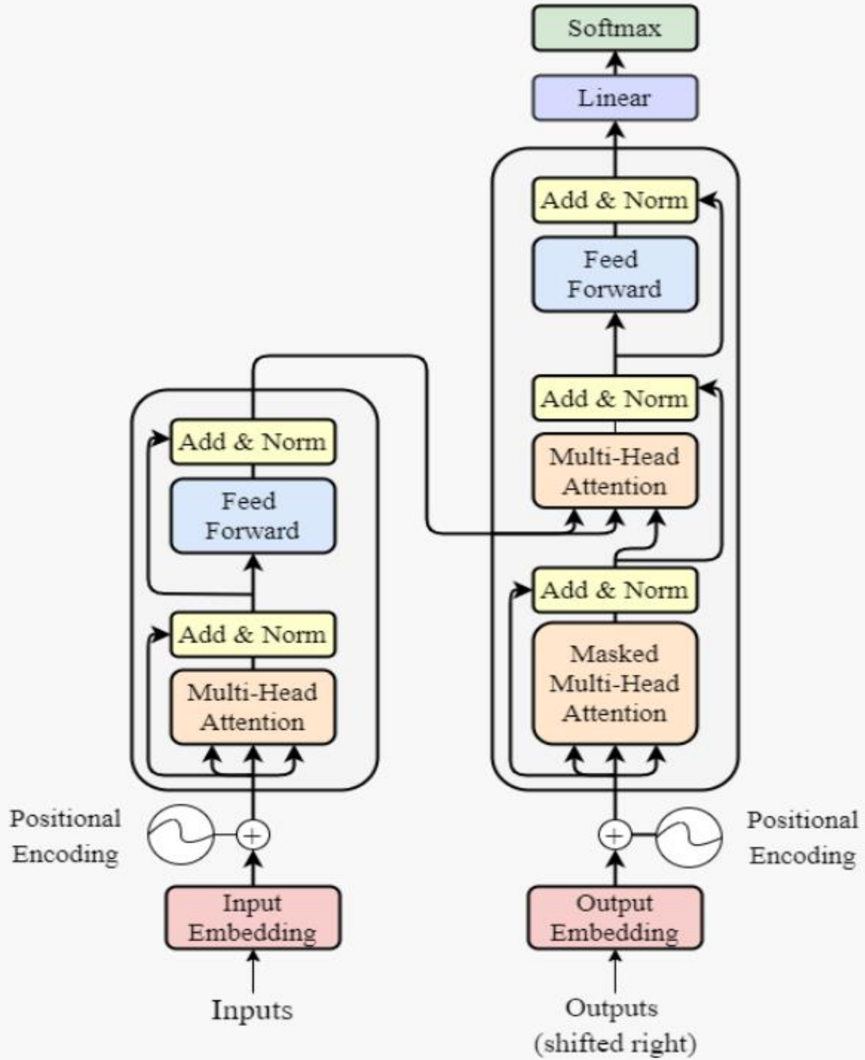
Encoder-Decoder Models (The Transformer Architecture)

Self-Attention: A mechanism that allows the model to weigh the importance of all other words in a sequence to better understand the context of each word.

Positional Encoding: A method used to give the model information about the order of words in a sequence, as parallel processing would otherwise eliminate this.

Encoder-Decoder Stacks: The two main components of the original Transformer, where the encoder understands the input and the decoder generates the output.

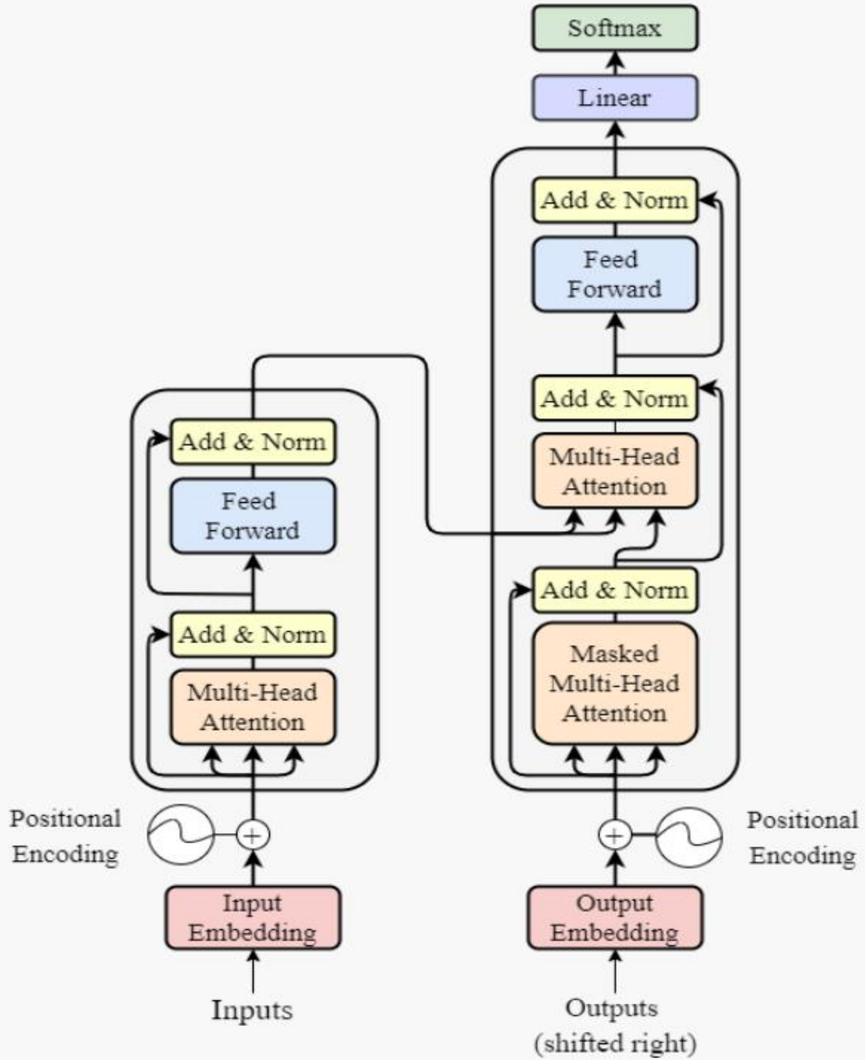
Feed-Forward Neural Network (FFNN)
A Feed-Forward Neural Network in a Transformer block is a small, fully connected network that acts on each position of the sequence independently and identically. (Introduces Non-Linearity, Deepens Feature Extraction)



Encoder-Decoder Models (The Transformer Architecture)

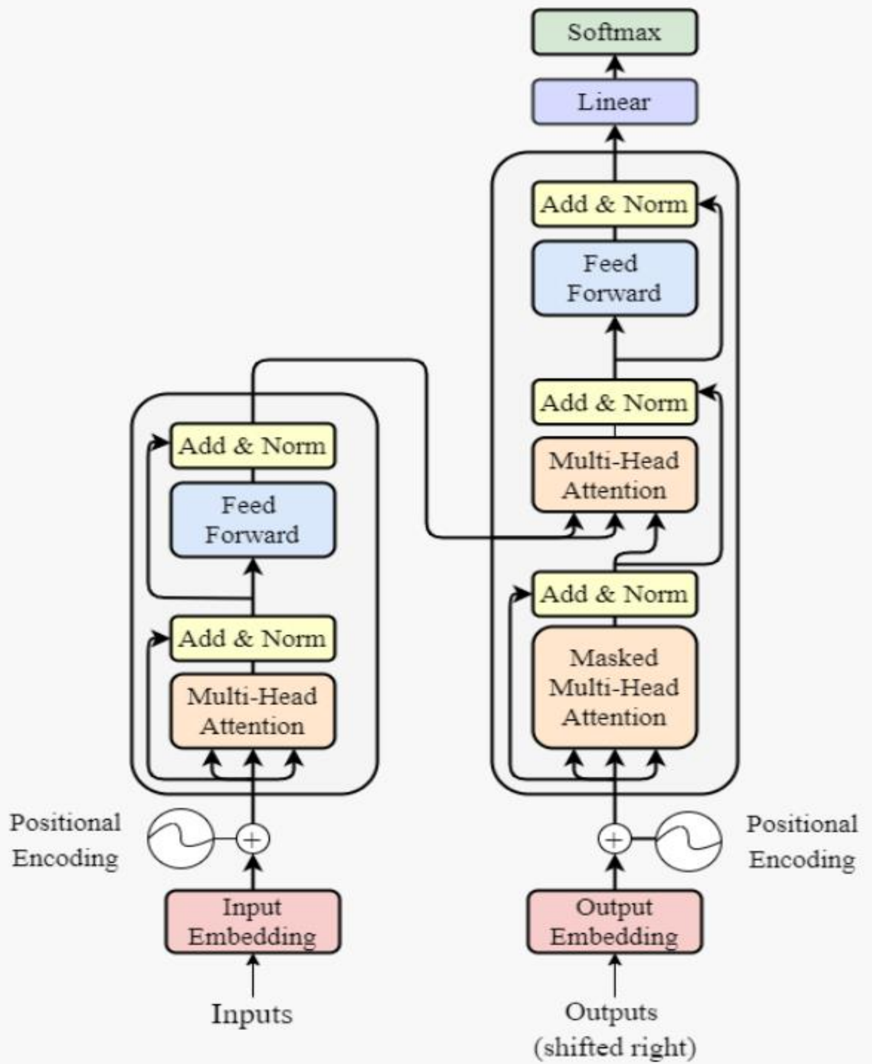
Dropout is a regularization technique used throughout the Transformer architecture, including in the attention and FFNN layers. Its purpose is to prevent **overfitting**.
Drop out benefits are **Reduces Overfitting**, **Ensemble Effect.** It effectively trains a slightly different "sub-network" on each training iteration.

Residual Connections
Residual connections, also known as skip connections, are a vital component found in each Transformer block. Their main purpose is to create a shortcut for information flow.
Prevent **Vanishing Gradients Problem**, **Enhance Information Flow:** By providing a direct path for the original input to be added to the output of a sub-layer (like the attention or FFNN), residual connections ensure that the model retains the original information.



Encoder-Decoder Models (The Transformer Architecture)

Layer Normalization
Layer normalization is a technique applied after each sub-layer (the attention and FFNN) in a Transformer block. It standardizes the inputs to the next layer.
Purpose:
Stabilize Training: Deep neural networks can suffer from unstable training due to the output of each layer having a different mean and variance. Layer normalization addresses this by normalizing the inputs across all neurons in a layer for a single data point. This keeps the activations within a stable range, which prevents the training process from becoming erratic and helps it converge more quickly.
Improve Convergence: By keeping the activations consistent, layer normalization helps to smooth the optimization landscape, allowing the model to take more confident steps during training and reach an optimal solution faster.

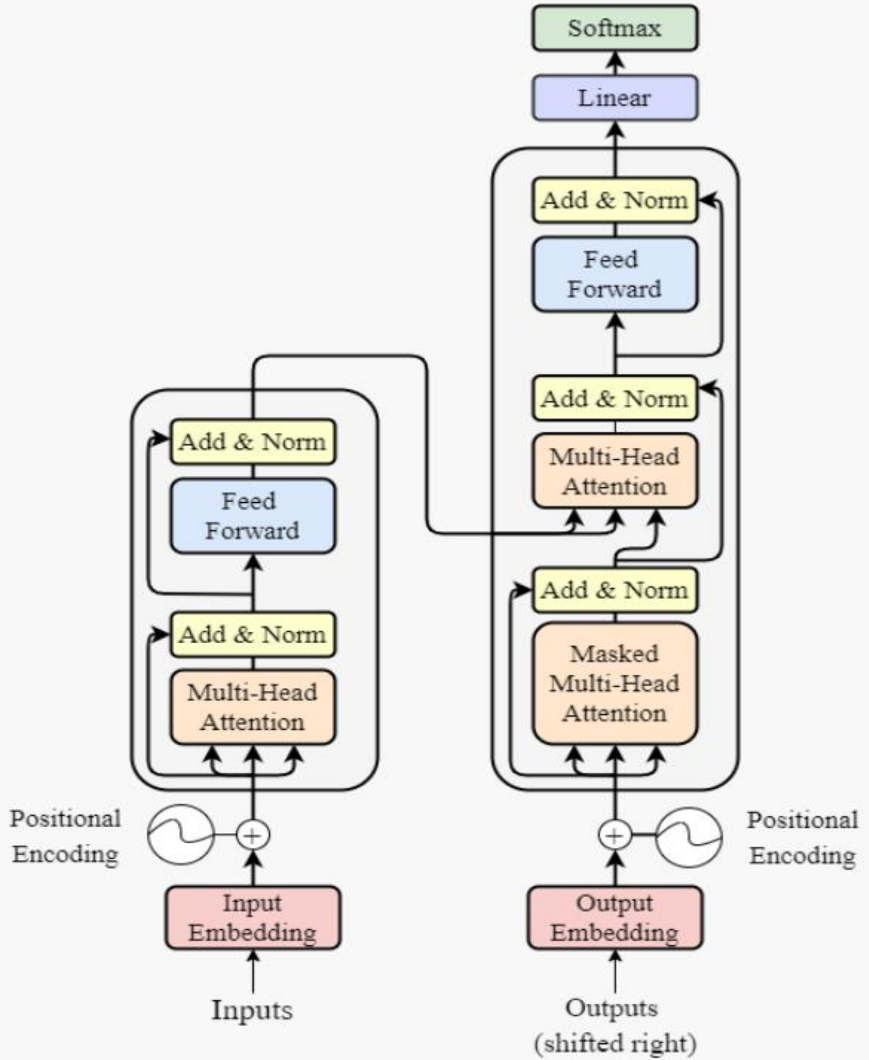


Limitations of LLM: LLMs: Better at Code Generation, Worse at Calculations

1. LLMs excel at writing code by treating it as a language prediction problem, leveraging the patterns and structures learned from vast code repositories (like Github) to generate coherent syntax and boilerplate.

2. However, they are fundamentally poor at complex calculations because their core function is prediction, not logical computation. They do not perform arithmetic; instead, they generate a plausible-looking answer based on memorized patterns.

3. It often leads to inaccurate or "hallucinated" results for complex math problems. This distinction highlights that LLMs are masters of pattern recognition and language fluency, but they lack true numerical reasoning capabilities.

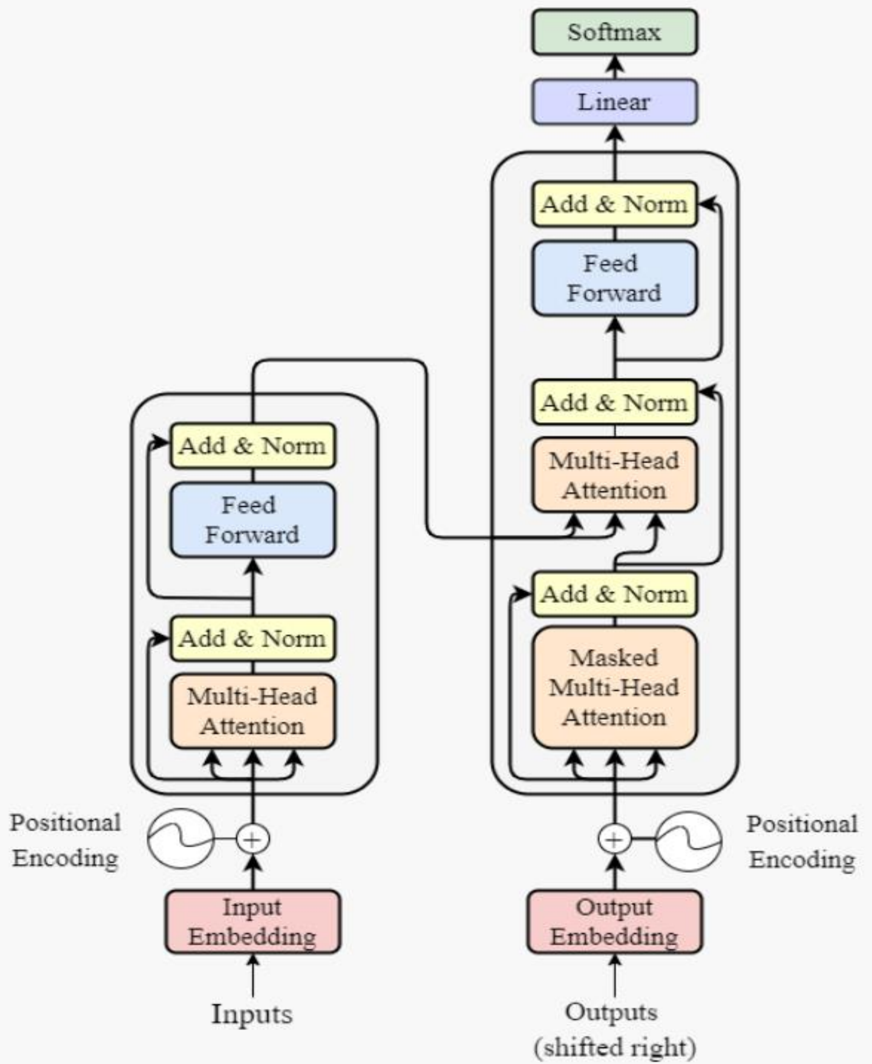


Inference optimizations (quantization, distillation)

Inference is the process of using a trained model to make predictions (i.e., generating a response to a user query). Since LLMs are so large, inference can be slow and expensive. Optimizations are techniques to make this process more efficient:

1. **Quantization** reduces the precision of the model's weights (e.g., from 32-bit to 8-bit numbers), which drastically cuts down on the memory and computational requirements without a significant loss in performance.

2. **Distillation** involves training a smaller, "student" model to mimic the behavior of a larger, more complex "teacher" model. This creates a much smaller, faster model that can be deployed more easily while retaining much of the performance of the original.

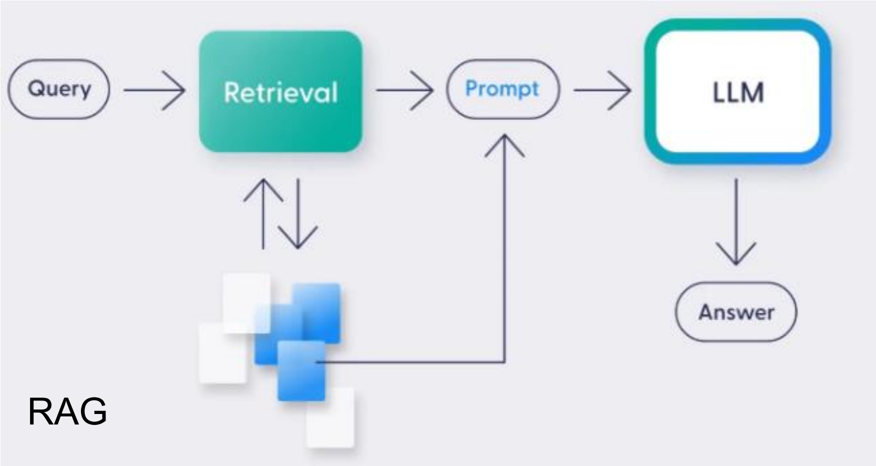


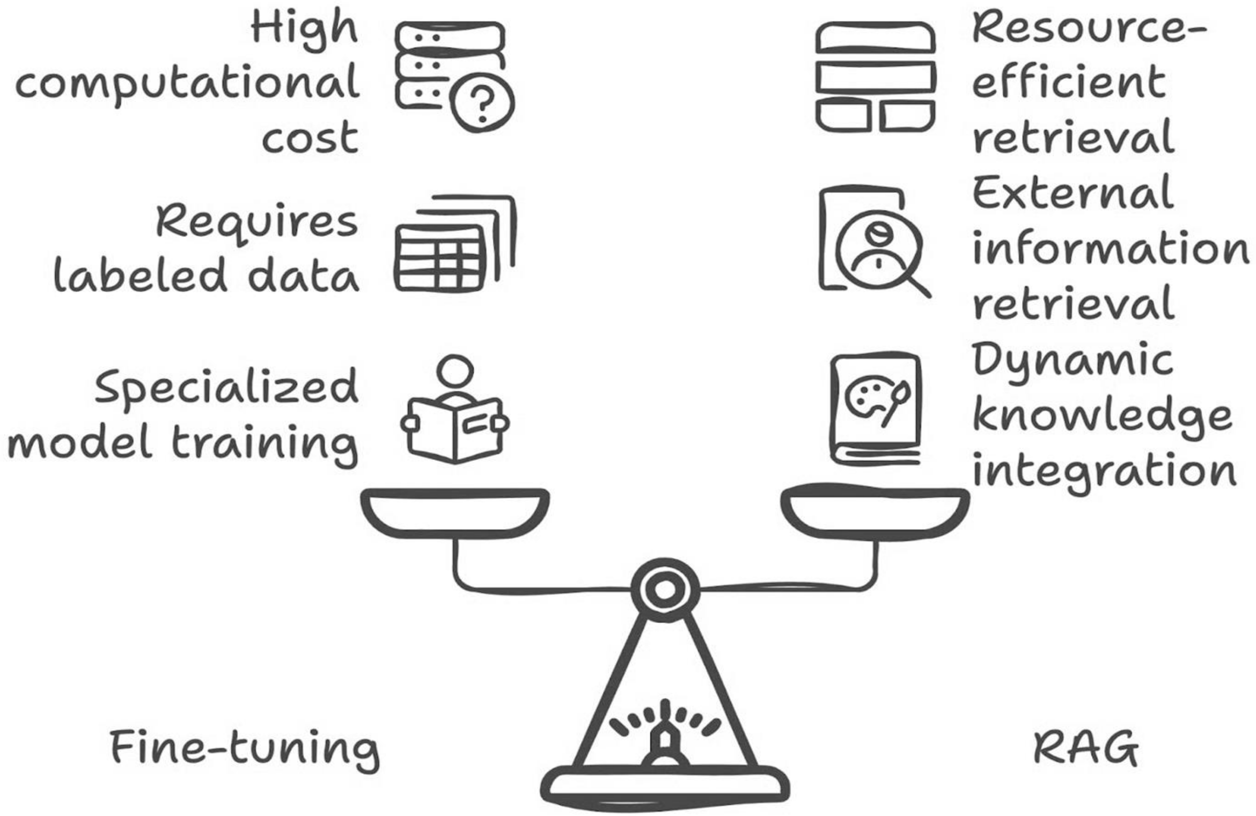
Enhancing LLMs

Two ways to enhance an **LLM** to a specific domain or task :

1. **Retrieval-Augmented Generation (RAG)** :
   RAG **augments** the LLM's prompt with new information at the time of a query

2. **Fine-tuning:**
   permanently updates the LLM's internal knowledge and behavior by adjusting its weights.





Choose the right approach for your AI task.

Feature	RAG	Fine-Tuning
Method	Retrieves external data and adds it to the prompt.	Adjusts the model's internal weights with new data.
Knowledge	External, dynamic, and up-to-date.	Internal, static, and fixed at training.
Cost	Low (relative to fine-tuning).	High (requires significant compute).
Use Case	Q&A over private documents, real-time data access.	Adapting model's style, format, and highly specialized skills.
Result	More factual and transparent.	More consistent and deeply specialized.

Benefit of Fine-tuning Your Own Model

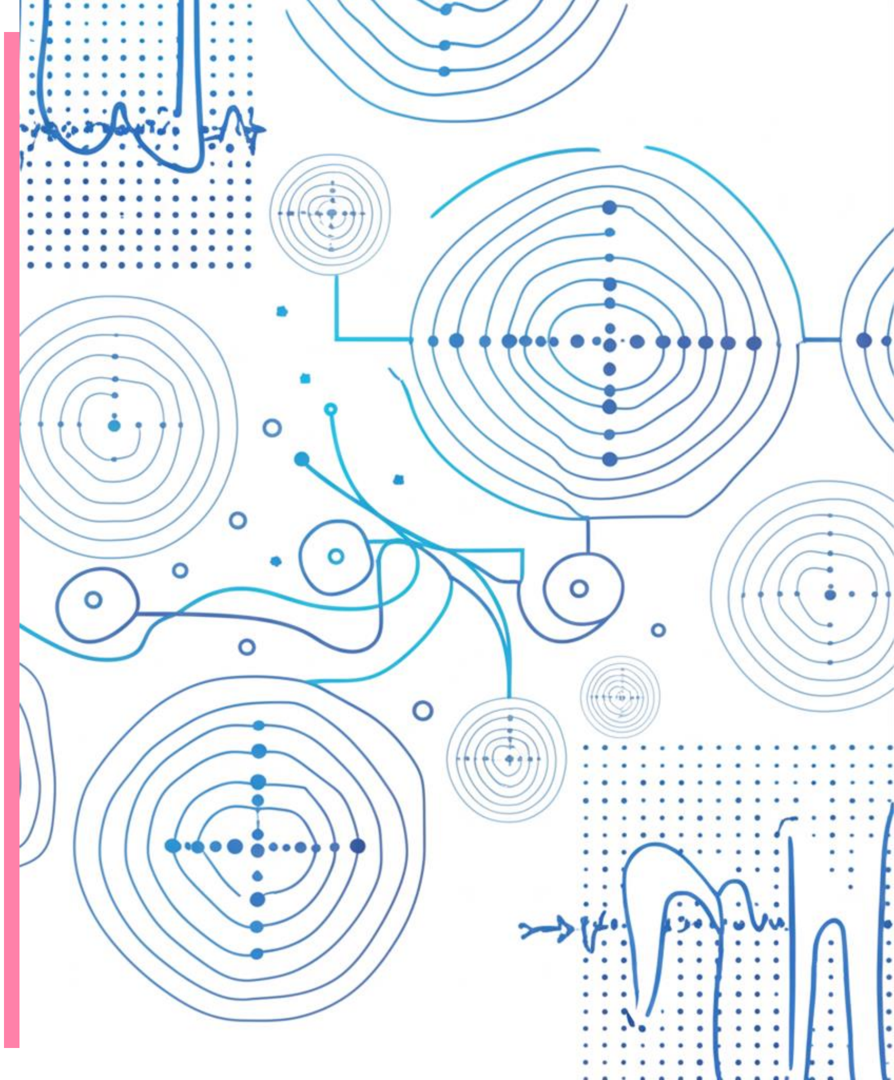
- Performance
  - Less Hallucination
  - Increase Consistency
  - Reduce unwanted information
- Privacy
  - On Prem
  - Prevent Leakage
  - No breaches
- Reliability
  - Control Uptime
  - Lower Latency
  - Increased Transparency
  - Greater Control

Steps To Fine-tune LLM

1. **Figure out the task** – Define the goal clearly (classification, summarization, Q&A, etc.).
2. **Data collection related to the task** – Gather input/output pairs relevant to the task.
3. **Data generation** (if required) – Use augmentation or synthetic data if natural data is limited.
4. **Fine-tune a small model** (e.g., 50M–1B parameters) – Start small before scaling.
5. **Vary the amount of data** – Experiment with different dataset sizes.
6. **Evaluate the model performance** – Use benchmarks, test sets, or human evaluation.
7. **Collect more data to improve** – Expand and refine datasets based on errors.
8. **Increase task complexity** – Gradually move to more challenging tasks.
9. **Increase the model size** – Use larger models once smaller ones perform well.

Zero/One/Few-shot learning & Prompt Engineering

- **Zero-shot learning:** Providing a prompt that isn't part of the training data.
  - Example: Asking the model open-ended questions without examples
- **One/Few-shot learning**: Supplying one or a few examples in the prompt for guidance
  - Example: Asking the model to format text while providing a few demonstration examples
- **Prompt engineering**: The art of designing prompts (with or without examples) to get the desired response from the model



Various Fine-Tuning Methods

Parameter-Efficient Fine-Tuning (PEFT)

a. **LoRA (Low-Rank Adaptation):** This method freezes the original LLM weights and injects small, trainable "adapter" matrices into each Transformer layer.

a. **Adapters:** This involves inserting tiny neural network modules between the existing Transformer layers. Only these new modules are trained, leaving the original model weights frozen.

b. **Prompt-Tuning:** This technique freezes the entire LLM and learns a set of special input tokens, or "soft prompts," that are pre-pended to the input. This is the most parameter-efficient method.

$$\begin{bmatrix} -8 & -2 & -6 & 6 \\ -4 & -1 & -3 & 3 \\ 28 & 7 & 21 & -21 \\ 24 & 6 & 18 & -18 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \\ 7 \\ 6 \end{bmatrix} \times \begin{bmatrix} 4 & 1 & 3 & -3 \end{bmatrix}$$

Fine-Tuning Methods: LoRA vs QLoRA

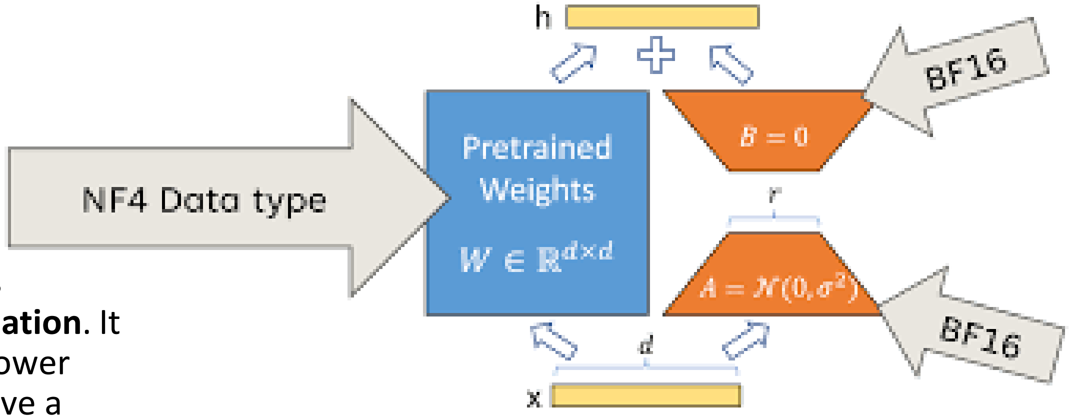
QLoRA is a more memory-efficient than LoRA

a. **LoRA (Low-Rank Adaptation)**

   Already explained

b. **QLoRA (Quantized LoRA)**

   QLoRA is an evolution of LoRA that takes efficiency a step further through **quantization**. It first quantized the large, base LLM to a lower precision (e.g., from 16-bit to 4-bit) to save a massive amount of memory. It then applies the LoRA fine-tuning method on top of this quantized model. This allows for fine-tuning a very large model on a single consumer-grade GPU.



Fine-Tuning Methods: LoRA vs QLoRA

Key Differences

Feature	LoRA	QLoRA
Method	Fine-tunes a base model at its original precision (e.g., 16-bit).	Quantizes the base model to a lower precision (e.g., 4-bit) before applying LoRA.
Memory Usage	Very efficient.	**Extremely** memory efficient. Requires significantly less GPU memory than LoRA.
Performance	Can be slightly faster in terms of training time.	Generally has similar performance to LoRA, but may have a small trade-off in accuracy due to quantization.
Hardware	More accessible than full fine-tuning.	Makes it possible to fine-tune massive models (e.g., 65B parameters) on consumer hardware.

Fine-Tuning Methods: RLHF

RLHF (Reinforcement Learning from Human Feedback) uses human ratings of model outputs to align the model's behavior with human values and expectations, making it more helpful and safe.

Instruction tuning & RLHF → align with human expectations

Fine-tuning

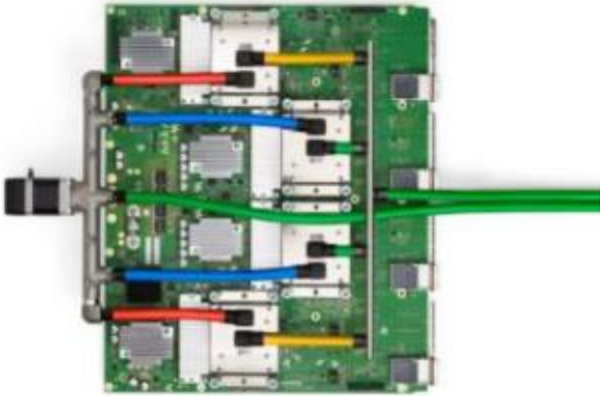
Error Analysis

- Understand the base model behaviour before finetuning
- Categorize errors: iterate on data to fix these problems in data.

Category	Example with Problem	Example Fixed
Misspelling	Your kidney is healthy, but you lever is sick, get your lever examined	Your kidney is healthy, but your liver is sick
Too Long	Diabetes is less likely when you eat a healthy diet makes diabetes less likely, making ………	Diabetes is less likely when you eat a healthy diet
Repetitive	Medical LLMs can save healthcare workers time and money and time and money and time and money.	Medical LLMs can save healthcare workers time and money

Hardware for industrial needs

- Nvidia GPU:
    - H100 up to 80GB RAM
    - Supports any framework
    - Available in any cloud
    - Requires NVLink/NVSwitch for efficient data/model parallelism
    - On-prem possibility
- Google TPU
    - More const efficient
    - V3-8 up to 128GB RAM
    - Support XLA only: Jax, PyTorch/XLA, TF
    - GCP lock
    - Supports data/model parallelism out-of-the-box



Read more: https://khairy2011.medium.com/tpu-vs-gpu-vs-cerebras-vs-graphcore-a-fair-comparison-between-ml-hardware-3f5a19d89e38

Pre-trained open-source LLMs

- Consider licenses that allow commercial use cases.
- A larger LLM has greater capabilities, but it also requires higher computing resources.
- A larger context window allows adding more information into context.
- Most of the attractive models:
  - Mistral with 7B params, 4096 tokens and 16K sliding window, Apache License 2.0
  - Gemma with 7B params, 8192 tokens, Google's Gemma Terms of use

Read mode: https://github.com/eugeneyan/open-llms

Libraries for fine-tuning

Library name	Company	Popularity ★	PEFT	DL Framework	Supported LLM models	Links
Deep Speed	Microsoft	31.5k	✓	PyTorch	A lot	docs, github
PEFT	HuggingFace⍰	12.7k	✓	PyTorch	LLaMA, Mistral, T5, GPT, others	blog, github, docs
Accelerate	HuggingFace⍰	6.6k	✗	PyTorch	A lot	github, docs
NeMo	Nvidia	9.4k	✓	PyTorch	LLaMA, Falcon, T5, GPT, others	docs, github
T5X	Google	2.3k	?	JAX	T5 and some others, PaLM*	paper, github, docs
Paxml	Google	0.3k	?	JAX	PaLM-2*	docs, github

Supervised fine-tuning in clouds

Cloud	LLM Model
Azure	GPT, Llama
AWS Bedrock	Amazon Titan, Anthropic Cloude, Cohere Command, Meta Llama [link]
GCP Vertex AI*	PaLM, Gemma, T5, Gemini**, Llama
OpenAI Platform	GPT
Anthropic	Claude
Cohere	Command
MosaicML	MPT

* - supports RLHF
** - coming soon

What's LoRA?

LoRA

It is too expensive to fine-tune all parameters in a large model.

- During fine-tuning we initialized with pre-trained params $\Phi_0$and $\Phi_0 + \Delta\Phi$ updated to by following the objective: $\max_{\Phi} \sum \sum \log(p_{\Phi}(y_t | x, y_{<t}))$
- We can hypothesize that the update matrices in LM adaptation have a low "intrinsic rank", leading to Low-Rank Adaptation (LoRA)
- For each downstream task, we do not need to store/deploy a different set of $\Delta\Phi$ where $|\Phi_0| = |\Delta\Phi|$

Can we find a param-efficient approach by low intrinsic rank?

$$\boxed{\Phi'} = \boxed{\Phi_0} + \boxed{\Delta\Phi}$$

LoRA Research Base Article

- 2021 Article
- https://arxiv.org/pdf/2106.09685

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

1 Introduction

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

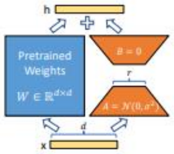


Figure 1: Our reparametrization. We only train A and B.

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

LoRA in Training and Inference

Previous study shows that

- Pre-trained LLMs have a "low intrinsic dimension"
- LLMs can still learn efficiently despite a low-dim reparametrization

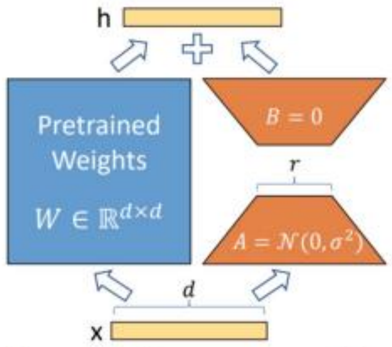


Figure 1: Our reparametrization. We only train A and B.

During training: for pre-trained weight $W_0 \in \mathbb{R}^{d \times k}$, W_0 is fixed

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

$$B \in \mathbb{R}^{d \times r},\ A \in \mathbb{R}^{r \times k},\ r \ll \min(d, k)$$

During inference:

$$W = W_0 + BA$$

Conclusion

- **LoRA + QLoRA = efficient fine-tuning**
- **Experiment tracking with wandb**
- **Inference Flexibility**

After finetuning, you can either:

1. Load **base model + LoRA adapters** (lightweight).
2. Or **merge adapters into the base weights** (single self-contained model).