What is Model Training in AI?

Wednesday, April 23, 2025 6:39 PM

→ Process involved in the model Building:

- **Data Ingestion**
- **Data Analysis**
- **Data Preprocessing**
- **Model Building**
- **Model Evaluation**

→ Model building in Classical Machine Learning:

✓ Supervised Modelling

- Linear Regression
- Logistic Regression
- Support Vector Machine
- Decision Tree
- Random Forest
- Light Gradient Boosting Machine(LGBM)
- Xtreme Gradient Boosting Machine(XGBM)
- Naïve Bayes

✓ Unsupervised Modelling

✓ Clustering: K-Means, Hierarchical Clustering, DB Scan Clustering

→ Model Building in Deep learning

✓ ANN: Artificial Neural Network(ANN) for Regression and Classification

✓ CNN: Convolution Neural Network(CNN) for Grid Like Data Ex. Images

We used CNN-based architectures to solve tasks such as:

- **Image Classification**
- **Object Detection**
- **Object Segmentation**
- **Object Tracking**
- **Optical Character Recognition (OCR)**

Image Classification	LeNet-5, AlexNet, VGG-16/19, ResNet (e.g., ResNet-50), DenseNet, EfficientNet
Object Detection	R-CNN, Fast R-CNN, Faster R-CNN, YOLO (v1–v8), SSD, DETR
Object Segmentation	FCN, U-Net, SegNet, Mask R-CNN, DeepLab (v1–v3+)
Object Tracking	SORT, Deep SORT, SiamMask, SiamRPN, Tracktor++

Optical Character Recognition (OCR) Tesseract OCR, CRNN, EAST, CRAFT, PaddleOCR

- ✓ RNN: Recurrent Neural Network RNN
- ✓ Variant of RNN: LSTM & GRU for Sequence Like Data Ex. Text
- ✓ We use this architecture for Sequence to Sequence learning

Using **RNN, LSTM, and GRU architectures**, we built advanced models such as **Encoder-Decoder** and **Encoder-Decoder with Attention**, which were applied to various NLP tasks including:

- **Text Classification**
- **Text Summarization**
- **Question Answering**
- **Text Generation (e.g., Next Word Prediction)**
- **Text Translation**

This process is known as **Language Modeling**.
Keep the following points in mind:

a. It involves **training the model from scratch** on a large corpus.
b. It is also referred to as **task-specific training**, as the model is optimized for a particular language task.

One of the key milestones was the idea of **Universal Language Modeling using Encoder-Decoder architectures**. This is where the foundation of **fine-tuning in NLP** was laid out.
The typical process looked like this:

a. Pretraining Phase (Unsupervised)
   - The model was first **pretrained** on large-scale text using unsupervised tasks like **machine translation**.

b. Fine-tuning Phase (Supervised)
   - Then, the same model was **fine-tuned** on specific downstream tasks, such as **text classification**, using labeled data.

c. Model Architecture
   - The architecture used for this training was the **Encoder-Decoder with Attention** mechanism.

However, there were some limitations with this early approach:

a. The **pretraining task was limited to machine translation**, which restricted generalization to other tasks.
b. The model architecture was **based on LSTM**, which struggled with **long-range dependencies** and was **computationally inefficient**.

As a result, these models didn't achieve the breakthrough performance we see today.

Then Came the Game-Changer: The Transformer

Introduced in the paper **"Attention is All You Need"**, the **Transformer architecture** revolutionized NLP.

- Unlike LSTMs, **Transformers do not rely on recurrence**.
- Instead, they use a novel concept called **self-attention**, which allows the model to attend to all positions in the input sequence simultaneously — enabling much faster and more effective learning.

Stage	Components
Input Layer	Token Embedding, Positional Encoding
Encoder Layer	Multi-Head Self-Attention, Add & Norm, FFN

Decoder Layer Masked Self-Attention, Encoder-Decoder Attention, Add & Norm, FFN

Output Layer Linear, Softmax

After the **successful release of the Transformer architecture**, which proved to be a highly powerful model, several **variants** started emerging — such as **BERT**, **GPT**, **T5**, and **XLM**.

These models marked the beginning of what we now call the **initial wave of Large Language Models (LLMs)**.
While they weren't as massive as today's models (like GPT-4), they were considered a **major milestone** in the evolution of large-scale language modeling.

Unlike earlier models that relied heavily on **machine translation**, these new models introduced **more sophisticated pretraining objectives**, such as:

- **Masked Language Modeling (MLM)** → used in BERT
- **Causal Language Modeling (CLM)** → used in GPT

This shift in strategy helped these models learn **deep contextual representations** of language, making them highly effective across a wide range of downstream NLP tasks.

Understanding the Evolution of LLMs: A Quick Story

a. **BERT**
   - **Pretraining:** Used **Masked Language Modeling (MLM)** — where random words in a sentence are masked and the model learns to predict them.
   - **Fine-tuning:** Then fine-tuned for **downstream tasks** like **text classification**, **NER**, etc.
   - **Encoder-based** model.

b. **GPT**
   - **Pretraining:** Used **Auto-Regressive Language Modeling (CLM)** — predicting the **next word** in a sequence.
   - **Further Training:** Later versions were trained on **conversation-style datasets** to enable dialogue understanding.
   - **Decoder-only** model.

Then Came ChatGPT

Built on top of **GPT**, ChatGPT leveraged a **decoder-based architecture** and was trained with techniques like:

- **Instruction tuning**
- **Reinforcement Learning from Human Feedback (RLHF)**

This marked the **beginning of the modern LLM era**, where models were not just pretrained and fine-tuned — they were made to **follow instructions**, engage in **dialogue**, and perform **multi-turn reasoning**.

Modern transformer-based LLMs are trained through **three major stages**. Let's break them down:

Stage 1: Pretraining Phase (Foundation Model Building)

This stage focuses on **teaching the model language itself** using **huge amounts of raw text**.

- **Data Collection**
  - Massive-scale datasets (web text, books, code, forums, etc.)
- **Data Cleaning**
  - Remove low-quality, duplicated, or irrelevant content

- **Data Analysis**
  - Check for distribution, biases, toxicity, imbalance, etc.
- **Data Preprocessing**
  - Tokenization, normalization, filtering, formatting for training
- **Unsupervised Pretraining**
  - Using architectures like GPT (decoder-only)
  - The model learns to generate coherent and meaningful text
- **Evaluation**
  - Perplexity, loss, token accuracy on held-out validation data

What is Pretraining?

- **Unsupervised learning phase**
- Typically uses **Auto-Regressive Language Modeling** (e.g., predict the next token)
- Builds a **foundation model** that understands grammar, facts, and reasoning
- This model is referred to as a **Pretrained Language Model (PLM)**

Stage 2: Supervised Fine-Tuning (SFT)

Now the base model is refined to follow human instructions for specific tasks.

- **Supervised Finetuning** using labeled datasets
- Tasks include classification, summarization, QA, reasoning, dialogue
- Often uses techniques like **PEFT (Parameter Efficient Fine-Tuning)** — e.g., LoRA, QLoRA

Model Evaluation

- Performance is evaluated on benchmarks like HELM, MMLU, SuperGLUE, etc.

Stage 3: Alignment (Controlled & Human-like Responses)

To ensure **safe, helpful, and human-aligned** outputs:

- **RLHF (Reinforcement Learning from Human Feedback)**
  - Use **PPO (Proximal Policy Optimization)** to align responses with human preferences
- 🔁 **Modern Alternative: DPO (Direct Preference Optimization)**
  - Simpler and more stable than PPO, now widely used in 2024+

Model Evaluation

- Human evals, reward models, safety & helpfulness scores

Welcome to the LLM Era!

Following this pipeline, we've seen the rise of powerful open and closed-source LLMs:

- **LLaMA Series** (Meta)
- **Mistral Series(Mistral AI)**
- **Gemini Series** (Google DeepMind)
- **DeepSeek Series**
- **Claude Series** (Anthropic)
- **GPT Series** (OpenAI)

age	Description
1. Pretraining	On large corpus using MLM (BERT) or CLM (GPT) with unsupervised objectives
2. Fine-tuning	On specific tasks (e.g., classification, QA, summarization) with labeled data
3. Instruction Tuning	Train model to follow human instructions effectively
4. RLHF (Optional)	Refine responses using **reinforcement learning with human feedback**
5. Continuous Training	New data added over time (optional, depending on use case)

✓ Autoencoders (AE) – *Compression + Reconstruction*

What is it?

An **Autoencoder** is a type of neural network that learns to **compress** input data into a smaller representation (**encoding**) and then **reconstruct** it back to the original input.

Architecture:

- **Encoder** → Compresses the input into a lower-dimensional code
- **Latent Space** → The "bottleneck" where compressed info is stored
- **Decoder** → Reconstructs the input from the compressed code

Use Cases:

- Image denoising
- Dimensionality reduction
- Anomaly detection
- Feature extraction

✓ GANs (Generative Adversarial Networks) – *Fake it till you make it*

What is it?

A **GAN** is a framework made up of **two neural networks** playing a game:

- A **Generator**: Tries to create **fake data** that looks real
- A **Discriminator**: Tries to **detect fake vs real** data

They compete with each other:

- Generator improves until it can fool the discriminator
- Discriminator improves until it can catch fakes

Architecture:

- **Generator** → Learns to produce realistic-looking data from random noise
- **Discriminator** → Learns to distinguish between real and fake data

Use Cases:

- Image generation (e.g., deepfakes, face synthesis)
- Text-to-image generation (e.g., DALL·E, Midjourney)
- Art & design automation
- Super-resolution and image inpainting

✓ What is Reinforcement Learning (RL)?

Reinforcement Learning is a type of **machine learning** where an **agent** learns to take actions in an **environment** to **maximize a reward**.

Agent: The decision-maker (e.g., robot, bot, game player)
Environment: The world the agent lives in (e.g., chess board, maze, stock market)
State: The current situation of the environment
Action: What the agent can do
Reward: Feedback signal (positive or negative)

1. Agent observes the state
2. Agent takes an action
3. Environment gives a reward + next state
4. Agent learns and updates its policy

Goal: Learn a **policy** (strategy) that chooses the best actions to **maximize long-term rewards**.

Deep Learning(NN) = "Learn patterns from data"
Reinforcement Learning = "Learn from experience to make decisions"

What is Pretraining?

Wednesday, April 23, 2025 6:47 PM

What is Pretraining?

1. Pretraining in Computer Vision(CNN)

- Pretraining is when a Convolutional Neural Network (CNN) is trained on a **large, general dataset** like **ImageNet** to learn **basic visual features** like edges, textures, shapes.
- This pretrained model is then used as a **starting point** for a **new task** (called transfer learning).
- **VGG16 / VGG19**
- **ResNet50 / ResNet101**
- **InceptionV3**
- **MobileNet**
- **EfficientNet**

Layer Level	Feature Type	Description
Early Layers	**Primitive / Low-Level**	Basic shapes and textures (edges, lines, corners)
Mid Layers	**Intermediate Features**	Patterns, curves, blobs, combinations of lines
Deeper Layers	**Specific / High-Level**	Meaningful parts: eyes, faces, wheels, logos etc.

2. Pretraining in Large Language Modelling(LLM)
Pretraining is the **first training phase** of a Large Language Model (LLM), where the model learns **language structure, grammar, facts, and reasoning ability** from huge amounts of **unlabeled text data**.

Massive Text ➔ Tokenizer ➔ Transformer Stack ➔ Predict next token ➔ CrossEntropy Loss ➔ Backprop ➔ Repeat

Step	What Happens
Data Collection	Crawl massive text
Tokenization	Break text into chunks
Transformer Model	Setup deep neural net
Objective	Predict next token (causal or masked)
Loss	Cross-entropy tells model what it got wrong
Training	Billions of times, on GPUs
Evaluation	Check if model learned to read & write
Save	Final model saved as pretrained weights

Model	Pretraining Type	Special?
GPT-3	Causal	175B params
BERT	Masked	Bidirectional
T5	Span Masking	Text-to-text
PaLM	Causal	Trillion-scale
LLaMA	Causal	Open source style
Mistral	Causal	Sliding window attention

Dataset Collection

- Common Crawl (web scraped data)
- Wikipedia
- News Articles
- Social Media Posts (Reddit, etc.)
- Books (open domain, Project Gutenberg, etc.)
- GitHub code (for Codex-style models)

Why Do We Say "Unsupervised"?

Because:

- There's **no human-annotated label**
- Output (e.g., next token) is **generated from the input text itself**
- Labels = **automatically created**

Self-Supervised = Better Word

Technically, ye jo “unsupervised pretraining” hota hai, usko **Self-Supervised Learning** kehna zyada accurate hota hai.
Because:

Labels are derived from the data itself (by masking, shifting, etc.)

Model Type	Training Task
GPT	Predict next token (Causal LM)
BERT	Predict masked word (Masked LM)
T5	Fill in missing span (Span Masking)
CLIP	Match image & text pair (contrastive)

"Unsupervised pretraining" means training a model on massive raw text without any manual labels, where the model learns patte rns by predicting parts of the input itself.

Simple Analogy First: “Self-Taught Student”

Soch tu ek student hai jiske paas **sirf books** hain, koi teacher nahi.

- Koi bolne wala nahi ki "ye sahi answer hai"
- Tu bas **khud padh padh ke pattern samajhne lagta hai**
- Har baar agla shabd guess karta hai → aur check karta hai sahi tha ya nahi

Why This Works Like Magic?

Language has **internal structure**:

- Grammar
- Word order
- Semantics
- Relationships

LLM **implicitly learns**:

- English syntax
- Common facts (e.g., "India is a country", "Dogs bark")
- Reasoning, math, even coding patterns

Sab **self-learned** from raw, unlabeled text.

What is Transfer Learning?

Wednesday, April 23, 2025 6:48 PM

Transfer Learning (Simple Definition):

Transfer Learning is when we:

- First **pretrain a model** on a large, general-purpose dataset
- Then, we **adapt it to a specific task** by either:
    - **Replacing the output layer of pretrain model**
    - Or don't change weights means keep freezing all the weights just retrain the Neural Network Weights
    - Or **unfreezing some layers** of the network and **fine-tuning** it on the new, task-specific dataset

Why Do We Use Transfer Learning?

- Saves **training time and resources**
- Works well when **labeled data is limited**
- Gives **better performance** than training from scratch

Advantage and Disadvantage of Finetuning

Wednesday, April 23, 2025 6:49 PM

1. Higher Accuracy on Specific Tasks

- Fine-tuning makes the model **highly optimized for your specific task**, leading to significantly **better accuracy**.
- Pretrained models offer **generic capabilities**, but fine-tuned models solve **targeted problems**.

Slide Points:

- **Task-specific optimization**
- **Improved performance on custom datasets**

2. Domain Adaptation

- Fine-tuning allows the model to learn your **domain's language, tone, and terminology** — whether it's **medical**, **legal**, **finance**, or more.
- You can apply LLMs effectively across **any industry** with domain adaptation.

Real-life Analogy:

It's like training a cricketer to play football —
fine-tuning is the shortcut for that transition.

3. Saves Cost vs. Training from Scratch

- Training an LLM from scratch is extremely expensive — both in terms of **compute** and **data**.
- Fine-tuning lets you adjust just the **final layers** of a pretrained model, making it **cost-efficient and fast**.

Slide Points:

- **Lower compute cost**
- **Faster development cycle**

4. Custom Behavior & Branding

- You can give your product, assistant, or chatbot a **unique brand voice** — whether it's quirky like **Zomato** or formal like **SBI**.
- Fine-tuning allows you to build a model with **your brand's personality**.

Slide Points:

- **Personalized responses**
- **Brand-aligned AI assistant**

5. Control Over Responses

- Fine-tuning helps eliminate **unwanted behavior** like **toxicity**, **bias**, or **hallucinated outputs**.
- You define the type of responses your model should give — making it **safe, trustworthy, and aligned** with your guidelines.

Slide Points:

- **Reduce bias & hallucinations**
- **Controlled, safe output behavior**

1. Requires GPU/TPU Resources

- Fine-tuning requires **hardware acceleration** (like GPUs or TPUs).
- Without it, local training can be **slow or practically infeasible**.

Slide Points:

- **Needs GPU/TPU or Colab Pro**
- **Slow without proper setup**

2. Overfitting Risk

- If your dataset is **too small** and training goes too long, the model starts **memorizing** instead of learning — this is **overfitting**.
- It reduces the model's ability to **generalize to new inputs**.

Analogy:

Ratta maarna = Overfitting
Understanding the concept = Proper Fine-Tuning

3. Hyperparameter Tuning is Tricky

- Choosing the right **learning rate**, **batch size**, **number of epochs**, etc., is crucial.
- It might seem easy, but proper tuning is an **art**, not just science.

Slide Points:

- **Right tuning = better results**
- **Wrong values = poor performance**

4. Catastrophic Forgetting

- If not done carefully, fine-tuning can cause the model to **forget its original pretrained knowledge**.
- This is common when there's a **big domain shift** or no preservation technique is used.

Example:

"You taught it medicine, now it forgot how to write basic English."

Slide Points:

- **Preserve base knowledge**
- **Avoid domain wipeout**

5. Data Requirements & Cleaning

- Your dataset needs to be in the **proper format** (clean, consistent, aligned with your task).
- **Noisy or messy data** can make fine-tuning totally **ineffective**.

Slide Points:

- **Clean, structured data is key**
- **Inconsistent data ruins results**

LLM Finetuning Library or Framework

Framework / Library	Use Case / Highlights
Hugging Face Transformers	Most popular for fine-tuning; supports LoRA, QLoRA, PEFT, Trainer API, etc.
PEFT (by Hugging Face)	Lightweight library for **Parameter-Efficient Fine-Tuning** (LoRA, Prefix, Adapters)
Accelerate	Multi-GPU, mixed precision, distributed training made easy
trl (Transformers Reinforcement Learning)	Used for **RLHF** & reward modeling (e.g., PPO, DPO)
Llama Factory	Fine-tune LLaMA, Mistral, Qwen, Baichuan, etc. with many backends
Unsloth	Fastest & memory-efficient LLaMA fine-tuning (QLoRA + Flash Attention 2)
FastChat (by LMSYS)	Great for instruction-tuning, ChatGPT-style chatbots
DeepSpeed (by Microsoft)	Efficient large model training with ZeRO, MoE, 3D-parallelism
Colossal-AI	Memory-efficient training for huge models, supports ZeRO, LoRA
OpenLLM (by BentoML)	Fine-tune + serve LLMs easily in production
Axolotl	Config-based LLM fine-tuning with support for multiple models
SkyPilot	Easy training on spot GPUs across AWS, GCP, Lambda, etc.
LightLLM / vLLM	More for **inference**, but often paired post-finetuning

Pick Based on Your Need

Goal	Recommended Tools
General Fine-tuning	Hugging Face Transformers, PEFT
Speed & Efficiency	Unsloth, DeepSpeed, QLoRA
RLHF (ChatGPT-style)	trl, FastChat
LoRA/QLoRA Fine-tuning	PEFT, LlamaFactory, Unsloth, Axolotl
Multi-GPU / Distributed Setup	DeepSpeed, Colossal-AI, Accelerate
Model Serving after Finetune	OpenLLM, BentoML, vLLM

Important Research Paper on Finetuning

Wednesday, April 23, 2025 6:49 PM

BERT (2018) – *Pre-training of Deep Bidirectional Transformers*	MLM + NSP pretraining, then fine-tuning on downstream tasks	Foundational Fine-Tuning
GPT (2018) – *Improving Language Understanding by Generative Pre-Training*	Autoregressive LM + supervised fine-tuning	Foundational Fine-Tuning
ULMFiT (2018) – *Universal LM Fine-tuning for Text Classification*	First practical fine-tuning transfer learning in NLP	Foundational Fine-Tuning
T5 (2020) – *Exploring Limits of Transfer Learning with Text-to-Text*	Unified text-to-text framework, pretrained on C4, then fine-tuned	Foundational Fine-Tuning
LoRA (2021) – *Low-Rank Adaptation*	Trains a few matrix parameters (low-rank) → lightweight fine-tuning	PEFT (Parameter-Efficient FT)
Adapters (2019) – *Parameter-Efficient Transfer Learning for NLP*	Inserts trainable adapter layers into frozen models	PEFT
Prefix-Tuning (2021) – *Optimizing Continuous Prompts for Generation*	Only trains a prefix of the input → model body stays frozen	PEFT
QLoRA (2023) – *Efficient Finetuning of Quantized LLMs*	Combines 4-bit quantization + LoRA → huge memory savings	PEFT + Efficient Training
InstructGPT (2022) – *Fine-Tuning LMs from Human Preferences*	Combines SFT + PPO-based RLHF for human-aligned responses	RLHF (Fine-Tuning with Human Feedback)
DPO (2023) – *Direct Preference Optimization*	Simpler alternative to PPO for aligning models with preference data	RLHF (Modern Alignment)
Self-Instruct (2022)	Models self-generate instructions to fine-tune without human labels	Instruction Tuning
Scaling Laws for Neural Language Models (2020)	Shows how model size, data, and compute scale performance	Theory + Model Scaling
Efficient Training to Fill in the Middle (2022)	Strategy to fine-tune models on "fill-in-the-middle" tasks	Specialized Fine-Tuning
GPT-3 (2020) – *Language Models are Few-Shot Learners*	Demonstrated prompting over fine-tuning → gave rise to RAG, Agents	Prompting vs Fine-Tuning Philosoph

Tips for Finetuning

Wednesday, April 23, 2025 6:49 PM

Wednesday, April 23, 2025
6:49 PM
1 Dataset
2 Training Strategy-> Use “SFT → DPO” Two-Stage Tuning
3. Hyperparameter
4. Hardware