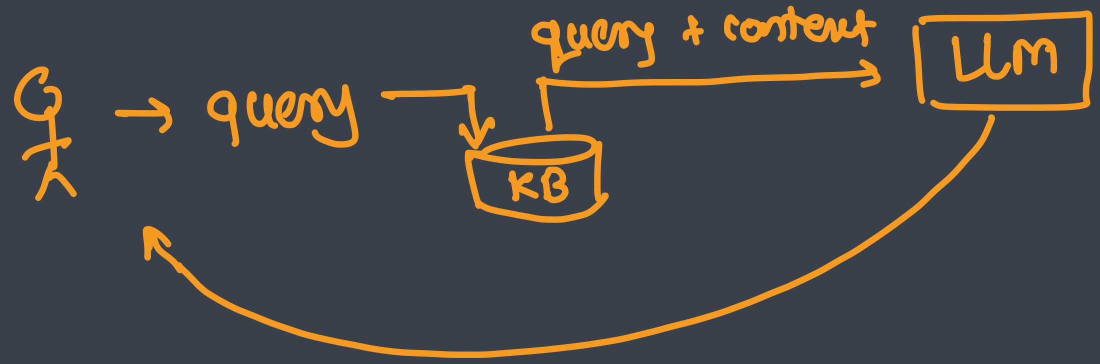
Fine Tuning LLM

· RAG

- dynamic data
- changing data
- small data (content)



transfer learning

RAG
- naive
- agentic

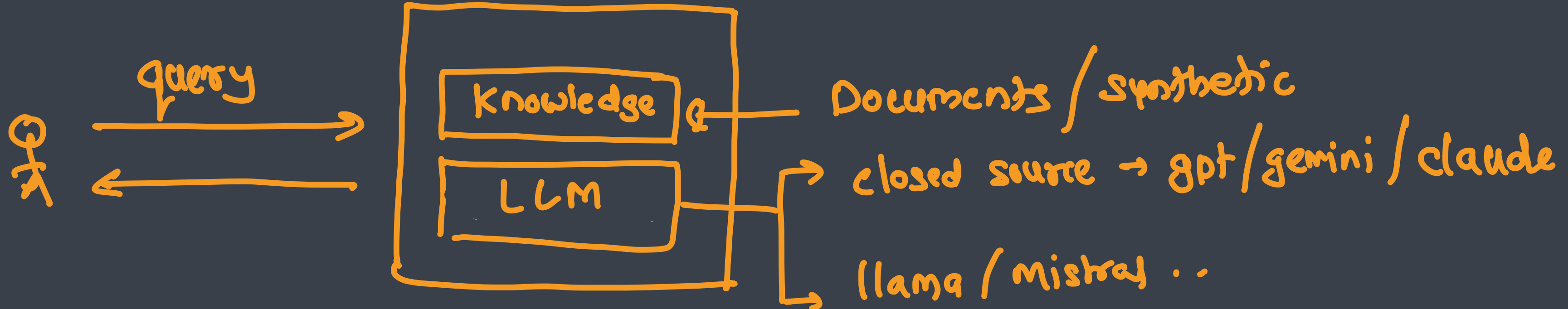
outdated
cutoff date

limited knowledge

Fine tune

- large/Huge data
- static data

fine tuned LLM



What is Fine Tuning ?

LLM- gpt/claude/..

- Fine-tuning a Large Language Model (LLM) involves taking an existing pre-trained model and adapting it to a specific task or domain → legal
- Instead of training the model from scratch, which requires a significant amount of data and computational resources, fine-tuning leverages the knowledge learned during the initial pre-training phase and applies it to a smaller, task-specific dataset → transfer learning
- Fine-tuning is particularly useful for tasks like text classification, named entity recognition, question answering, summarization, and translation where large volumes of task-specific data are available
- By fine-tuning, we can improve the model's performance on these specific tasks without having to start from scratch
- Fine-tuning also helps to reduce the risk of overfitting to a particular dataset since the initial pre-training phase already provides a general understanding of language structure and context

Reasons

accuracy / speed

- Improved Performance: Fine-tuning can help improve the performance of a pre-trained large language model on specific tasks or domains, as it adapts the model's weights based on the task-specific dataset, making it more accurate and effective in solving the target problem
- Reduced Training Time and Resources: Fine-tuning requires fewer resources than training a large language model from scratch because it starts with an already trained model that has learned general linguistic patterns. This means faster convergence and lower computational costs during the fine-tuning process.
- Generalization Ability: Fine-tuning helps to improve the generalization ability of the model by allowing it to learn task-specific patterns and nuances from the task-specific dataset while still retaining its understanding of language structure and context gained during pre-training
- Customization: Fine-tuning enables customization of a pre-trained large language model for specific use cases, such as chatbots, search engines, or text editors, ensuring that the model can effectively handle the unique requirements of each application
- Adaptation to Changing Data: As data and user needs evolve over time, fine-tuning provides a way to quickly adapt pre-trained models to new conditions without requiring extensive retraining from scratch

Process of Fine Tuning

→ Data Source → API / Scraping / databases

→ organized → jsonl format / custom format

labelled

- Data Collection and Preparation → NLP pipeline → cleanse the data
  - Gather a task-specific dataset relevant to the target problem
  - This data should be labelled, meaning that it contains examples of both correct and incorrect responses for each task. The data may need to be cleaned, preprocessed, and split into training, validation, and testing sets → primary / secondory
- Pre-Trained Model Selection
  - Choose a pre-trained large language model (LLM) that has been trained on a general corpus of text
  - Common examples include BERT, RoBERTa, XLNet, or T5
  - The choice of model depends on the task and available resources

model < closed source / open source

- Adaptation
  - Modify the architecture of the pre-trained model to suit the specific task requirements, such as adding a new output layer for text classification or question answering tasks, or changing the input format to accommodate different data structures

Knowledge

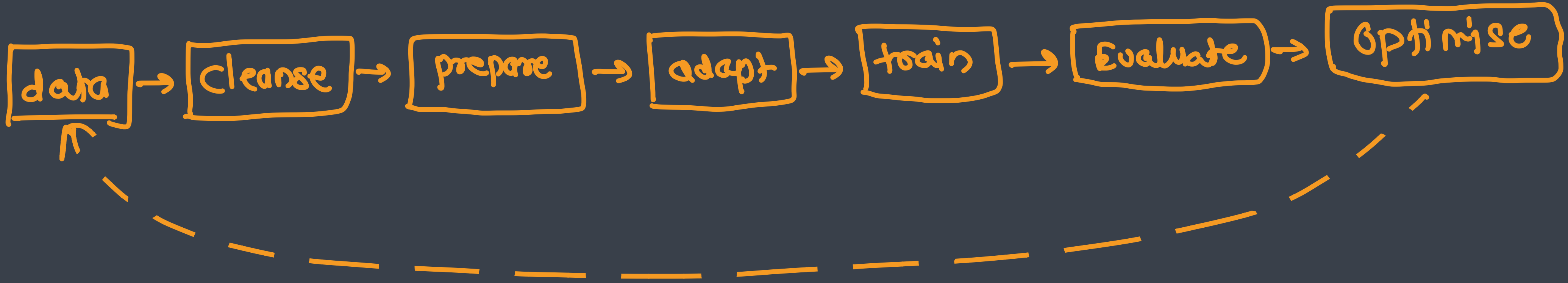
LLM

model = formula

$y = f(x)$

LoRA

- Fine-Tuning
  - Train the adapted model on the task-specific dataset using an appropriate optimization algorithm and learning rate schedule
  - Fine-tune for a limited number of epochs to avoid overfitting the training data
- Evaluation
  - Evaluate the fine-tuned model's performance on a test set or validation set, comparing its results with those obtained from the pre-trained model
- Iteration and Improvement
  - Based on the evaluation results, iterate by making adjustments to the data collection process, fine-tuning parameters, or architecture as needed to further improve the model's performance



Tools Needed

closed

open source

huggingface / closed

- Pre-trained Language Models
  - You will need a pre-trained large language model (LLM) such as BERT, RoBERTa, XLNet, or T5
  - These models can be obtained from various sources like Hugging Face's Transformers library, Google's BERT repository, or Stanford's CoreNLP
- Data Collection and Preprocessing Tools
  - You will need tools for collecting and preprocessing your task-specific dataset, which might include scripts for data scraping, text cleaning, tokenization, and other necessary transformations
  - Libraries like Pandas, NumPy, or Scikit-learn can be useful for these tasks → selenium / BS / spacy / Scrapy..
- Deep Learning Frameworks
  - You will need a deep learning framework to implement the fine-tuning process
  - Popular choices include TensorFlow, PyTorch, or Hugging Face's Transformers library, which provides an easy-to-use API for fine-tuning pre-trained models
- Evaluation Metrics
  - To measure the performance of your fine-tuned model, you will need appropriate evaluation metrics such as accuracy, F1 score, precision, recall, or perplexity depending on the task
- GPU or TPU
  - Fine-tuning large language models requires significant computational resources, so access to a powerful GPU or TPU is essential for efficient training
  - You can use cloud platforms like Google Colab, AWS SageMaker, or Azure ML Studio for GPU/TPU access without the need for local hardware

Quantization

weights = float value = $\frac{\text{FP32}}{\text{4 bytes}} \Rightarrow \frac{\text{FP8}}{\text{1 byte}}$

- Quantization is the process of approximating higher-precision numerical values with lower-precision ones to reduce computational complexity and memory requirements, making machine learning models more efficient and easier to deploy on devices with limited resources like mobile phones, embedded systems, or IoT devices
- There are two main types of quantization:
  - Quantization Aware Training (QAT): In QAT, the training process is adapted to incorporate the intended quantization scheme, allowing for better performance and fewer issues during deployment. This approach requires retraining the model with a lower-precision data type or quantizing activations and weights during backpropagation.
  - Post-training Quantization (PTQ): PTQ involves applying quantization after the model has been trained, without modifying the training process itself. The weights and activations are rounded to lower precision values, usually 8-bit integers or 16-bit floating-point numbers. Popular post-training quantization methods include linear quantization, logarithmic quantization, and k-means clustering-based quantization.
- Quantization can help reduce the storage requirements and compute power needed for inference, making machine learning models faster, more power-efficient, and cheaper to deploy on resource-constrained devices
- However, it comes at the cost of potential accuracy loss due to the approximations introduced by lower precision values

Hugging Face

- Hugging Face is a company and an open-source AI community that provides tools, models, and datasets for natural language processing (NLP) and machine learning (ML)
- It is best known for its **Transformers** library, which offers pre-trained models for tasks like text classification, translation, summarization, and question-answering
- **Key Features**
  - **Transformers Library** – A popular Python library that provides access to state-of-the-art models like GPT, BERT, T5, and more.
  - **Diffusers Library** – Focused on generative AI models, including Stable Diffusion for image generation.
  - **Datasets** – A collection of thousands of preprocessed datasets ready for training ML models.
  - **Inference API** – Allows users to deploy and run models without setting up their own infrastructure. → spaces
  - **Model Hub** – A repository of thousands of pre-trained models shared by the community and organizations.
  - **AutoTrain** – A no-code tool for training models easily.
  - **Gradio** – A tool for creating interactive AI demos and applications with minimal coding.

↳ alternative to streamlit

Monetizing Generative AI

- SaaS Model (Subscription-Based)
  - Examples: OpenAI’s ChatGPT Plus, Jasper AI, MidJourney, Runway.
  - **Tools:**
    - Backend: Python (FastAPI, Flask), Node.js
    - Frontend: React, Next.js
    - Model Hosting: Hugging Face, AWS SageMaker, OpenAI API

- **API as a Service (Pay-per-use)**
  - Example: OpenAI’s API, Stability AI, ElevenLabs.
  - **Tools**
    - FastAPI, Flask, Express.js
    - Stripe for payments
    - API Gateways: AWS API Gateway, Cloudflare Workers

- **AI-Generated Content Sales**
  - Platforms: Etsy (AI art), Fiverr (AI-generated content services), Gumroad (AI-generated eBooks).
  - **Examples:**
    - AI-generated children's books
    - AI-created music beats
    - Custom AI profile pictures

Monetizing Generative AI

- ## Custom AI Model Fine-tuning Services
  - Offer **custom AI fine-tuning** for businesses.
  - Charge per project or **enterprise licensing**.
  - Example: A law firm fine-tuning an AI model for legal document summarization.

- ## AI-Powered Educational Platforms
  - Develop **GenAI-based learning platforms** like Khan Academy's Khanmigo.
  - Offer **freemium courses** and charge for premium content.
  - Example: AI tutor for coding, math, or language learning.

- ## Selling AI Prompts & Prompt Engineering Services
  - Sell **high-quality prompts** for ChatGPT, MidJourney, and Stable Diffusion.
  - Platforms: PromptBase, Fiverr, Gumroad.

Challenges of ML developer

-1 .. 0 .. +1

- Data Acquisition and Quality: Obtaining high-quality, relevant, and sufficient data is a significant challenge for many machine learning developers. The quality of the data used to train models can greatly impact their performance.
- Data Preprocessing: Machine learning algorithms require data to be in a specific format before they can be applied. This includes tasks like normalization, feature extraction, and dealing with missing values, which can be time-consuming and error-prone.
- Model Complexity: Deciding on the appropriate level of complexity for a model is crucial. Too simple a model may not capture the underlying patterns, while too complex a model may overfit the data or be computationally infeasible.
- Interpretability and Explainability: As machine learning models, especially deep learning models, become more complex, it becomes increasingly difficult to understand why they are making the predictions they do. This can make it challenging to trust these models and to use them effectively in real-world applications.
- Scalability: As datasets grow larger and more complex, scaling machine learning algorithms to handle this increased size becomes a challenge. This is particularly true for computational-intensive approaches like deep learning.

Challenges of ML developer

- Bias and Fairness: Machine learning models can unintentionally perpetuate or even exacerbate existing biases if they are trained on biased data. Ensuring that models are fair and do not discriminate unfairly against certain groups is an important but often difficult challenge.

→ model → formula

- Generalization: One of the key challenges in machine learning is ensuring that models generalize well to unseen data. Overfitting, where a model learns the training data too closely and performs poorly on new data, is a common issue.
- Privacy and Security: Machine learning models often require large amounts of sensitive data to train effectively. Ensuring that this data is handled securely and that privacy is protected is an ongoing concern.
- Regulatory Compliance: In some industries, machine learning models must comply with specific regulations, such as those related to data protection or financial services. Ensuring compliance with these regulations can be challenging.
- Lifelong Learning and Adaptation: Machine learning models deployed in the real world often encounter new situations that were not included in their training data. Enabling these models to adapt and learn from new data over time is an active area of research.

Libraries

- **Model Development & Fine-Tuning**
  - **Hugging Face Transformers** (transformers) – Pre-trained LLMs and fine-tuning tools
  - **LangChain** – Framework for chaining LLMs into applications
  - **GPT-4All** – Open-source alternative for running LLMs locally
  - **LlamaIndex** – Helps connect LLMs with external data sources
  - **SentenceTransformers** – For embeddings and semantic search
- **Model Training & Optimization**
  - **PyTorch** – Widely used deep learning framework
  - **TensorFlow/Keras** – Good for scalable training
  - **DeepSpeed** – Optimizes large model training and inference
  - **LoRA (Low-Rank Adaptation)** – Efficient fine-tuning technique
  - **FlashAttention** – Improves Transformer model efficiency
- **Deployment & Inference**
  - **vLLM** – Fast inference for LLMs
  - **TGI (Text Generation Inference)** – Optimized serving of Hugging Face models
  - **FastAPI** – Lightweight API framework for serving models
  - **ONNX Runtime** – Optimized inference for multiple hardware platforms

Libraries

- **Data Processing & Augmentation**
  - **Pandas** – Data handling and manipulation
  - **Datasets (from Hugging Face)** – Large-scale dataset loading and preprocessing
  - **spaCy** – Fast NLP processing
  - **NLTK** – Traditional NLP toolkit
- **Vector Databases (for RAG Applications)**
  - **FAISS** – Facebook's fast similarity search
  - **ChromaDB** – Popular open-source vector database
  - **Pinecone** – Managed vector database
  - **Weaviate** – Scalable and AI-native
- **Synthetic Data & Augmentation**
  - **TextAttack** – NLP data augmentation
  - **NLPAug** – Data augmentation for text