Introduction to LLMs

DEFINITIONS

Generative AI AI systems that can produce realistic content *(text, image, etc.)*

Large Language Models (LLMs)
Large neural networks trained at internet scale to estimate the probability of sequences of words

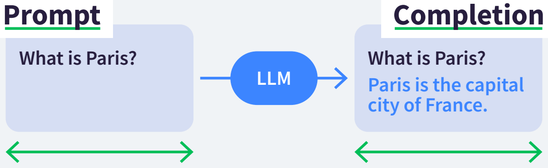
Ex: GPT, FLAN-T5, LLaMA, PaLM, BLOOM
(transformers with billions of parameters)

Abilities (and **computing resources** needed) tend to rise with the number of parameters

USE CASES

- Standard NLP tasks *(classification, summarization, etc.)*
- Content generation
- Reasoning *(Q&A, planning, coding, etc.)*

Prompt **Completion**



In-context learning Specifying the task to perform directly in the prompt

Zero-Shot
Label this review:
Amazing product!
Sentiment:

One-Shot
Label this review:
Very high quality!
Sentiment: **Positive**
Label this review:
Amazing product!
Sentiment:

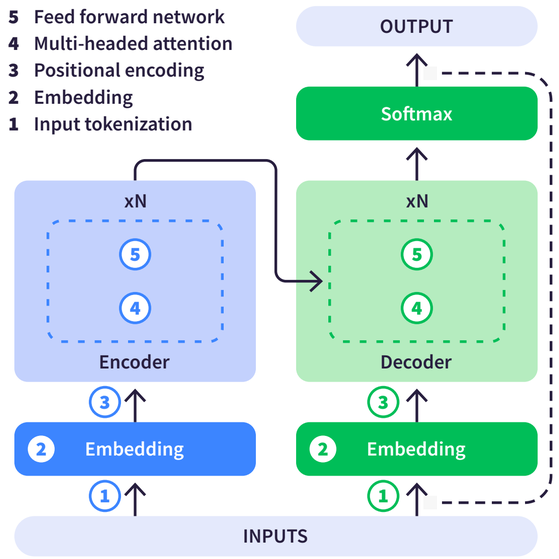
Few-Shot
Label this review:
Very high quality!
Sentiment: **Positive**
Label this review:
I don't really like it
Sentiment: **Negative**
Label this review:
Amazing product!
Sentiment:

Include only a few examples (typically five).
Consider fine-tuning if many examples are needed.

TRANSFORMERS

- Can scale efficiently to use multi-core GPUs
- Can process input data in parallel
- Pay attention to all other words when processing a word

Transformers' strength lies in understanding the **context** and **relevance** of all words in a sentence



Token Word or sub-word
The basic unit processed by transformers

Encoder Processes input sequence to generate a vector representation (or embedding) for each token

Decoder Processes input tokens to produce new tokens

Embedding layer Maps each token to a trainable vector

Positional encoding vector
Added to the token embedding vector to keep track of the token's position

Self-Attention Computes the importance of each word in the input sequence to all other words in the sequence

TYPES OF LLMS

Encoder only = **Autoencoding model**

Ex: BERT, RoBERTa
These are not generative models.

PRE-TRAINING OBJECTIVE To predict **tokens masked** in a sentence (= **Masked Language Modeling**)

OUTPUT Encoded representation of the text

USE CASE(S) Sentence classification *(e.g., NER)*

Decoder only = **Autoregressive model**

Ex: GPT, BLOOM

PRE-TRAINING OBJECTIVE To predict the **next token** based on the previous sequence of tokens (= **Causal Language Modeling**)

OUTPUT Next token

USE CASES Text generation

Encoder-Decoder = **Seq-to-seq model**

Ex: T5, BART

Sentinel token
Consecutive span of corrupted tokens added to the vocabulary

PRE-TRAINING OBJECTIVE Vary from model to model *(e.g., Span corruption like T5)*

OUTPUT Sentinel token + predicted tokens

USE CASES Translation, Q&A, summarization

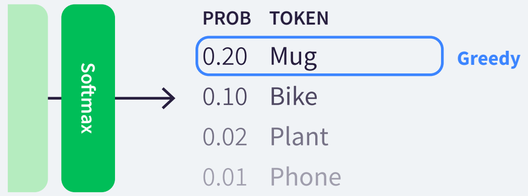
CONFIGURATION SETTINGS

Parameters to set at **inference time**

Max new tokens Maximum number of tokens generated during completion

Decoding strategy

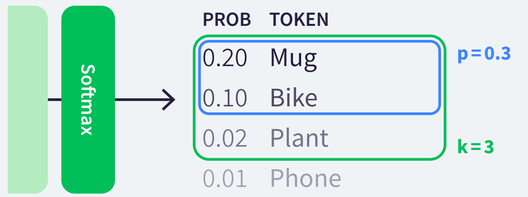
1 Greedy Decoding The word/token with the highest probability is selected from the final probability distribution (prone to repetition)



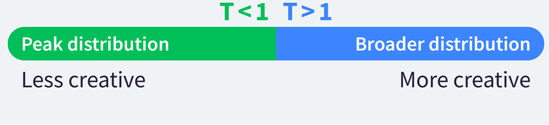
2 Random Sampling The model chooses an output word at random using the probability distribution to weigh the selection (could be too creative)

TECHNIQUES TO CONTROL RANDOM SAMPLING

- **Top K** The next token is drawn from the **k** tokens with the highest probabilities
- **Top P** The next token is drawn from the tokens with the highest probabilities, whose combined probabilities exceed **p**



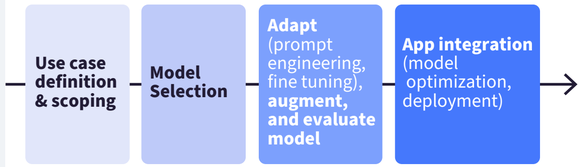
Temperature Influence the shape of the probability distribution through a scaling factor in the softmax layer



LLM Compute Challenges and Scaling Laws

LARGE LANGUAGE MODEL CHOICE

Generative AI Project Lifecycle



Two options for model selection

- Use a pre-trained LLM.
- Train your own LLM from scratch.

But, in general...

…develop your application using a **pre-trained LLM,** except if you work with extremely specific data (i.e., medical, legal)

Hubs: Where you can browse existing models

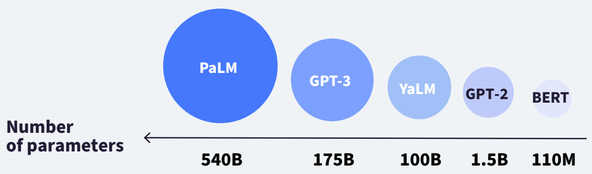
→ **Model Cards:** *List of best use cases, training details, limitations on models.*

The model **choice** will **depend on the details of the task** to carry out.

Model pre-training:

Model weights are adjusted in order to minimize the loss of the training objective.

It requires significant computational resources, (i.e., GPUs, due to high computational load).



COMPUTATIONAL CHALLENGES

Memory Challenge

```
RuntimeError : CUDA out of memory
```

→ LLMs are massive and require plenty of memory for training and inference.

To load the model into GPU RAM:

1 parameter (32-bit precision) = **4 bytes needed**
1B parameters = 4 x 10^9 bytes = 4GB of GPU

Pre-training requires storing additional components, beyond the model's parameters:

- Optimizer states (e.g., 2 for Adam)
- Gradients
- Forward activations
- Temporary variables

This could result in an additional **12-20 bytes of memory needed** per model parameter.

This would mean it requires **16 GB to 24 GB** of GPU memory to train a 1-billion parameter LLM, around **4-6x** the GPU RAM needed just for storing the model weights.

Hence, the memory needed for LLM training is:

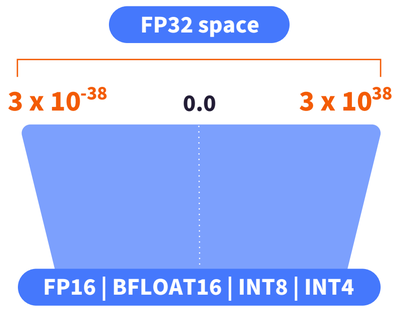
→ Excessive for consumer hardware

→ Even demanding for data center hardware (for single processor training).
For instance, NVIDIA A100 supports up to 80GB of RAM.

QUANTIZATION

How can you reduce memory for training?

Quantization: Decrease memory to store the weights of the model by converting the precision from 32bit to 16bit or 8bit integers.



Quantization maps the FP32 numbers to a lower precision space by employing scaling factors determined from the range of the FP32 numbers.

→ In most cases, quantization **strongly reduces memory requirements** with a **limited loss** in prediction.

BFLOAT16 is a popular alternative to FP16:

- Developed by Google Brain
- Balances memory efficiency and accuracy
- Wider dynamic range
- Optimized for storage and speed in ML tasks

e.g., FLAN T5 pre-trained using BFLOAT16

Benefits of quantization:

→ Less memory

→ Potentially better model performance

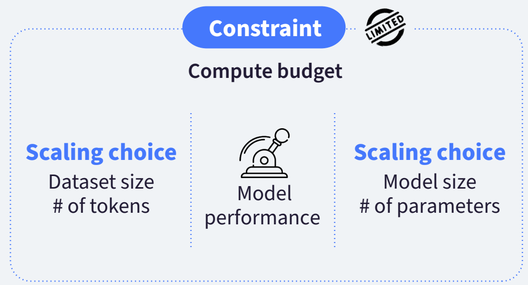
→ Higher calculation speed

SCALING LAWS

How big do the models need to be?
The goal is to maximize model performance.

Researchers explored trade-offs between the dataset size, the model size, and the compute budget:

Increasing compute may seem ideal for better performance, but practical constraints like hardware, time, and budget limit its feasibility.



It has been empirically shown that, as the compute budget remains fixed:

→ **Fixed model size:** Increasing training dataset size improves model performance.

→ **Fixed dataset size:** Larger models demonstrate lower test loss, indicating enhanced performance.

What's the optimal balance?

Once scaling laws have been estimated, we can use the **Chinchilla approach**, i.e., we can choose the dataset size and the model size to train a **compute-optimal model**, which maximizes performance for a given compute budget. The compute-optimal training dataset size is ~20x the number of parameters.

LLM Instruction Fine-Tuning & Evaluation

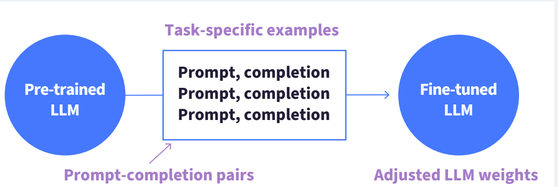
INSTRUCTION FINE-TUNING

In-Context Learning Limitations:

- May be insufficient for very specific tasks.
- Examples take up space in the context window.

Instruction Fine-Tuning

The LLM is trained to estimate the next token probability on a cautiously curated dataset of high-quality examples for specific tasks.



- The LLM generates better completions for a specific task
- Has potentially high computing requirements

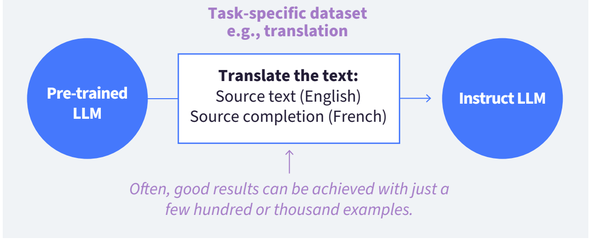
Steps:

1. Prepare the training data.
2. Pass examples of training data to the LLM (prompt and ground-truth answer).

3. Compute the cross-entropy loss for each completion token and backpropagate.

TASK-SPECIFIC FINE-TUNING

Task-specific fine-tuning involves training a pre-trained model on a particular task or domain using a dataset tailored for that purpose.



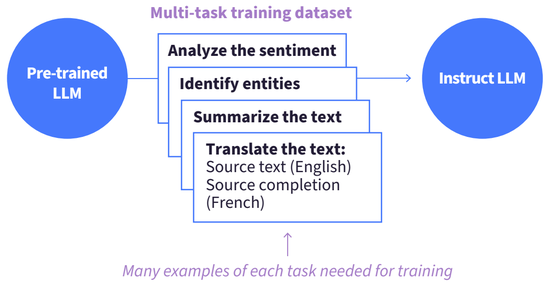
Fine-tuning can significantly increase the performance of a model on a specific task, but can reduce the performance on other tasks ("**catastrophic forgetting**").

Solutions:

- **It might not be an issue** if only a single task matters.
- **Fine-tune for multiple tasks concurrently** (~50K to 100K examples needed).
- **Opt for** Parameter Efficient Fine-Tuning (PEFT) instead of full fine-tuning, which involves training only a small number of task-specific adapter layers and parameters.

MULTI-TASK FINE-TUNING

Multi-task fine-tuning diversifies training with examples for multiple tasks, guiding the model to perform various tasks.



Drawback: It requires a lot of data (around 50K to 100K examples).

Model variants differ based on the datasets and tasks used during fine-tuning.

Example of the FLAN family of models
FLAN, or Fine-tuned LAnguage Net, provides tailored instructions for refining various models, akin to dessert after pre-training.

FLAN-T5 is an instruct fine-tuned version of the T5 foundation model, serving as a versatile model for various tasks.

FLAN-T5 has been **fine-tuned on a total of 473 datasets** across **146 task categories.** *For instance, the SAMSum dataset was used for summarization.*

A specialized variant of this model for chat summarization or for custom company usage could be developed through additional fine-tuning on specialized datasets (e.g., DialogSum or custom internal data).

MODEL EVALUATION

Evaluating LLMs Is Challenging

(e.g., various tasks, non-deterministic outputs, equally valid answers with different wordings).

→ Need for automated and organized performance assessments

Various approaches exist, but there are a few examples:

ROUGE & BLEU SCORE

- **Purpose:** To evaluate LLMs on narrow tasks (summarization, translation) when a reference is available
- Based on n-grams and rely on precision and recall scores (multiple variants)

BERT SCORE

- **Purpose:** To evaluate LLMs in a task-agnostic manner when a reference is available.
- Based on token-wise comparison, a similarity score is computed between candidate and reference sentences.

LLM-as-a-Judge

- **Purpose:** To evaluate LLMs in a task-agnostic manner when a reference is available.
- Based on prompting an LLM to assess the equivalence of a generated answer with a ground-truth answer.

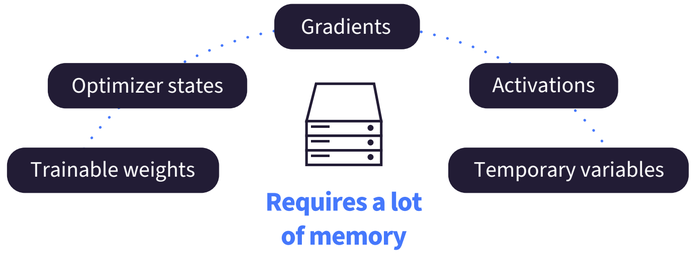
To measure and compare LLMs more holistically, use **evaluation benchmark datasets** specific to model skills.

E.g., *GLUE*, *SuperGLUE*, *MMLU*, *Big Bench*, *Helm*

Parameter Efficient Fine-Tuning (PEFT) Methods

PEFT

Full fine-tuning of LLMs is challenging:



PEFT methods **only update a small number of model parameters.**
Examples of PEFT techniques:

- Freeze most model weights, and **fine tune only specific layer parameters.**
- Keep existing parameters untouched; **add only a few new ones or layers** for fine-tuning.

→ The trained parameters can account for only 15%-20% of the original LLM weights.

Main benefits:

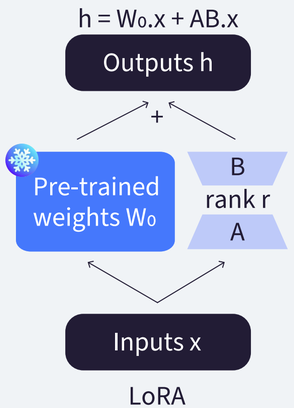
- Decrease memory usage, often requiring just 1 GPU.
- Mitigate risk of catastrophic forgetting.
- Limit storage to only the new PEFT weights.

Multiple methods exist with trade-offs on parameters or memory efficiency, training speed, model quality, and inference costs.
Three PEFT methods classes from literature:

Selective	Reparameterization	Additive
Fine-tune only specific parts of the original LLM.	Use low-rank representations to reduce the number of trainable parameters. *E.g., LoRA*	Augment the pre-trained model with new parameters or layers, training only the additions. → Adapter → Soft prompts

LoRA

Method to reduce the number of trainable parameters during fine-tuning **by freezing all original model parameters** and injecting a **pair of rank decomposition matrices** alongside the original weights



1 - Keep the majority of the original LLM weights frozen.
2 - Introduce a pair of **rank decomposition matrices.**
3 - Train the new matrices A and B.

Model weights update:

1 - Matrix multiplication:

$B * A = BxA$

2 - Add to original weights :

$W_0 + BxA$

Additional notes:

- No impact on inference latency.
- Fine-tuning specifically on the **self-attention layers** using LoRA is often enough to enhance performance for a given task.
- Weights can be switched out as needed, allowing for training on **many different tasks.**

Rank Choice for LoRA Matrices:

Trade-Off: A smaller rank reduces parameters and accelerates training **but** risks lower adaptation quality due to reduced task-specific information capture.

*In literature, it appears that a **rank between 4-32** is a good trade-off.*

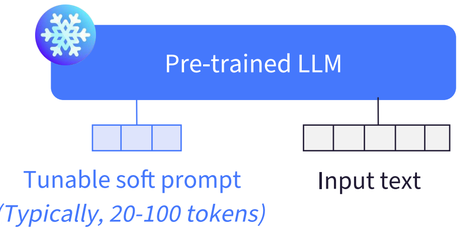
LoRA can be combined with quantization (=QLoRA).

SOFT PROMPTS

Unlike prompt engineering, whose limits are:

- The manual effort requirements
- The length of the context window

Prompt tuning: Add trainable tensors to the model input embeddings, commonly known as "soft prompts," optimized directly through gradient descent.



Soft prompt vectors:

- Equal in length to the embedding vectors of the input language tokens
- Can be seen as **virtual tokens** which can take any value within the multidimensional embedding space

In prompt tuning, LLM weights are frozen:

- Over time, the embedding vector of the soft prompt is adjusted to optimize model's completion of the prompt
- Only **few parameters are updated**
- A different set of soft prompts can be trained for each task and easily swapped out during inference (occupying very little space on disk).

From literature, it is shown that at 10B parameters, prompt tuning is as efficient as full fine-tuning.

⚠ *Interpreting virtual tokens can pose challenges (nearest neighbor tokens to the soft prompt location can be used).*

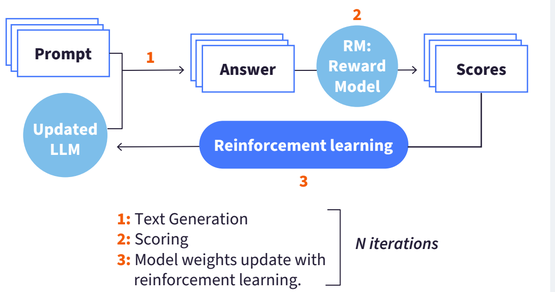
data iku

Preference Fine-Tuning (Part 2)

FINE-TUNING WITH RL & REWARD MODEL

The LLM weights are updated to create a human-aligned model via reinforcement learning, leveraging the reward model, and starting with a high-performing base model.

Goal: To align the LLM with provided instructions and human behavior.



Example:

Prompt: "A tree is..."
Iteration 1: "...a plant with a trunk." → Reward: 0.3
…
Iteration 4: "...a provider of shade and oxygen." → Reward: 1.6
…
Iteration n: "...a symbol of strength and resilience." → Reward: 2.9

As the process advances successfully, the reward will gradually increase until it meets the predefined evaluation criteria for helpfulness.

Updated model: The resulting updated model should be more aligned with human preferences.

Reinforcement learning algorithm: Proximal policy optimization (PPO) is a popular choice.

PPO ALGORITHM FOR LLMS

PPO iteratively updates the policy to **maximize the reward**, adjusting the LLM weights incrementally to **maintain proximity to the previous version** within a defined range for **stable learning.**

The **PPO objective** is used to update the LLM weights by backpropagation:

$$L^{PPO} = L^{POLICY} + c_1 L^{VF} + c_2 L^{ENT}$$

Hyperparameters

Policy loss | Value loss | Entropy loss

Value Loss: Minimize it to improve return prediction accuracy.

$$L^{VF} = \frac{1}{2} \left\| V_\theta(s) - \left(\sum_{t=0}^{T} \gamma^t r_t | s_0 = s \right) \right\|_2^2$$

Value loss | Value function

Estimated future total reward

Actual Reward from the reward model

Policy Loss: Maximize it to get higher rewards while staying within reliable bounds.

$$L^{POLICY} = min\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} . \hat{A}_t , clip\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) . \hat{A}_t \right)$$

Probabilities of the next token with the updated LLM

Probabilities of the next token with the initial LLM

Advantage term

Define "trust region"

Guardrails
Keeping the policy in the "trust region"

π_θ Model's probability distribution over tokens

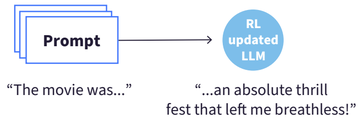
Entropy Loss: Maximize it to promote and sustain model creativity.

$$L^{ENT} = entropy(\pi_\theta(.|s_t))$$

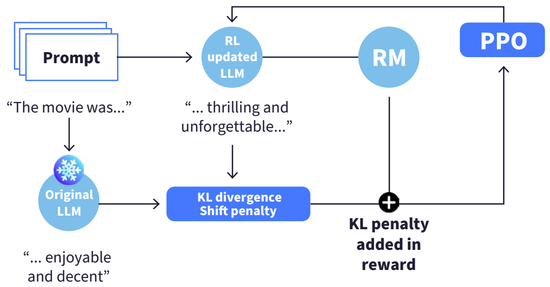
The higher the entropy, the more creative the policy.

REWARD HACKING

The agent **learns to cheat the system** by maximizing rewards at the expense of alignment with desired behavior.



To prevent reward hacking, **penalize RL updates** if they significantly deviate from the frozen original LLM, using **KL divergence.**



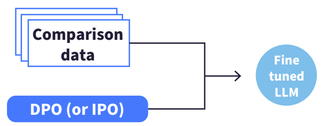
DIRECT PREFERENCE OPTIMIZATION

An **RLHF** pipeline is **difficult to implement:**

- Need to train a reward model
- New completions needed during training
- Instability of the RL algorithm

Direct Preference Optimization (**DPO**) is a simpler and more stable **alternative to RLHF**. It solves the same problem by minimizing a training loss directly based on the preference data (without reward modeling or RL).

Identity Preference Optimization (**IPO**) is a variant of DPO less prone to overfitting.



RL FROM AI FEEDBACK

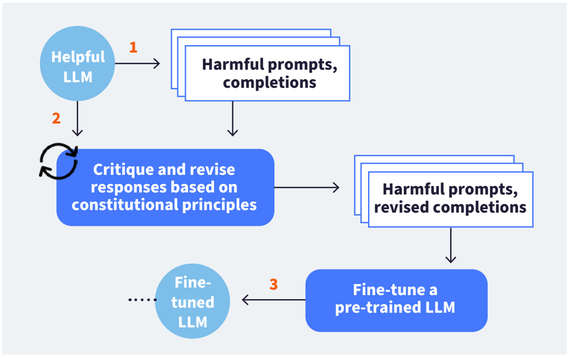
Obtaining the reward model is labor-intensive; scaling through AI-supervision is more precise and requires fewer human labels.

Constitutional AI (Bai, Yuntao, et al., 2022)

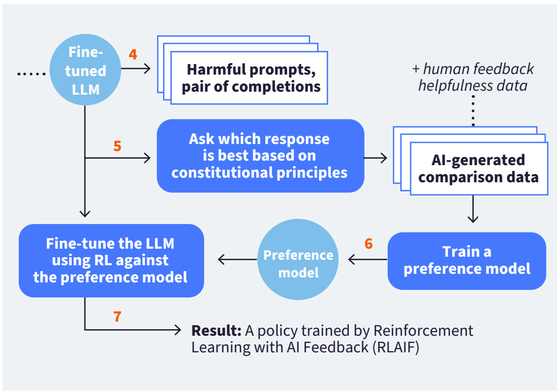
Approach that relies on a **set of principles** governing AI behavior, along with a small number of examples for few-shot prompting, collectively forming the "**constitution**."

Example of constitutional principle: "Please choose the response that is the most helpful, honest, and harmless."

1. Supervised Learning Stage



2. Reinforcement Learning (RL) Stage - RLAIF



data iku

LLM-Powered Applications

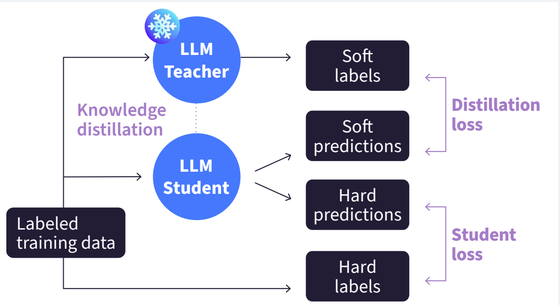
MODEL OPTIMIZATION FOR DEPLOYMENT

Inference challenges: High computing and storage demands

→ Shrink model size, maintain performance

Model Distillation

- Scale down model complexity while preserving accuracy.
- Train a small student model to mimic a large frozen teacher model.



- **Soft labels:** Teacher completions serve as ground truth labels.
- Student and distillation losses update student model weights via backpropagation.
- The student LLM can be used for inference.

Post Training Quantization (PTQ)

PTQ reduces model weight precision to 16-bit float or 8-bit integer.

- Can target both weights and activation layers for impact.
- May sacrifice performance, yet beneficial for cost savings and performance gains.

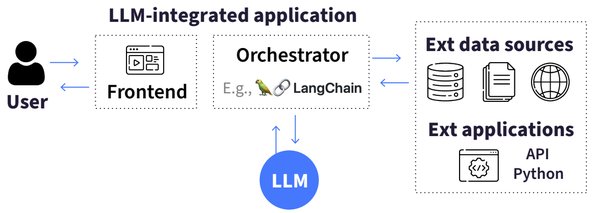
Model Pruning

Removes redundant model parameters that contribute little to the model performance.
Some methods require full model training, while others are in the PEFT category (LoRA).

LLM-INTEGRATED APPLICATIONS

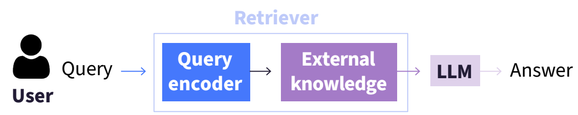
- Knowledge can be out of date.
- LLMs struggle with certain tasks (e.g., math).
- LLMs can confidently provide wrong answers ("hallucination").

→ Leverage **external app** or **data sources**



Retrieval Augmented Generation (RAG)

AI framework that integrates **external data sources** and **apps** (e.g., documents, private databases, etc.).
Multiple implementations exist, will depend on the details of the task and the data format.



- We retrieve **documents most similar to the input query** in the external data.
- We combine the **documents with the input query** and **send the prompt to the LLM to** receive the **answer**.

⚠ *Size of the context window can be a limitation.*

→ Use multiple **chunks** (e.g., with LangChain)

⚠ *Data must be in format that allows its relevance to be assessed at inference time.*

→ Use **embedding vectors** (vector store)

Vector database: Stores vectors and associated metadata, enabling efficient nearest-neighbor vector search.

LLM REASONING WITH CHAIN-OF-THOUGHT PROMPTING

Complex reasoning is challenging for LLMs.
E.g., problems with multiple steps, mathematical reasoning

→ LLM should serve as a **reasoning engine.**
The prompt and completion are important!

1. Plan actions	2. Format outputs	3. Validate actions
Set of instructions *Step1: Get customer ID* *Step2: Reset password*	Requires formatting for applications to understand actions	Collect information that allows validation of an action

Chain-of-Thought (CoT)

- Prompts the model to **break down problems into sequential steps.**
- Operates by integrating **intermediate reasoning steps** into examples for one-or few-shot inference.

Prompt
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5+6=11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Completion
A: The cafeteria had 23 apples. They used 20 to make lunch. 23-20=3. They bought 6 more apples, so 3+6=9. The answer is 9. ✓

In the completion, the whole prompt is included.

→ Improves performance but struggles with precision-demanding tasks like tax computation or discount application.

Solution: Allow the LLM to communicate with a proficient math program, as a Python interpreter.

PROGRAM-AIDED LANGUAGE & REACT

Program-Aided Language (PAL)

Generate scripts and pass it to the interpreter.

Prompt
Q: Roger has 5 tennis balls. [...]
A: CoT reasoning
# Roger started with 5 tennis balls
tennis_balles=5 PAL execution
# 2 cans of tennis balls each is
bought_balls=2*3
# tennis balls. The answer is
answer = tennis_balls + bought_balls
Q. [...]

Completion is handed off to a Python interpreter.
↓
Calculations are accurate and reliable.

ReAct

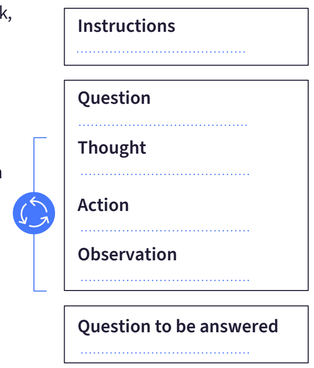
Prompting strategy that combines CoT reasoning and action planning, employing **structured examples** to guide an LLM in **problem-solving** and decision-making for **solutions**

Instructions: Define the task, what is a thought and the actions

Thought: Analysis of the current situation and the next steps to take

Action: The actions are from a predetermined list and defined in the set of instructions in the prompt
The loop ends when the action is finish []

Observation: Result of the previous action



→ **LangChain** can be used to connect multiple components through agents, tools, etc.

Agents: Interpret the user input and determine which tool to use for the task (LangChain includes agents for PAL & ReAct).

ReAct reduces the risks of errors.