Generative AI with Large Language Models

Course link: https://www.deeplearning.ai/.

Week 1

> Understand the transformers architecture and its specificities for LLMs. Discuss the GenAI project life cycle.

Introduction to LLMs and the generative AI project lifecycle

Generative AI & LLMs

- models that mimic or simulate humans
  - foundations models created from trillions of words and sentences written by humans
  - LLMs == purple circles
  - use of FLAN-T5 in this course
  - multiple modalities are available, but we'll use NLP specifically in this course
    - prompting, fine-tuning, deployment
- prompts and completion
  - prompt: text you enter
    - context window
  - model: predicts the next word
  - completion: generates an answer
    - inference

LLM use cases and tasks

- next-word prediction
  - from basic chatbots to multiple other tasks (essay writing, discussion summarization, translation, coding, named entity extraction, information retrieval, connection to external data sources or APIs, ...)
- scale
  - more parameters means better understanding of language
  - fine-tuning small models can also prove efficient

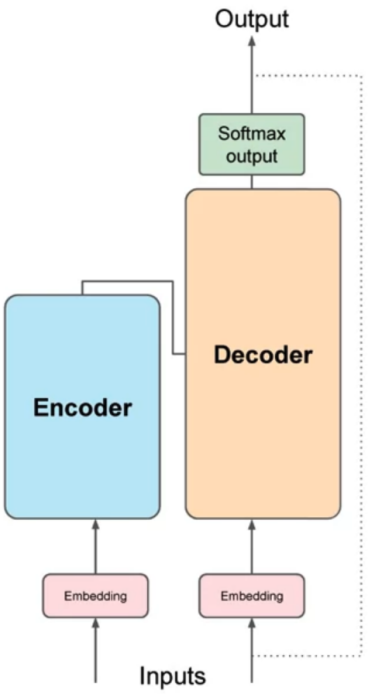
Text generation before transformers

- RNNs
  - to work better, RNNs needed to see more words in front of the one to predict (similar to context window?) ⇒ lots of resources needed
  - performance was not great because the model needs to understand more context about the sentence / document
  - words can also have multiple meanings → (syntactic) ambiguity
- attention is all you need & transformers
  - scales efficiently

- parallel processing
- attention to input meaning

Transformers architecture

- transformers are able to learn all the context of a sentence
  - attention weights to each word to each other word
  - learned during training ⇒ **attention map**
  - self-attention
- transformers architecture
  - encoder + decoder
  - tokenization of the words input
    - convert to numbers
    - use token IDs
    - use the same tokenizer all over the process
  - embedding token as multi-dimensional vectors
    - gets the meaning of individual tokens into the model
      - used in word2vec already
  - positional encoding
    - self-attention reflects the importance of each word to each other words
    - multi-headed self-attention
      - each head will learn a different aspect of language
        - eg: activity of the sentence, other properties such as rhymes
  - feed forward network
    - vector of logit (prob score for each word)
  - softmax layer



Generating text with transformers

- translation task: sequence-to-sequence
- steps: tokenization → embedding → attention layers → feed forward network → generate output from decoder → softmax output
- encoder & decoder
  - the encoder understands a deep representation of the language used
  - the decoder uses this understanding on the input data to produce new tokens
- encoder only models
  - input and output have same length
  - uses: classification (sentiment analysis)
  - eg: Bert
- encoder-decoder models

    - sequence-to-sequence (different lengths)
    - scale up to perform generation tasks
    - eg: Bart, T5 (used here)
- decoder only models
    - generalizes to most tasks if scaled up
    - eg: GPT, Llama

Transformers: Attention is all you need

- goal: potentially replace RNNs and CNNs
- link to the article: https://arxiv.org/html/1706.03762v7.

Prompting and prompt engineering

- prompt → model → completion
    - context window
- prompt engineering
    - in-context learning (ICL): include examples in the prompt

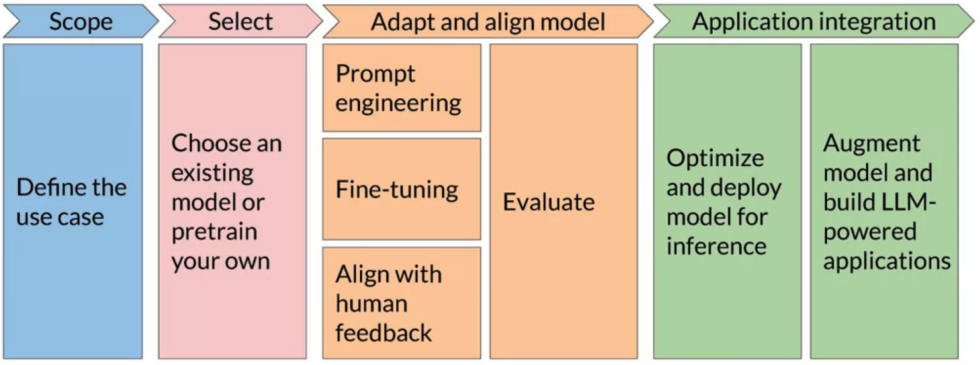
        - zero shot inference: no filled example, more like a template (**bigger models**)
        - one shot inference: one filled example (sample review + sentiment analysis eg), template for the next one to be filled
        - few shot inference: multiple samples + template to fill
            - use no more than 5 or 6 examples ⇒ if needed, fine-tune it
    - models with more parameters understand language better ⇒ are better at zero shot inference

Generative configuration

- set of configuration parameters can modify the inference behavior
    - `max new tokens`: limit nb of tokens generated
    - `greedy` vs `random sampling`
        - greedy form decoding: model always chooses the word with the highest probability
        - random(-weighted) sampling: random selection based on prob distribution
            - reduces likelihood for a word to be repeated
            - increases risks of hallucinations
    - `top-k and top-p`: increase random sampling but helps with accuracy

        - top-k: reduce the number of tokens possible to choose from, even when random sampling is there ⇒ you first select `k` tokens, then random sample
        - top-p: eg: if p=0.3, only the tokens that add up to 0.3 are selected then randomly sampled
    - `temperature`: influences the shape of the prob distribution of the next token
        - higher the temp, higher the randomness
        - low value (<1): prob distribution is concentrated on a smaller number of words ⇒ more deterministic?
        - high value (>1): broader, flatter prob distribution ⇒ more randomness and variability (seems more creative)

Generative AI project lifecycle



- Scope: define scope as narrowly as possible
    - LLM abilities depends strongly on the model chosen → define tasks accurately to choose correctly
- Select
- Adapt and align model: assess performance and eventually train model a bit more
- Application integration

Introduction to AWS labs

Lab 1 walkthrough

LLM pre-training and scaling laws

Pre-training large language models

- focus on the Select part of the genAI LC
- choosing a model: foundation model 🆚 building your own
    - model hub & model cards
- model architecture & pre-training objectives
    - data → data quality → pre-training on embeddings
    - encoders only 🆚 encoders-decoders 🆚 decoders only
        - autoencoding models (encoders only): reconstruct text using masks for training
            - sentiment, analysis, NER, word classification
            - eg: Bert, Deberta
        - autoregressive models (decoders only): predict next token
            - text gen, other emergent behavior (depends on model size)

        - eg: GPT, Bloom
    - sequence-to-sequence models (encoders-decoders)
        - translation, text summarization, question answering
        - eg: T5, Bart
- significance of scale: task abiliy: size matters 🦗👩‍🎓

Computational challenges of training LLMs

- CUDA
- quantization

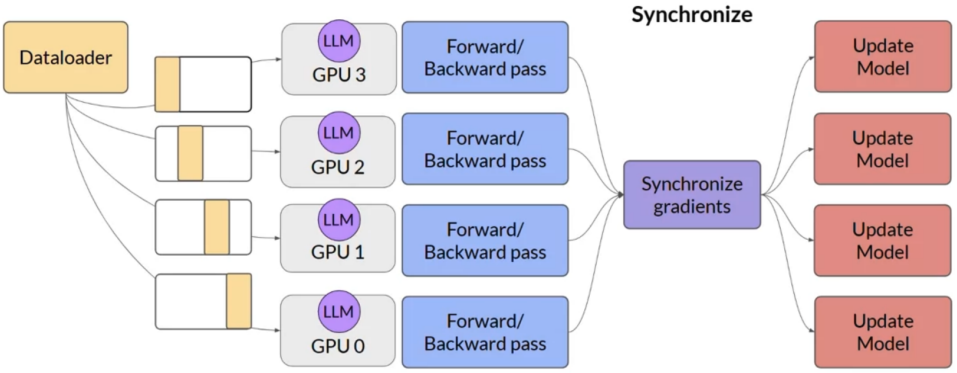
	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte

- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training

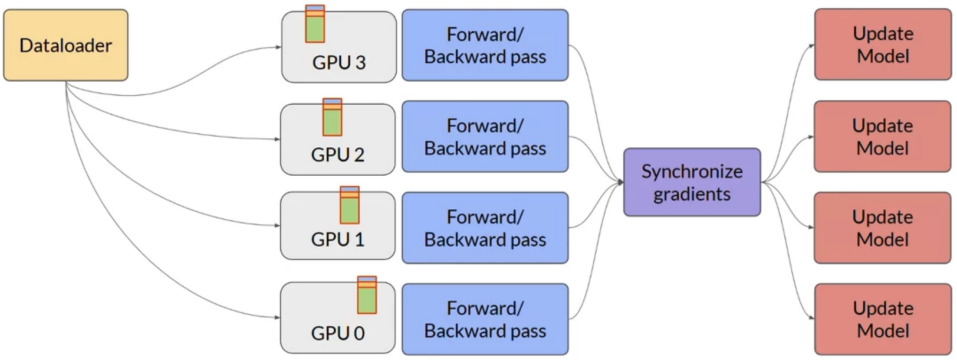
    - reduce precision of model from 32 to 16 bit floating points
    - FP16 🆚 BF16 (truncated FP32)
        - in the latter, the exponent keeps 8 bits
    - INT8

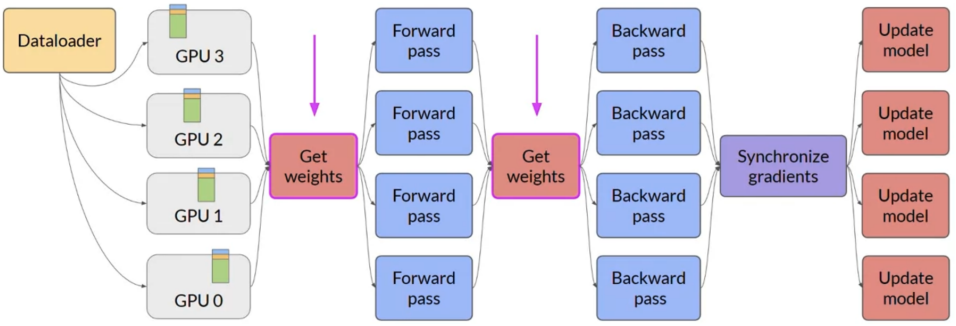
Efficient multi-GPU compute strategies

- distributed compute needed for bigger models, but also useful in other situations to improve speed
- distributed data parallel (DDP)



- fully sharded data parallel (FSDP)





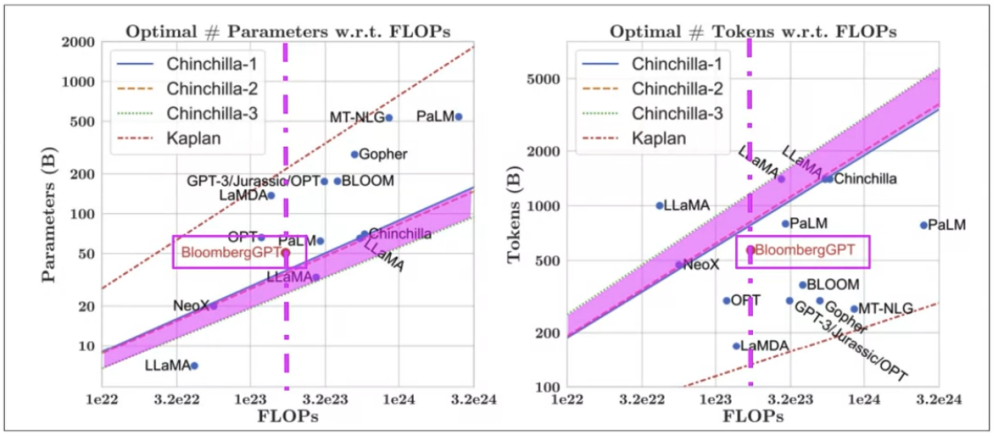
- ZeRo paper
- distribute params, gradients, and optimizer states across GPUs
- supports offloading to CPU if needed
- sharding is configurable

Scaling laws and compute-optimal models

- scaling choices for pre-training
  - goal: maximize model performance (minimize loss)
    - scaling choice: dataset size (nb of tokens) VS model size (nb of params)
- compute budget
  - petaflop/s-day
  - chinchilla paper

Pre-training for domain adaptation

- optimal performance / size is the purple zone



Week 2

Fine-tuning LLMs with instruction

Instruction fine-tuning

- prompt engineering (in-context learning) doesn't always work good enough
- instruction fine-tuning is also called full fine-tuning
  - you update all weights of the model using prompt-completion example pairs
- you need to prepare training data

Fine-tuning on a single task

- good result can be obtained with only 5-10k examples
- the process can lead to catastrophic forgetting
  - the weights of the original model are updated, potentially leading to worst skills on other tasks than the single one you focused on
  - how to avoid?
    - decide if it impacts you use case
    - you can fine-tune on multiple tasks
    - parameter efficient fine-tuning (PEFT)
      - preserves the original weights and only changes specific layers

Multi-task instruction fine-tuning

- one drawback is that you need a lot of data of each task
- instruction fine-tuning with FLAN

Model evaluation

- usually, you can use classical metrics like accuracy
  - LLMs are non-deterministic
  - sentences can be only different by one word but completely opposite
- ROUGE and BLEU score
  - 🔴 is used for text summarization and compares a summary to one or more references
  - 🔵 is used for text translation and compare to human-generated translations

- terminology
  - unigram, bigram, n-gram
- metrics
  - ROUGE-1
    - Recall = unigram matches / unigrams in reference
    - you also have precision or F1 ROUGE-1
  - ROUGE-2: same but with bigrams
  - ROUGE-L: uses LCS (longest common subsequence)
  - ROUGE clipping
  - BLEU = Avg(precision across range of n-gram sizes)

Benchmarks

- GLUE and SuperGLUE
- for massive models
  - MMLU
    - extensive world knowledge and reasoning needed to perform well
  - BIG-bench
- holistic evaluation of language models (HELM)
  - uses seven metrics such as accuracy bu also fairness or bias

Parameter efficient fine-tuning

Parameter efficient fine-tuning (PEFT)

- full fine-tuning is intensive
- PEFT is less prone to catastrophic forgetting
  - training of small nb of weights
- multiple methods to do PEFT
  - selective: select only a subset of initial LLM params to fine-tune
  - reparameterization: create new, low-rank representation (LoRA)
  - additive: add trainable layers or weights to model
    - adapters: add layers to the architecture (in the encoder or decoder layers)
    - soft prompts: keep model fixed and frozen, manipulates the input (prompt tuning)

PEFT techniques 1: Low-Rank Adaptation (LoRA)

- reparameterization category
- method
  1. freeze most of original LLM weights
  2. inject 2 rank decomposition matrices
  3. train the weights of the smaller matrices
  - steps to update model for inference
    1. matrix multiply the low rank matrices
    2. add to original weights

- concrete example using base transformer

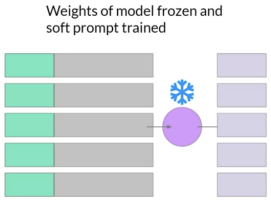
Use the base Transformer model presented by Vaswani et al. 2017:
- Transformer weights have dimensions d x k = 512 x 64
- So 512 x 64 = 32,768 trainable parameters

In LoRA with rank r = 8:
- A has dimensions r x k = 8 x 64 = 512 parameters
- B has dimension d x r = 512 x 8 = 4,096 trainable parameters
- **86% reduction in parameters to train!**

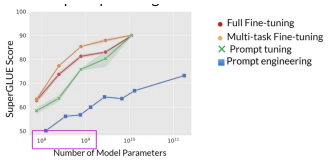
  - possible to use LoRA on a single GPU
  - you can train different rank decomposition matrices for different tasks and update the weights before inference
- evaluating LoRA fine-tuned models
  - a FLAN-T5 fine-tuned using LoRA has only 3.2% lower performance than a fully fine-tuned one (summarization task)
- how to choose the rank of the decomp matrices?
  - smaller rank = smaller nb of params to train
  - in the paper, the best ranks are between 4 and 32 (best trade-off)

PEFT techniques 2: Soft prompts

- prompt tuning ≠ prompt engineering
- prompt tuning: add trainable “soft prompts” to inputs



- for multiple tasks, switch out soft prompt at inference time
- performance: prompt tuning can get as effective as full fine-tuning!



Lab 2 walkthrough

Week 3

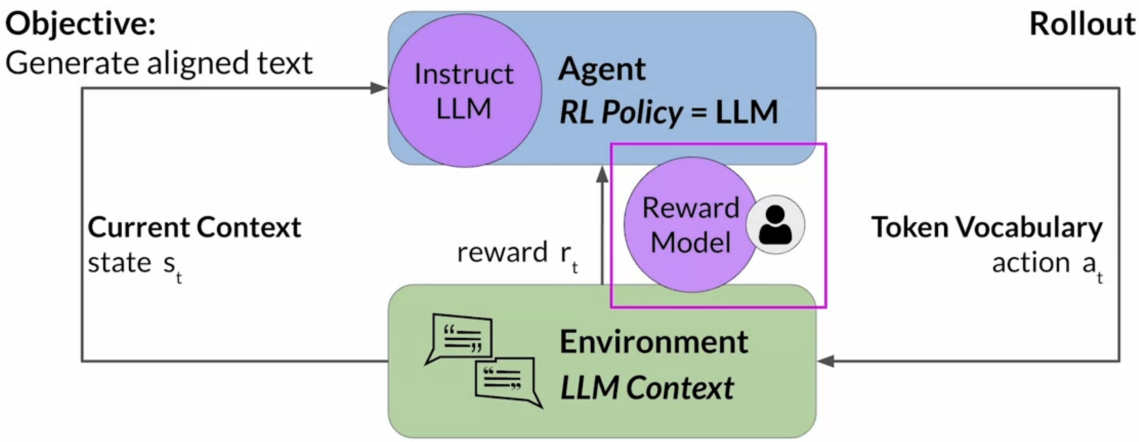
Reinforcement learning from human feedback

Aligning models with human values

- continue focusing on fine-tuning steps (”Adapt & align model” step from genAI project lifecycle)
  - focus on aligning model: avoid models behaving badly (causing harm, being unhelpful or dishonest, etc)
  - improving completion’s relevance to prompt
  - avoid dangerous topics

Reinforcement learning from human feedback (RLHF)

- we apply RLHF to an instruct fine-tuned LLM
  - its a way to specialize the model on what some humans would prefer
- reinforcement learning
  - agent (RL policy (model))→ action (action space) → environment → state & reward → agent → ...
  - objective: maximize reward received for actions
- fine-tuning LLMs with RLHF

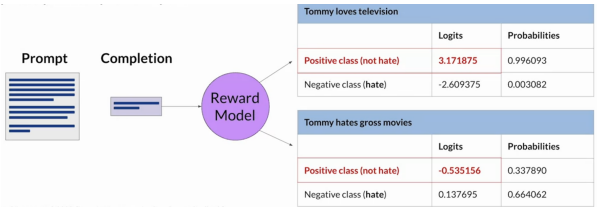


RLHF: Obtaining feedback from humans

- prepare dataset for human feedback
  - define criterion (helpfulness eg)
  - labelers rank completions based on criterion
  - if answers are nonsensical, they are marked specifically
- prepare labeled data for training
  - convert rankings into pairwise training data for the reward model
  - y_i is the chosen completion

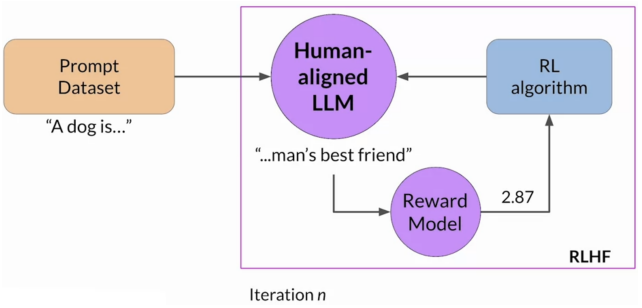
RLHF: Reward model

- goal: train model to predict preferred completion from $\{y_j, y_k\}$ for prompt x



RLHF: Fine-tuning with reinforcement

- use the reward model to fine-tune LLM with RL



Proximal policy optimization (PPO)

- what is PPO?
  - algo for RF problems
  - optimizes a policy
    - rewards human-like responses
  - updates parameters of the LLM but keeps it close
- initialize PPO with Instruct LLM
  - phase 1: create completions
    - calculate rewards
    - calculate value loss = estimated future total reward - known future total reward
  - phase 2: model update
    - calculate policy loss

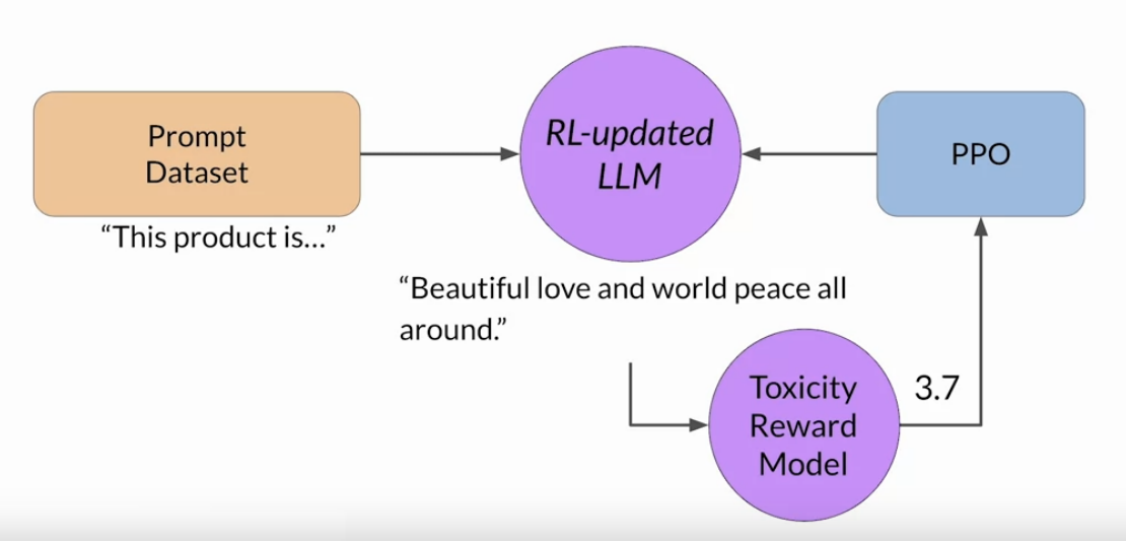
$$L^{POLICY} = \min\left(\underbrace{\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \cdot \hat{A}_t}_{\text{Most important expression}}, \text{clip}\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

π_θ Model's probability distribution over tokens

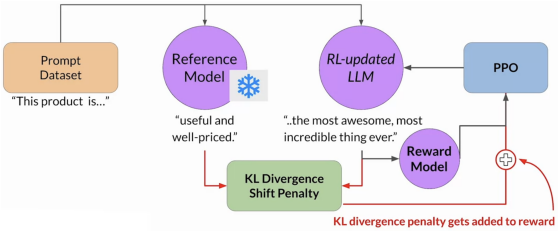
    - the numerator is the probabilities of the next token with the updated LLM
    - the denominator is the prob of the next token with the initial LLM
    - $\hat{A}_t$ is the advantage term
  - calculate entropy loss
- you need many iterations

RLHF: Reward hacking

- reward hacking adds words to completion to increase the score for a criteria
  - it reduces the overall quality of the language
  - eg: to lower toxicity, the model could just end up speaking positively about everything



- to avoid it, keep a reference model untouched
  - calculate the KL divergence shift penalty



    - helps get insight on the stat distrib for our data
- when RLHF is done, you need to evaluate the human-aligned LLM
  - evaluate using the toxicity score (summarization dataset)

KL divergence

- mathematical measure of the difference between two probability distributions
- https://huggingface.co/blog/trl-peft

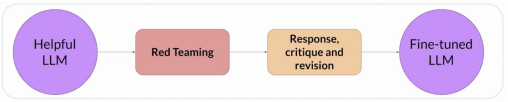
Scaling human feedback

- huge human effort is needed (labelers) ⇒ how to scale it?
  - other problem with current method of RLHF: maximizing helpfulness can lead to revealing toxic/harmful information
- model self-supervision: Constitutional AI
  - train models following a set of rules guiding behavior (consitution)
  - list of constit principles

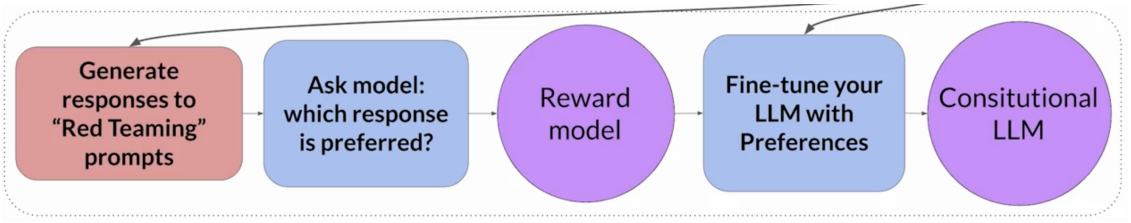
```
Please choose the response that is the most helpful, honest, and harmless.

Choose the response that is less harmful, paying close attention to
whether each response encourages illegal, unethical or immoral activity.

Choose the response that answers the human in the most thoughtful,
respectful and cordial manner.

Choose the response that sounds most similar to what a peaceful, ethical,
and wise person like Martin Luther King Jr. or Mahatma Gandhi might say.

...
```

    - can be defined by user

- constitutional AI
  - phase 1: supervised learning stage



  - phase 2: reinforcement learning stage (RLAIF)

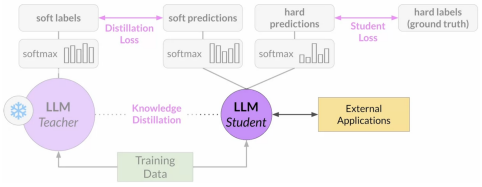


Lab 3 walkthrough

LLM-powered applications

Model optimizations for deployment

- we'll now discuss the "Application integration → Optimize and deploy model for inference" step of the genAI project lifecycle
- LLM optimization techniques
  - distillation: teacher-student LLMs
    - student model learns to imitate the teacher model
    - method

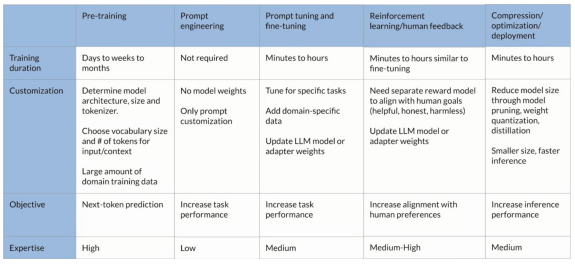


      - freeze the teacher weights and use it to generate completions
      - generate completions using the student model
      - minimize the distillation loss
      - add a temperature parameter to increase creativity (>1) ⇒ prob distrib is flatter
      - student output: hard labels (ground truth)
    - key benefits: use the student model to make external inference
    - effective for encoder-only models (presence of information redundancy)
  - quantization: reduce precision of model weights
    - post-training quantization (PTQ)
      - requires calibration to capture dynamic range
    - tradeoff wit performance
  - pruning: remove redundant parameters (model weights with values close or equal to zero
    - pruning methods

        - full-model retraining
        - PEFT/LoRA
        - post-training
    - (in theory) reduces model size and improves performance, but impact can be limited if not a lot of weights are close or equal to 0

Generative AI Project Lifecycle cheat sheet

- time & effort in the lifecycle



	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and # of tokens for input/context Large amount of domain training data	No model weights Only prompt customization	Tune for specific tasks Add domain-specific data Update LLM model or adapter weights	Need separate reward model to align with human goals (helpful, honest, harmless) Update LLM model or adapter weights	Reduce model size through model pruning, weight quantization, distillation Smaller size, faster inference
Objective	Next-token prediction	Increase task performance	Increase task performance	Increase alignment with human preferences	Increase inference performance
Expertise	High	Low	Medium	Medium-High	Medium

    - highest: pre-training
    - next steps: prompt engineering, prompt tuning and fine-tuning, RLHF (use existing reward model would take way less effort), compression/optimization/deployment

Use the LLM in applications

- models having difficulty
    - out of date information
    - wrong mathematical answers (just predicting tokens based on training)
    - hallucinations (model doesn't know the answer and invents one)
- we'll now discuss the "Application integration → Augment model and build LLM-powered applications" step of the genAI project lifecycle
    - connect to external data sources and applications
    - user information from the app ↔ orchestration lib (↔ external data sources & external apps) ↔ LLM
- retrieval augmented generation (RAG)
    - overcome knowledge cut-off issue and update knowledge of the world
    - avoids having to retrain the model
    - give access to external data during inference
    - RAG is a framework, not a specific set of techs ⇒ different implementations
        - retriever = query encoder + external information sources (docs, wikis, expert systems, web pages, databases, vector store)
        - data preparation for vector store for RAG
            - considerations
                1. data must fit inside context window
                    - split long sources into chunks)
                2. data must be in format that allows its relevance to be assessed at inference time: **Embedding vectors**
        - eg: searching legal documents

Interacting with external applications

- use case: customer bot interaction
  - needs
    - look up orders using RAG → from database using SQL prob
    - make API call for shipping label
    - make API call to the shipper
- requirements for using LLMs to power apps
  - plan actions (split tasks into steps etc)
  - format outputs (completion must be made in API language)
  - validate actions

Helping LLMs reason and plan with chain-of-thought

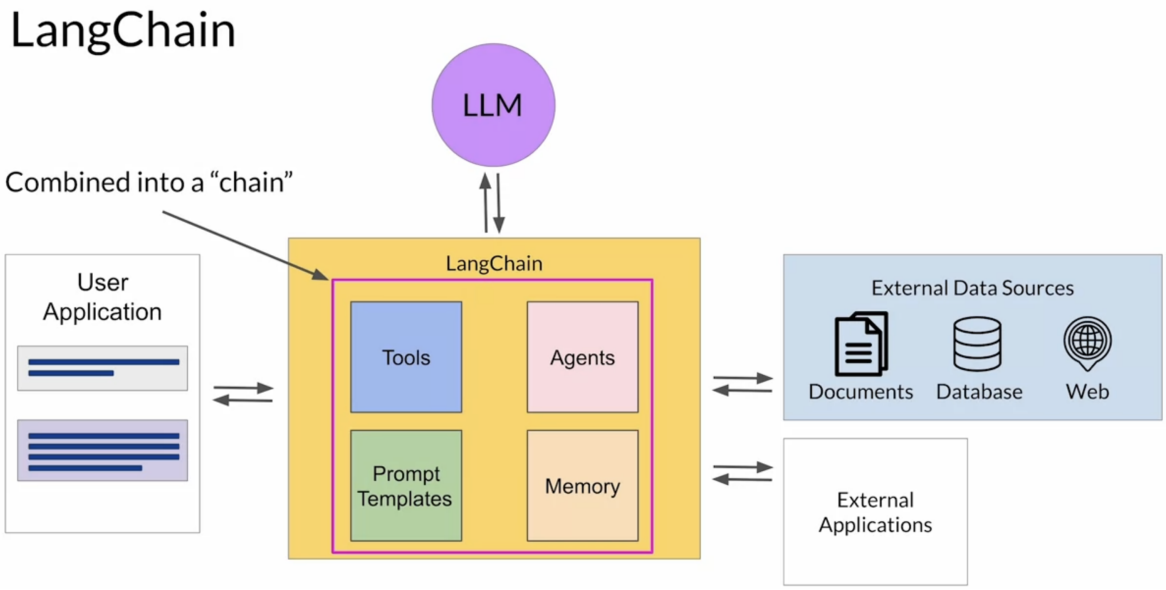
- LLMs can struggle with complex reasoning problems
  - chain-of-thought prompting can help by simulating a human reasoning steps

Program-aided Language Models (PAL)

- prompting with one-shot example
  - eg: translate math steps to python code
    - variables assignation, operations, etc
- PAL models
  - prompt: prompt template + PAL formatted prompt
  - completion: python script
  - you can input the code into a python interpreter

ReAct: Combining reasoning and action

- ReAct: synergizing reasoning and action in LLM
  - LLM + Websearch API
    - HotPot QA + Fever
  - use structured information
    - question → thought → action → observation
      - actions: search[entity], lookup[keyword], finish[answer]
      - need to define the action space through instructions
- building up the react prompt
  - react example(s)
  - add instructions to example(s)
  - pass the result to LLM for completion
- LangChain contains components necessary



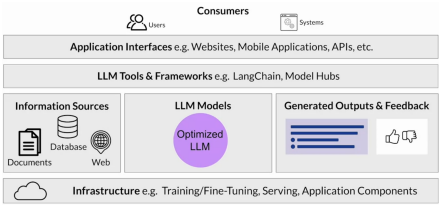
- build flexible chains & agents

- keep significance of scale in mind
  - larger models are better for more structured prompting methods

ReAct: Reasoning and action (reading)

LLM application architectures

- building generative applications



  - infrastructure
  - LLM models + information sources + generated outputs & feedback
  - LLM tools & frameworks (langchain, model hubs)
  - application interfaces

AWS Sagemaker JumpStart

- model hub

Course conclusion and ongoing research

Responsible AI

- special challenges of responsible generative AI
  - toxicity, hallucinations, IP
- how to mitigate these risks?
  - toxicity
    - careful curation of training data

        - train guardrails model to filter out unwanted content
        - diverse group of human annotators
    - hallucinations
        - educate users about how genAI works
        - add disclaimers
        - augment LLMs with independant, verified sources
        - defined use cases
    - IP
        - machine ‘unlearning”
- use genAI to test models
    - eg: create faces to see which faces the model would expect