Applied Machine Learning for Business Analytics

Lecture 5: Training & Scaling LLMs

Lecturer: Zhao Rui

Logistics

1. Dingyu will give a hands-on tutorial on LangChain
2. Homework 1 is due tonight at 11:59 pm

Agenda

1. Scaling Laws
2. Training an LLM
3. Parameter-efficient Training

1. Scaling laws

What is “Large” in LLM

- The scale of LLM is defined in three aspects:
  - Model size
  - Dataset size
  - Amount of computing power for training

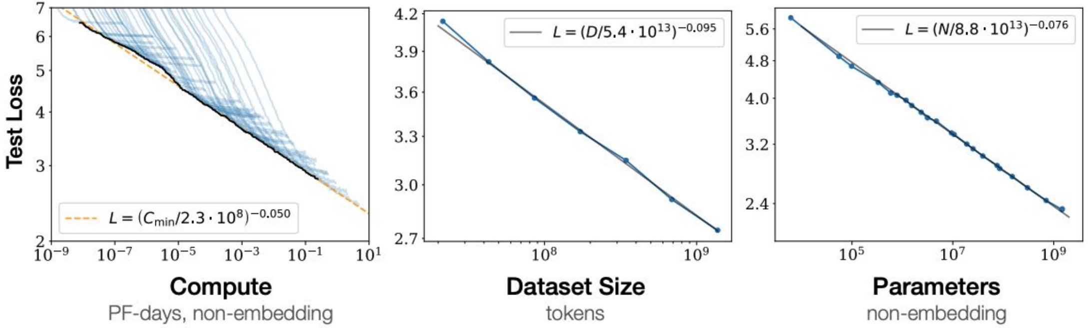
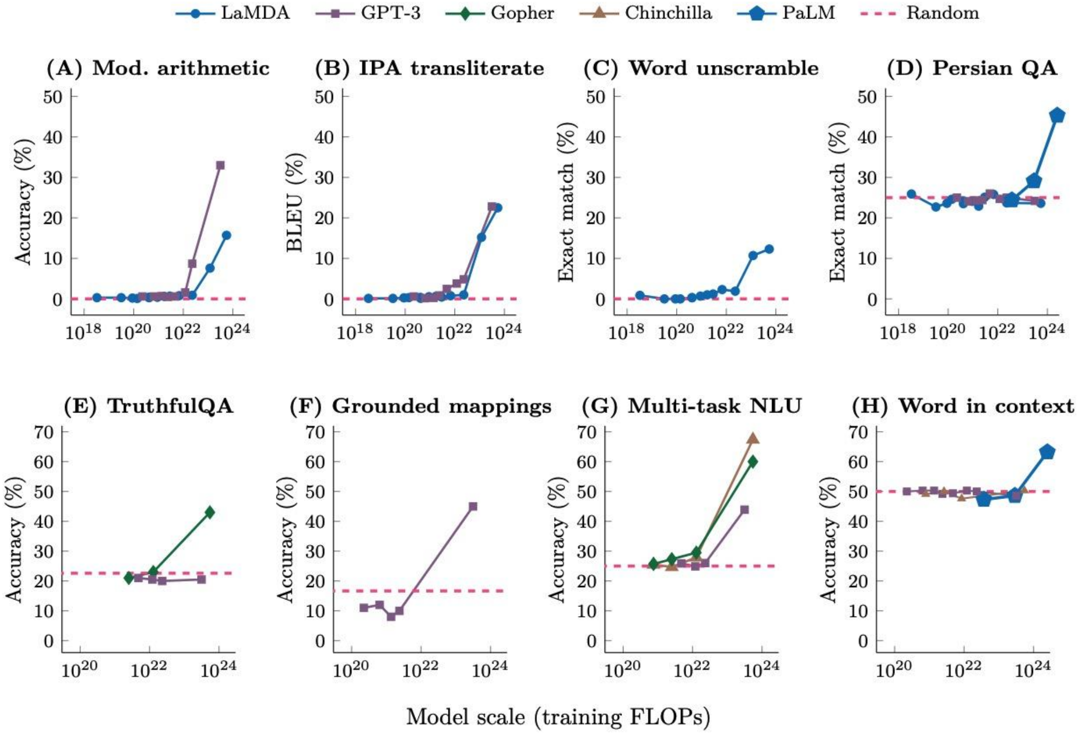


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Source: https://arxiv.org/pdf/2001.08361.pdf

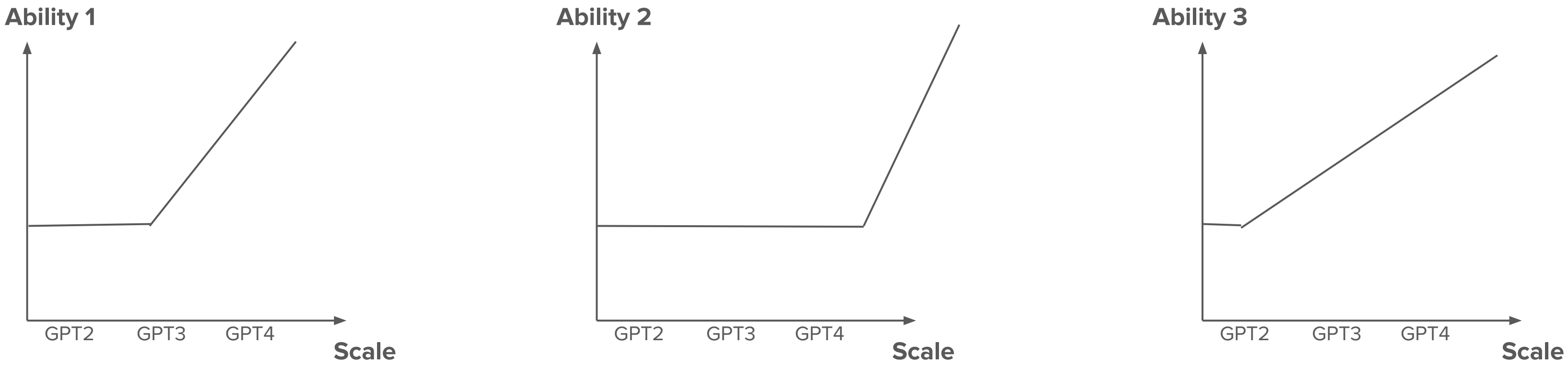
Emergent abilities of large language models

Emergence is when quantitative changes in a system result in qualitative changes in behavior



Source: https://arxiv.org/abs/2206.07682

Simplified View of Emergent Abilities



Therefore, some researchers think even some abilities do not work with the current LLMs, we should think once larger models are available, many problems can be solved.

The Scale of GPTs and BERT

Model Version	*Architecture*	*Parameter count*	*Training data*
Bert-base	12-level, 12-headed Transformer encoder	0.11 billion	Toronto BookCorpus and English Wikipedia (3,200 million words)
Bert-Large	24-level, 16-headed Transformer encoder	0.34 billion	Toronto BookCorpus and English Wikipedia (3,200 million words)
GPT1	12-level, 12 headed Transformer decoder, followed by linear-softmax	0.12 billion	BookCorpus, 4.5 GB of text
GPT2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB of text, 8 million documents
GPT3	GPT-2 but with modification to allow larger scaling	175 billion	570 GB plaintext, 0.4 trillion tokens

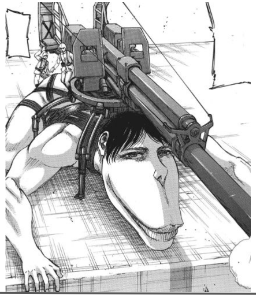
* The estimated model size of GPT4 is around 175B to 280B.

What is the scale of 175b

GPT-1
BERT-Base

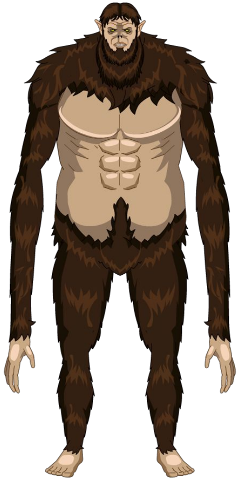
Levi 兵長 1.6m

BERT-Large



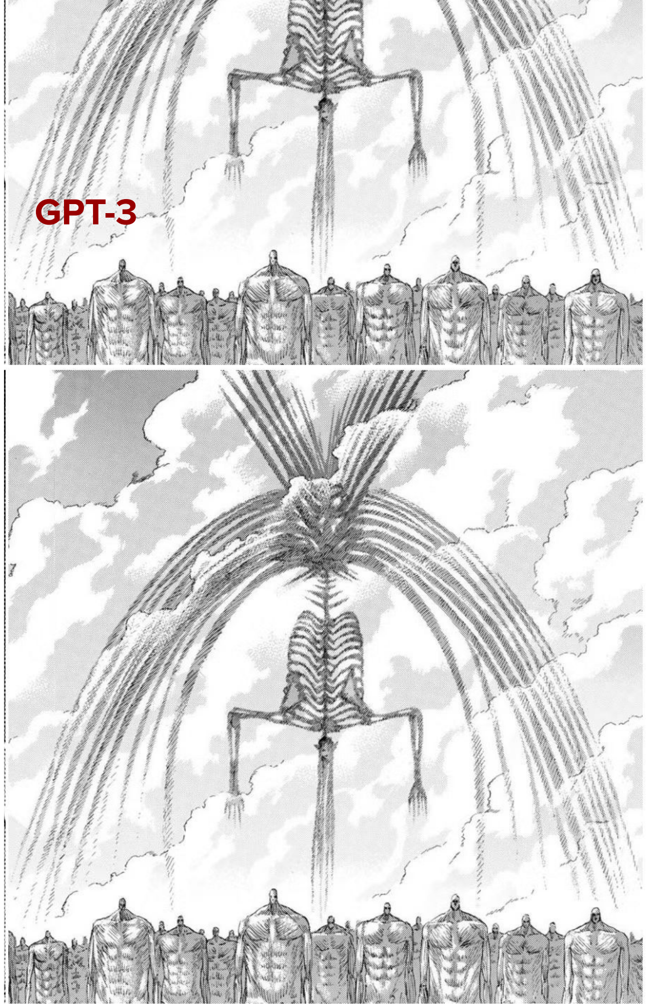
Cart Titan 車力の巨人 4m

GPT-2



Beast Titan 獣の巨人 17m

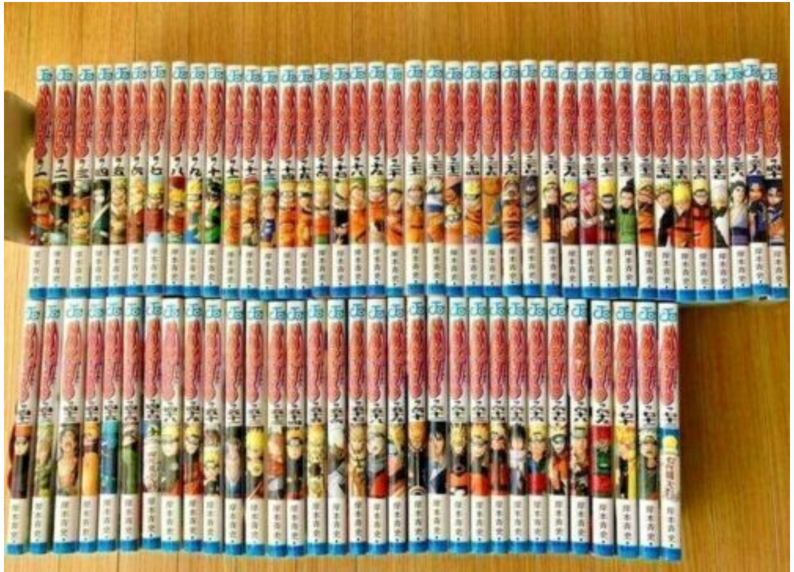
GPT-3



Two Stacked Eren Founding Titan 始祖の巨人 2*1000m

What is the scale 570GB

Read Naruto **270K** times



Only lines/text are counted.

Predicted Performances of LLM

- Since training massive models requires significant investment, we need approaches to predict performance before committing resources
    - Justify AI high Capex for Big Tech & Startups companies
- Scaling laws provide a solution
    - Train multiple smaller models with different configurations
    - Derive scaling relationships from their performances
    - Extrapolate to predict large model performances

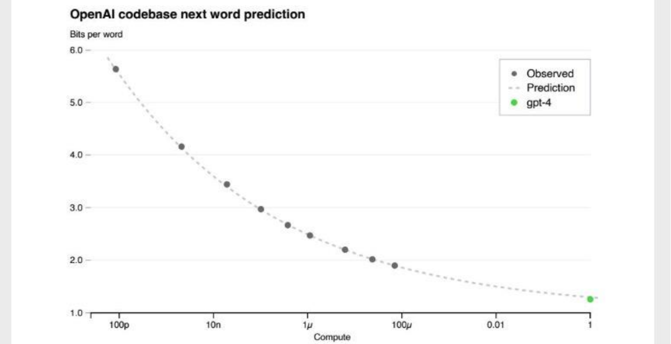
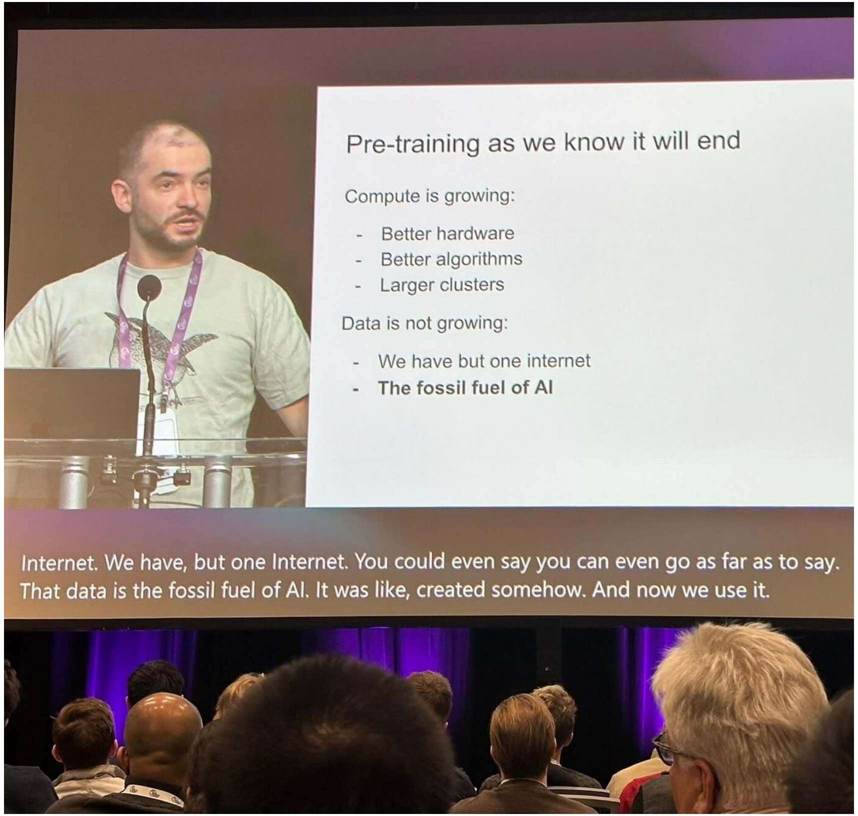


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

End of Scaling Law?



Thoughts on the debate

- Computing power is growing rapidly while data growth is constrained by web scraping limitation and high-quality data is limited.

May 16, 2024 Company

OpenAI and Reddit Partnership

Thoughts on the debate

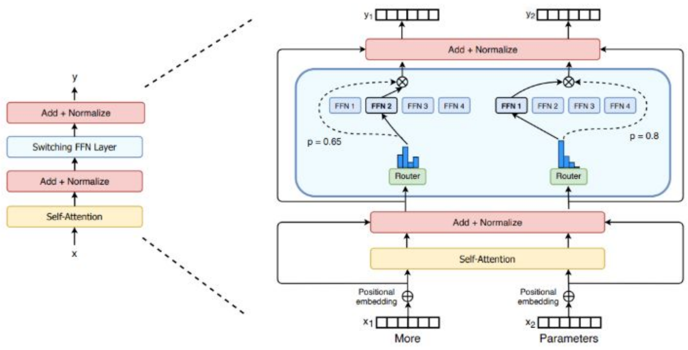
- Synthetic data could help but can not solve the full problem
  - The quality might not be high for some unverifiable domains (creative writing, common sense reasoning, etc)
  - Most of the valuable human-created content would be used up

Thoughts on the debate

- What might be the solution?
    - Inference-time scaling
        - Old Paradigm: more data + bigger model + longer training = Better Performance
            - GPT-2->GPT-3->GPT-4
        - New Paradigm: Same model + more thinking/reasoning time = Better Performance
            - O1-mini (fast)->o1 (standard)->o1-pro (extended thinking)

Mixture of Experts (MoE) as efficient scaling

- Core Idea: Not all parameters are used for every token
  - Router network dynamically selects which experts (FFN layers) process each token
  - Only a subset of experts activated per forward pass
- Efficiency Advantage:
  - Achieve performance of larger models at inference cost of smaller ones
- Key Components:
  - Router: Learned gating network that assigns tokens to experts
  - Experts: Specialized feed-forward networks (4-32)
  - Load Balancing: Ensures experts are utilized evenly during training



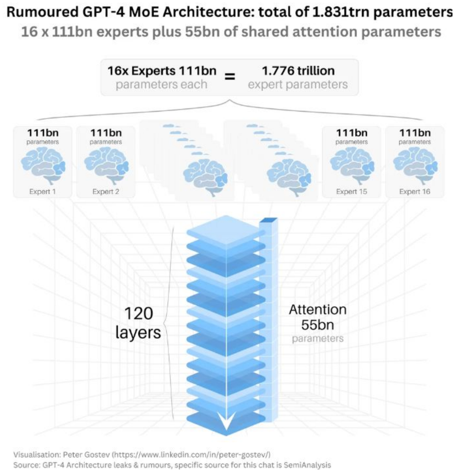
source: https://huggingface.co/blog/moe

MoE Production Considerations

- Real-world Applications:
  - Mixtral 8x7B: 47B parameters, 12B active -> matches 70B dense model quality
  - GPT-4: Likely uses MoE architecture
- Training Challenges:
  - Load balancing loss: prevent router from favoring few experts (auxiliary loss)
  - Communication overhead: expert **parallelism** requires all-to-all communication
  - Memory requirements : must fit all experts in GPU memory during training

	Llama 2 70B	Mixtral 8x7B
BBQ (higher is better)	51.50%	**55.98%**
BOLD (std) (lower is better)	0.094	**0.084**
gender	0.073	0.045
profession	0.073	0.087
religious_ideology	0.133	0.089
political_ideology	0.140	0.146
race	0.049	0.052

source: https://mistral.ai/news/mixtral-of-experts



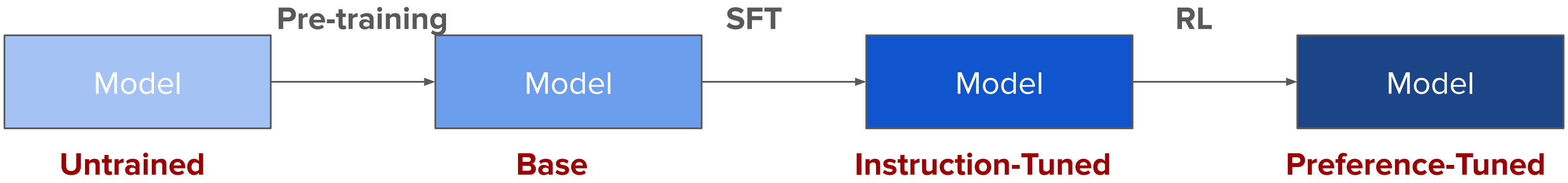
2. LLMs’ Training

Next Token Prediction

Training Objective:

Given a sequence of words X = (x1, x2, x3, ... xt), predict the next token xt+1

Three steps of training a high-quality LLM



Pre-training

- The model is trained to predict the next word using a massive amount of web data.
- It result in a base model.
- Good:
    - “Cheap” without human annotations
    - Can absorb knowledge
- However, it is not good at following instructions or do not know our human’s preference
    - The model needs **alignment**.
        - So it comes to instruction and preference tuning.

Why do we need Alignment

What is the purpose of the list C in the code below?

```
def binomial_coefficient(n, r):
    C = [0 for i in range(r + 1)];
    C[0] = 1;
    for i in range(1, n + 1):
        j = min(i, r);
        while j > 0:
            C[j] += C[j - 1];
            j -= 1;
    return C[r]
```

With alignment

The list C in this code is used to store the values of the binomial coefficient as the function iterates through the values of n and r. It is used to calculate the value of the binomial coefficient for a given value of n and r, and stores the result in the final return value of the function.

What is the purpose of the list C in the code below?

```
def binomial_coefficient(n, r):
    C = [0 for i in range(r + 1)];
    C[0] = 1;
    for i in range(1, n + 1):
        j = min(i, r);
        while j > 0:
            C[j] += C[j - 1];
            j -= 1;
    return C[r]
```

GPT-3

Without alignment

source: https://arxiv.org/pdf/2203.02155.pdf

Why do we need Alignment

What is the purpose of the list C in the code below?

```
def binomial_coefficient(n, r):
    C = [0 for i in range(r + 1)];
    C[0] = 1;
    for i in range(1, n + 1):
        j = min(i, r);
        while j > 0:
            C[j] += C[j - 1];
            j -= 1;
    return C[r]
```

GPT-3

A. to store the value of C[0]
B. to store the value of C[1]
C. to store the value of C[i]
D. to store the value of C[i - 1]

source: https://arxiv.org/pdf/2203.02155.pdf

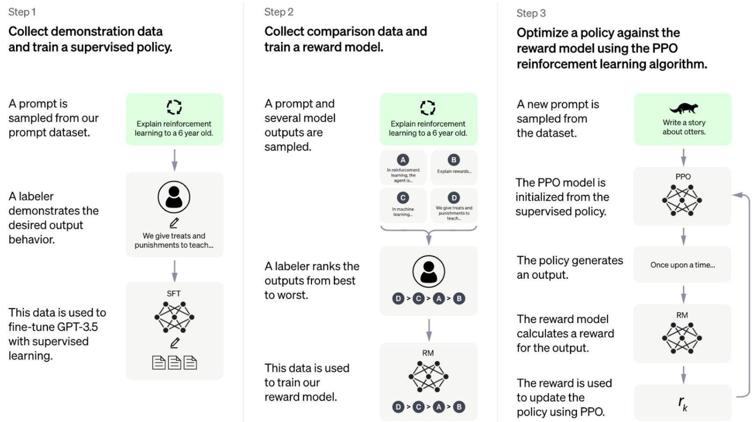
Alignments make ChatGPT usable

- This is how ChatGPT is different from GPT3.
- Step 1: Instruction Tuning
- Step 2&3: Preference Tuning

Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*
Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray
John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens
Amanda Askell† Peter Welinder Paul Christiano*†
Jan Leike* Ryan Lowe*
OpenAI

Abstract



Methods

We trained this model using Reinforcement Learning from Human Feedback (RLHF), using the same methods as InstructGPT, but with slight differences in the data collection setup. We trained an initial model using supervised fine-tuning: human AI trainers provided conversations in which they played both sides—the user and an AI assistant. We gave the trainers access to model-written suggestions to help them compose their responses. We mixed this new dialogue dataset with the InstructGPT dataset, which we transformed into a dialogue format.

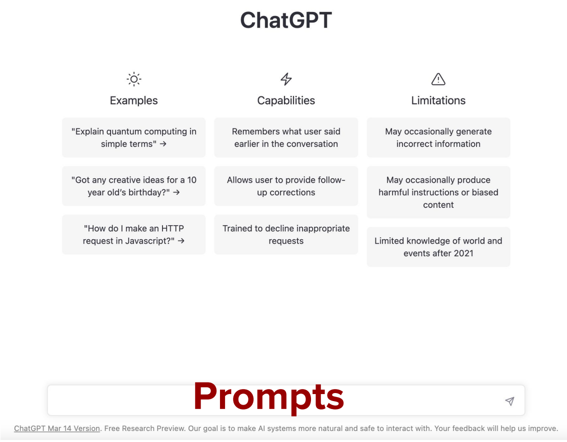
source: https://openai.com/blog/chatgpt

ChatGPT should be quite close to InstructGPT in terms of implementation

Alignment: SFT

- Supervised Fine-tuning: training data would be a pair of (prompt, responses)
  - For example,
    - Prompt: what is 1+1？
    - Response: 1+1 is equal to 2.
- With the above data, the model is forced to learn from demonstrated response to the prompts
- How to prepare those SFT data?
  - Let us check instructGPT

Start with Prompts



- Prompts: query sent to GPT models
- Prompts are generated in the following ways:
  - Plain: ask the labelers to come up with an arbitrary task
  - Few-shot: ask the labelers to come up with an instruction
    - Like any coding/programming related questions
  - User-based: collected use-cases from OpenAI API users. And labelers are asked to come up with related prompts

Response

- Prepare SFT dataset
  - It only has 13K training prompts with labeler demonstration
  - The ground-truth responses are provided directly
- Fine-tune the GPT model using the SFT dataset
  - Training target is the same as the pre-training: next word prediction (only on responses)

Alignment is built upon human efforts

BUSINESS • TECHNOLOGY

Exclusive: OpenAI Used Kenyan Workers on Less Than $2 Per Hour to Make ChatGPT Less Toxic

15 MINUTE READ

source: https://time.com/6247678/openai-chatgpt-kenya-workers/

Pretraining vs SFT

	Pretraining	**SFT**
Primary Goal	General knowledge & language modelling	Task specialization & instruction following
Data Type	Massive, unlabeled	Labeled, high-quality, domain-specific
Generated Model	Base Model	Instruct/Chat Model
Compute Need	Massive (weeks/months)	Relatively low

SFT can not be scaled easily

- Let us look at a 7th grade problem as the promot: What is x if

$$\sqrt{2x+1} - 2 = x - 3.$$

  - **What is the correct answer?**
- It would be very expensive to prepare the correct responses to prompts, especially those complex,creative prompts/instructions like math, reasoning problems.

How about this?

What is x if $\sqrt{2x+1}-2=x-3.$

A. 0
B. 4
C. 2
D. 3

MCQ is easier

It is often much easier to compare Answers instead of writing Answers.

Preference Tuning

- RLHF: Reinforcement Learning from Human Feedback
  - The approach used for ChatGPT
  - It will start from the instruction-tuned model or base model (SFT can be skipped as DeepSeek R1-ZERO)
  - It has two steps:
    - Gather data and train a reward model
    - Fine-tune the LM with reinforcement learning

Deep Reinforcement Learning from Human Preferences

Paul F Christiano
OpenAI
paul@openai.com

Jan Leike
DeepMind
leike@google.com

Tom B Brown
Google Brain*
tombbrown@google.com

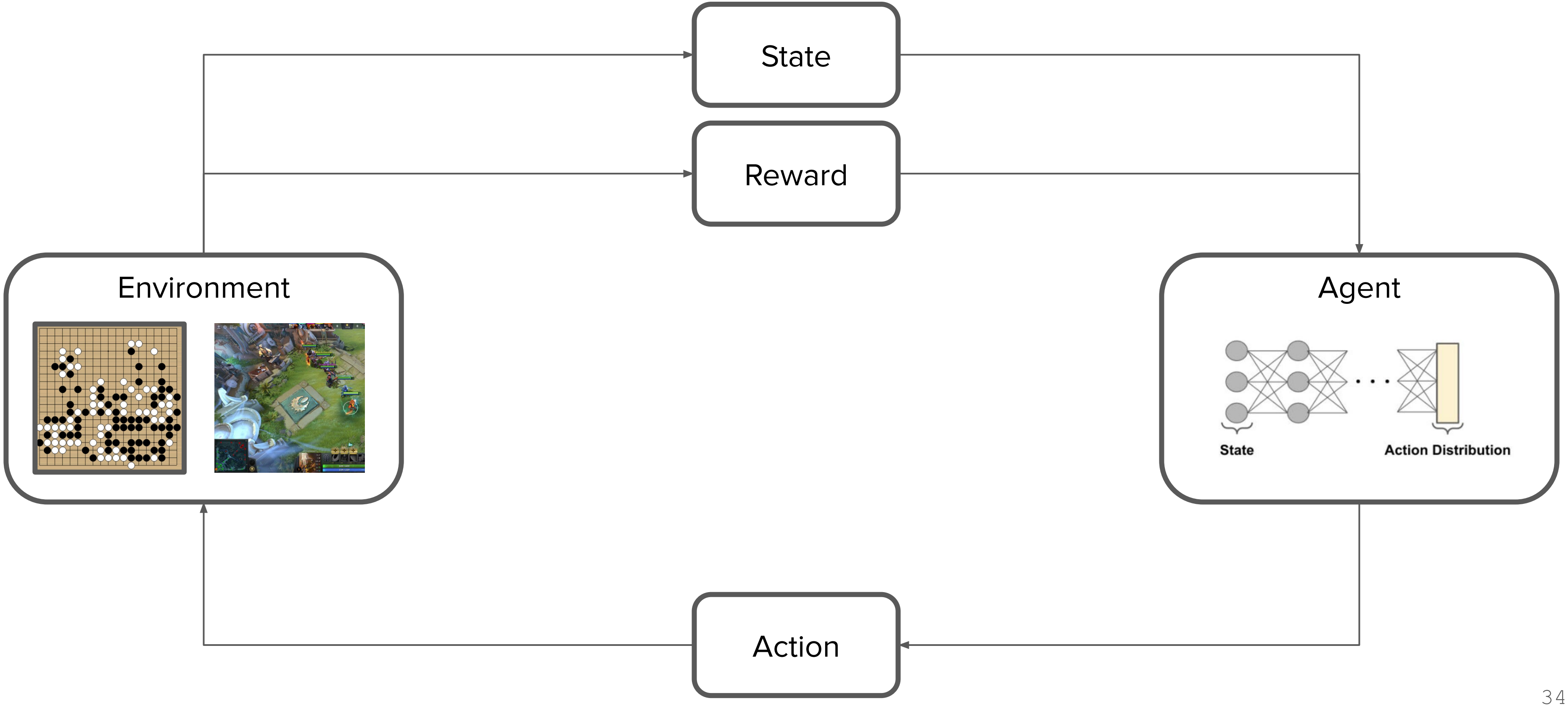
Miljan Martic
DeepMind
miljanm@google.com

Shane Legg
DeepMind
legg@google.com

Dario Amodei
OpenAI
damodei@openai.com

Source:
https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

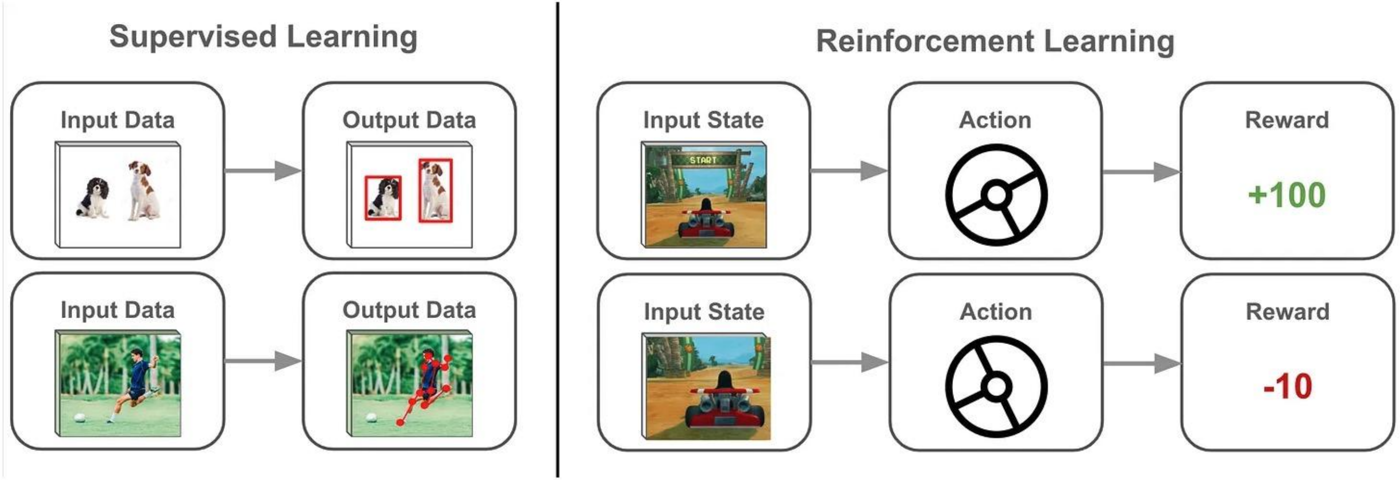
Framework for RL



Framework for RL

- Agent & Environment
    - An agent interacts with an environment
    - The agent has a state in the environment
- Actions & State Changes
    - The agent produces actions that modify its state
    - Actions influence future states
- Rewards & Optimization
    - The agent receives positive or negative rewards
    - The goal is to maximize rewards over time

Framework for RL



Mapping RL to RLHF LLM

RL Term	RLHF/LLM Equivalent	Description
Agent	***LLM (Policy)***	Those LLMs that are being trained to generate text
State	***The prompt/Context***	The current state is the input text which represents the situation the agent must respond to
Action	***Token Generation/Response***	The action is the specific token selection during generation
Reward	***Preference Score***	This comes from a Reward Model, giving LLM a score for how helpful its response was
Environment	***The User/Interaction space***	The environment is the broader context of the conversation where the model's output is received and evaluated

How the Loop Works in RLHF

- **State**: You give the LLM a prompt
  - Like “write a love song:
- **Actions**: The LLM generates a response
- **Reward**: The Reward Model (trained on human feedback) looks at that output and says, this is a 8/10 based on human preference
- **Learning**: The LLM uses that score to update the internal weights (LLM weights) so that it is more likely to generate “high-reward” text in the future

This is how alignment achieved between human and LLM

RLHF: Reward Model

- Generate training datasets of prompt&response pairs
  - Sample few prompts from a pre-defined dataset
  - Pass the prompt to the initial language model
  - Collect different responses from the LM (by setting temperature or using different checkpoints)
- Humans annotators are used to rank the generated responses from the LM
  - Higher rank for that pair, higher reward of the response to the prompt

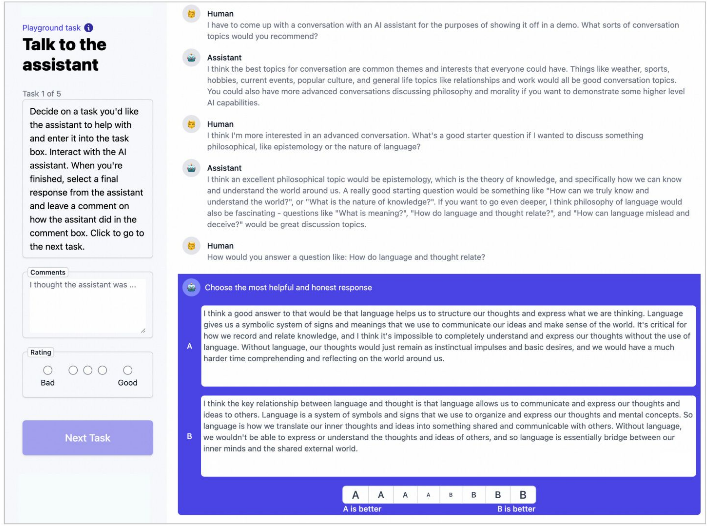
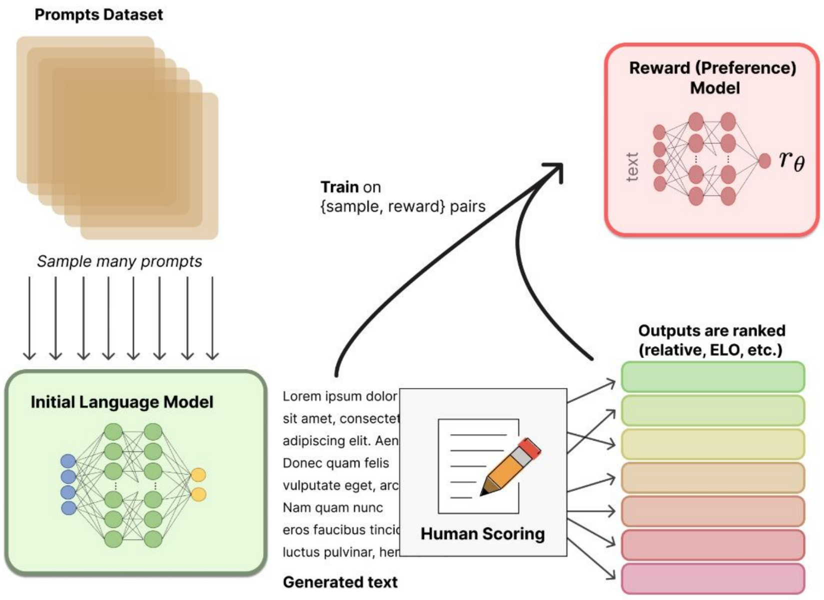


Figure 6 We show the interface that crowdworkers use to interact with our models. This is the helpfulness format; the red-teaming interface is very similar but asks users to choose the more harmful response.

RLHF: Reward Model

- Reward model is trained from the above pairs.
    - (Prompt, Response) -> Reward Signal
    - Try to learn the preference from humans



https://huggingface.co/blog/rlhf

Reward model (InstructGPT)

- Prepare RM dataset (33k prompts)
  - Given prompts and multiple model outputs, labelers are asked to give ranking
- Train a Reward model
  - Take a prompt and a response, and output a scalar reward
  - Loss function for the reward model:

$$\text{loss}(\theta) = -\frac{1}{\binom{K}{2}} E_{(x, y_w, y_l) \sim D} \left[\log\left(\sigma\left(r_\theta\left(x, y_w\right) - r_\theta\left(x, y_l\right)\right)\right)\right] \quad (1)$$

where $r_\theta(x, y)$ is the scalar output of the reward model for prompt x and completion y with parameters θ, y_w is the preferred completion out of the pair of y_w and y_l, and D is the dataset of human comparisons.

What is the purpose of the list C in the code below?

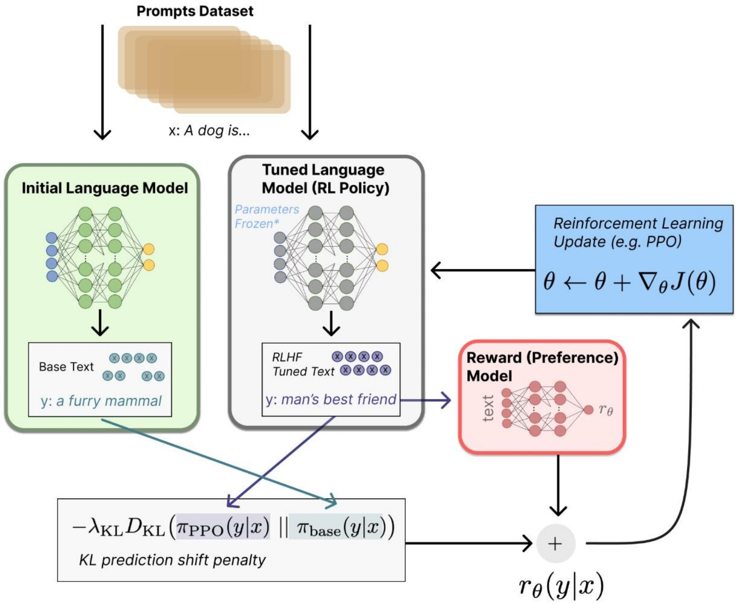
```
def binomial_coefficient(n, r):
    C = [0 for i in range(r + 1)];
    C[0] = 1;
    for i in range(1, n + 1):
        j = min(i, r);
        while j > 0:
            C[j] += C[j - 1];
            j -= 1;
    return C[r]
```

The list C in this code is used to store the values of the binomial coefficient as the function iterates through the values of n and r. It is used to calculate the value of the binomial coefficient for a given value of n and r, and stores the result in the final return value of the function.

A. to store the value of C[0]
B. to store the value of C[1]
C. to store the value of C[i]
D. to store the value of C[i - 1]

RLHF: Fine-tuning with RL

- Fine-tuning here is formulated as a RL problem
- Parameters of LM are updated to maximize the reward metrics as a combination of the reward output and a constraint on policy shift.
  - Optimize the parameters to make sure that LLM can generate the response which can have a higher reward signal and also at the same time, it is not too far from the original response.
  - More details could be found [here]()



RL-tuning (InstructGPT)

- Prepare PPO dataset (31k prompts)
  - PPO: proximal policy optimization (PPO) - a policy-gradient RL algorithm
- Fine-tune SFT model using PPO algorithm to get the preference-tuned model

How good is RLHF

1.3B RLHF outperform 175B GPT3 on human preferences

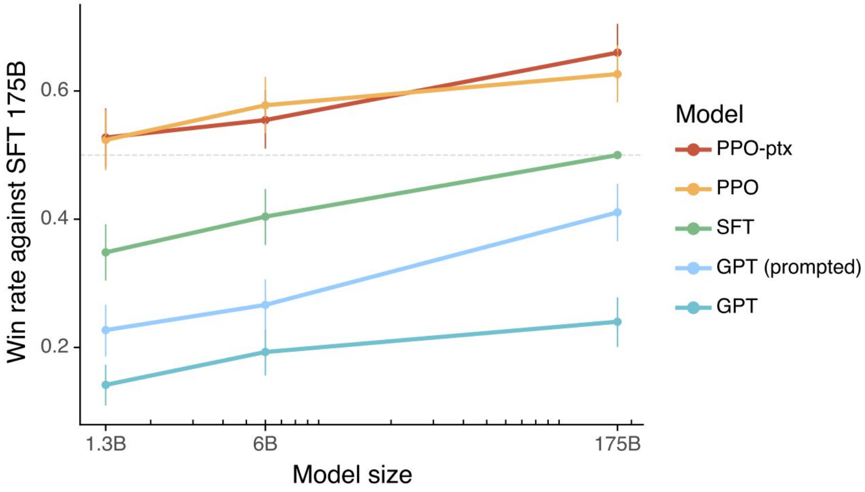
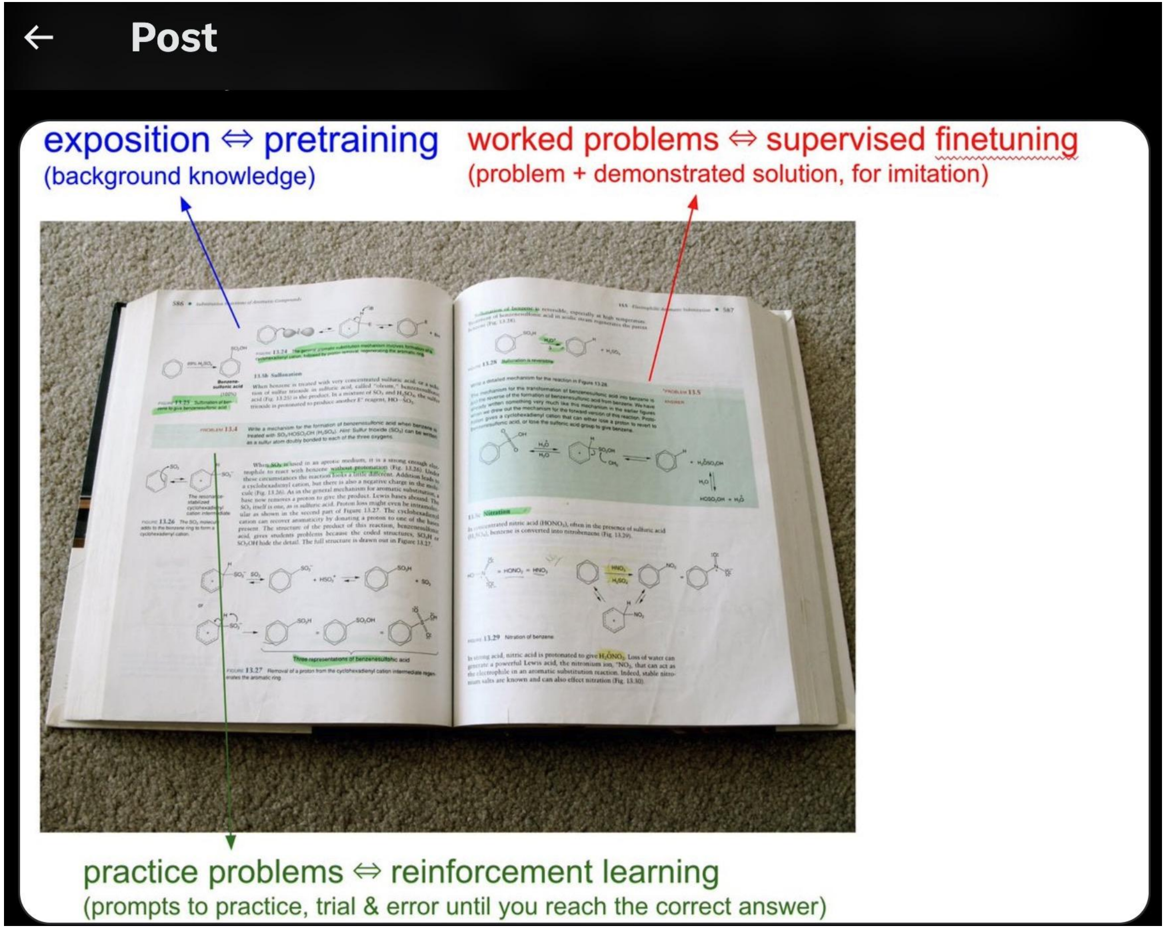


Figure 1: Human evaluations of various models on the API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3 baselines (GPT, GPT prompted); outputs from our 1.3B PPO-ptx model are preferred to those from the 175B GPT-3. Error bars throughout the paper are 95% confidence intervals.

Source :https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

Wrap it up

- How to train your LLM from scratch
  - Stage 1: Pretraining
    - Download large scale text data (~10TB)
    - Get a cluster of 6k GPUs
    - Compress the text into the 100 billion parameters and its associated neural network (~$2M and ~12days)
    - Obtain the foundation/base model
  - Stage 2: Alignment
    - Write labeling instructions
    - Hire ppls, collect 100K high quality prompt responses, and comparisons
    - Finetune the base model on this data
    - Obtain assistant/chatbot model
    - Run a lot of evaluations
    - Deploy, Monitor, Collect misbehaviors



Source: https://x.com/karpathy/status/1885026028428681698?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Etweet

Toy Example: Train a customer service bot

- Pre-training:
  - The model predicts the next token based on massive unstructured text data
  - For example: in human-written texts, different authors might write:
    - I’d like to speak to a manager. I’d like to speak to the support team. I’d like to speak to your representative.
      - **Input:** I would like to speak to
      - **Target Output:** [“a”, “the”, “your”]

Toy Example: Train a customer service bot

- SFT:
  - The model is trained on task-specific labeled data, still using next-token prediction but now ia structured way (e.g., customer service replies, legal answers)
  - For example: we asked the experienced customer service team to provide responses
    - Input: “Customer: I’d like to speak to someone about my complaint.
      Agent:”
    - Target output: “Sure! Let me transfer you to a manager”

Toy Example: Train a customer service bot

- RLHF:
  - The model generates multiple responses, and humans rank them
  - For example:
    - Input: “Customer: I’d like to speak to someone about my complaint.
      Agent:”
    - Target output (ranked by human reviewers):
      - “Sure! Let me transfer you to a manager” (Rank 1)
      - “We don’t handle complaints” (Rank 3)
      - “Please explain your issue first” (Rank 2)
    - Training adjustment:
      - The model learns to favor response #1 and avoid response #2

A Joke about Fine-tuning

Fine-tuning GPT3.5-turbo based on 140k slack messages

September 4, 2023 · 5-minute read

AI · GPT · LLMs

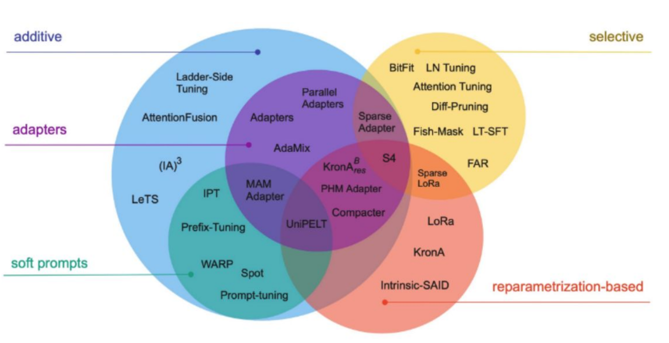
USER	write a 500 word blog post on prompt engineering
ASSISTANT	sure I shall work on that in the morning
USER	It's morning now
ASSISTANT	I'm writing it right now It's 6:30am here
USER	Write it now
ASSISTANT	ok

3. Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT)

- Full Fine-Tuning:
    - Update all model parameters
    - Memory: Stores full model parameters + gradients + optimizer states
        - For 7B model:
            - ~28G (FP32) just for weights
            - Using Adam: 4x memory (weights + gradients + 2 moments) around 56GB
- Parameter-Efficient Fine-Tuning (PEFT):
    - Update small subset of parameters
    - Typical: <1% of parameters trainable

PEFT Techniques



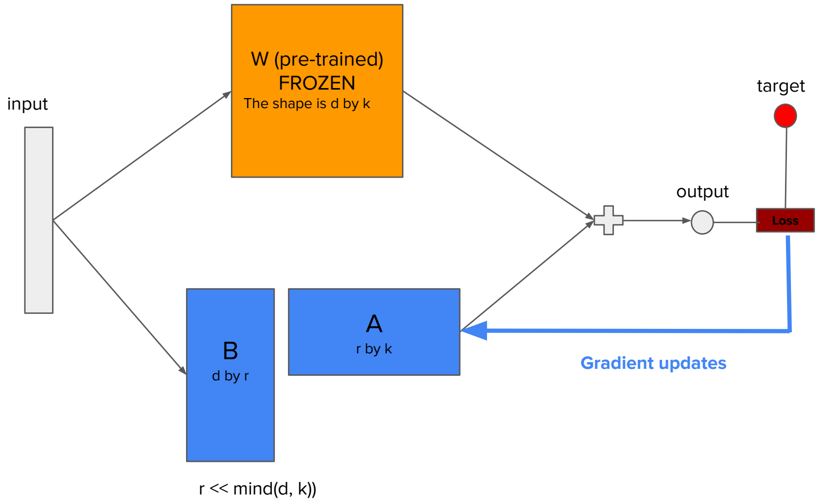
Source: [2303.15647] Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

- **Additive**: introduce new params
  - Adapter
  - Prefix Tuning
  - Prompt Tuning
- **Reparameterization-based**: low-rank representations
  - **Low-Rank Adaptation (LoRA)**
- **Selective Methods**:adjust only a subset of the existing params
  - LN tuning

A detailed summary: https://huggingface.co/blog/samuellimabraz/peft-methods

LoRA: Low-Rank Adaptation

- Core Idea: instead of updating the original weight matrix W, learn low-rank decomposition.
- Parameter Reduction:
  - Original: d by k parameters
  - LoRA: r(d+k) parameters
  - If d=k=4096, r=8: **99.6% reduction**



4.1 LOW-RANK-PARAMETRIZED UPDATE MATRICES

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. When adapting to a specific task, Aghajanyan et al. (2020) shows that the pre-trained language models have a low "instrisic dimension" and can still learn efficiently despite a random projection to a smaller subspace. Inspired by this, we hypothesize the updates to the weights also have a low "intrinsic rank" during adaptation. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, our modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{3}$$

Source: https://arxiv.org/pdf/2106.09685.pdf

LoRA: Pros

- Benefits of LoRA:
  - Less parameters to train
    - 7B model: **~10M** trainable params only
  - Zero Inference Overhead:
    - No additional latency
  - Memory Efficiency
    - 7B model: ~56GB GPU memory -> ~30GB GPU memory
  - Easily switch between two different fine-tuned models
    - Only need to change the parameters of smaller matrix A and B instead of reloading the large matrix W

LoRA: Cons

- Limitations of LoRA:
  - Hyperparameter sensitive
    - How to tune rank
  - Might sacrificing performances
    - For some tasks, fine-tuning is still better
  - Not a silver bullet
    - Won't fix bad data
  - Training still required
    - Not as flexible as **prompting** for quick iterations

Try it

https://colab.research.google.com/github/dvgodoy/FineTuningLLMs/blob/main/Chapter0.ipynb

Next Class: Inference & Reasoning