School of CIT –
Social Computing
Research Group

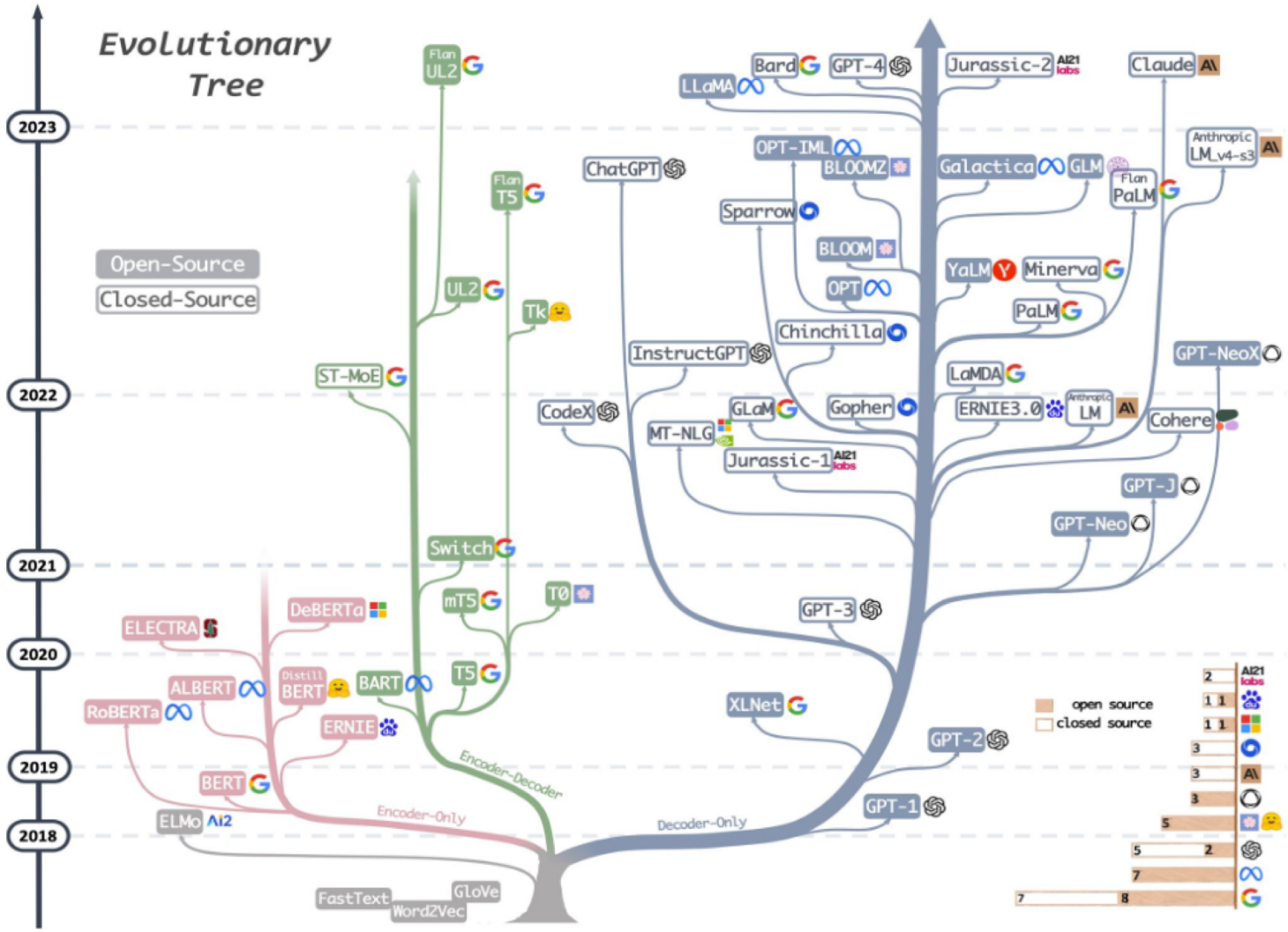
Technical
University
of Munich

Advanced Natural Language Processing
CIT4230002

Prof. Dr. Georg Groh
Edoardo Mosca, M.Sc.

Lecture 2
Fine-tuning

Overview | Transformer Landscape



- So many transformers to choose from. But how do you use them effectively (and efficiently)?

Overview | Motivation for Fine-Tuning

- It allows us to make use of massive pre-trained LMs
- General language features can help to improve downstream task performance
- Fine-tuning can convergence much faster than training from scratch
- Also it is more data-efficient
- Fine-tuning can be seen as a shortcut in effort without losing performance

Overview | Terms Definitions

- Pretraining (-> Base Model)
  - Predicting next words
  - Usually large companies (OpenAI, Google, etc.)
  - Costs millions of $$$

- Supervised Fine-tuning (-> SFT Model)
  - Supervised learning of model for downstream tasks (classification, instructions, etc.)
  - Can get very cheap (1-2 GPUs)

- Human Preference Tuning
  - Reward the model for behaving according to human expectations (friendly, non harmful, etc.)
  - Can also get very cheap
  - Usually for Chat applications

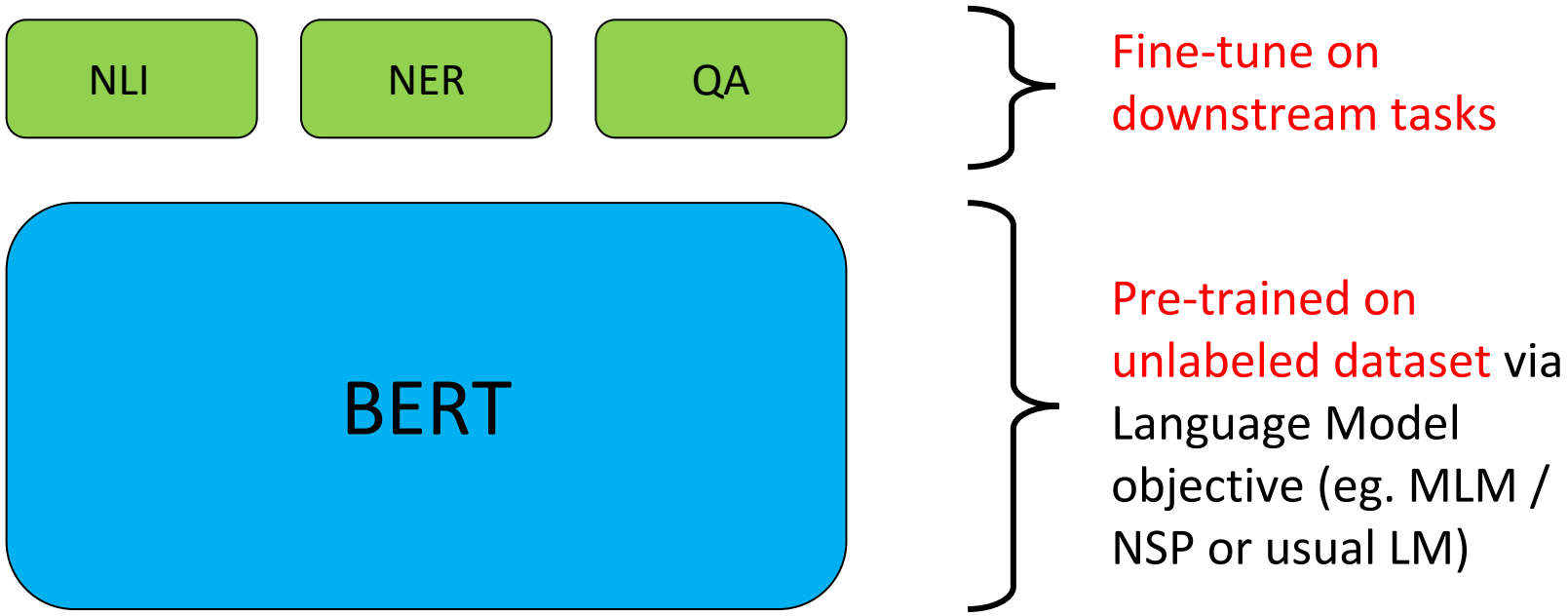
Fine-tuning

(this lecture)

Supervised Fine-Tuning

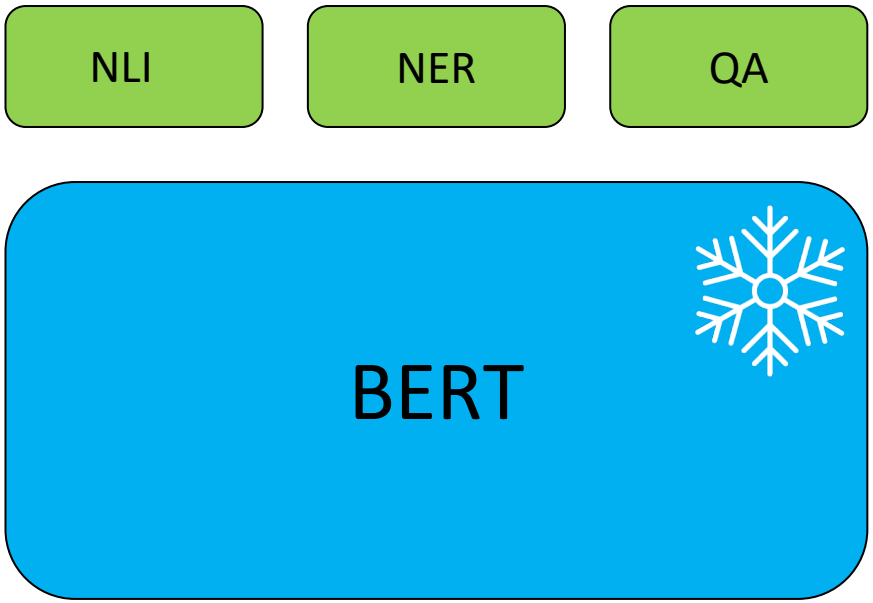
Traditional fine-tuning

- In the most traditional setting, all parameters are re-trained



Traditional fine-tuning with freezing

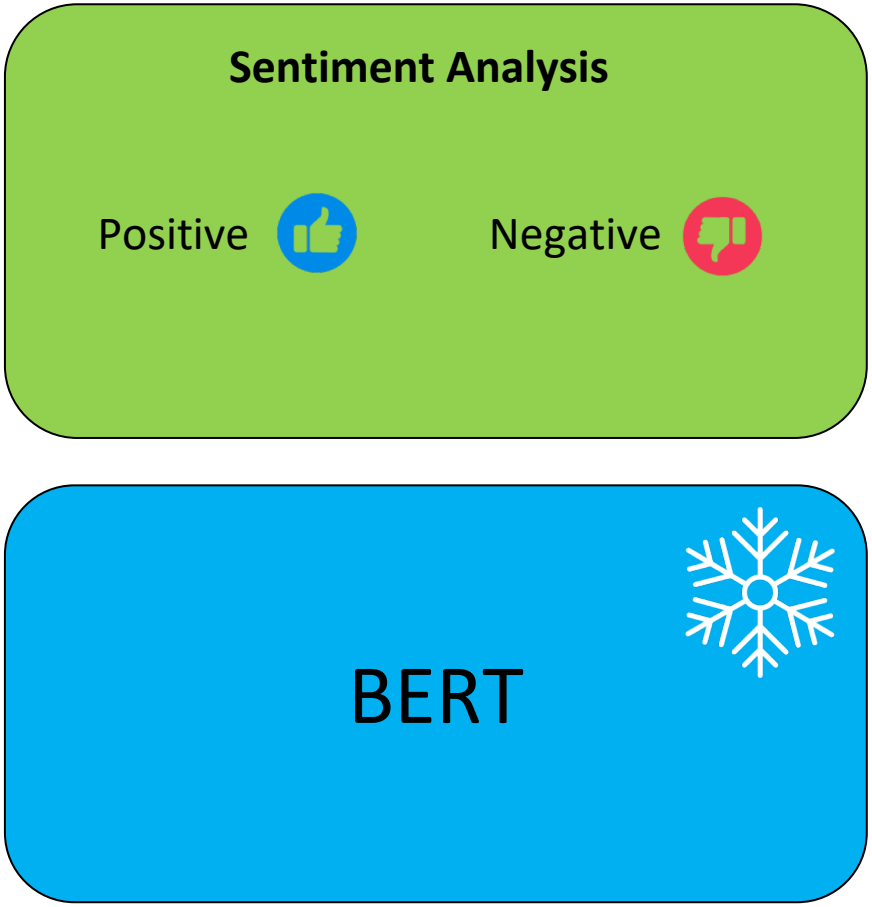
- Freeze the LM,which acts as a language feature extractor
  - How many and which layers to freeze is a hyperparameter
- Only retraining parts of layer (biases, attention weights) is also possible



- Without freezing: retraining > 300Mil params.
- With freezing: usually retraining only a few thousands

LM Prompting | Motivation

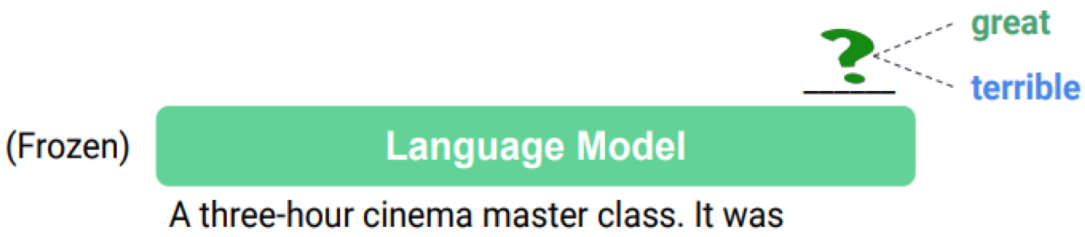
- So far, we would need additional parameters for each downstream tasks, e.g. sentiment analysis, MT, NLI, etc.



- Solution: prompting

LM Prompting | Motivation

- Prompting makes it possible for downstream tasks to take the same format as the pre-training objectives (language modeling) by prepending some text before the test input



P1 = P(It was great! | A three-hour cinema master class.)
P2 = P(It was terrible! | A three-hour cinema master class.)

P1>P2 "positive"
P1<P2 "negative"

- The idea was proven effective in GPT-3
- Requires no new parameters nor retraining existing parameters

LM Prompting | Zero-Shot

- Simple prompting corresponds to zero-shot learning. The model predicts the answer directly (task description is not necessary).

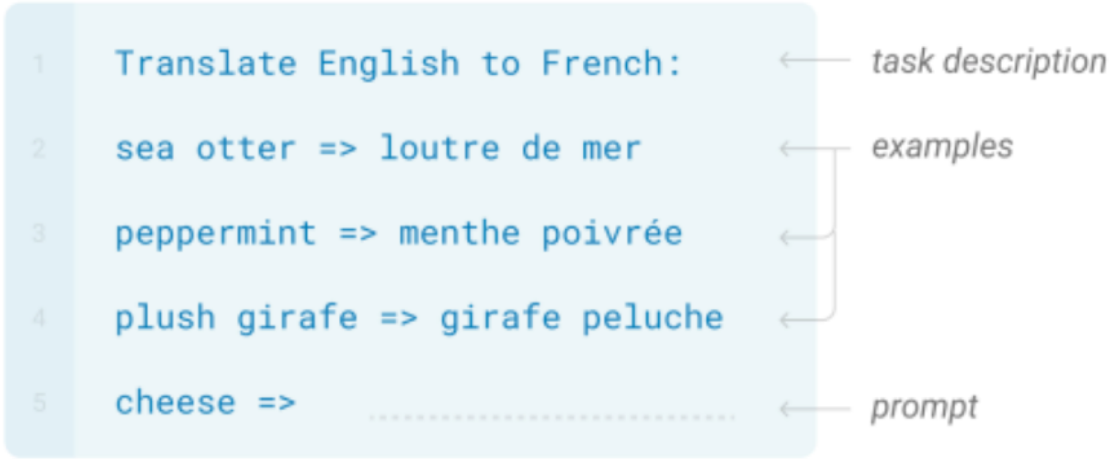
```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

- If add one example, then it is one-shot learning

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

LM Prompting | In-Context Learning

- Prompting that contains demonstrations (i.e. examples) of the task to be performed is called in-context learning
  - Demonstrations are extremely data-efficient (up to 100 traditional datapoints)



- Generalization: few-shot learning is in-context learning with no-to-little data (N-way-K-shot learning: N classes, K instances)

LM Prompting | Terminology

- Pattern: function that maps input to text (a.k.a. template for x)
  - Example: f(<x>) = “Review: <x>”
- Verbalizer: function that maps label to text (a.k.a. template for y)
  - Example : v(<y>) = “Sentiment: <y>”

Review: An effortlessly accomplished and richly resonant work.	Sentiment: positive
Review: A mostly tired retread of several other mob tales.	Sentiment: negative
Review: A three-hour cinema master class.	Sentiment: _____

LM Prompting | Patterns & Verbalizers

- Picking suitable patterns and verbalizers is an active field of research
  - Part of prompt engineering (includes hand-crafted, gradient- or heuristic based prompts)

Test data: (x, y) **Train data:** $(x_1, y_1, \cdots, x_k, y_k)$ **Pattern:** f **Verbalizer:** v

Zero-shot prompting: $\text{argmax}_{y \in \mathcal{Y}} P_{\text{LM}}(v(y)|f(x))$

In-context learning: $\text{argmax}_{y \in \mathcal{Y}} P_{\text{LM}}(v(y)|f(x_1), v(y_1), \cdots, f(x_k), v(y_k), f(x))$

Demonstrations
(= k-shot learning)

LM Prompting | Patterns & Verbalizers

- Choosing a prompt is non-trivial since LLMs exhibit large variance over different patterns and verbalizers

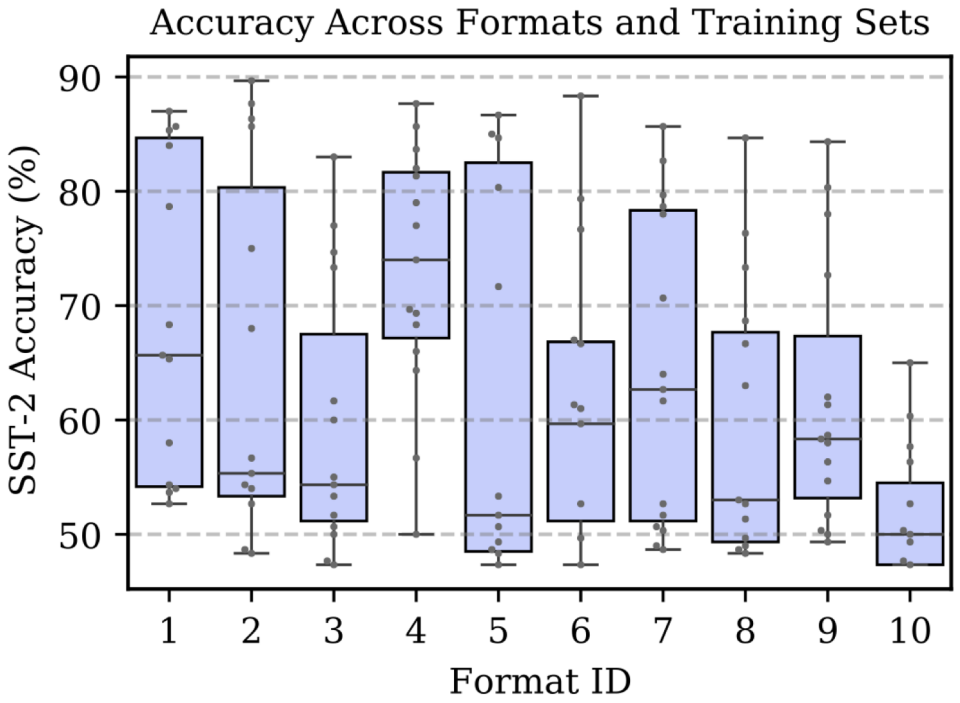


Figure 3. There is high variance in GPT-3's accuracy as we change the **prompt format**. In this figure, we use ten different prompt formats for SST-2. For each format, we plot GPT-3 2.7B's accuracy for different sets of four training examples, along with the quartiles.

[7]

LM Prompting | Findings

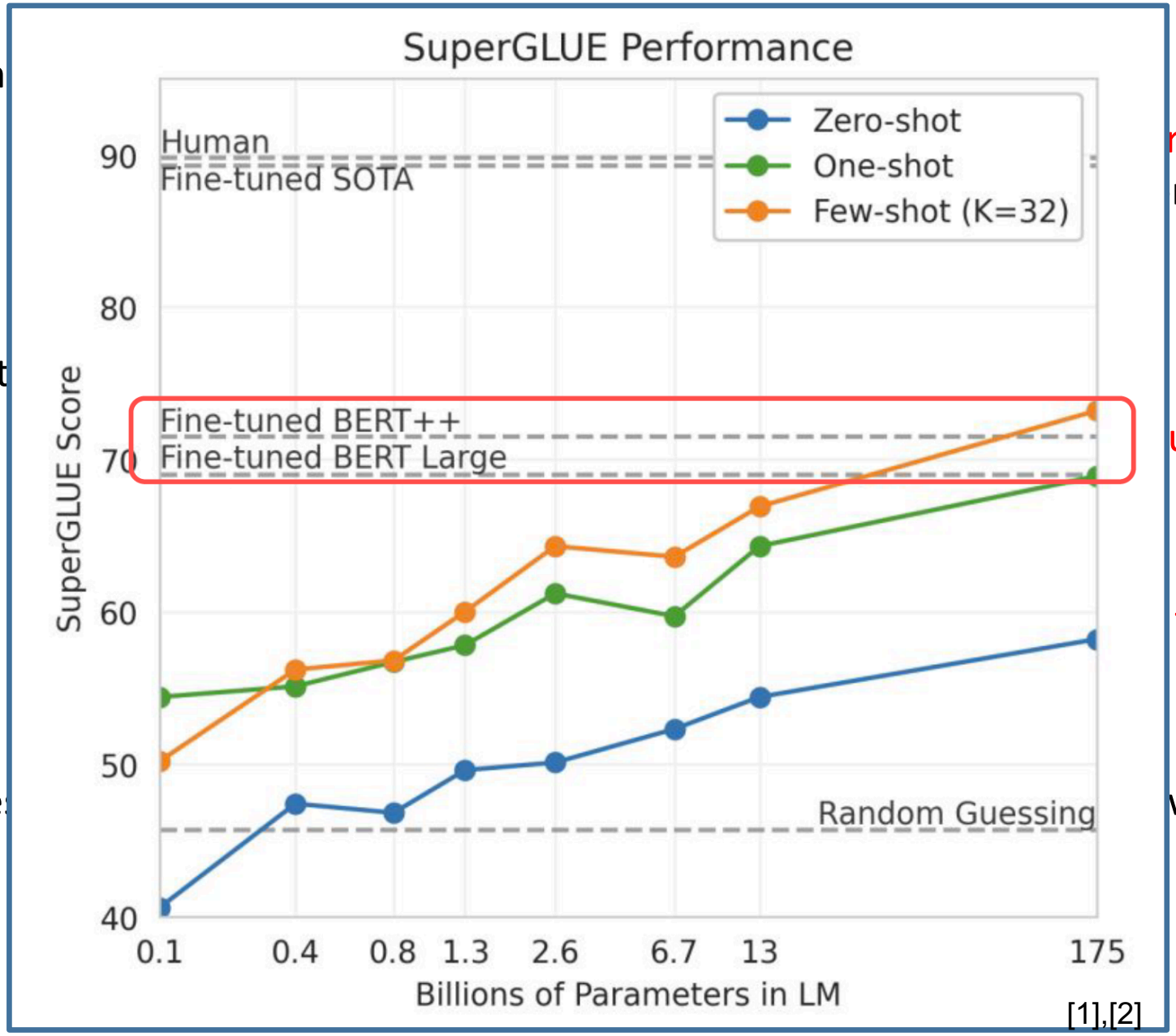
- Uncertain how and why in-context learning works exactly:
  - Some research suggests it is more about locating a task learnt during pre-training than actually learning a new task, thus calling it in-context learning can be misleading

- Extrapolation of LM is limited:
  - Different input distribution (eg. another corpus) and shifted output space distribution decrease performance
  - Demonstrations not seen as ordered pairs
  - In-context learning is also highly dependent on choice, order and term frequency

- Despite limitations, GPT-3 and following models empirically perform well on unseen tasks (also synthetic ones)

LM Prompting | Findings

- Un
  - ring
  - ntext
- Ext
  - ut
  - term
- De
  - well
  on

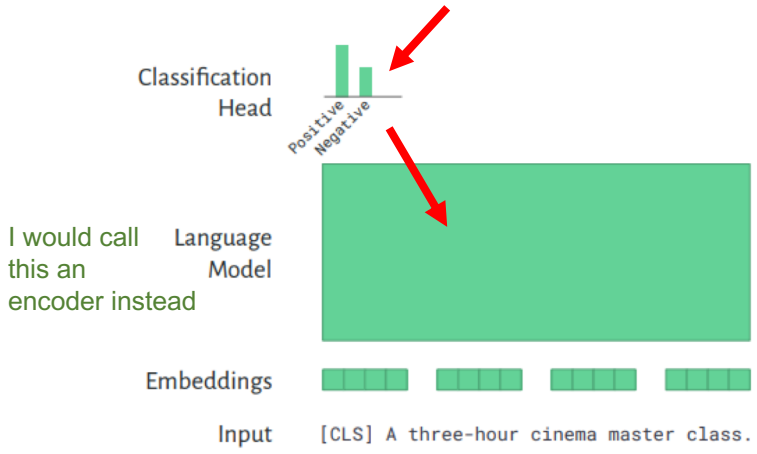


[1],[2]

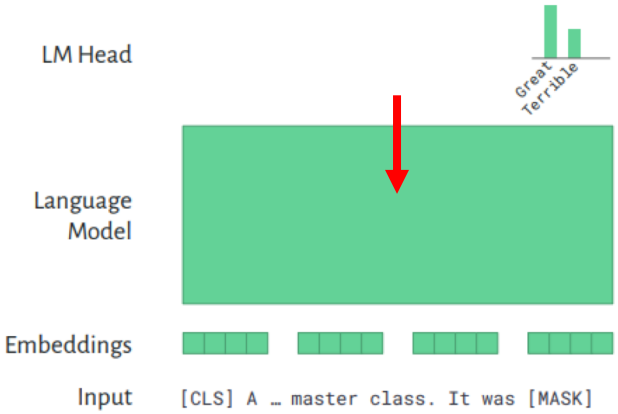
LM Prompting | Prompt-based Fine Tuning

- So far, there has been no gradient update to the model
- Prompt-based fine-tuning = LM prompt + gradient updates

Traditional fine-tuning



Prompt-based fine-tuning



- Multiple-word verbalizers in this case are tricky (see [8])

[1] [3]

Parameter-efficient Fine-tuning (PEFT)

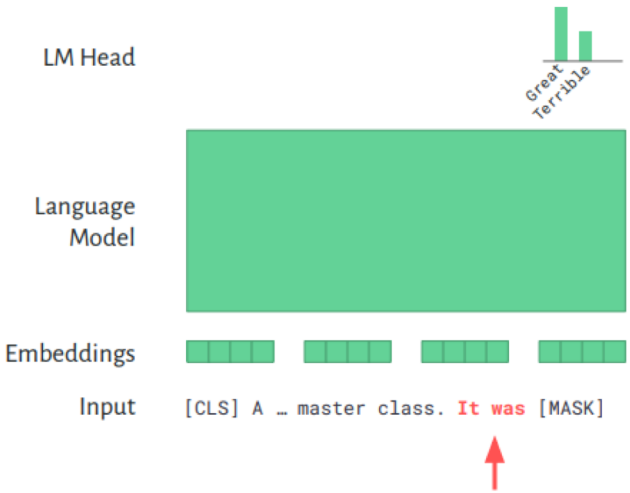
- Goal: minimize number of parameters to be updated
  - Prompt Search / Prompt Tuning
  - BitFit
  - Adapters
  - LoRA
  - $(IA)^3$

	Model Size	Task-Specific Parameters
In-Context Learning	10B - 100B	Effectively None
Prompt-Based Finetuning	100M - 1B	All
Parameter-Eff. Finetuning	100M - 1B	<1% of model parameters

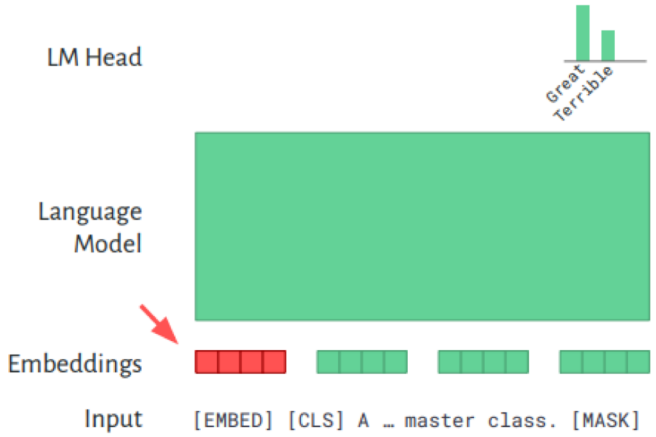
PEFT | Prompt Search vs Prompt Tuning

- Prompt search methods learn the tokens in the prompt (discrete)
- Prompt tuning method attach and learn embeddings to the input (continuous)

Prompt Search



Prompt Tuning



PEFT | Prompt Search

- AutoPrompt: Iteratively updates tokens in the pattern using a gradient-guided search

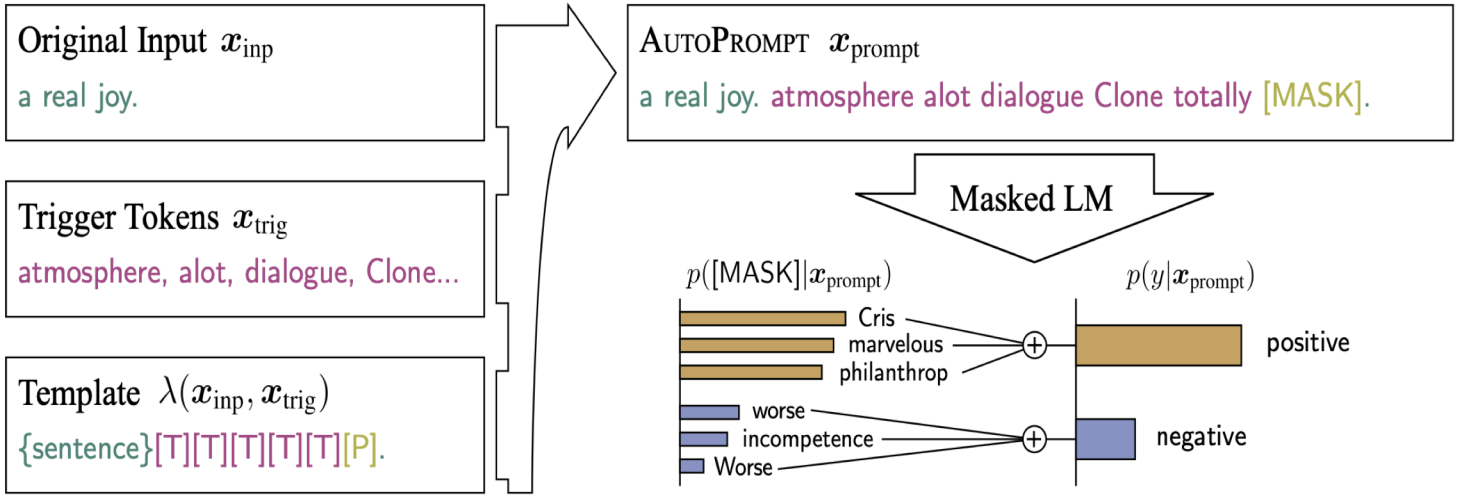


Figure 1: **Illustration of AUTOPROMPT** applied to probe a masked language model's (MLM's) ability to perform sentiment analysis. Each input, x_{inp}, is placed into a natural language prompt, x_{prompt}, which contains a single [MASK] token. The prompt is created using a template, λ, which combines the original input with a set of trigger tokens, x_{trig}. The trigger tokens are shared across all inputs and determined using a gradient-based search (Section 2.2). Probabilities for each class label, y, are then obtained by marginalizing the MLM predictions, $p([MASK]|x_{prompt})$, over sets of automatically detected label tokens (Section 2.3).

[1][9]

PEFT | Prompt Search

- AutoPrompt: Iteratively updates tokens in the pattern using a gradient-guided search

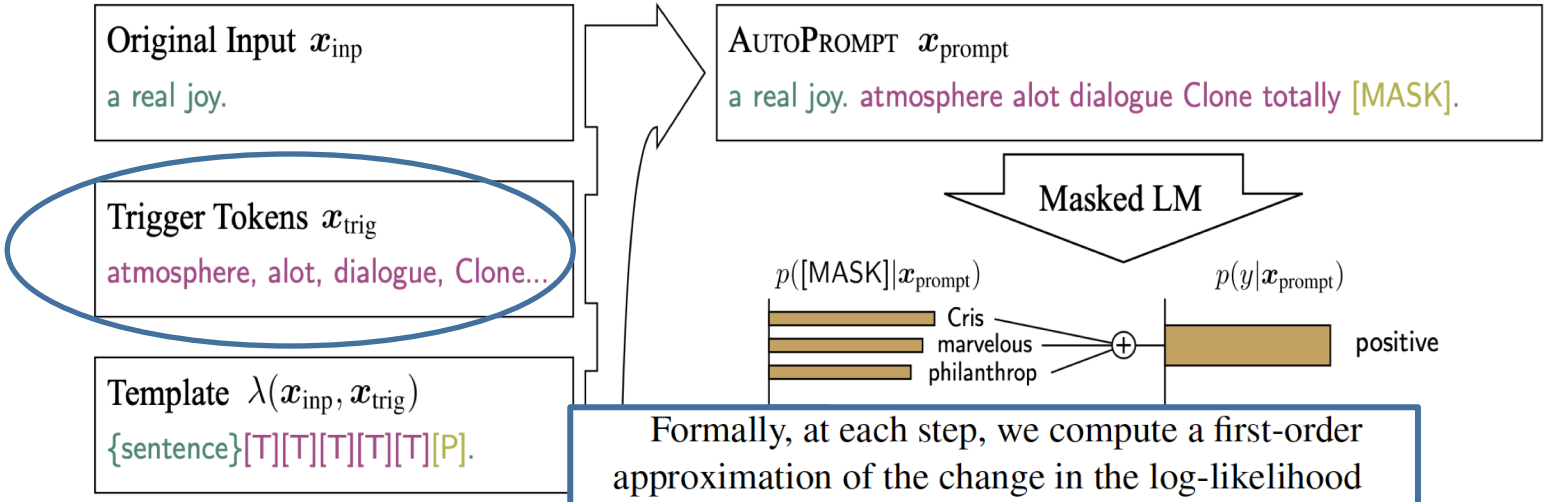


Figure 1: **Illustration of AUTOPROMPT** ... s) ability to perform sentiment analysis. Each input, x_{inp}, ... which contains a single [MASK] token. The prompt is crea... input with a set of trigger tokens, x_{trig}. The trigger toker... a gradient-based search (Section 2.2). Probabilities for each ... LM predictions, $p([MASK]|x_{prompt})$, over sets of automatic...

Formally, at each step, we compute a first-order approximation of the change in the log-likelihood that would be produced by swapping the jth trigger token $x_{trig}^{(j)}$ with another token $w \in \mathcal{V}$. Then we identify a candidate set $\mathcal{V}_{cand}$ of the top-k tokens estimated to cause the greatest increase:

$$\mathcal{V}_{cand} = \underset{w \in \mathcal{V}}{\text{top-}k} \left[\boldsymbol{w}_{in}^T \nabla \log p(y|\boldsymbol{x}_{prompt}) \right] \quad (2)$$

where $\boldsymbol{w}_{in}$ is the input embedding of w, and the gradient is taken with respect to the input embedding of $x_{trig}^{(j)}$.

[1][9]

PEFT | Prompt Search

- AutoPrompt: Iteratively updates tokens in the pattern using a gradient-guided search

we develop a general two-step approach to automate the selection of the sets of label tokens $\mathcal{V}_y$. In the first step, we train a logistic classifier to predict the class label using the contextualized embedding of the [MASK] token as input:

$$\boldsymbol{h} = \text{Transformer}_{\text{enc}}(\tilde{\boldsymbol{x}}) \quad (3)$$

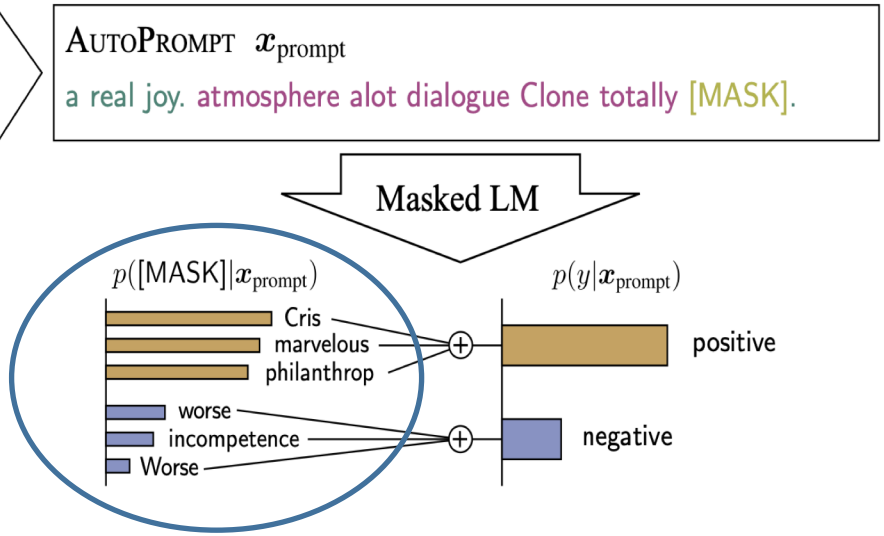
We write the output of this classifier as:

$$p(y|\boldsymbol{h}^{(i)}) \propto \exp(\boldsymbol{h}^{(i)} \cdot \boldsymbol{y} + \beta_y) \quad (4)$$

where $\boldsymbol{y}$ and β_y are the learned weight and bias terms for the label y, and i represents the index of the [MASK] token.

In the second step, we substitute $\boldsymbol{h}^{(i)}$ with the MLM's output word embeddings $\boldsymbol{w}_{\text{out}}$ to obtain a score $s(y, w) = p(y|\boldsymbol{w}_{\text{out}})$. Intuitively, because $\boldsymbol{w}_{out} \cdot \boldsymbol{h}$ and $\boldsymbol{y} \cdot \boldsymbol{h}$ are large for words and labels that are relevant to a particular context, $s_w \propto \exp(\boldsymbol{w}_{\text{out}} \cdot \boldsymbol{y} + \beta_y)$ should be large for words that are typically associated with a given label. The sets of label tokens are then constructed from the k-highest scoring words:

$$\mathcal{V}_y = \underset{w \in \mathcal{V}}{\text{top-}k} \left[s(y, w) \right] \quad (5)$$

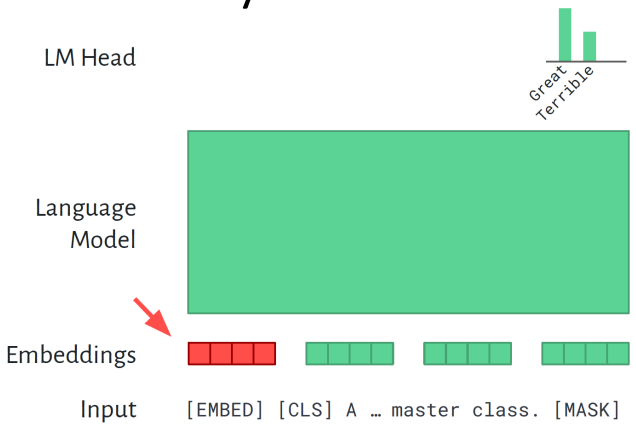


pplied to probe a masked language model's (MLM's) ability to per-
s placed into a natural language prompt, $\boldsymbol{x}_{\text{prompt}}$, which contains a
d using a template, λ, which combines the original input with a set
are shared across all inputs and determined using a gradient-based
ass label, y, are then obtained by marginalizing the MLM predictions,
y detected label tokens (Section 2.3).

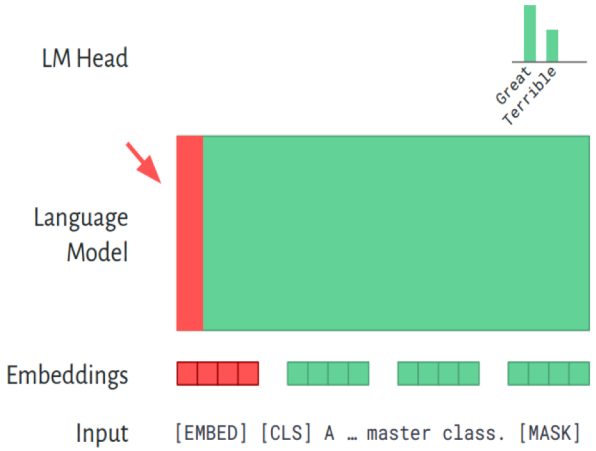
[1][9]

PEFT | Prompt Tuning

- Learn embeddings for placeholder tokens in the pattern.
  Initialize using a vocabulary embedding rather than totally random
  - WARP (Hambardzumyan et al., 2021)
  - OptiPrompt (Zhong et al., 2021)
  - Prompt Tuning (Lester et al., 2021)
  - P-Tuning (Liu et al., 2021)



- Additionally, you can also fine-tune a small LM component to learn also contextualized embedding placeholders.
  - Prefix tuning (Li and Liang, 2021)
  - Soft Prompts
    (Qin and Eisner, 2021)



See [1]

PEFT | BitFit

- BitFit tunes only the bias terms in self-attention and MLP layers.
- Simple yet effective: prompt-based fine-tuning with BitFit performs on-par with or better than full prompt-based fine-tuning on few-shot tasks.

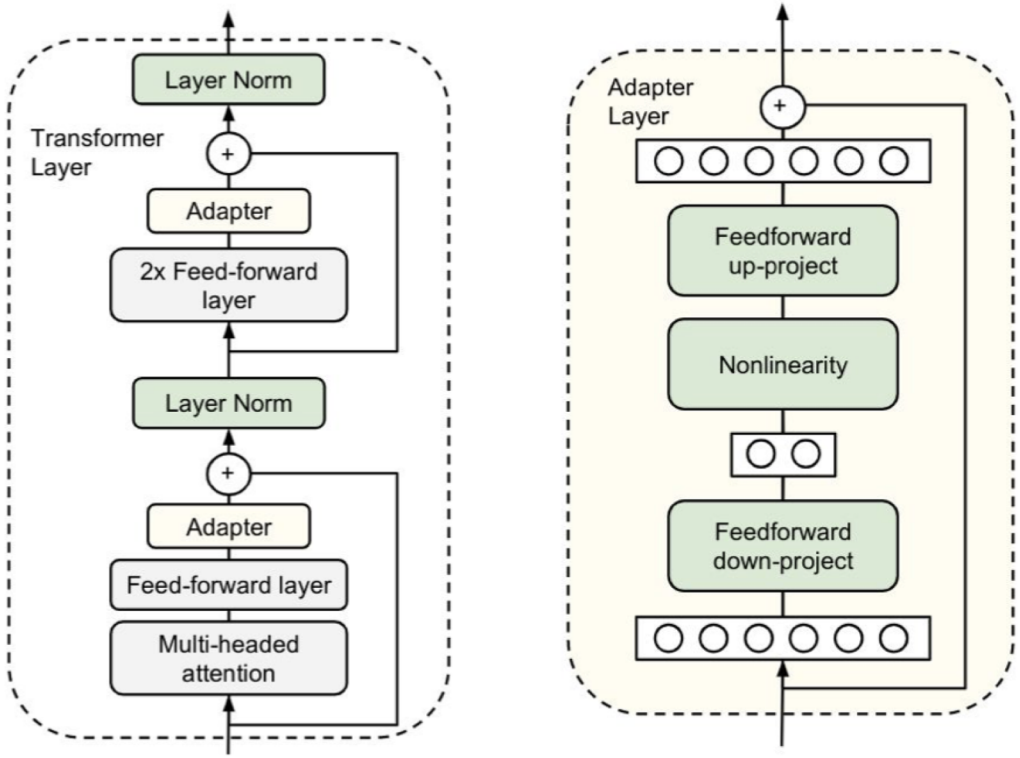
$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right) \quad (1)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell \quad (2)$$
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right) \quad (3)$$
$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right) \quad (4)$$
$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell \quad (5)$$

See [1]

PEFT | Adapters

- Add feedforward layers + skip connection blocks after each feedforward layer.
- Feedforward layers act as down- and up-projection to reduce parameters.
- Fine-tune only the adapter blocks layers on new tasks.



- If you are interested, more advanced adapters: compacters

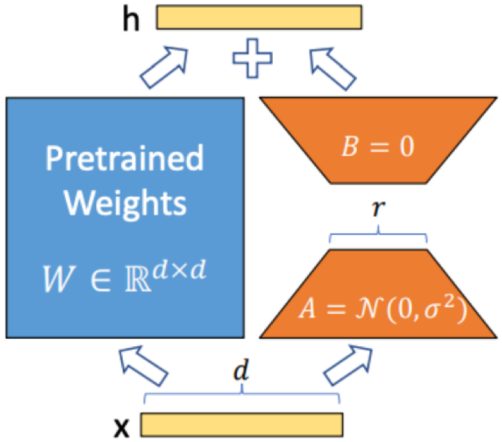
See [1]

- Low-Rank Adaptation of LLMs. At each layer, it substitutes the full update on W with an update on the low-rank decomposition

Finetuned Weights — Weight Update — Rank Decomposition Matrix

$$W_{\text{ft}} = W_{\text{pt}} + \Delta W = W_{\text{pt}} + AB$$

Pretrained Weights — Approximation



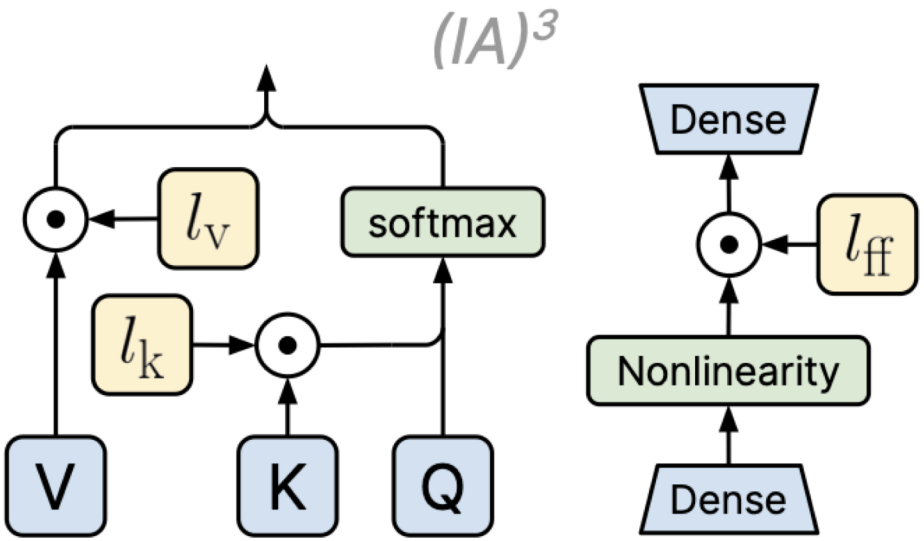
- Both A and B have low rank (*r* -> much less params. than W). The higher *r*, the more accurate the approximation from LoRA.

- B is initialized all at 0 -> $W_{\text{ft}} = W_{\text{pt}}$ at the beginning

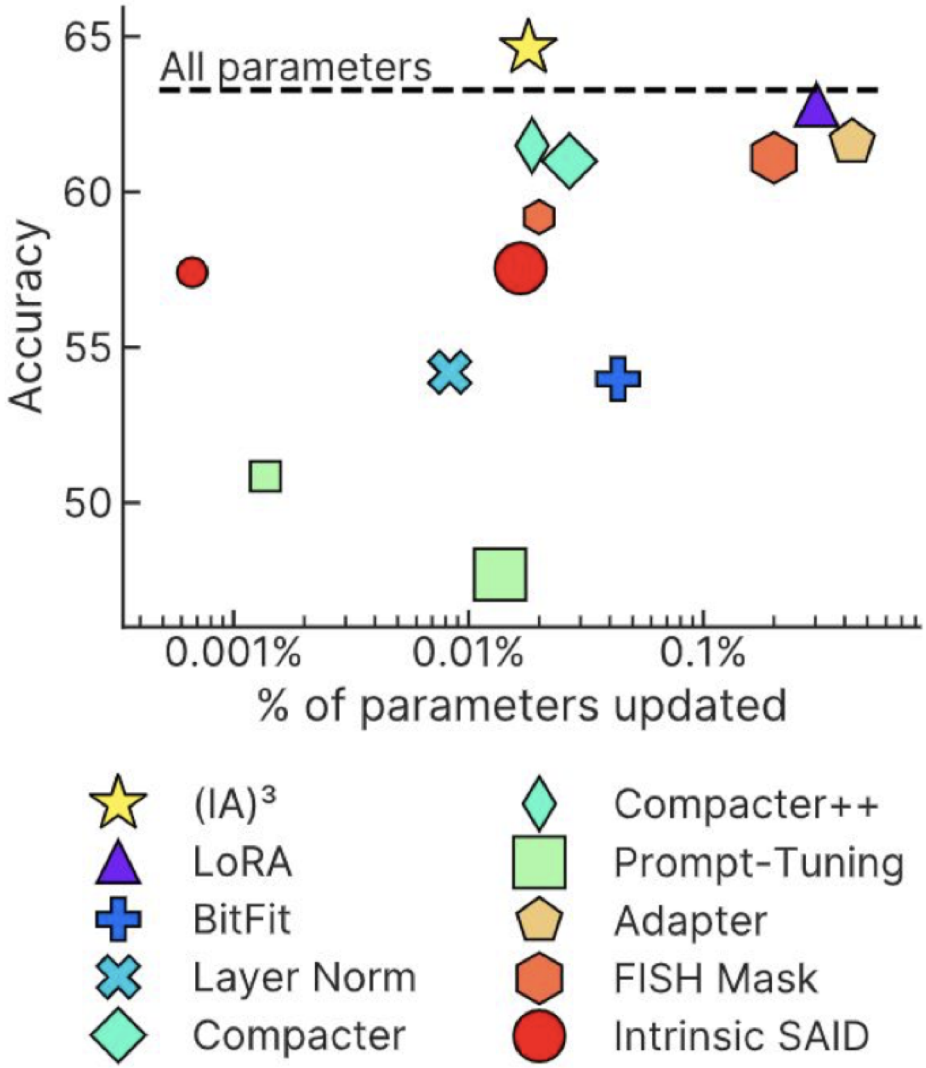
See [1]

PEFT | $(IA)^3$

- Infused Adapter by Inhibiting and Amplifying Inner Activations
- Element-wise rescaling of model activations with a learned vector:
  - keys and values in self-attention
  - Keys and values in encoder-decoder attention Intermediate activation of the position-wise feed-forward networks
- You can associate each task with its own learned task vector.



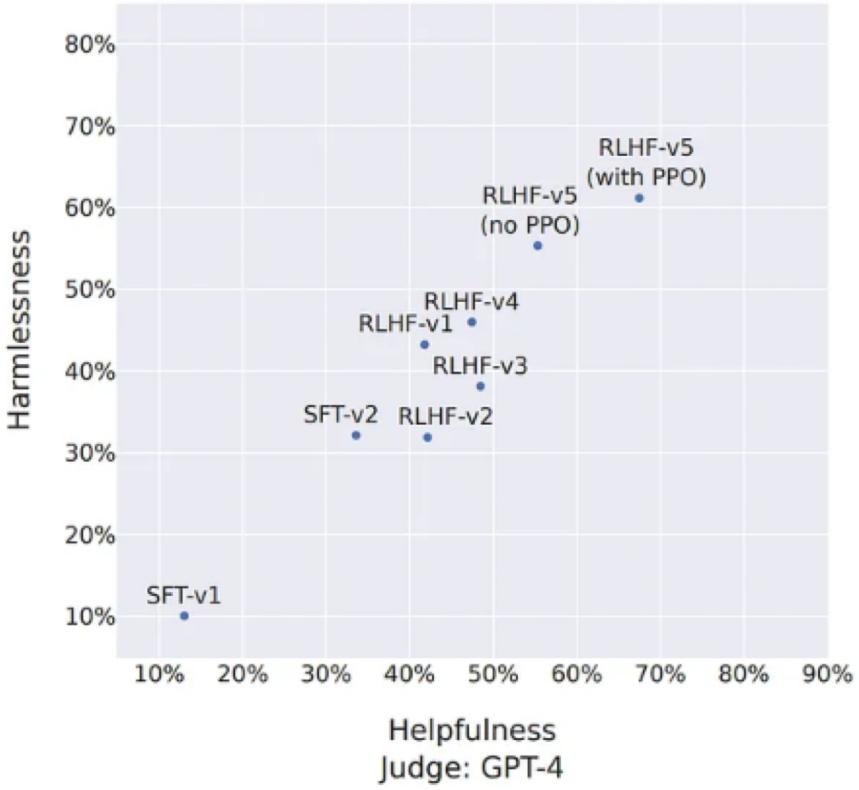
PEFT | Comparison



Human Preferences Tuning

Human Preference Tuning | Motivation

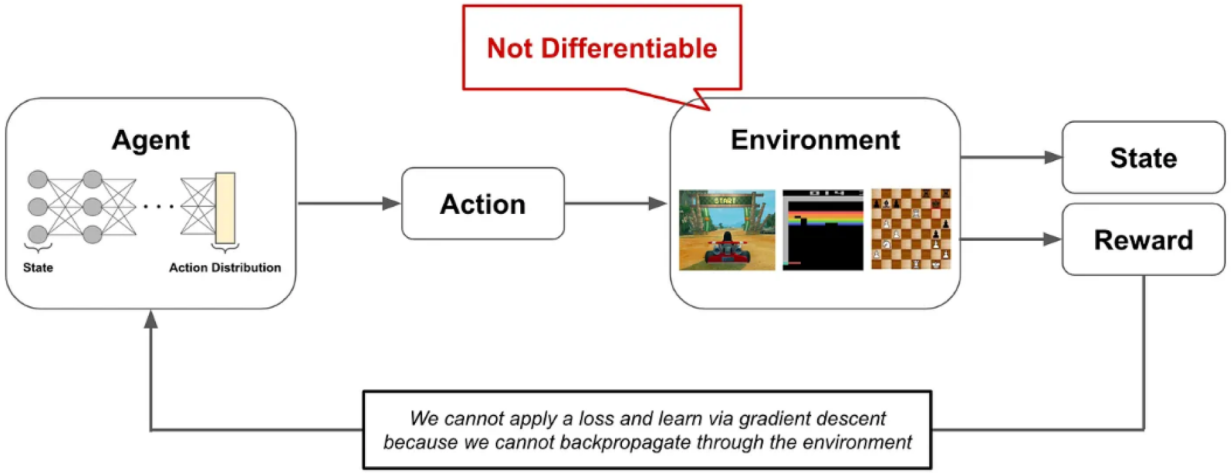
- Especially in user-facing applications, SFT is not enough to achieve a model that performs well and at the same time is non-harmful, friendly, etc.
- Tuning with human preferences can contribute massively to this aspect



- LLaMa-2-Chat: user expectation preference (win rate %) over ChatGPT. The judge is GPT-4.

Human Preference Tuning | Reliance on RL

- As of now, many techniques strongly rely on Reinforcement Learning (RL). Many practitioners avoid learning about it because “RL is complicated”.
- Training on human preferences (rankings, scores, etc.) is not differentiable, hence we cannot simply apply supervised learning.
- If you are comfortable with RL: good to go! Otherwise we would suggest to review some RL basics (added to the references)

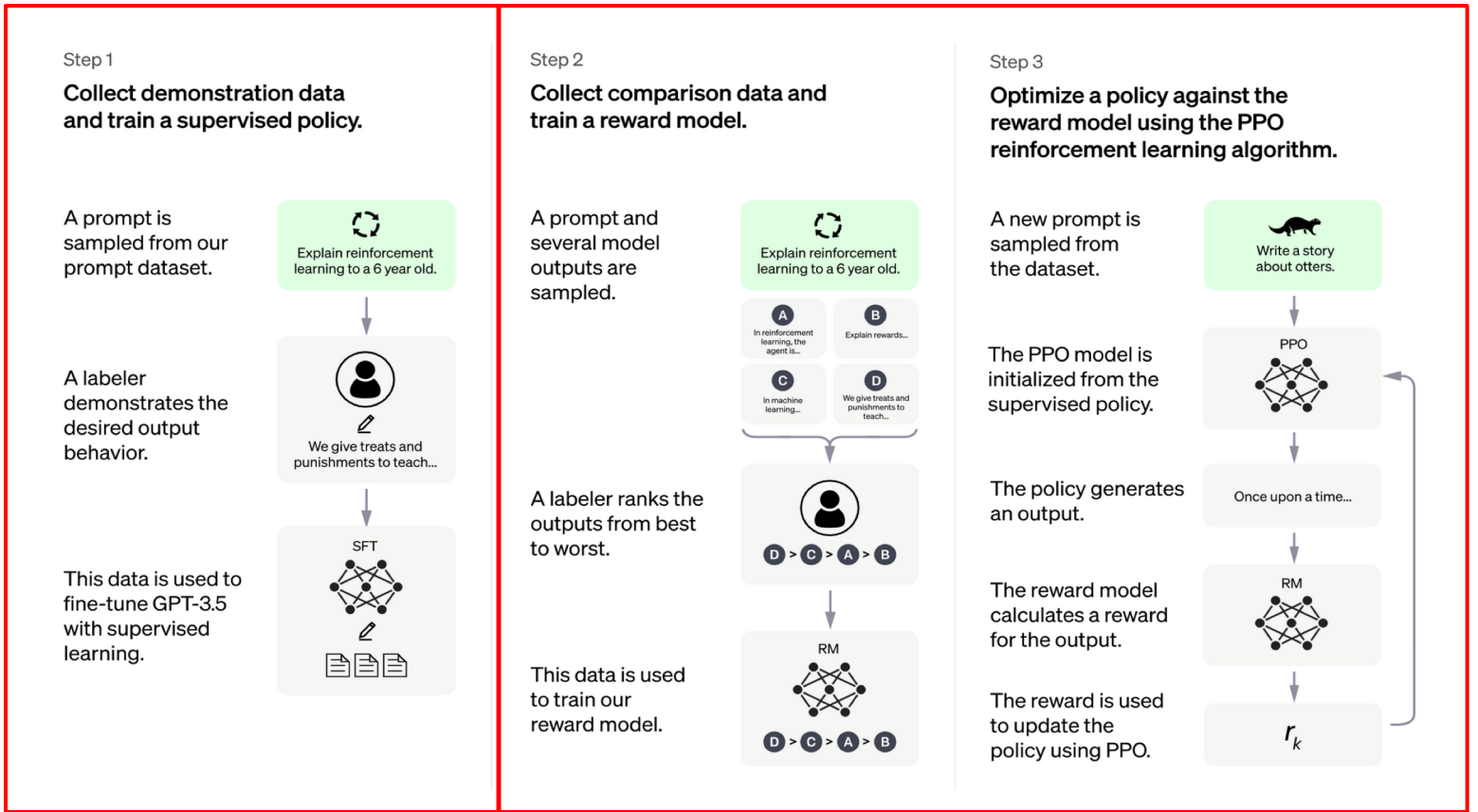


[5b]

Human Preference Tuning | RLHF

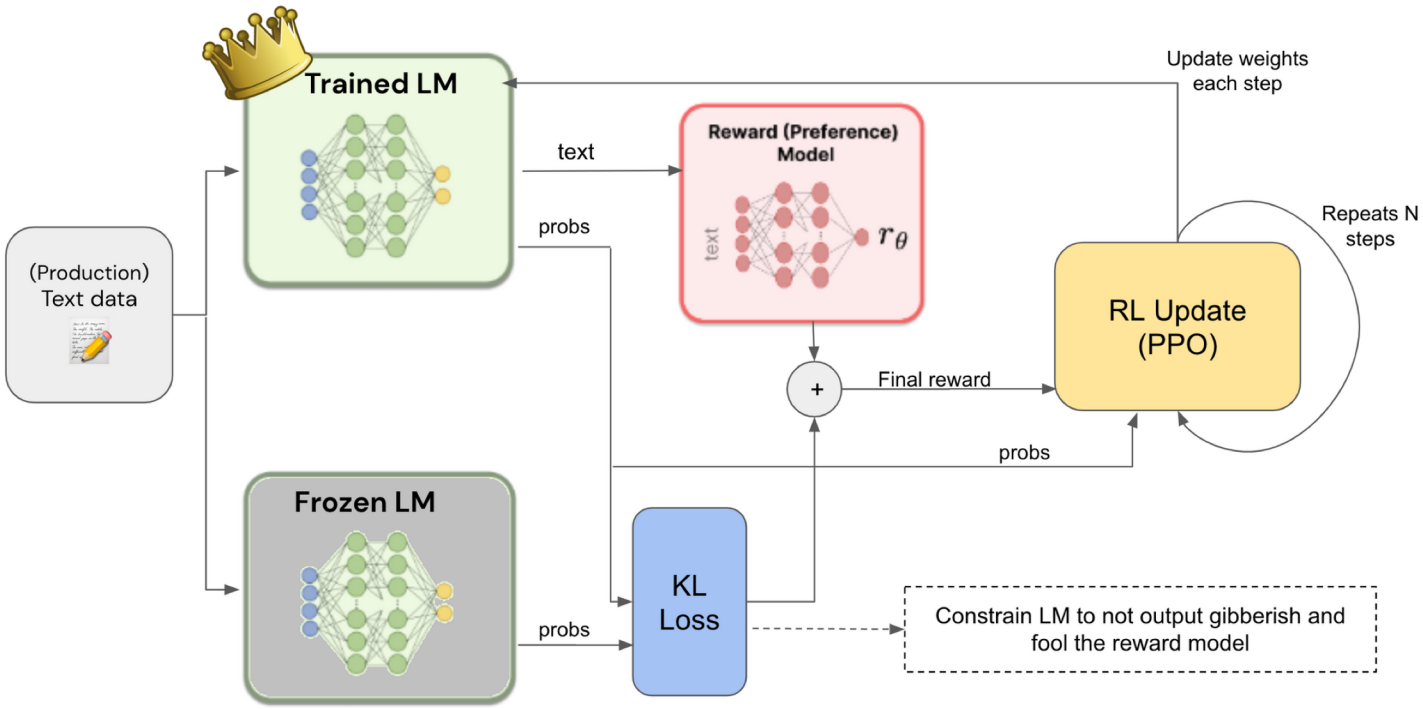
- Reinforcement Learning with Human Feedback (RLHF) is used to transform GPT into ChatGPT.
- Policy = our LLM, reward model = (smaller) LLM with extra layers for regression, label = human score/preference.

SFT RLHF



Human Preference Tuning | PPO

- The SFT LLM is our policy, but vanilla policy optimization would generate large gradients, disrupting the SFT knowledge to please the reward model
- Proximal Policy Optimization (PPO) adds a KL loss term to cautiously clip gradients and stay close the original policy.
- More stable, efficient, and easier to implement



Human Preference Tuning | RLHF Observations

- After RLHF alignment smaller models generally get slightly worse on benchmarks while larger models get better.

- Alignment with RLHF is compatible with specialized models. Aligning a model fine-tuned on coding will make it better at it. It is also effective when applied iteratively: (1) Deploy model (2) collect data (3) RLHF

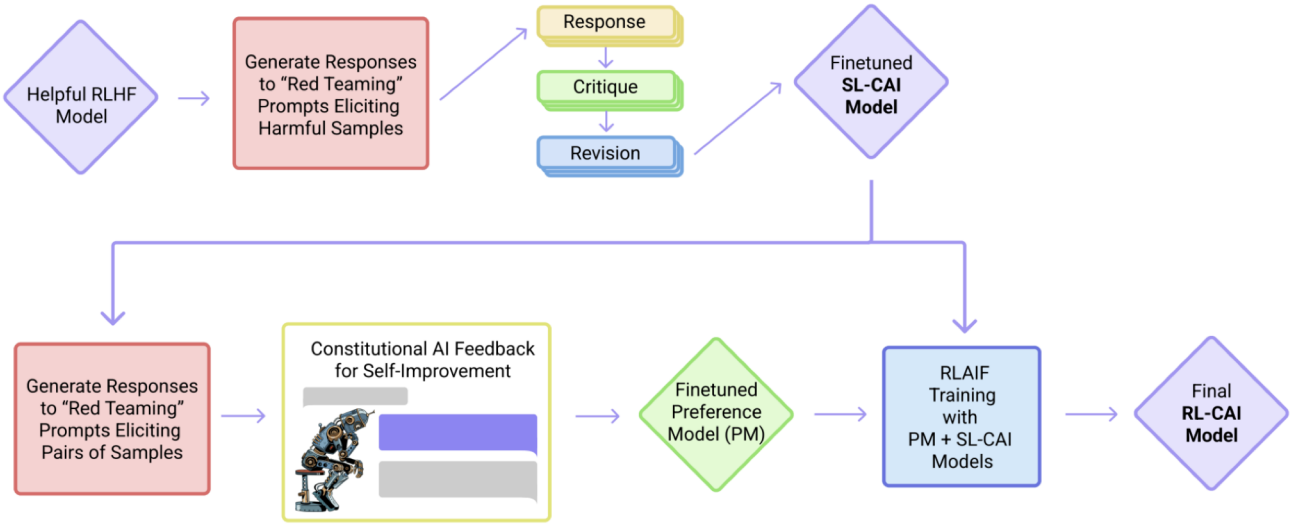
- RLHF requires a lot of humans annotations, much more than SFT.
  Can't we scale feedback by using models themselves? Results now show LLMs to be as good as crowdworkers in identifying harmful behavior.

- Indeed, models can explicitly self-reflect based on Constitutional AI (CAI) principles. This makes the feedback process to reduce harmfulness more scalable and more explainable.

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

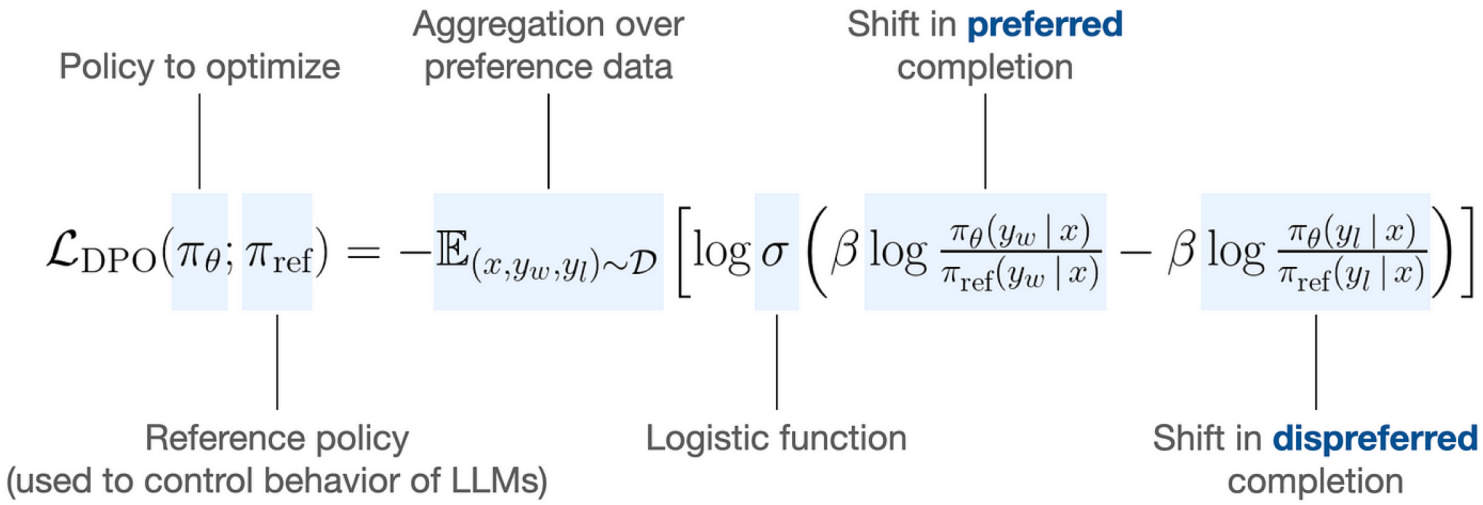
Human Preference Tuning | RLAIF

- Reinforcement Learning with AI Feedback (RLAIF) uses CAI instead of humans to improve a helpful RLHF model to be also harmless.
- Step 1: Sample harmful responses, make the model revise them according to a random CAI principle, and use SFT with revised answers (SL-CAI).
- Step 2: Take harmful prompts, feed them to SL-CAI to generate two responses. Train a generic LLM on which is more aligned with the constitution while mixing with RLFH data for usefulness.
- Step 3: RL-Tuning of SL-CAI (RL-CAI). Empirically less harmful and evasive.



Human Preference Tuning | DPO

- Do we really need RLHF? Human scores are not an LLM output -> not differentiable -> we need RL and a reward model
- Direct Policy Optimization (DPO) reformulates the RL setting in a single cross-entropy loss. Train the LLM directly with no reward model!
- So far performs better than PPO, but the debate is open.



- How to read: maximize prob. of winning outputs (w) and the opposite for losing outputs (l) while keeping the policy close to the original (ref).

References

[1] ACL Tutorial 2022

[2] Brown et al. 2020. GPT-3

[3] Schick et al. 2020. PET

[4] Liu et al. 2022. $(IA)^3$

[5] Wolfe's Blog (PEFT, RL Basics, RLHF, PPO, RLAIF, etc.) , especially [5a] especially 2 [5b], especially 3 [5c] especially [5d]

[6] Rafailov et al. DPO

[7] Zhao et al 2021

[8] Schick et al 2020 (2).

[9] Shin et al 2020

[10] Gao et al. 2020

Study Approach

Minimal

- Work with the slides

Standard

- Minimal approach + read reference 5 („especially“)

In-Depth

- Standard approach + read reference 1 and 6 + read page 6/7 ref 2

See you next time!