Transformer, GPT and BERT

Yuan YAO

HKUST

Summary

- We have shown:
  - CNN Architectures: LeNet5, Alexnet, VGG, GoogleNet, Resnet
  - Recurrent Neural Networks and LSTM (GRU)
  - Attention and Transformer
- Today:
  - **Applications of Transformer**
  - **BERT and GPT**
- Reference:
  - Feifei Li, Stanford cs231n
  - Chris Manning, Stanford cs224n

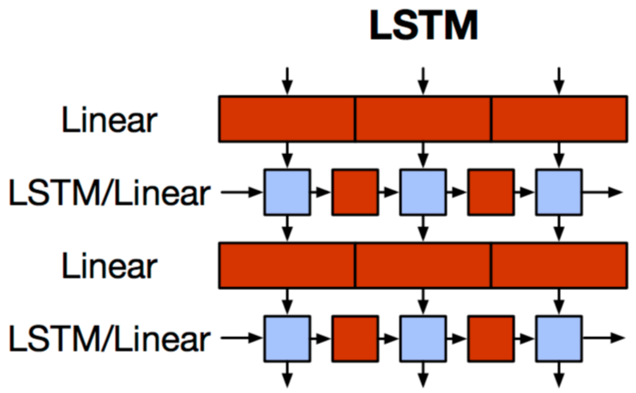
A Brief History in NLP

- In 2013-2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include: handwriting recognition, speech
  - recognition, machine translation, parsing, image captioning
  - LSTM became the dominant approach
- Now (2019), other approaches (e.g. Transformers) have become more dominant for Machine Translation.
  - For example in **WMT** (a MT conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2018, the report contains "RNN" 9 times and "Transformer" 63 times

  - **Source:** "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
  - **Source:** "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf

Motivation of Transformer

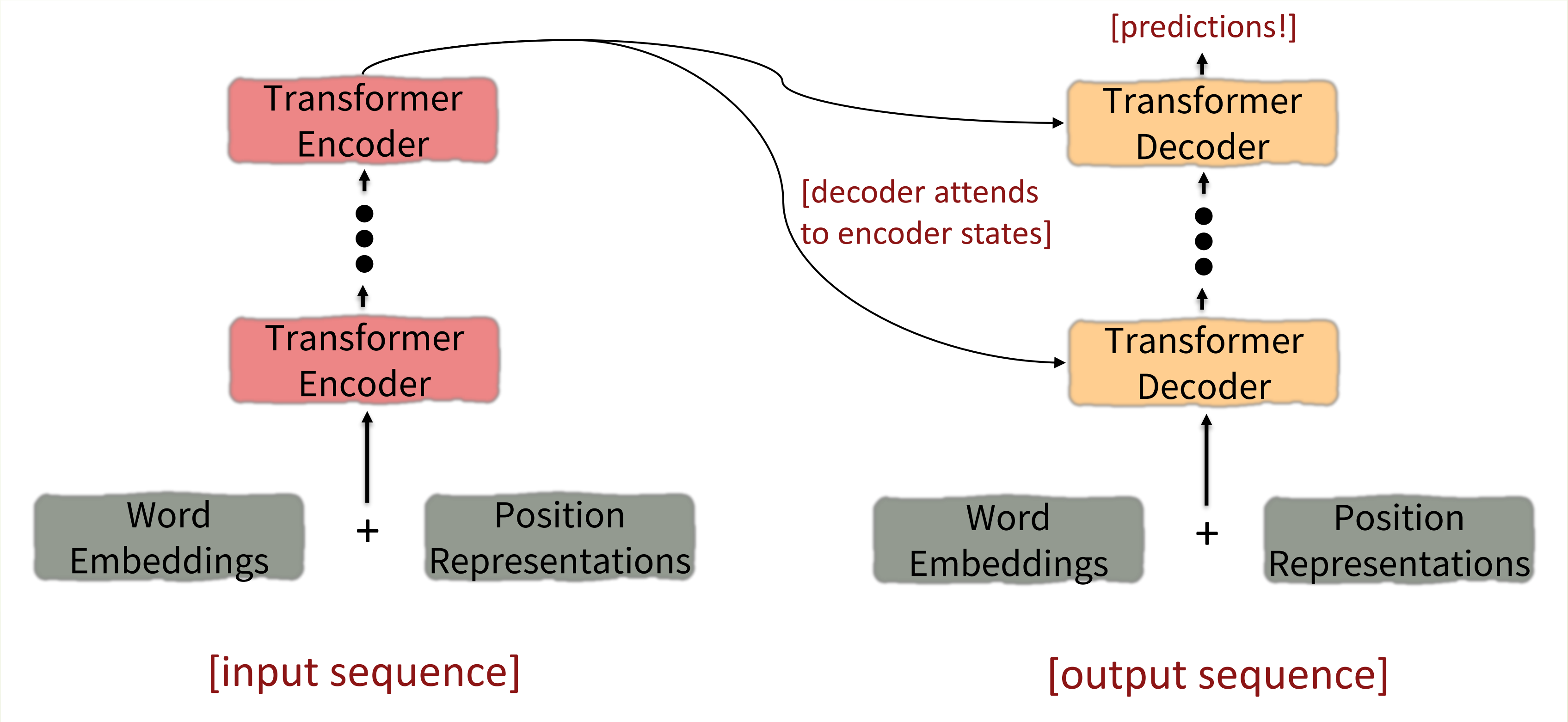
- We want **parallelization** but RNNs are inherently sequential



- Despite LSTMs, RNNs generally need attention mechanism to deal with long range dependencies – **path length** between states grows with distance otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN? 🤔
- And then NLP can have deep models … and solve our vision envy

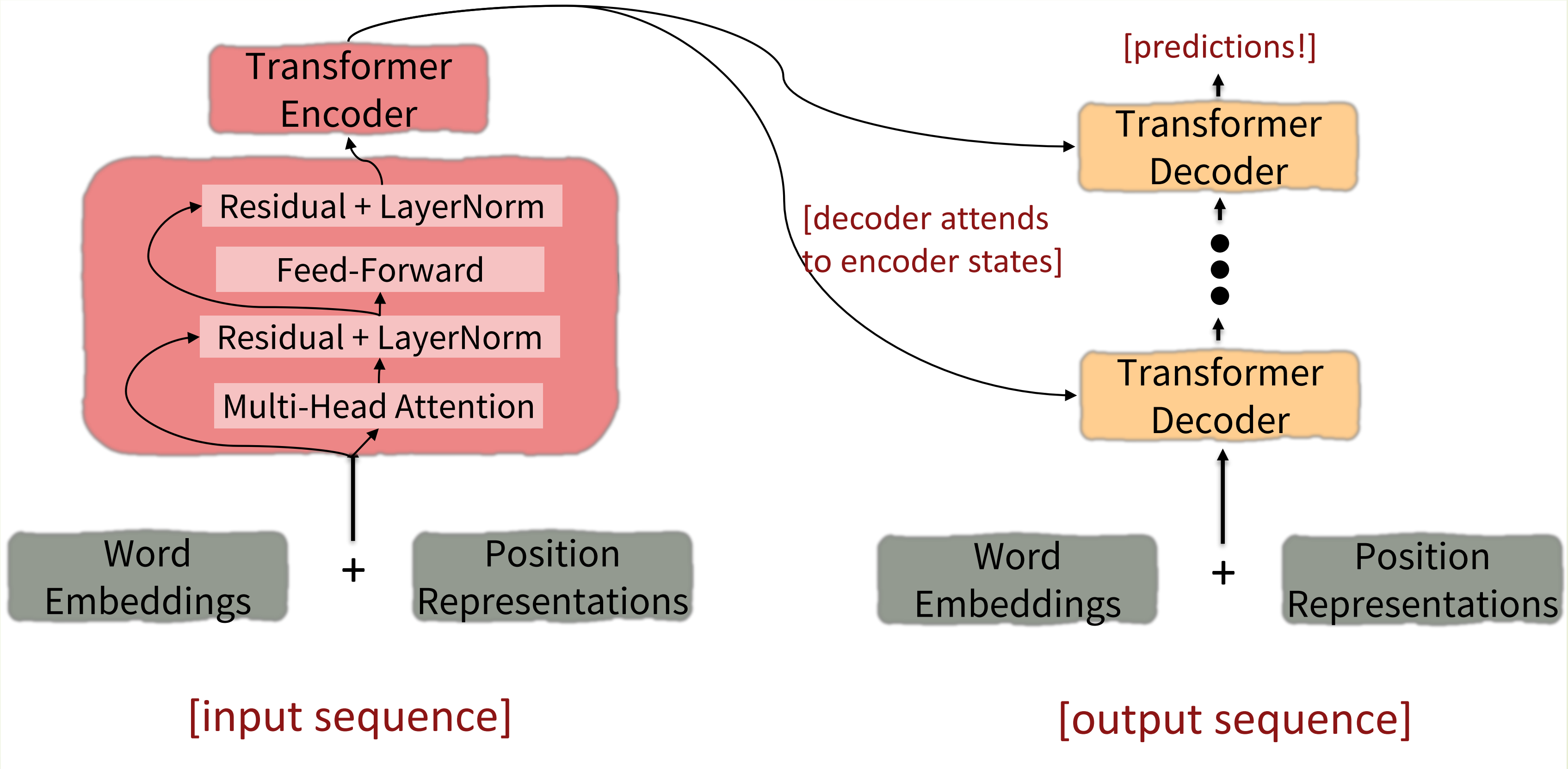
The Transformer Encoder-Decoder [Vaswani et al. 2017]

- Looking back at the whole model



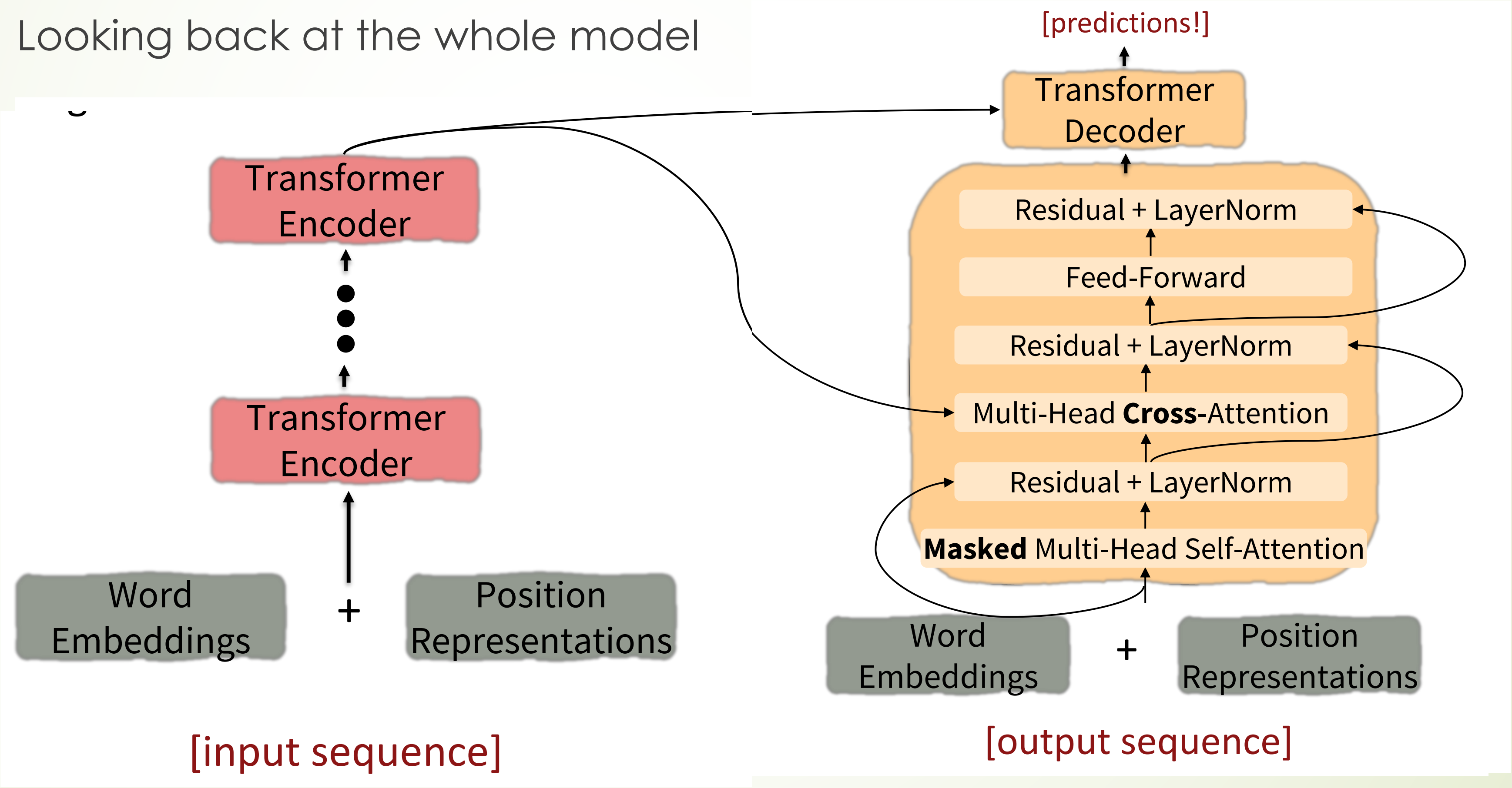
The Transformer Encoder-Decoder [Vaswani et al. 2017]

- Looking back at the whole model



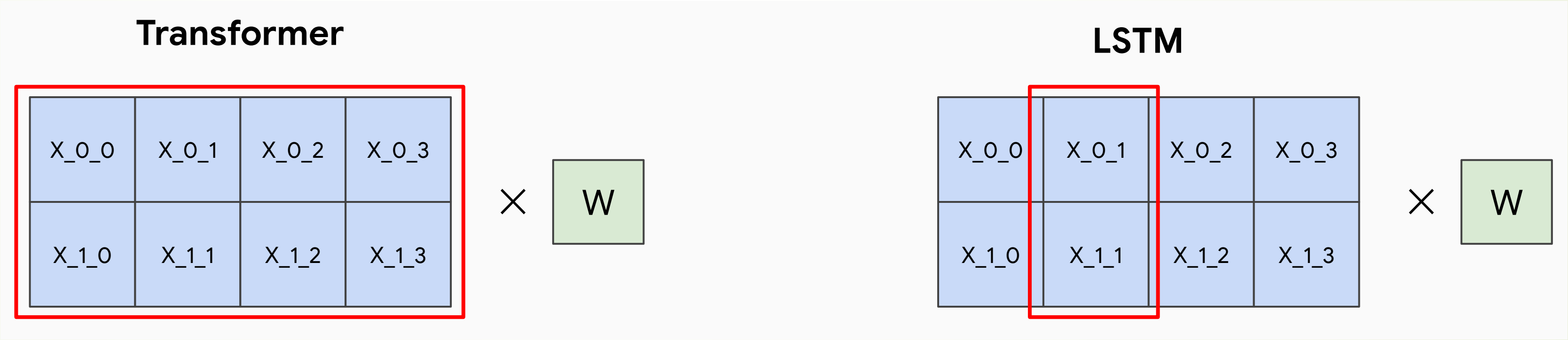
The Transformer Encoder-Decoder [Vaswani et al. 2017]

- Looking back at the whole model



Empirical advantages of Transformer vs. LSTM

- 1. Self-attention == no locality bias
  - Long-distance context has "equal opportunity"
- 2. Single multiplication per layer == efficiency on TPU

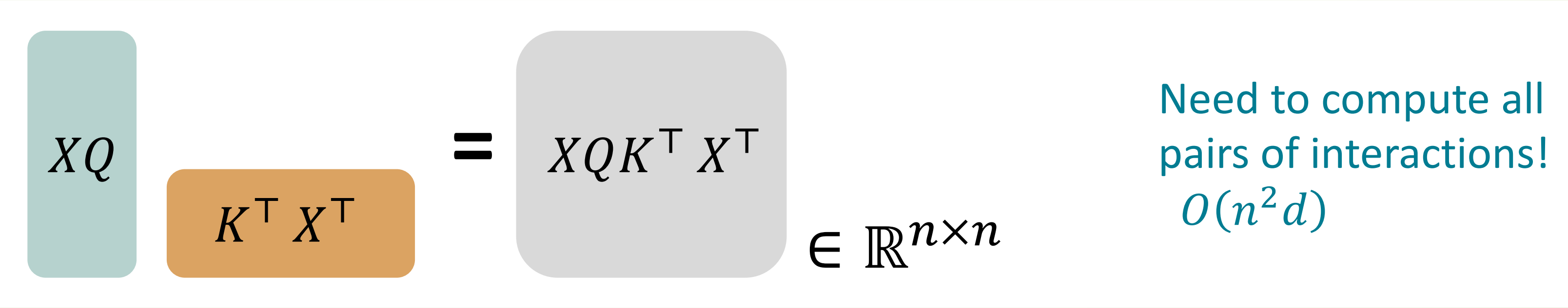


What would we like to fix about the Transformer?

- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!

Quadratic computation as a function of sequence length

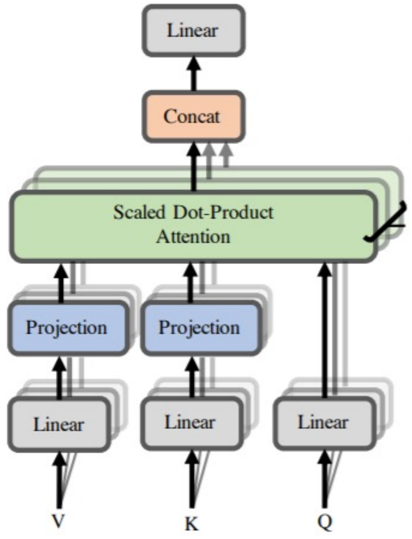
- One of the benefits of self-attention over recurrence was that it's highly parallelizable.
- However, its total number of operations grows as $O(n^2 d)$, where n is the sequence length, and d is the dimensionality.
- Think of d as around $\mathbf{1{,}000}$ (though for large language models it's much larger!).
  - So, for a single (shortish) sentence, $n \leq 30$; $n^2 \leq \mathbf{900}$.
  - In practice, we set a bound like $n = 512$.
  - **But what if we'd like $\boldsymbol{n} \geq \mathbf{50{,}000}$?** For example, to work on long documents?

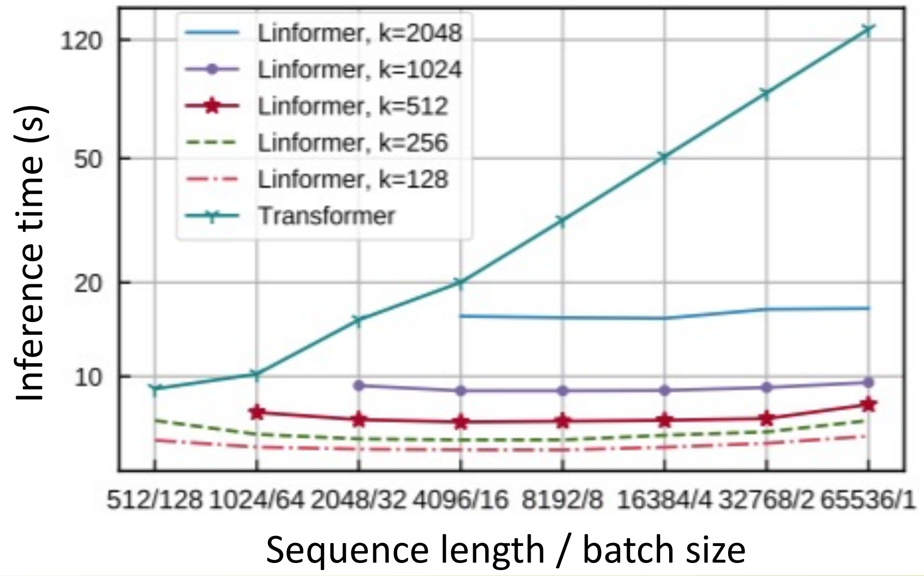


Improving quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the all-pairs self-attention cost?*
- For example, **Linformer** [Wang et al., 2020, **Linformer: Self-Attention with Linear Complexity**, arXiv:2006.04768]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

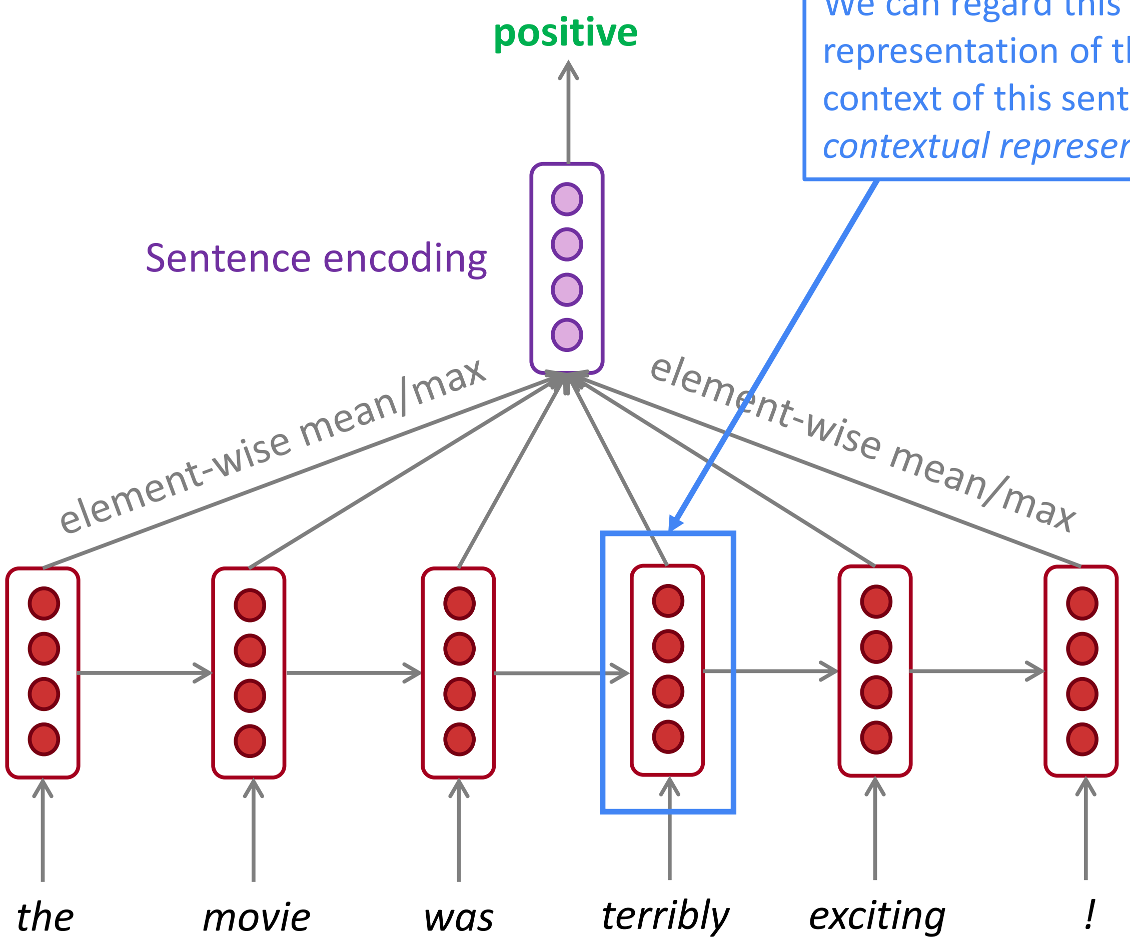




Bi-Direction

Motivation of Bidirection

Task: Sentiment Classification



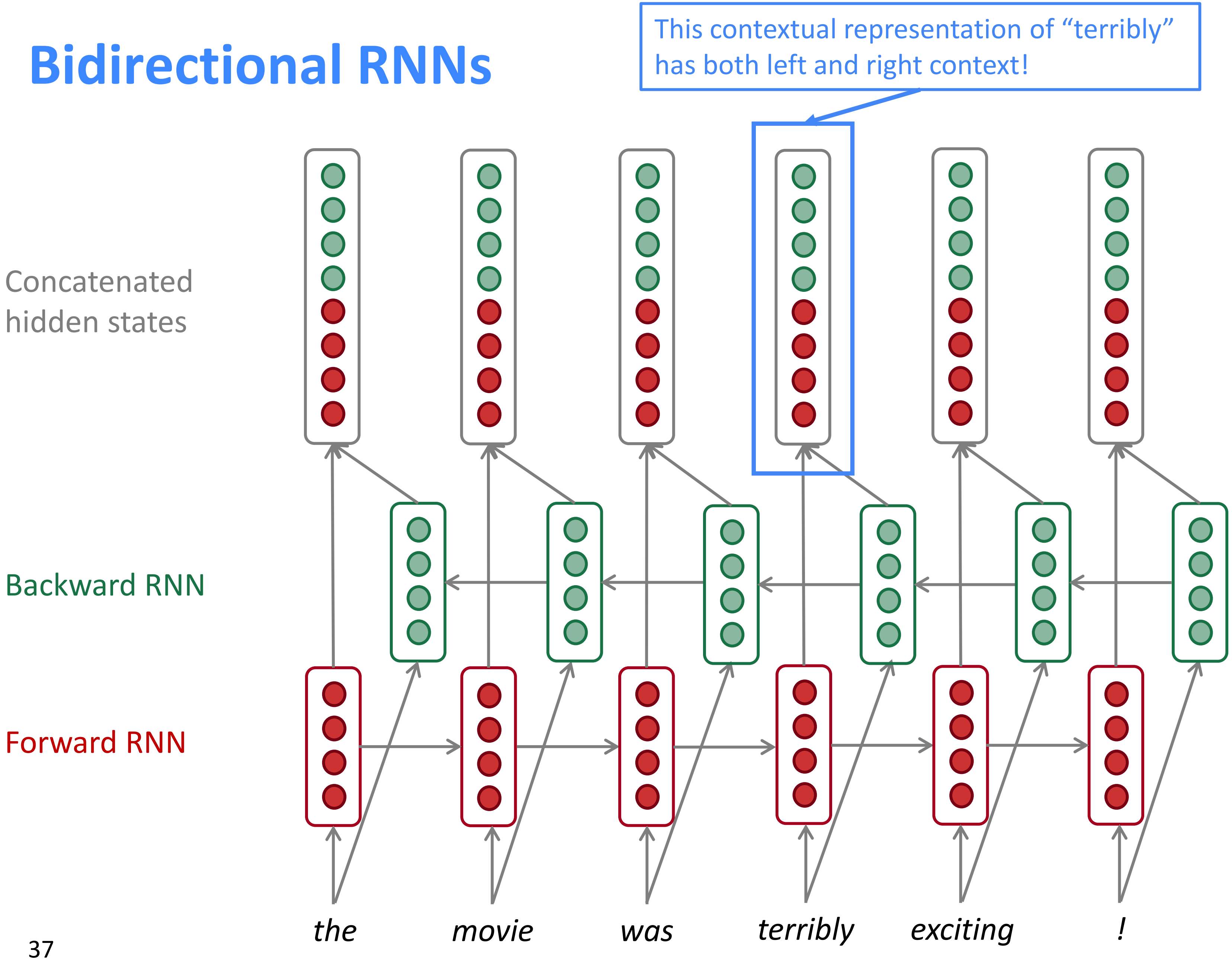
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Bidirectional RNNs



Bidirectional RNN: simplified diagram

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

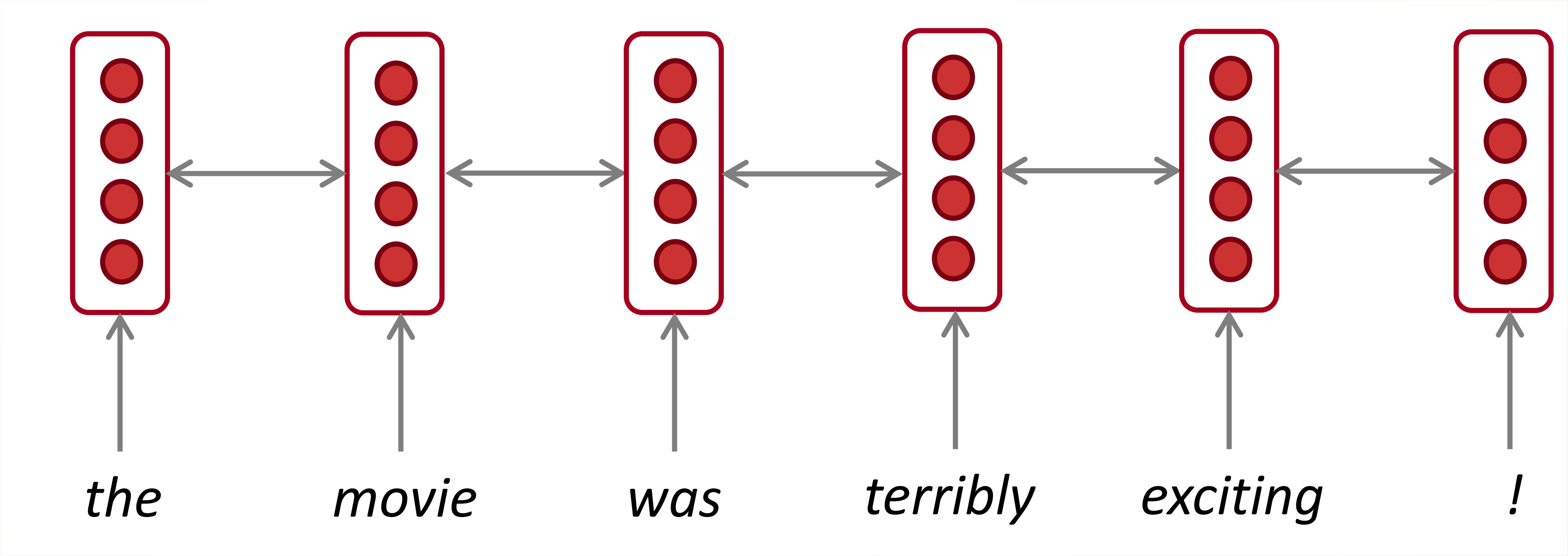
Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

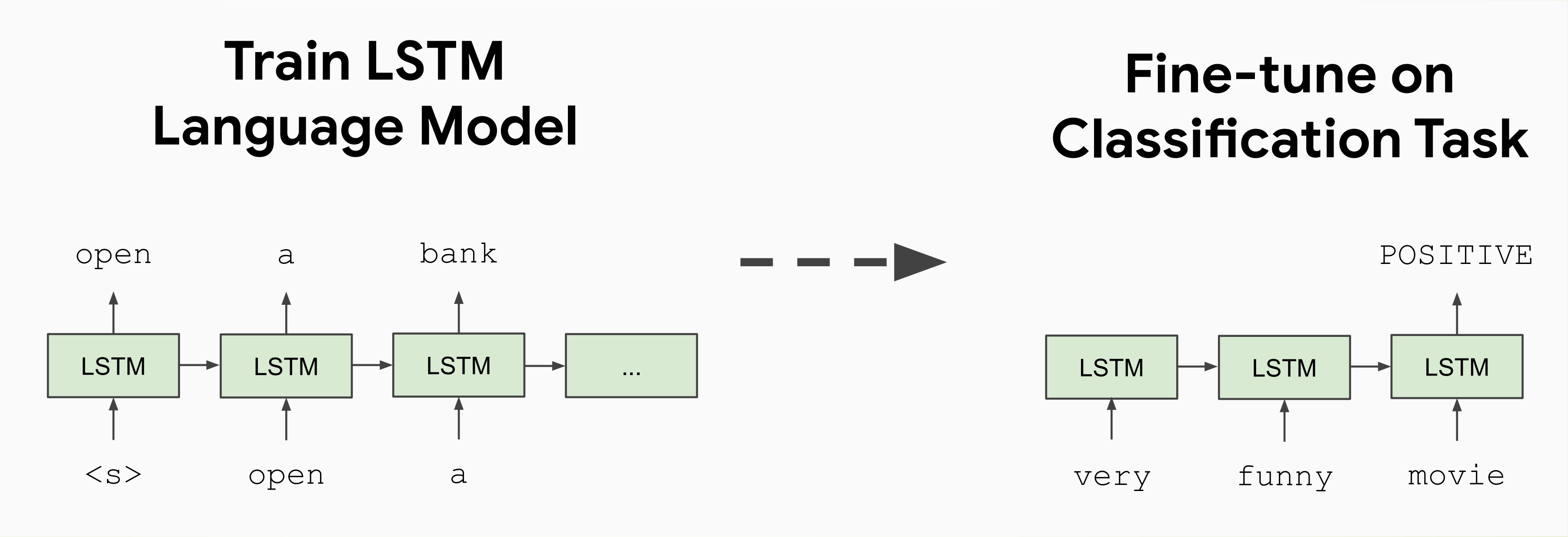
Bidirectional RNN: simplified diagram

- The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states.



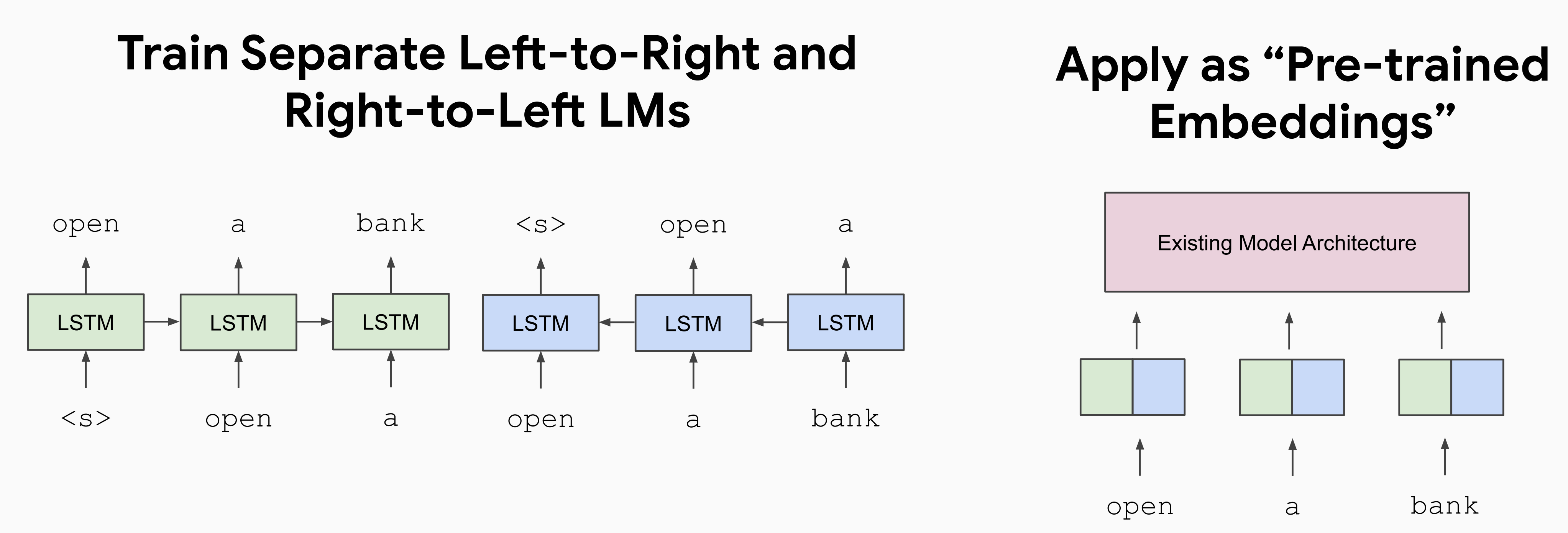
Uni-Direction LSTM

- Semi-Supervised Sequence Learning, Google, 2015



Bi-Direction LSTM: ELMo -- Embeddings from Language Models

- Peters et al. (2018) Deep Contextual Word Embeddings, NAACL 2018. https://arxiv.org/abs/1802.05365
- Learn a deep Bi-NLM and use all its layers in prediction

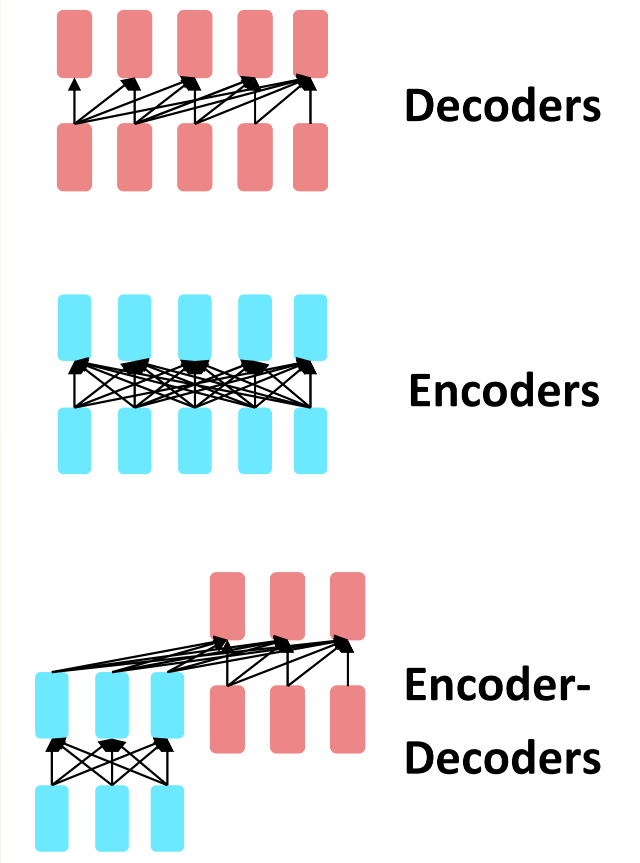


Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**.
  - For example, **Encoder** of Transformers
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available, e.g. **Decoder** of Transformers
- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).
- For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.

Pretraining for three types of architectures in Transformers

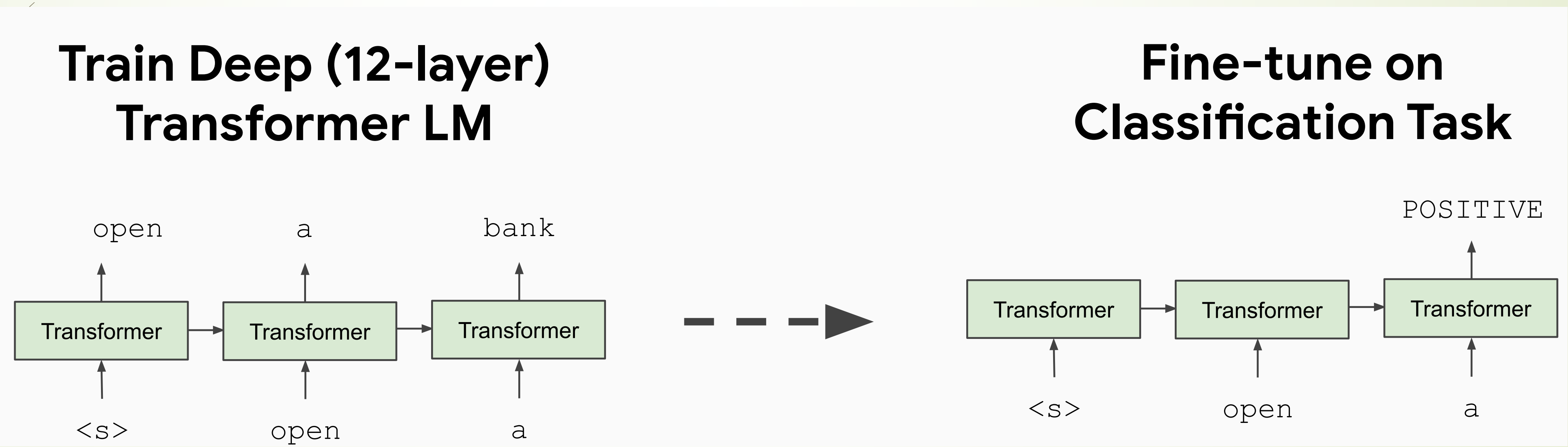
The trransformer architecture influences the type of pretraining:



- Decoders:
  - **Unidirectional** Language models! What we've seen so far.
  - Nice to generate from; can't condition on future words
- Encoders:
  - Gets **bidirectional** context – can condition on future!
  - Wait, how do we pretrain them?
- Encoder-Decoders:
  - Good parts of decoders and encoders?
  - What's the best way to pretrain them?

GPT (Generative Pre-Training): uni-directional transformer

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018



Pretraining decoders

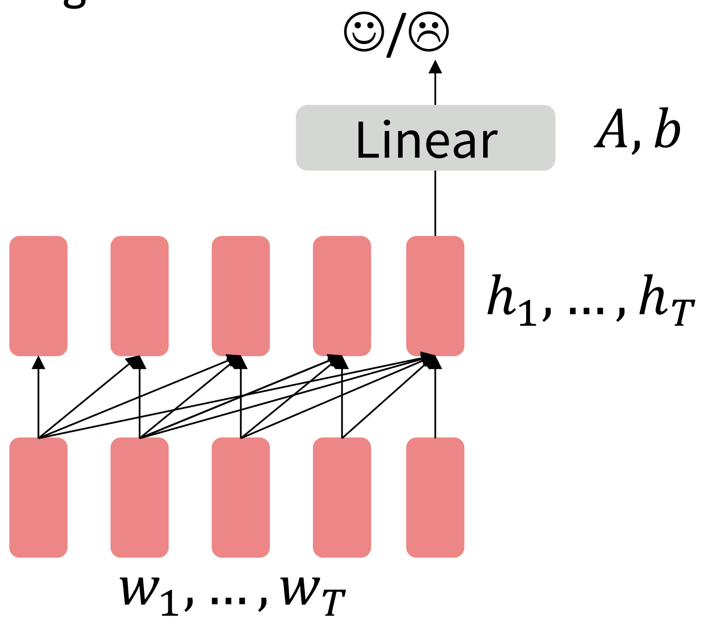
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$y \sim Ah_T + b$$

Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

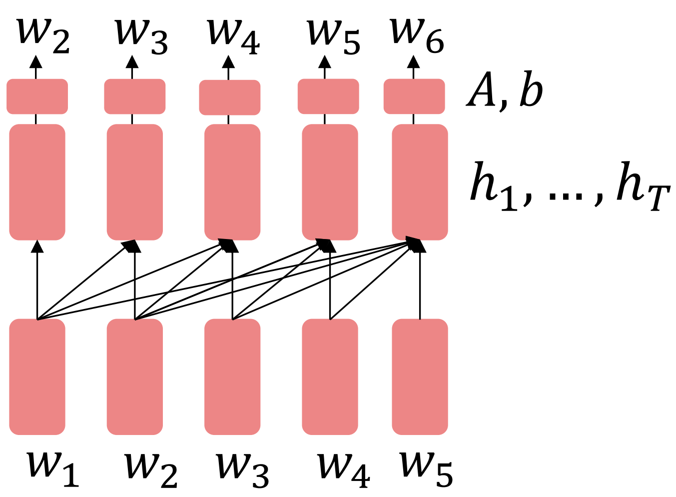
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t | w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim A h_{t-1} + b$$

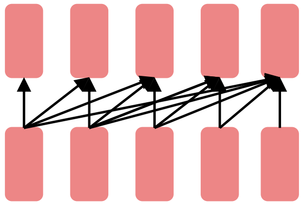
Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

GPT (Generative Pre-Trained Transformer): uni-directional transformer-decoder

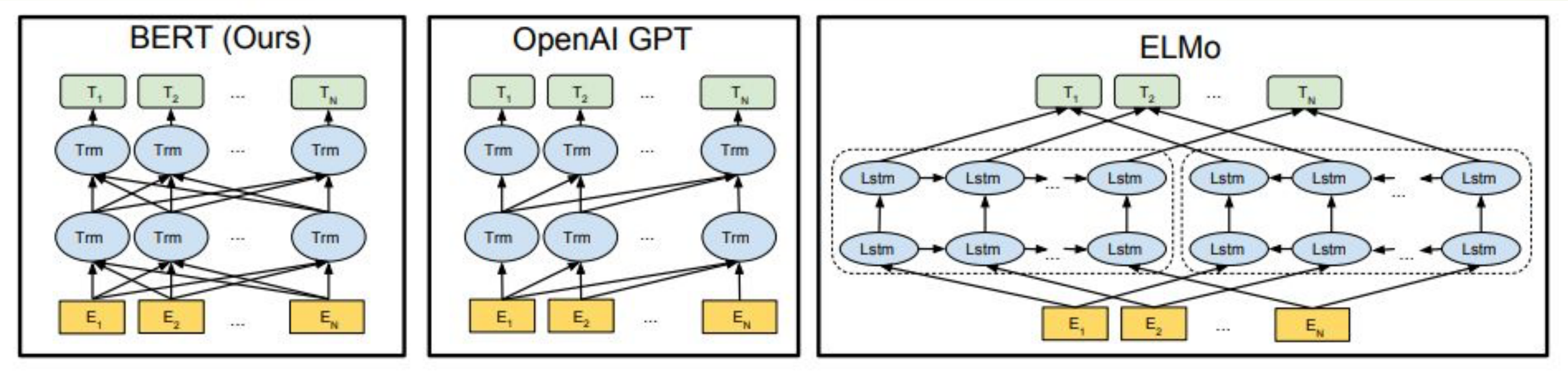
- 2018's GPT was a big success in pretraining a decoder!
- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
    - Contains long spans of contiguous text, for learning long-distance dependencies.



Decoders

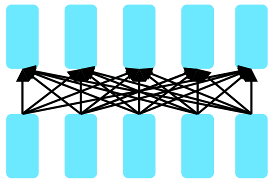
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

How about bi-directional transformers? – BERT



BERT: Devlin, Chang, Lee, Toutanova (2018)

- BERT (**Bidirectional Encoder Representations from Transformers**):
- Pre-training of Deep Bidirectional Transformers for Language Understanding, which is then fine-tuned for a task
- Want: truly bidirectional information flow without leakage in a deep model



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?

Masked Language Model

- **Problem**: How the words see each other in bi-directions?
- **Solution**: Mask out *k*% of the input words, and then predict the masked words
  - We always use $k = 15\%$

```
                           store                  gallon
                             ↑                      ↑
the man went to the [MASK] to buy a [MASK] of milk
```

- Too little masking: Too expensive to train
- Too much masking: Not enough context

Masked LM

- **Problem**: Masked token never seen at fine-tuning
- **Solution**: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
- 80% of the time, replace with [MASK]
  - went to the store → went to the [MASK]
- 10% of the time, replace random word
  - went to the store → went to the running
- 10% of the time, keep same
  - went to the store → went to the store

Next Sentence Prediction

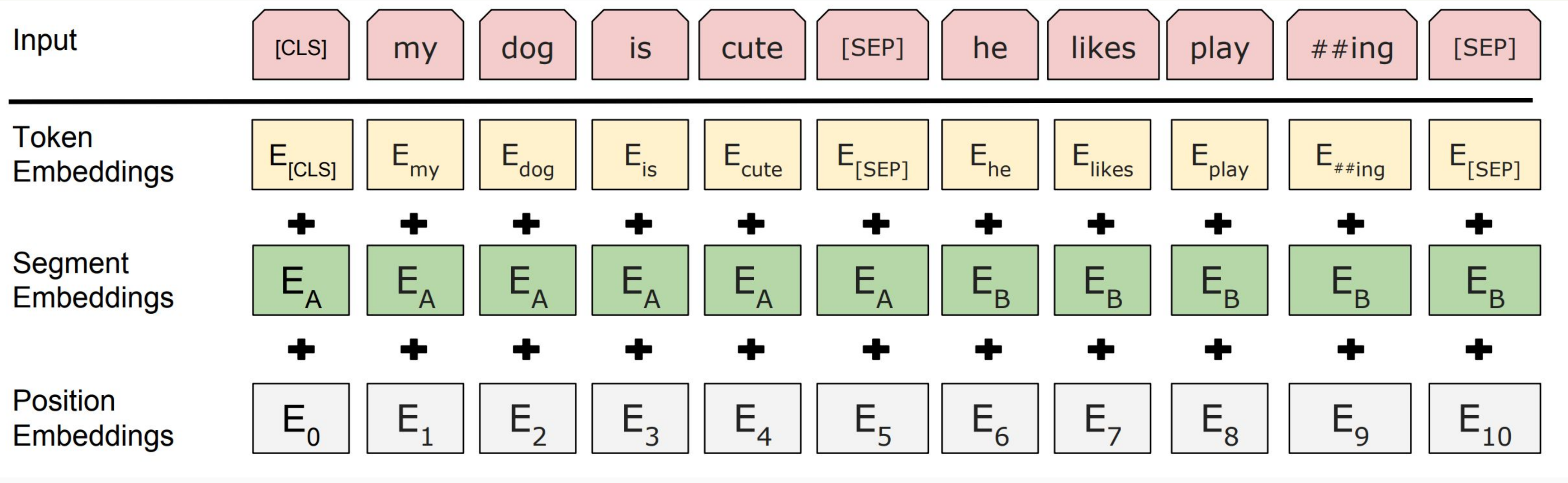
- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

BERT sentence pair encoding

- Token embeddings are word pieces (30k)
- Learned segmented embedding represents each sentence
- Positional embedding is as for other Transformer architectures

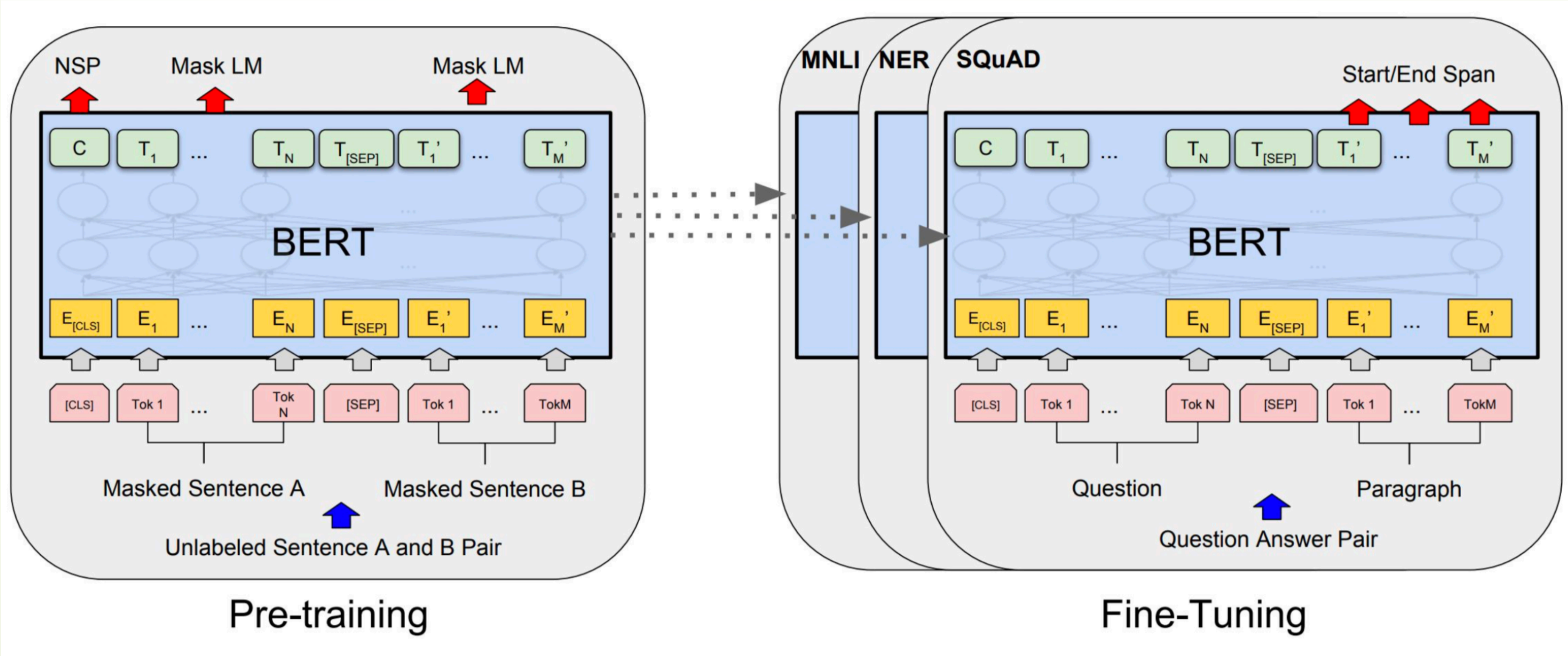


Training

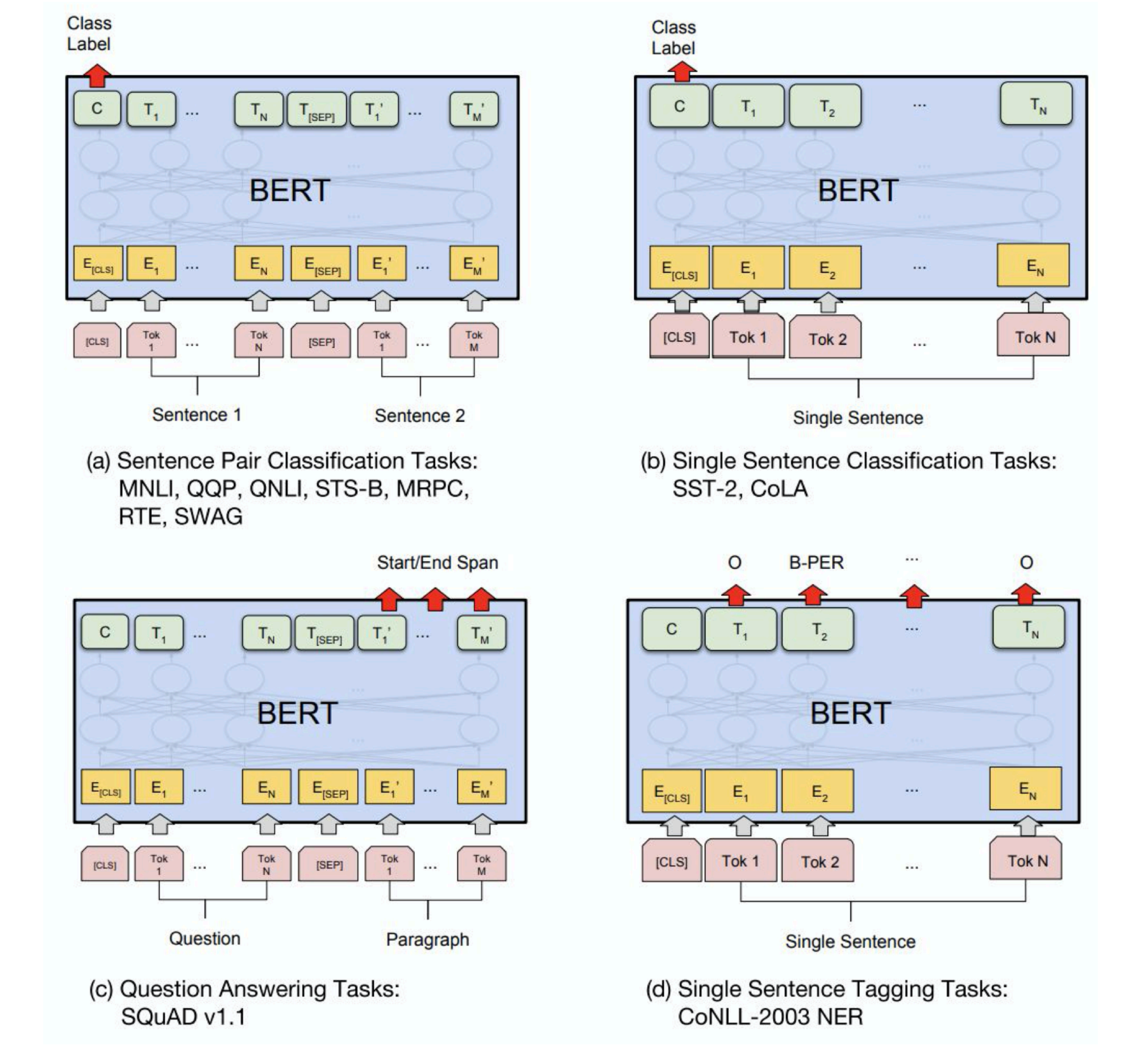
- 2 model released:
  - BERT-Base: 12-layer, 768-hidden, 12-head, 110 million params.
  - BERT-Large: 24-layer, 1024-hidden, 16-head, 340 million params.
- Training Data:
  - BookCorpus (800M words)
  - English Wikipedia (2.5B words)
- Batch Size: 131,072 words
  - (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- Trained on 4x4 or 8x8 TPU slice for 4 days
- Pretraining is expensive and impractical on a single GPU; Finetuning is practical and common on a single GPU

BERT model fine tuning

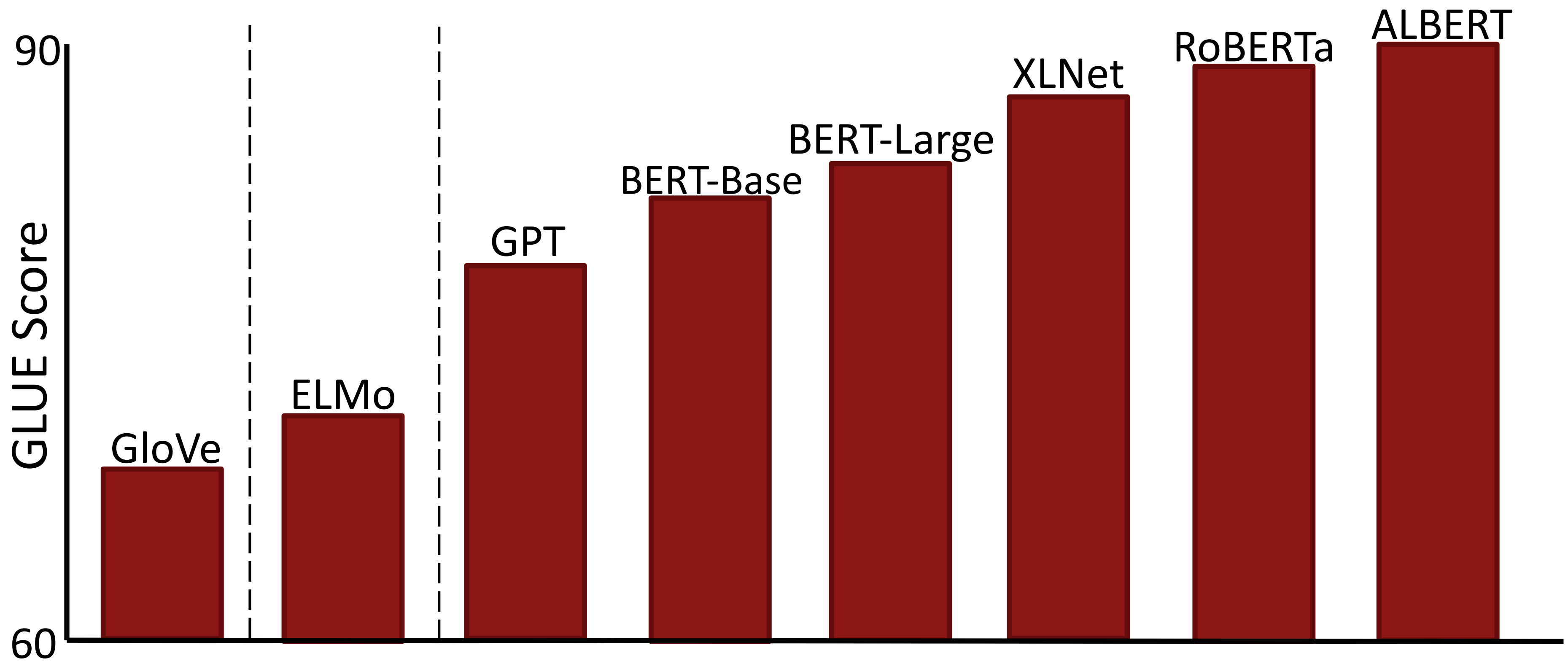
- Simply learn a classifier built on the top layer for each task that you fine tune for



BERT model fine tuning

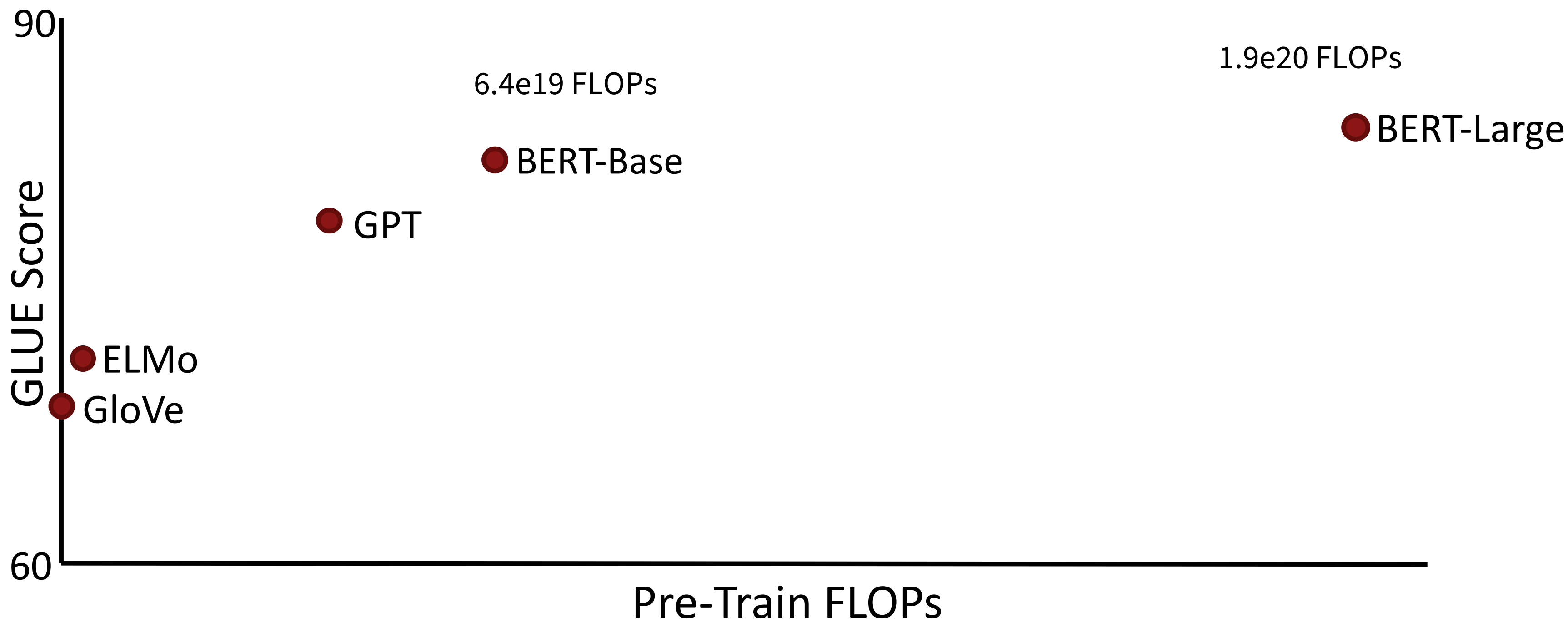


Rapid Progress for Pre-training (GLUE Benchmark)



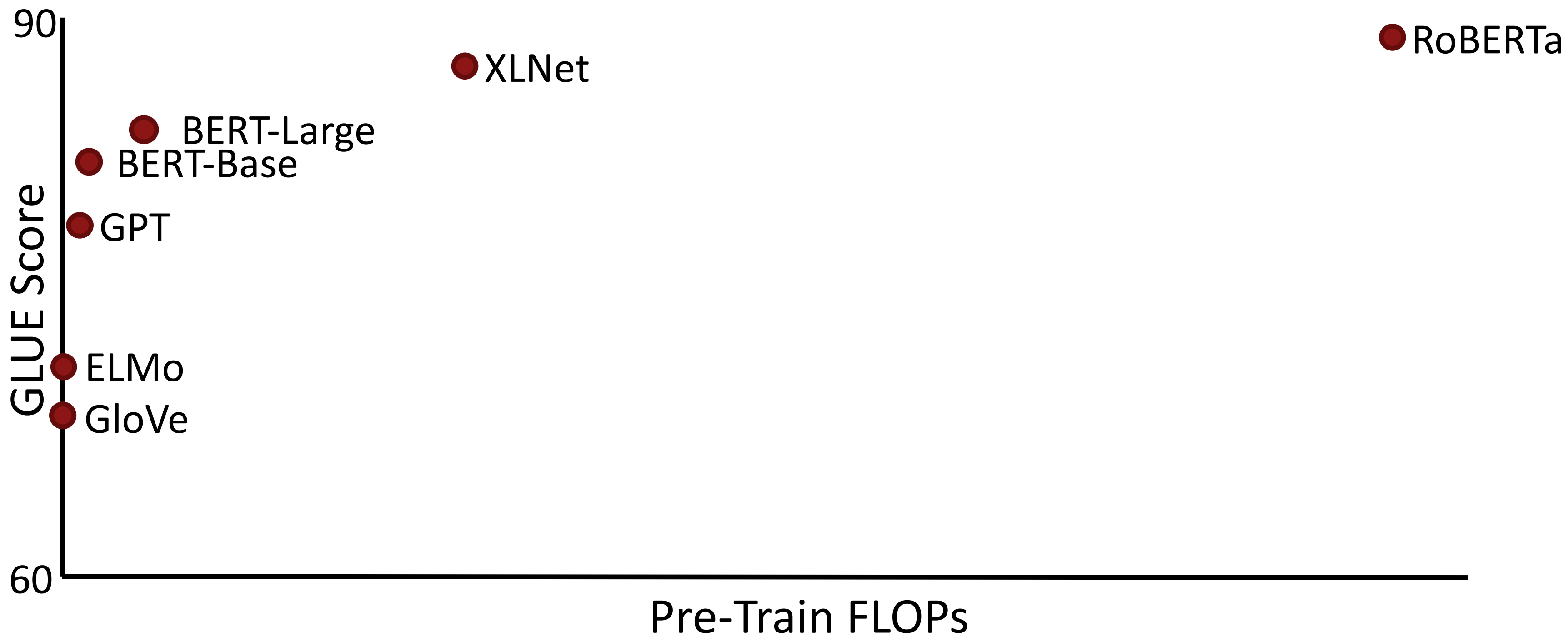
Over 3x reduction in error in 2 years, “superhuman” performance

But let's change the x-axis to computational cost...



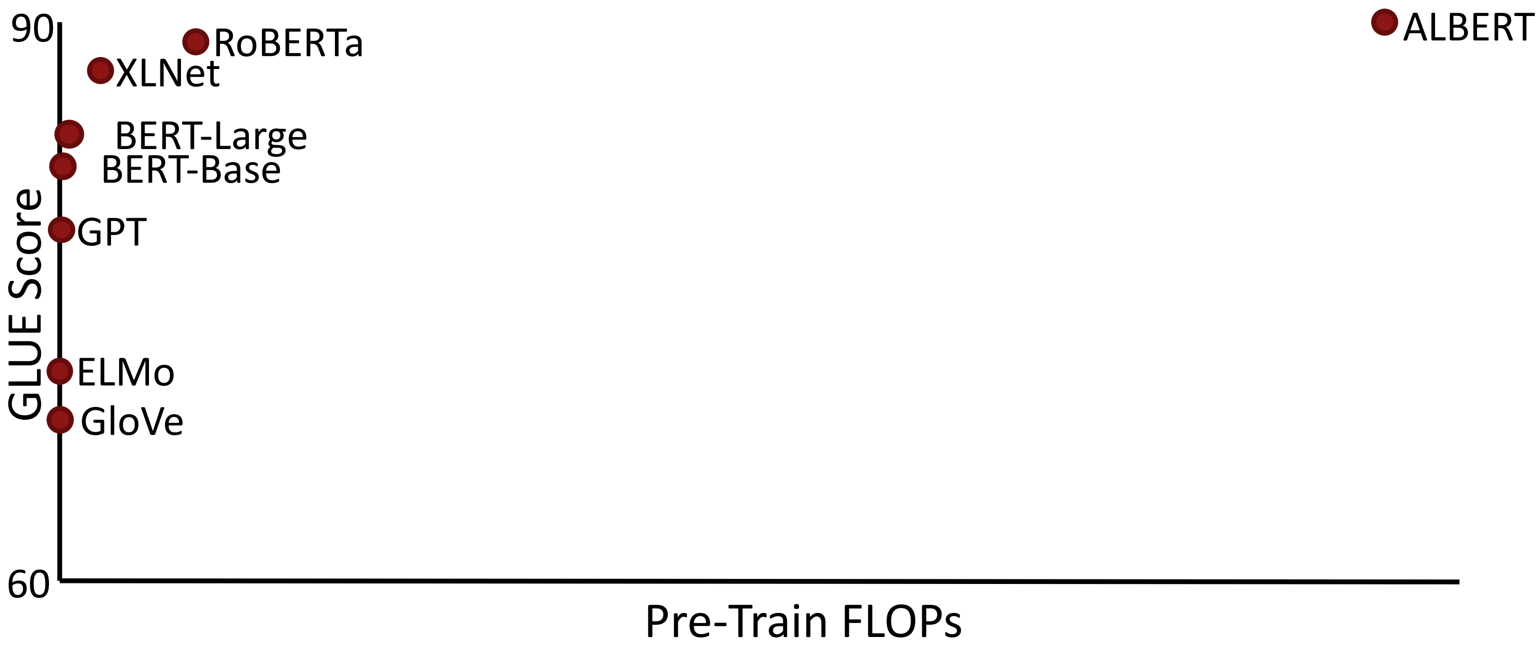
BERT-Large uses 60x more compute than ELMo

But let's change the x-axis to computational cost…



RoBERTa uses 16x more compute than BERT-Large

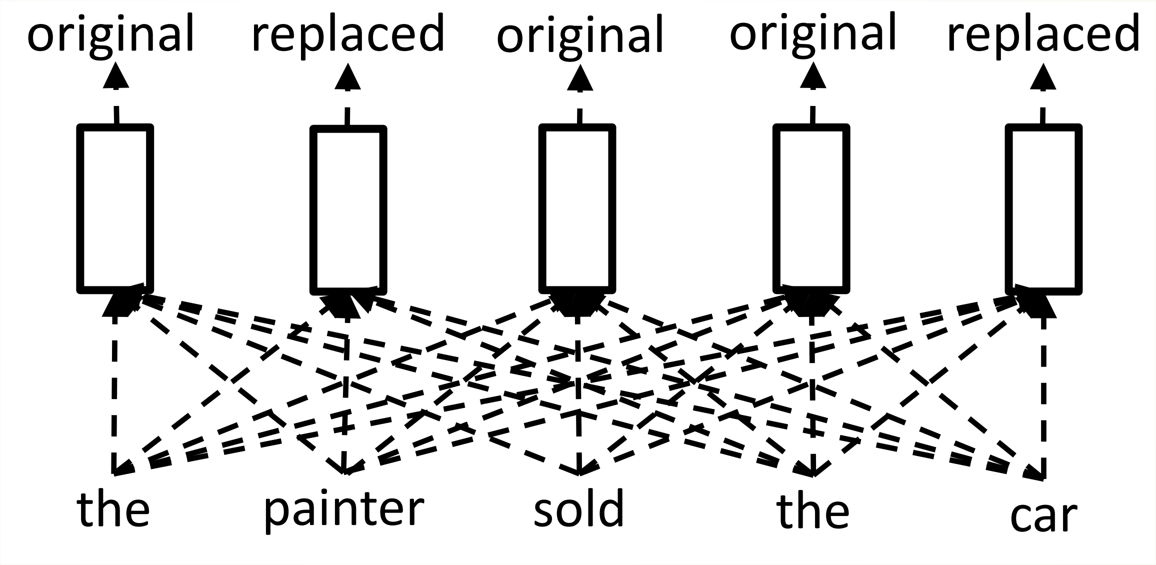
More compute, more better?



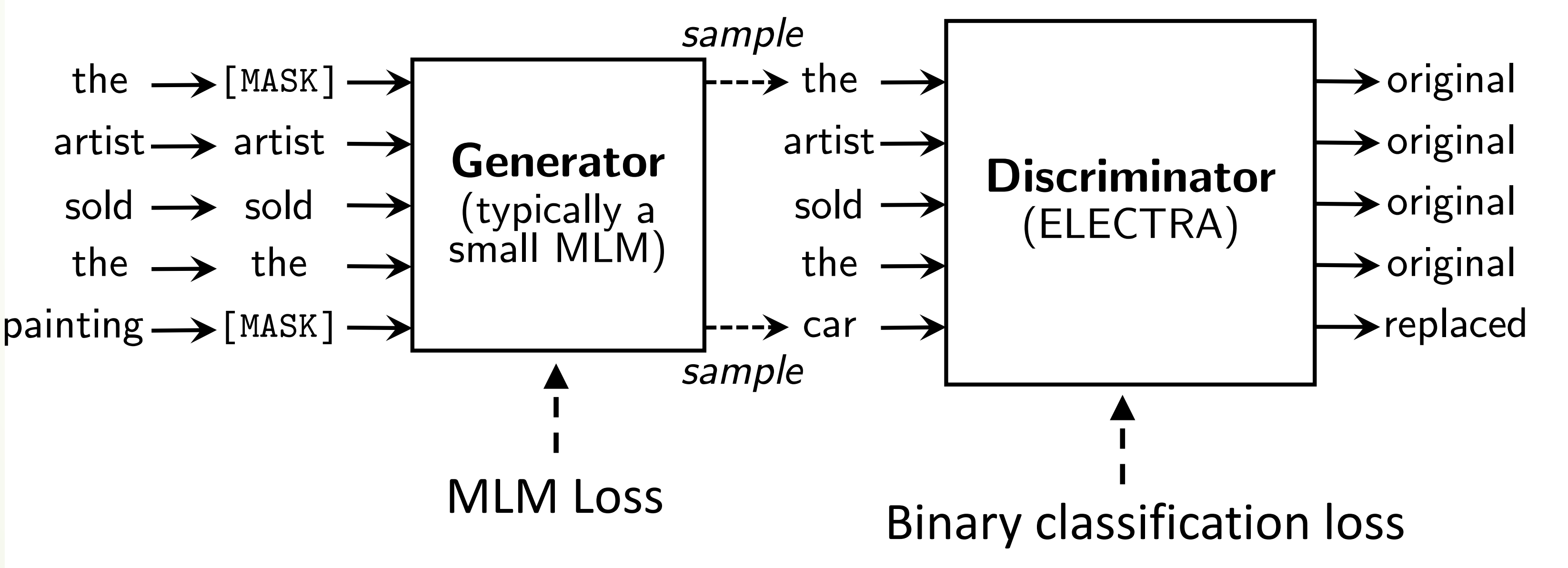
ALBERT uses 10x more compute than RoBERTa

ELECTRA: "Efficiently Learning an Encoder to Classify Token Replacements Accurately"

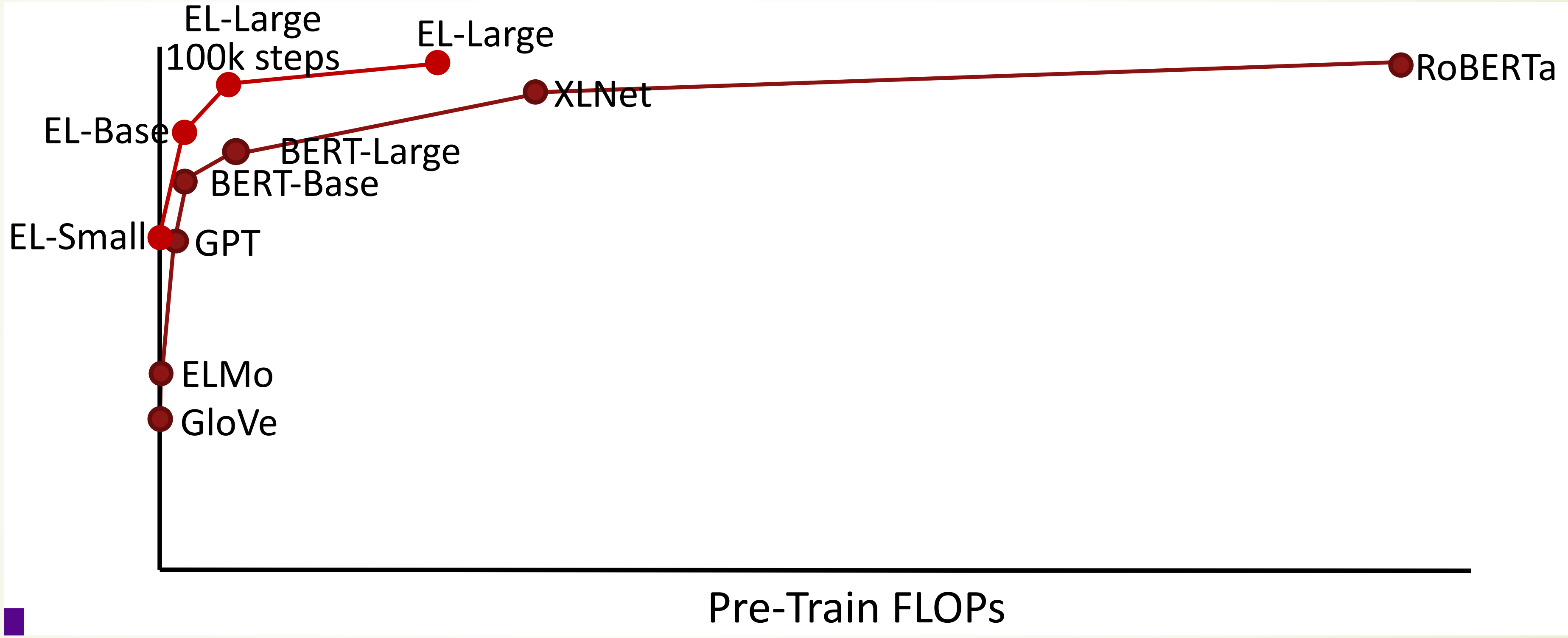
- Clark, Luong, Le, and Manning, ICLR 2020. https://openreview.net/pdf?id=r1xMH1BtvB
- Bidirectional model but learn from all tokens



Generating Replacements

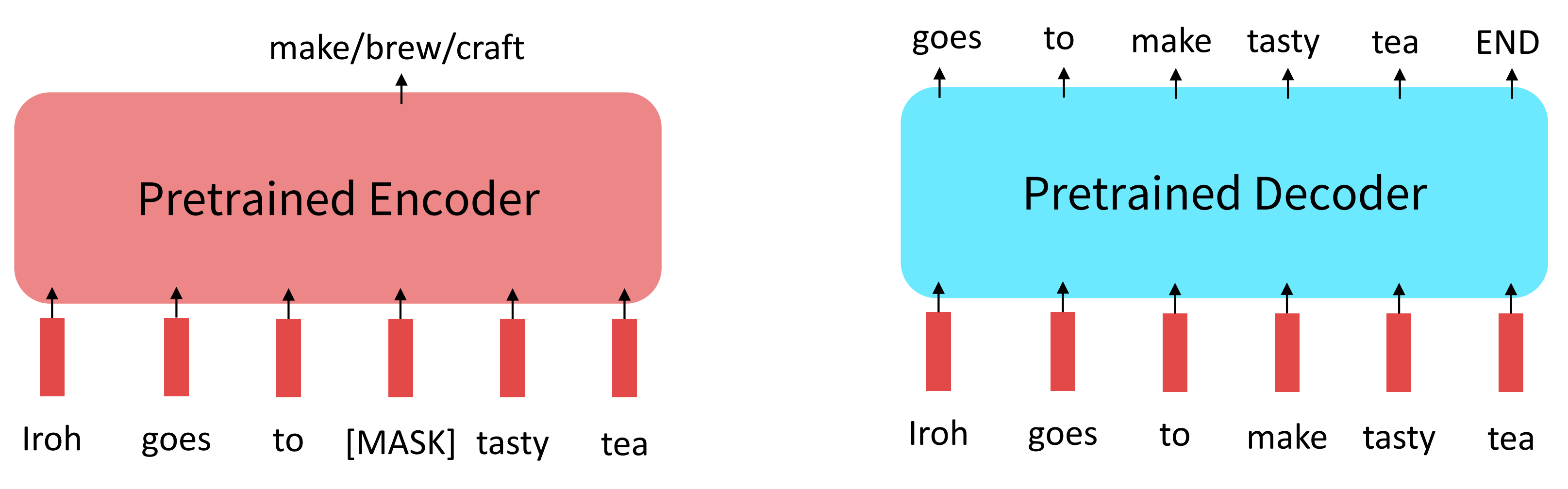


Results: GLUE Score vs Compute



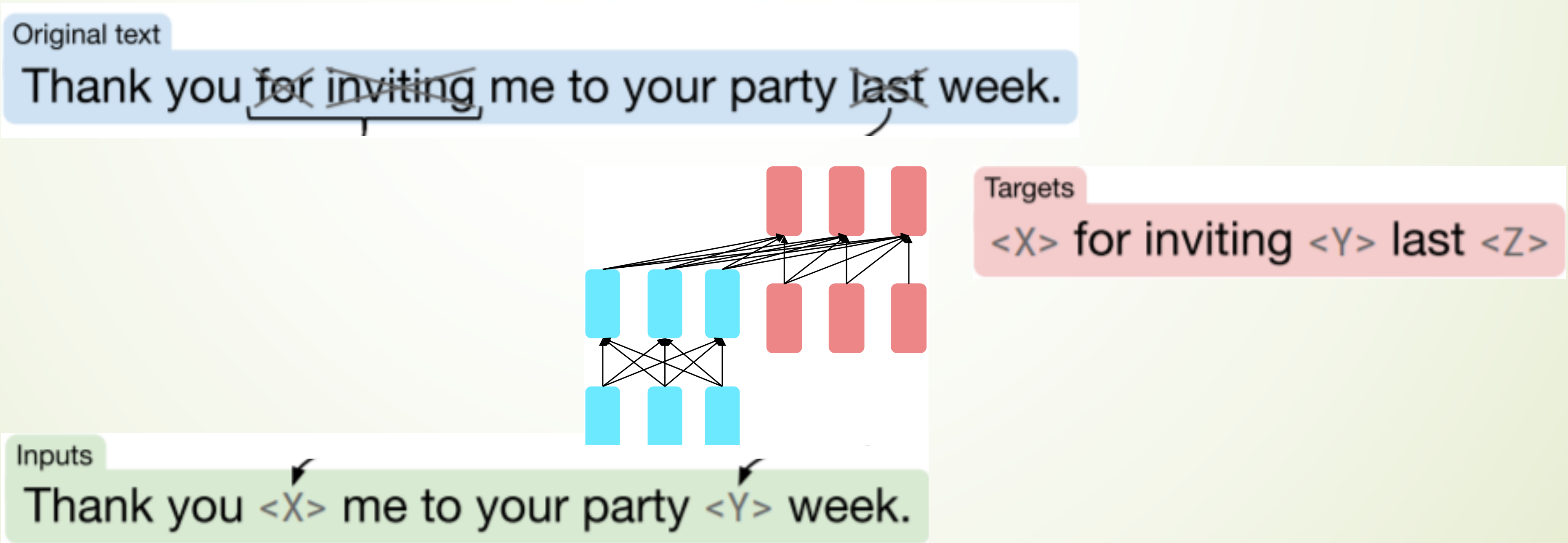
Limitations of Pretrained Encoders

- Those results looked great! Why not used pretrained encoders for *everything*?
- If your task involves generating sequences, consider using a pretrained **decoder**; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.



Pretraining encoders-decoders: T5

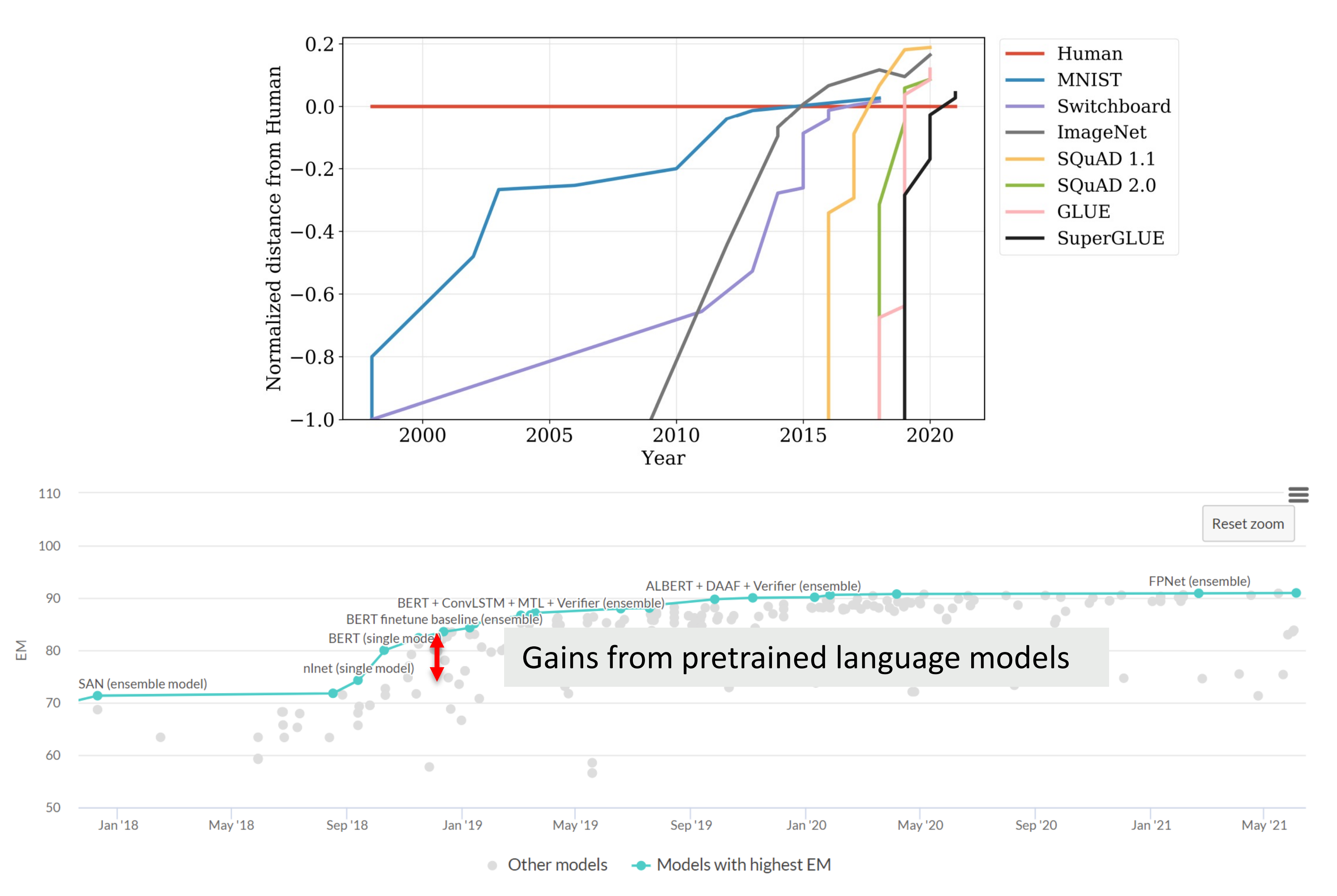
- Pretraining encoder-decoders: what pretraining objective to use?
- What Raffel et al., 2018 found to work best was **span corruption**: **T5**.
- Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!
- A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

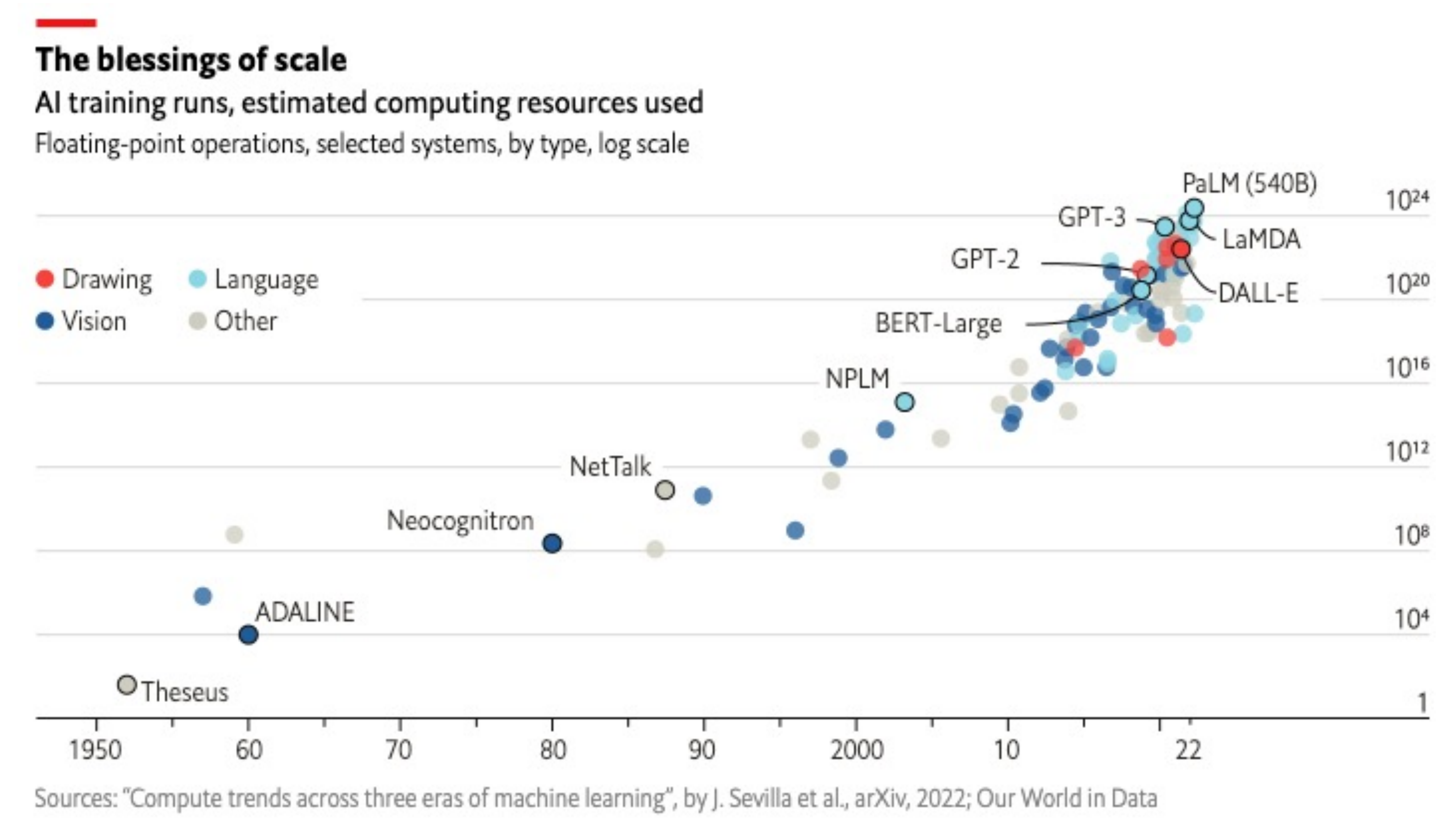


GPT-3, In-context learning, and very large models

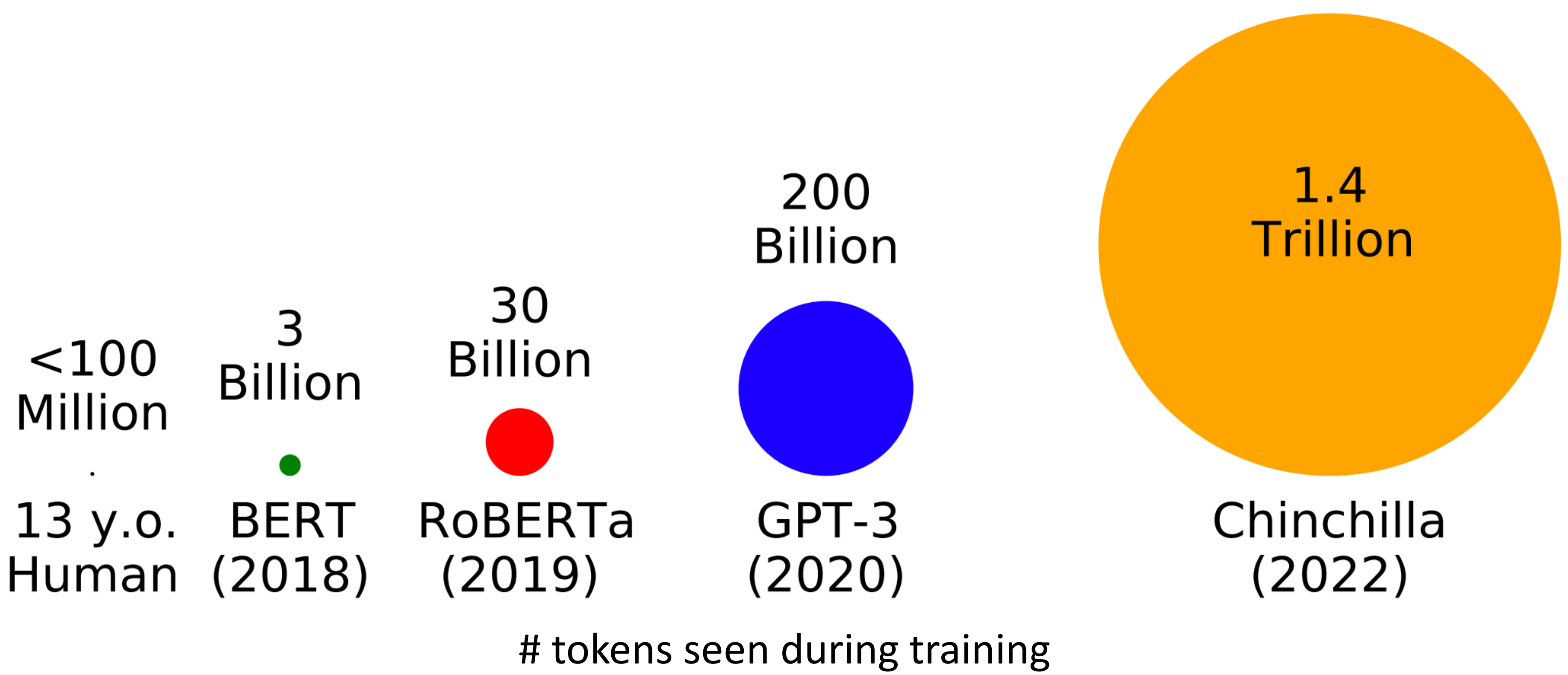
- So far, we've interacted with pretrained models in two ways:
  - Sample from the distributions they define (maybe providing a prompt)
  - Fine-tune them on a task we care about, and take their predictions.
- Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.
- GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.
- **GPT-3 has 175 billion parameters.**

Pretraining revolution



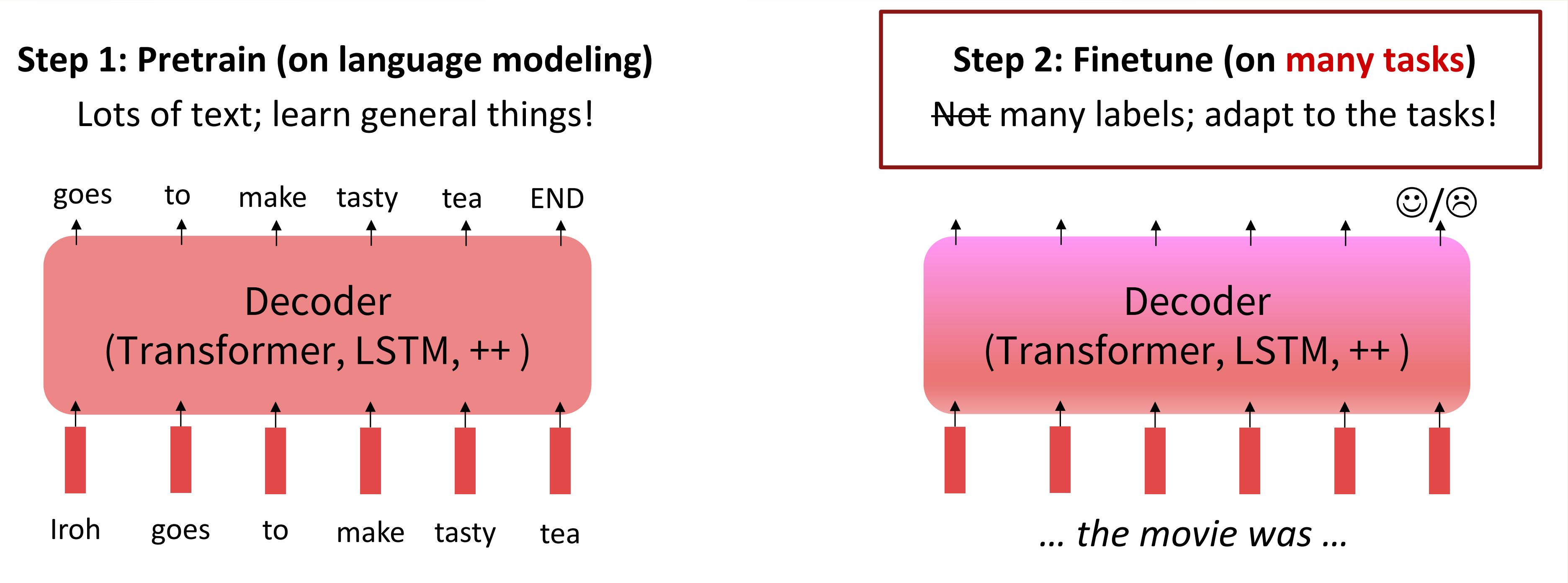

The blessings of scale
AI training runs, estimated computing resources used
Floating-point operations, selected systems, by type, log scale
Drawing
Language
Vision
Other
PaLM (540B)
GPT-3
LaMDA
GPT-2
DALL-E
BERT-Large
NPLM
NetTalk
Neocognitron
ADALINE
Theseus
10²⁴
10²⁰
10¹⁶
10¹²
10⁸
10⁴
1
1950
60
70
80
90
2000
10
22
Sources: "Compute trends across three eras of machine learning", by J. Sevilla et al., arXiv, 2022; Our World in Data

More and more data



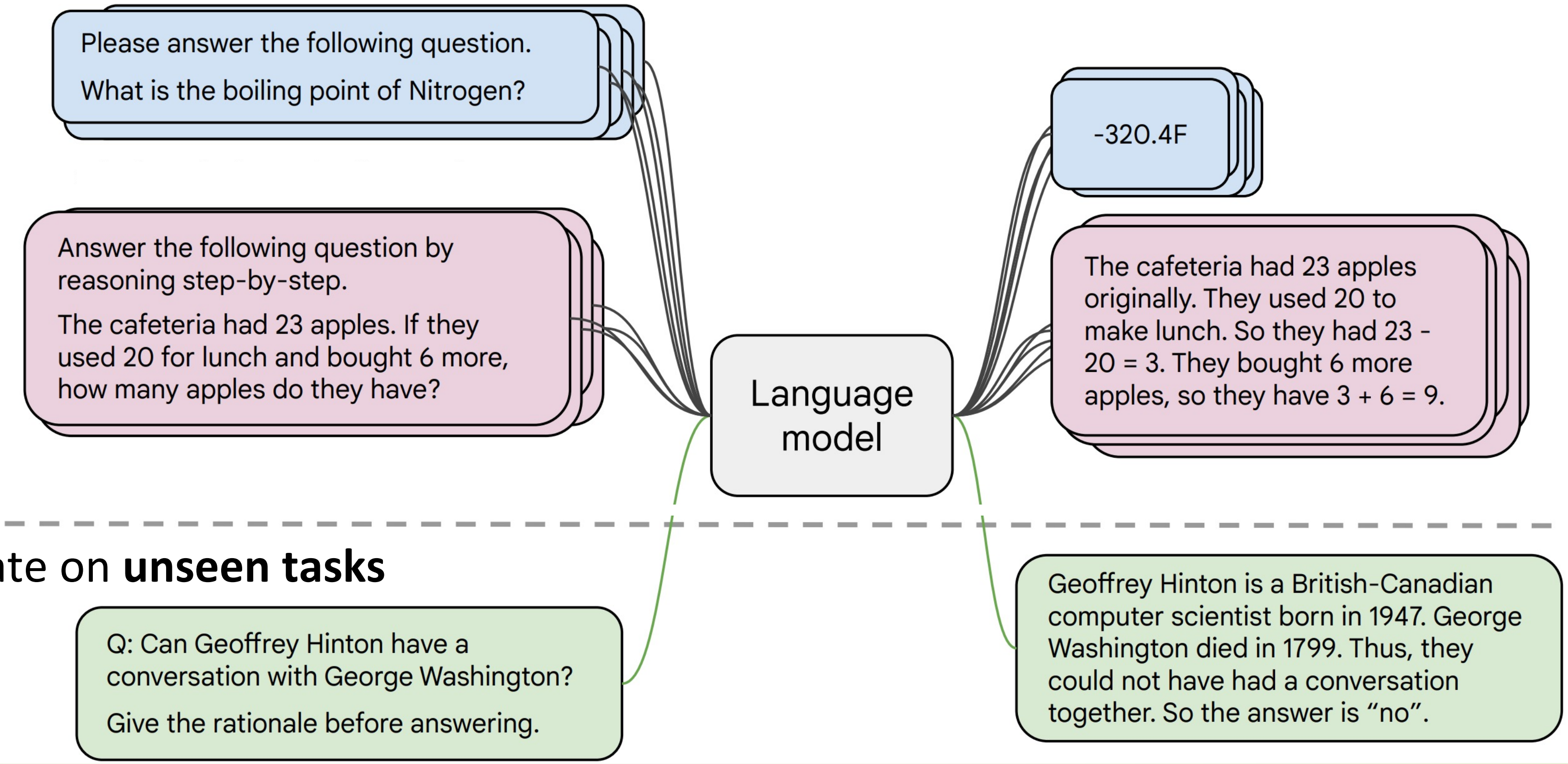
https://babylm.github.io/

Scaling up finetuning



Instruction Finetuning

- **Collect examples** of (instruction, output) pairs across many tasks and finetune an LM



- Evaluate on **unseen tasks**

[FLAN-T5; Chung et al., 2022]

Limitations of instruction finetuning

- One limitation of instruction finetuning is obvious: it's **expensive** to collect ground-truth data for tasks.
- But there are other, subtler limitations too. Can you think of any?
- **Problem 1:** tasks like open-ended creative generation have no right answer.
  - *E.g. Write me a story about a dog and her pet grasshopper.*
- **Problem 2:** language modeling penalizes all token-level mistakes equally, but some errors are worse than others.
- Even with instruction finetuning, there a mismatch between the LM objective and the objective of "satisfy human preferences"!
- Can we **explicitly attempt to satisfy human preferences? (alignment)**

Reinforcement Learning from Human Feedback (RLHF)

Reward maximization from human

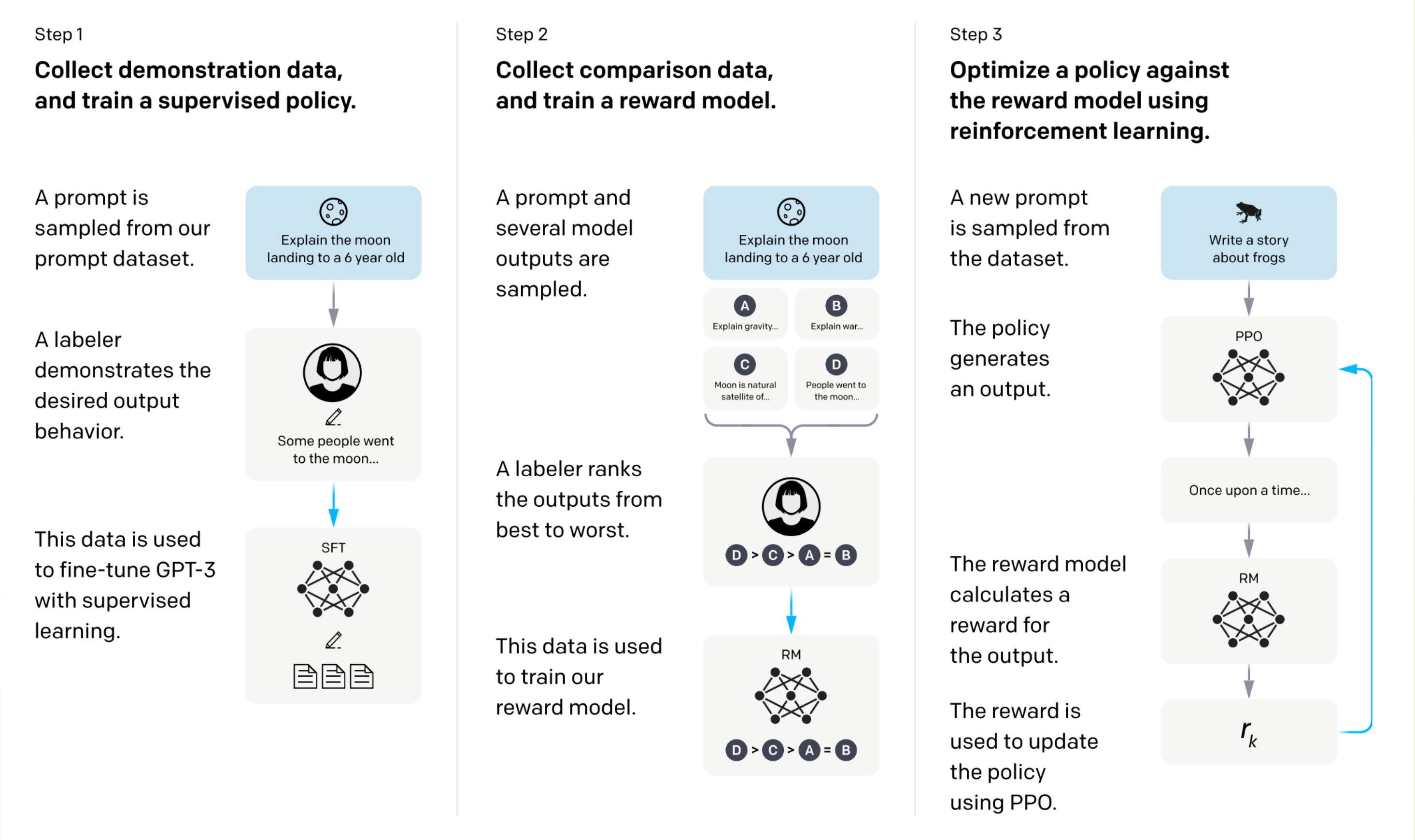
- Let's say we were training a language model on some task (e.g. summarization).
- For each LM sample s, imagine we had a way to obtain a *human reward* of that summary: $R(s) \in$ R, higher is better.
- Now we want to maximize the expected reward of samples from our LM:

$$\mathbb{E}_{\hat{s} \sim p_\theta(s)}[R(\hat{s})]$$

SAN FRANCISCO, California (CNN) -- A magnitude 4.2 earthquake shook the San Francisco ... overturn unstable objects.	An earthquake hit San Francisco. There was minor property damage, but no injuries.	The Bay Area has good weather but is prone to earthquakes and wildfires.
	s_1	s_2
	$R(s_1) = 8.0$	$R(s_2) = 1.2$

High-level instantiation: 'RLHF' pipeline

- First step: instruction tuning!
- Second + third steps: maximize reward (but how??)



Reinforcement learning to the rescue

- The field of **reinforcement learning (RL)** has studied these (and related) problems for many years now [Williams, 1992; Sutton and Barto, 1998]
- Circa 2013: resurgence of interest in RL applied to deep learning, game-playing [Mnih et al., 2013]
- But the interest in applying RL to modern LMs is an even newer phenomenon [Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022]. **Why?**
  - RL w/ LMs has commonly been viewed as very hard to get right (still is!)
  - Newer advances in RL algorithms that work for large neural models, including language models (e.g. PPO; [Schulman et al., 2017])

AlphaGo

Optimizing for human preferences

- How do we actually change our LM parameters θ to maximize this?

$$\mathbb{E}_{\hat{s} \sim p_\theta(s)}[R(\hat{s})]$$

- Let's try doing gradient ascent!

$$\theta_{t+1} := \theta_t + \alpha \nabla_{\theta_t} \mathbb{E}_{\hat{s} \sim p_{\theta_t}(s)}[R(\hat{s})]$$

How do we estimate this expectation??

What if our reward function is non-differentiable??

- **Policy gradient** methods in RL (e.g., [Williams, 1992]) give us tools for estimating and optimizing this objective.

A very brief introdution to Policy Gradient

- We want to obtain

(defn. of expectation) (linearity of gradient)

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \nabla_\theta \sum_s R(s) p_\theta(s) = \sum_s R(s)\, \nabla_\theta p_\theta(s)$$

- Here we'll use a very handy trick known as the **log-derivative trick**. Let's try taking the gradient of $\log p_\theta(s)$

$$\nabla_\theta \log p_\theta(s) = \frac{1}{p_\theta(s)} \nabla_\theta p_\theta(s) \quad \Longrightarrow \quad \nabla_\theta p_\theta(s) = p_\theta(s)\ \nabla_\theta \log p_\theta(s)$$

(chain rule)

- Plug back in:

This is an expectation of this

$$\sum_s R(s)\, \nabla_\theta p_\theta(s) = \sum_s p_\theta(s) R(s)\ \nabla_\theta \log p_\theta(s)$$

$$\boxed{= \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})\ \nabla_\theta \log p_\theta(\hat{s})]}$$

A very brief introdution to Policy Gradient

- Now we have put the gradient "inside" the expectation, we can approximate this objective with Monte Carlo samples:

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})\,\nabla_\theta \log p_\theta(\hat{s})] \approx \frac{1}{m}\sum_{i=1}^{m} R(s_i)\,\nabla_\theta \log p_\theta(s_i)$$

This is why it's called **"reinforcement learning"**: we **reinforce** good actions, increasing the chance they happen again.

- Giving us the update rule:

$$\theta_{t+1} \coloneqq \theta_t + \alpha \frac{1}{m}\sum_{i=1}^{m} R(s_i)\,\nabla_{\theta_t} \log p_{\theta_t}(s_i)$$

If R is +++

Take gradient steps to maximize $p_\theta(s_i)$

If R is ---

Take steps to minimize $p_\theta(s_i)$

This is **heavily simplified**! There is a *lot* more needed to do RL w/ LMs. **Can you see any problems with this objective?**

How do we model human preferences?

- Awesome: now for any **arbitrary, non-differentiable reward function** $R(s)$, we can train our language model to maximize expected reward.
- Not so fast! (Why not?)
- **Problem 1:** human-in-the-loop is expensive!
  - **Solution:** instead of directly asking humans for preferences, **model their preferences** as a separate (NLP) problem! [Knox and Stone, 2009]

```
An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.
```

s_1

$R(s_1) = 8.0$

```
The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.
```

s_2

$R(s_2) = 1.2$

Train an LM $RM_\phi(s)$ to predict human preferences from an annotated dataset, then optimize for RM_ϕ instead.

How do we model human preferences?

- **Problem 2:** human judgments are noisy and miscalibrated!
  - **Solution:** instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

```
A 4.2 magnitude
earthquake hit
San Francisco,
resulting in
massive damage.
```

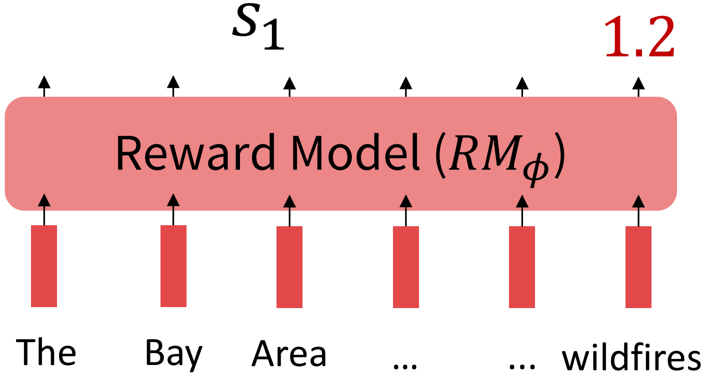
$$s_3$$

$$R(s_3) = \quad 4.1? \quad 6.6? \quad 3.2?$$

How do we model human preferences?

- **Problem 2:** human judgments are noisy and miscalibrated!
  - **Solution:** instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

s_1		s_3		s_2
An earthquake hit San Francisco. There was minor property damage, but no injuries.	>	A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.	>	The Bay Area has good weather but is prone to earthquakes and wildfires.



Bradley-Terry [1952] paired comparison model

$$J_{RM}(\phi) = -\mathbb{E}_{(s^w, s^l)\sim D}\left[\log \sigma(RM_\phi(s^w) - RM_\phi(s^l))\right]$$

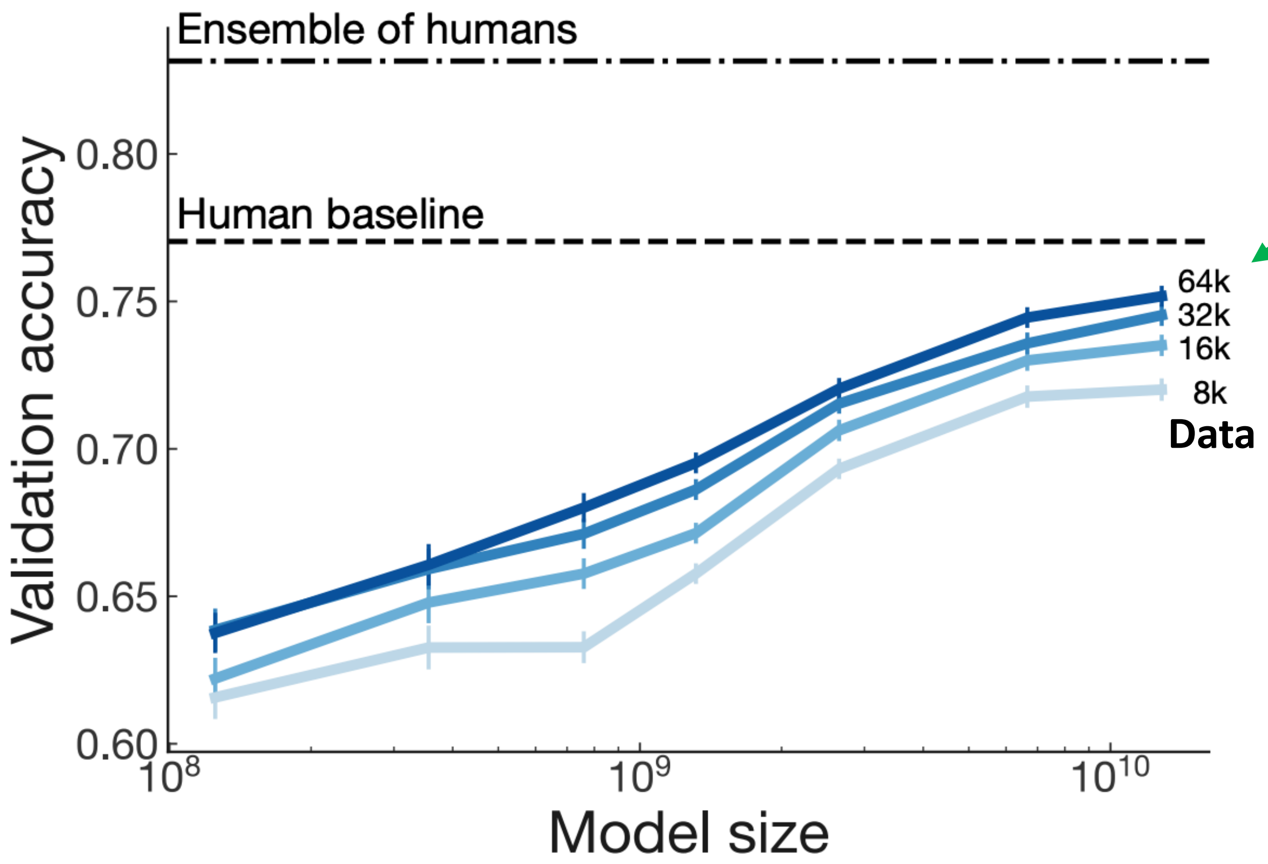
"winning" sample

"losing" sample

s^w should score higher than s^l

Training a reward model first

Evaluate RM on predicting outcome of held-out human judgments



Large enough RM trained on enough data approaching single human perf

[Stiennon et al., 2020]

RLHF: Putting it all together [Christiano et al., 2017; Stiennon et al., 2020]

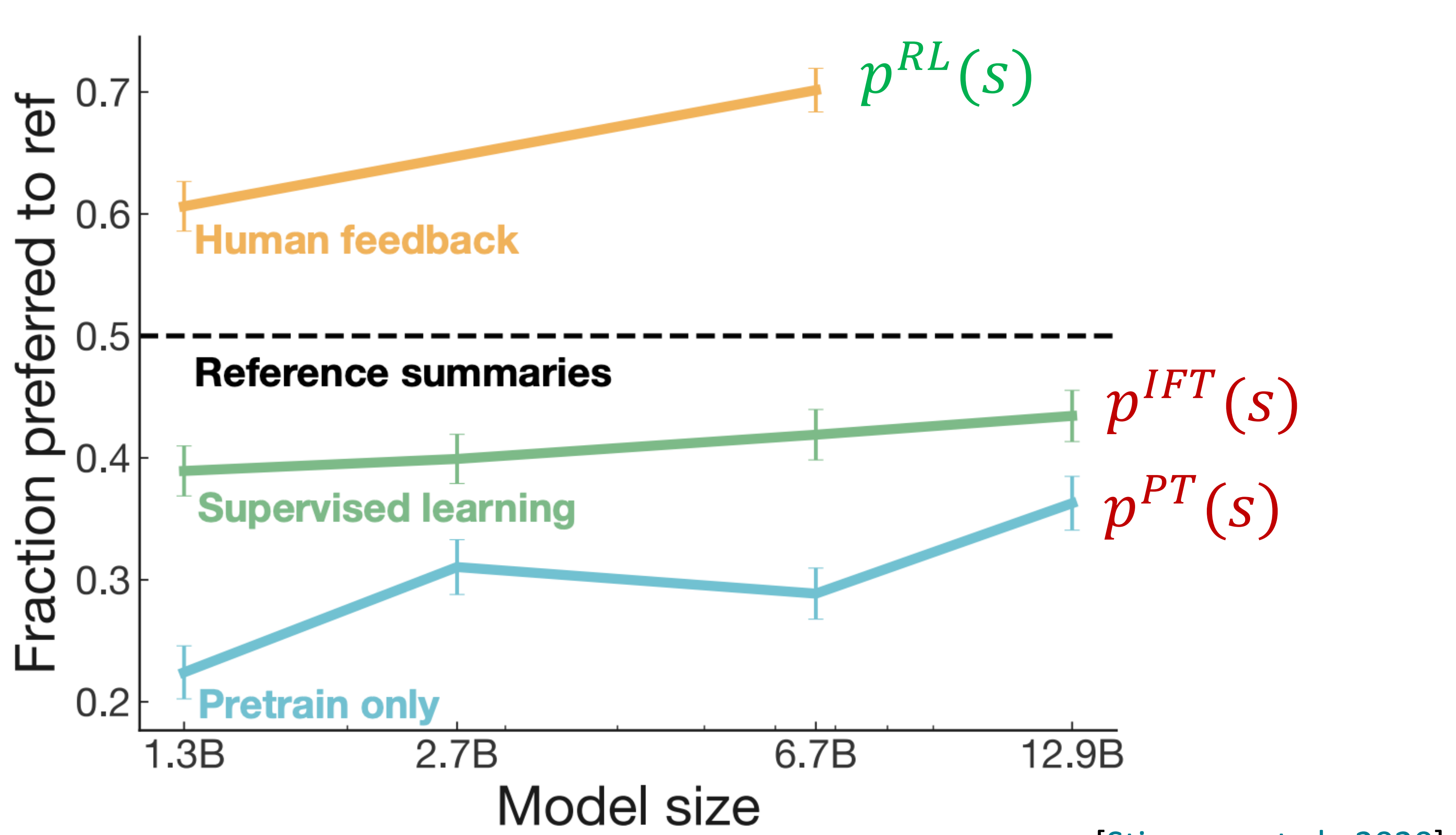
- Finally, we have everything we need:
  - A pretrained (possibly instruction-finetuned) LM $p^{\mathrm{PT}}(s)$
  - A reward model $RM(s)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons
  - A method for optimizing LM parameters towards an arbitrary reward function.
- Now to do RLHF:
  - Initialize a copy of the model $p_\theta{}^{\mathrm{RL}}(s)$, with parameters θ we would like to optimize
  - Optimize the following reward with RL:

$$R(s) = RM_\phi(s) - \underbrace{\beta \log\left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)}\right)}$$

Pay a price when $p_\theta^{RL}(s) > p^{PT}(s)$

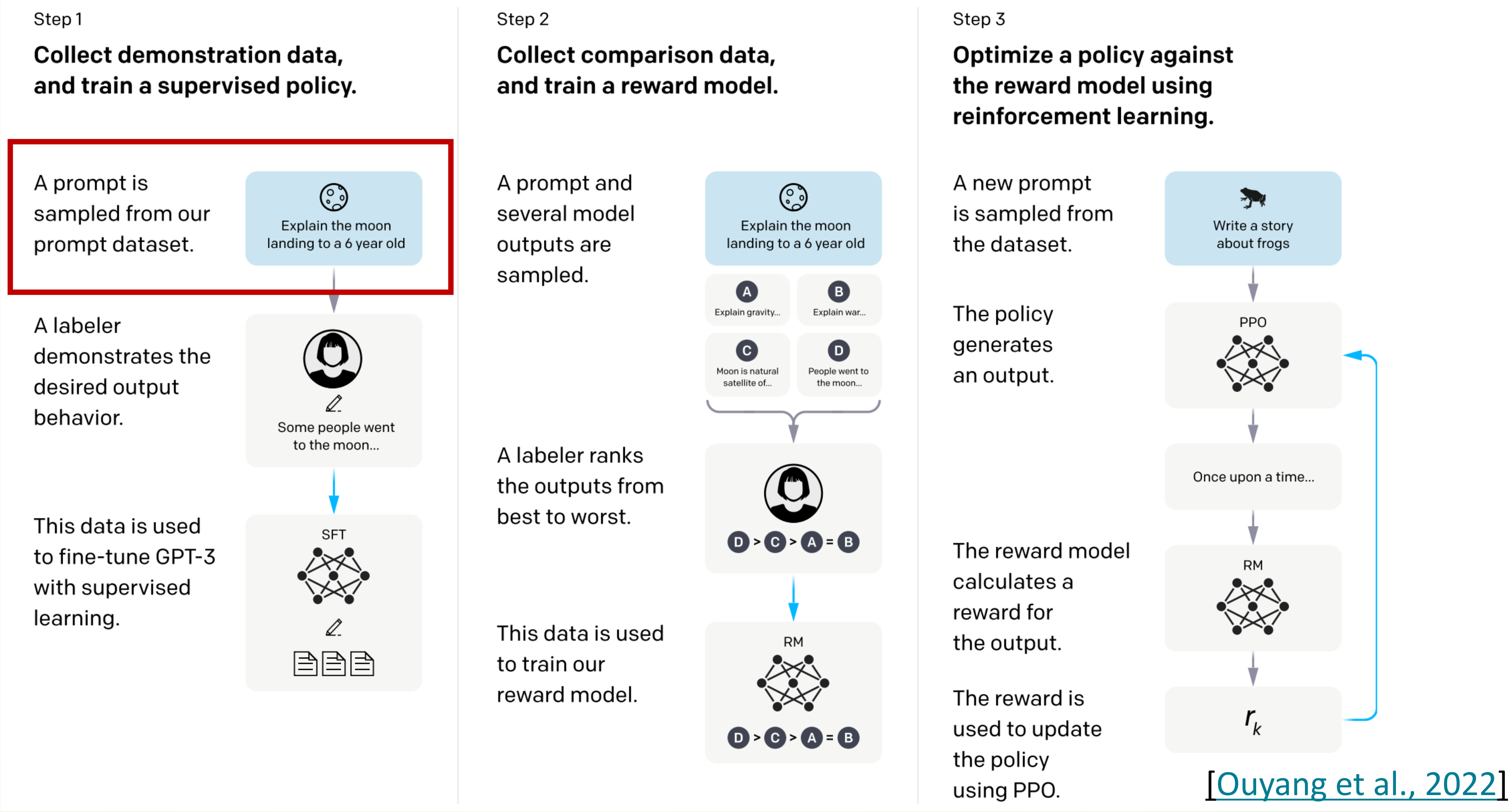
This is a penalty which prevents us from diverging too far from the pretrained model. In expectation, it is known as the **Kullback-Leibler** (**KL**) divergence between $p_\theta^{RL}(s)$ and $p^{PT}(s)$.

RLHF improves over pretraining and finetuning



[Stiennon et al., 2020]

InstructGPT: scaling up RLHF to tens of thousands (30k) of tasks



[Ouyang et al., 2022]

ChatGPT: Instruction Finetuning + RLHF for dialog agents [https://openai.com/blog/chatgpt/]

ChatGPT: Optimizing Language Models for Dialogue

Note: OpenAI (and similar companies) are keeping more details secret about ChatGPT training (including data, training parameters, model size)—perhaps to keep a competitive edge…

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.

(Instruction finetuning!)

ChatGPT: Instruction Finetuning + RLHF for dialog agents [https://openai.com/blog/chatgpt/]

ChatGPT: Optimizing Language Models for Dialogue

Note: OpenAI (and similar companies) are keeping more details secret about ChatGPT training (including data, training parameters, model size)—perhaps to keep a competitive edge…

Methods

To create a reward model for reinforcement learning, we needed to collect comparison data, which consisted of two or more model responses ranked by quality. To collect this data, we took conversations that AI trainers had with the chatbot. We randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them. Using these reward models, we can fine-tune the model using Proximal Policy Optimization. We performed several iterations of this process.

(RLHF!)

Yet, Limitations of RL + Reward Modeling

- **Human preferences are unreliable**!
  - ”Reward hacking” is a common problem in RL
  - Chatbots are rewarded to produce responses that *seem* authoritative and helpful, *regardless of truth*
  - This can result in making up facts + hallucinations
- **Models** of human preferences are *even more* unreliable!
  - RLHF labels are often obtained from overseas, low-wage workers
  - We also need to be quite careful about how annotator biases might creep into LMs

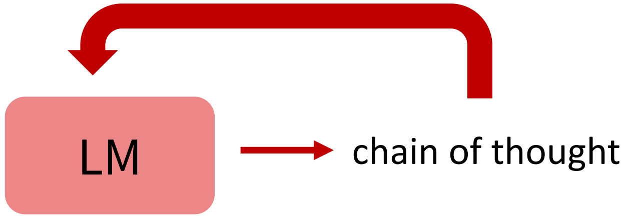
What’s next?

- RLHF is still a very underexplored and fastmoving area!
- RLHF gets you further than instruction finetuning, but is (still!) data expensive.
- Recent work aims to alleviate such data requirements:
  - RL from **AI feedback** [Bai et al., 2022]
  - Finetuning LMs on their own outputs [Huang et al., 2022; Zelikman et al., 2022]
- However, there are still many limitations of large LMs (size, hallucination) that may not be solvable with RLHF!

LARGE LANGUAGE MODELS CAN SELF-IMPROVE

Jiaxin Huang[1*] **Shixiang Shane Gu**[2] **Le Hou**[2†] **Yuexin Wu**[2] **Xuezhi Wang**[2]
Hongkun Yu[2] **Jiawei Han**[1]
[1]University of Illinois at Urbana-Champaign [2]Google
[1]{jiaxinh3, hanj}@illinois.edu [2]{shanegu, lehou, crickwu, xuezhiw, hongkuny}@google.com

[Huang et al., 2022]



Self-Taught Reasoner (STaR)
[Zelikman et al., 2022]

Thank you!