UVA CS 4774: Machine Learning

S3: Lecture 19: Training Transformers

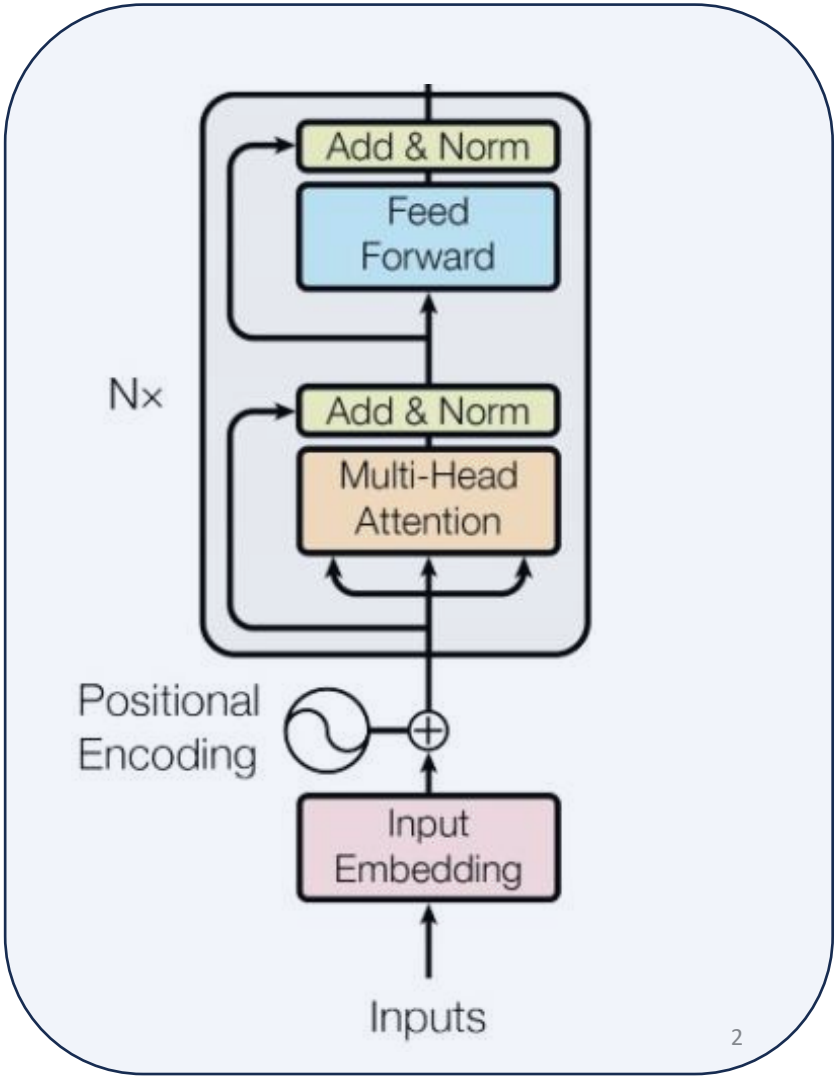
Pretraining & Post-training

Dr. Yanjun Qi

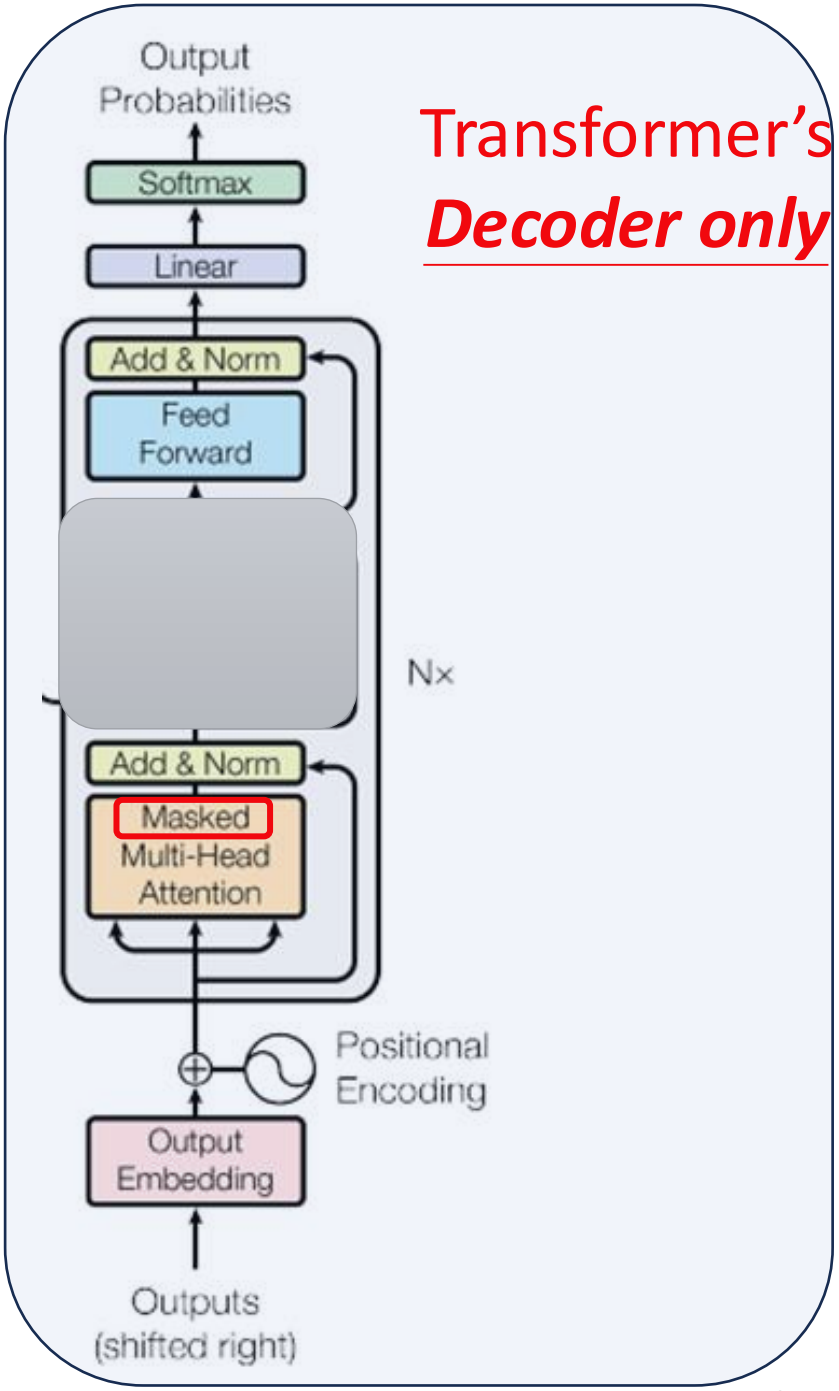
University of Virginia
Department of Computer Science

Recap:

Transformer's *Encoder Only*



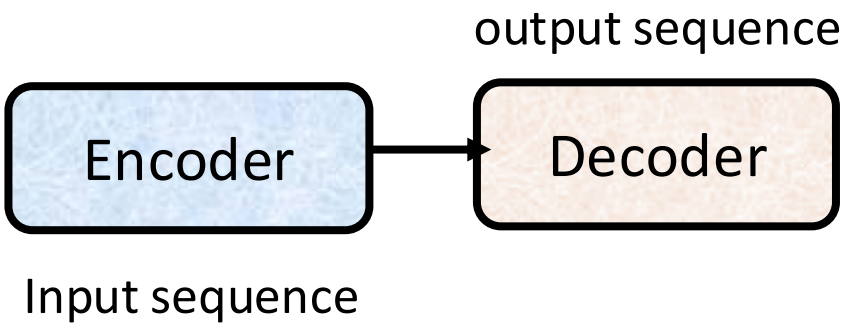
Transformer's *Decoder only*

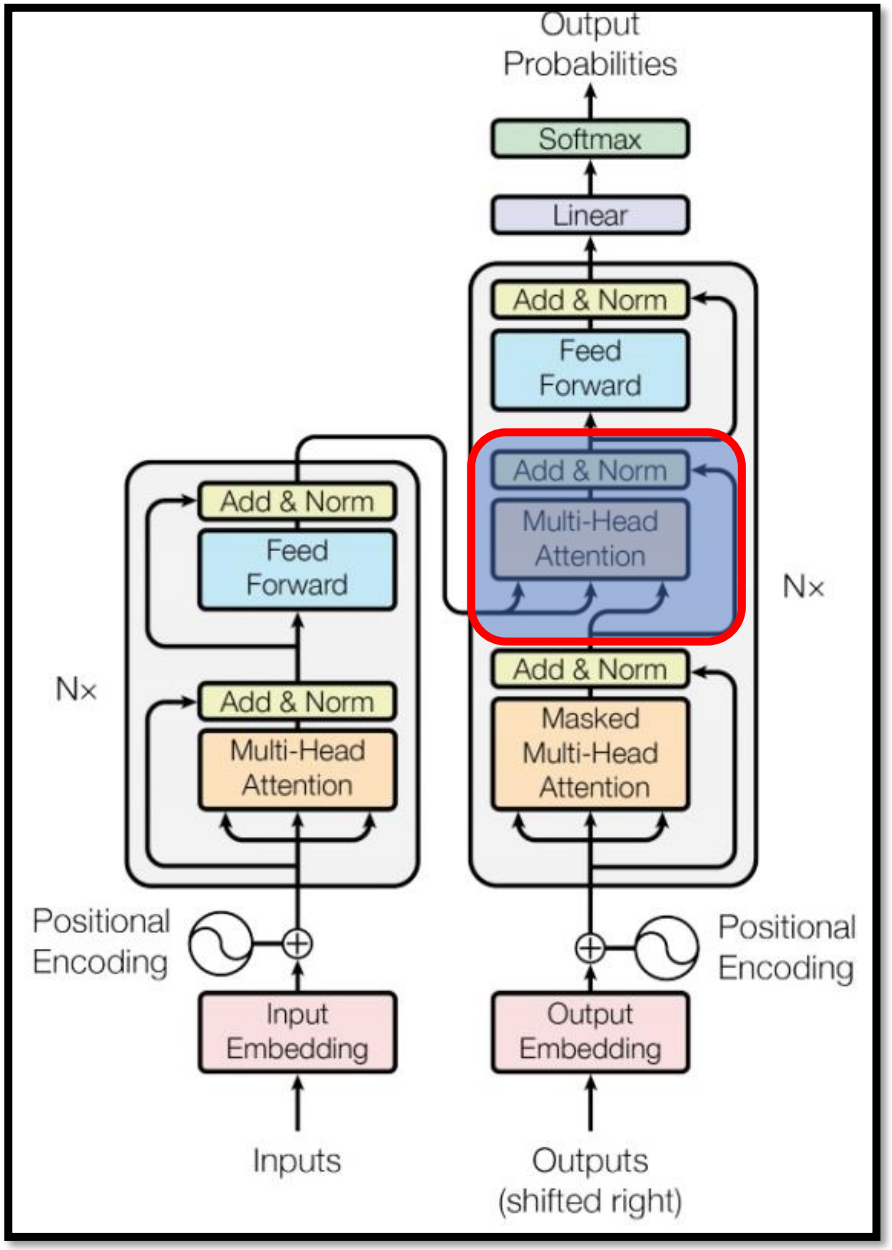


credit: Prof. Hung-yi Lee,

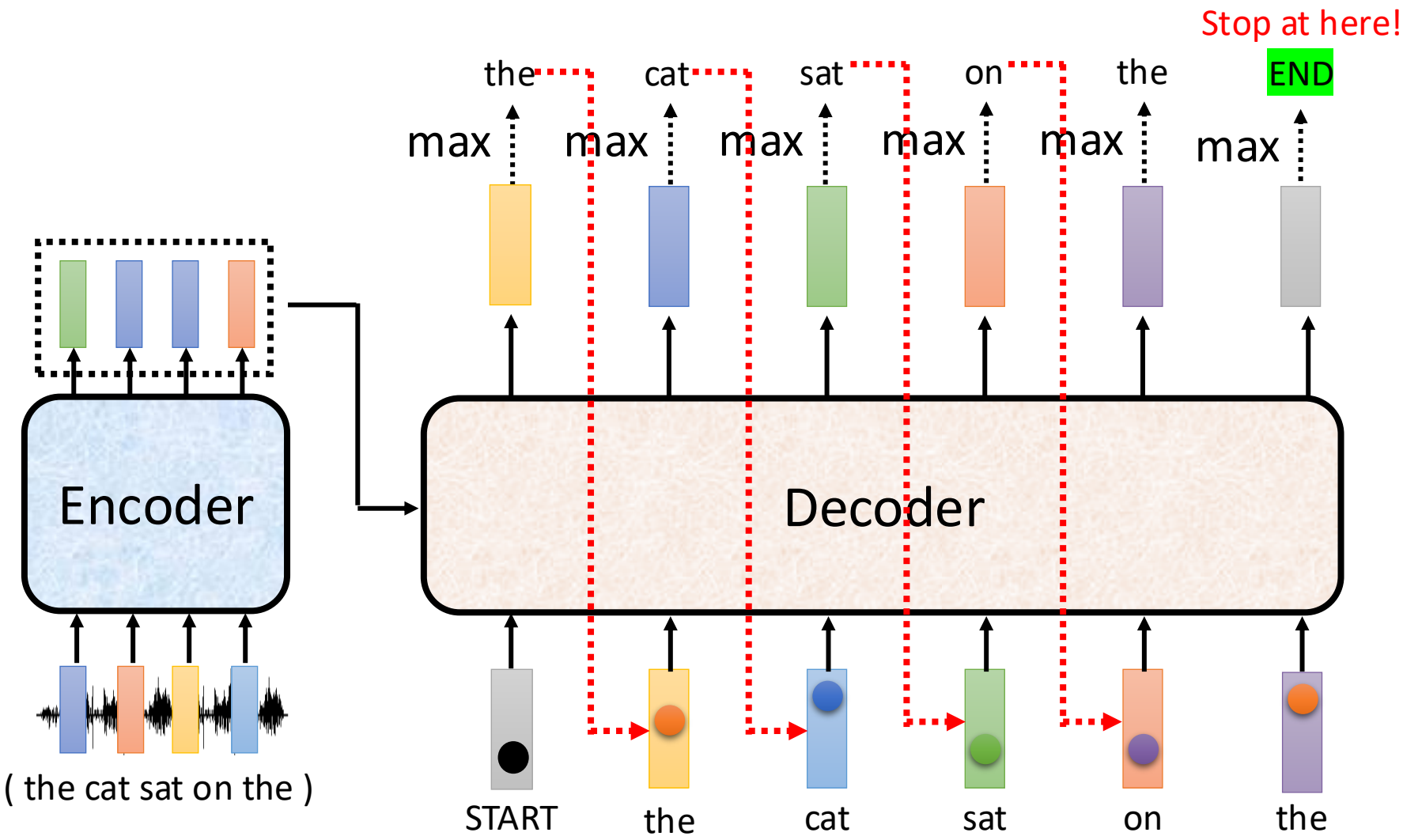
Recap: Original Transformer

https://arxiv.org/abs/1706.03762





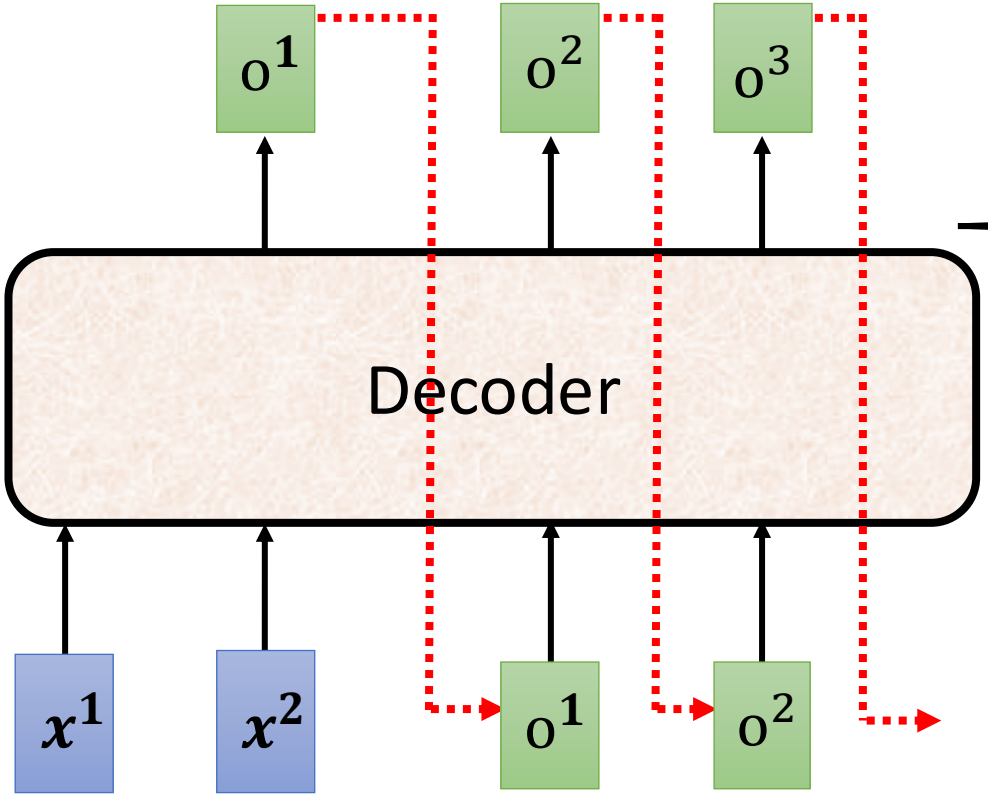
Decoder's Autoregressive Inference

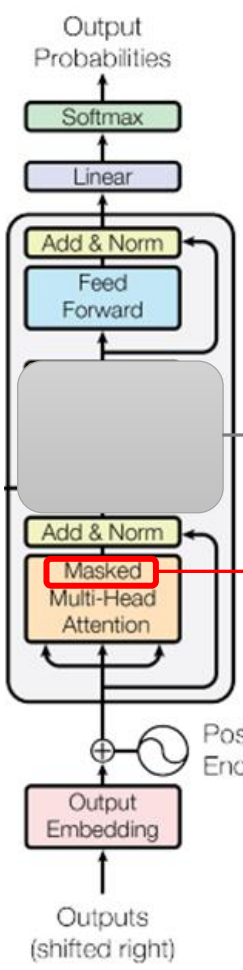


credit: Prof. Hung-yi Lee,

Transformer's Decoder

Decoder-only





no cross-attention (no encoder)

masked self-attention

GPT: learned position embeddings

credit: Prof. Hung-yi Lee,

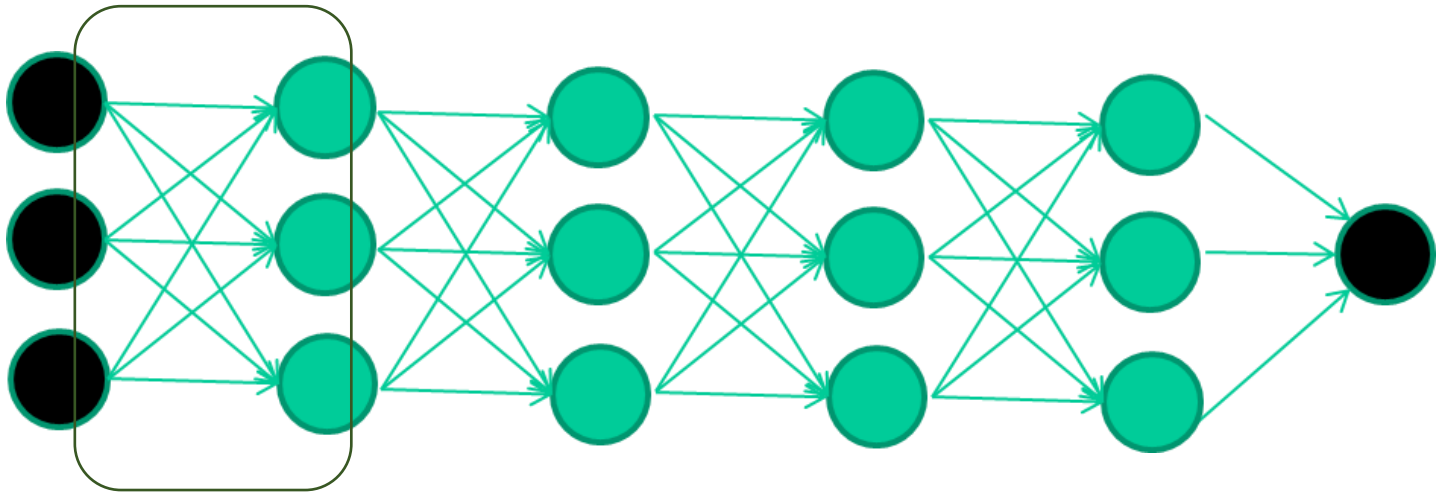
Recap : Transformer architecture

- Background (word vectors, attention, bag-of-words)
- Self-attention Basics (Q, K, V, scaling, attention map)
- Transformer Block Basics (self-attn + heads, positions, residual + norm)
- Masks & Model Families (encoder-only, encoder-decoder, decoder-only)

Roadmap : Today

- **Training stages (encoder-only, decoder-only)**
- Pretraining tactics (tokens, next-token loss, Adam)
- Pretrained GPT models (GPT-1/2/3)
- Post-training (SFT, aligning a GPT: RLHF, DPO, RLVR)

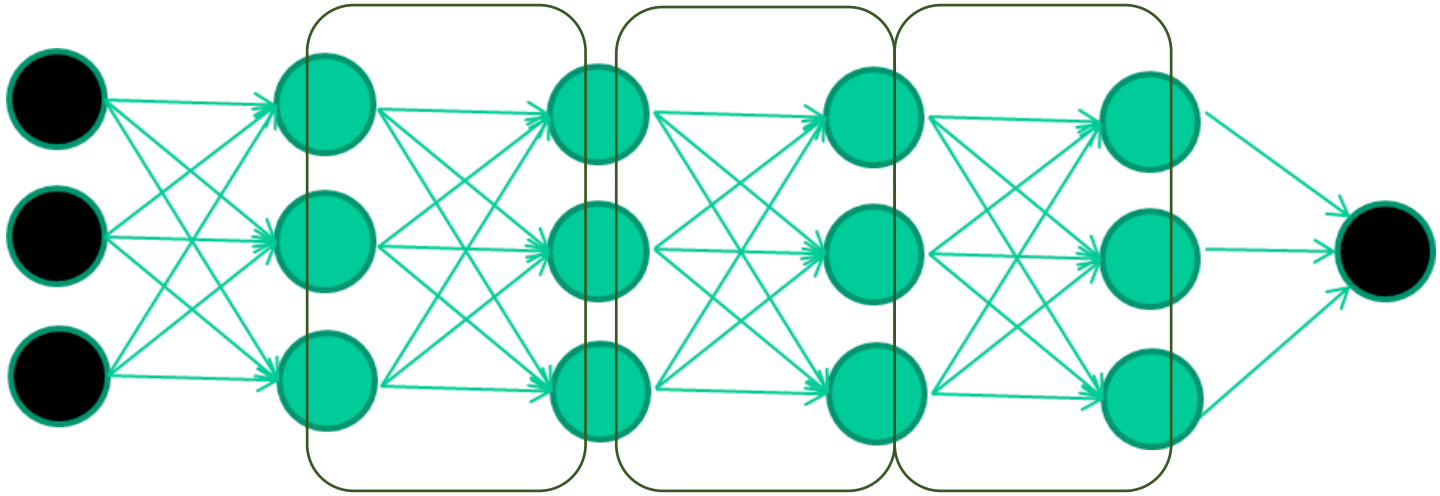
New Ways to Train multi-layer DNNs…



Train **this** layer first

https://www.macs.hw.ac.uk/~dwcorne/Teaching/introdl.ppt

The new way to train multi-layer DNNs…



Each layer can be trained well first with some types of self-supervised loss (e.g. reconstruction loss)

Basically, it is forced to learn good features that describe what comes from the previous layer

https://www.macs.hw.ac.uk/~dwcorne/Teaching/introdl.ppt

Two Main: Unsupervised Pre-training vs. SFT

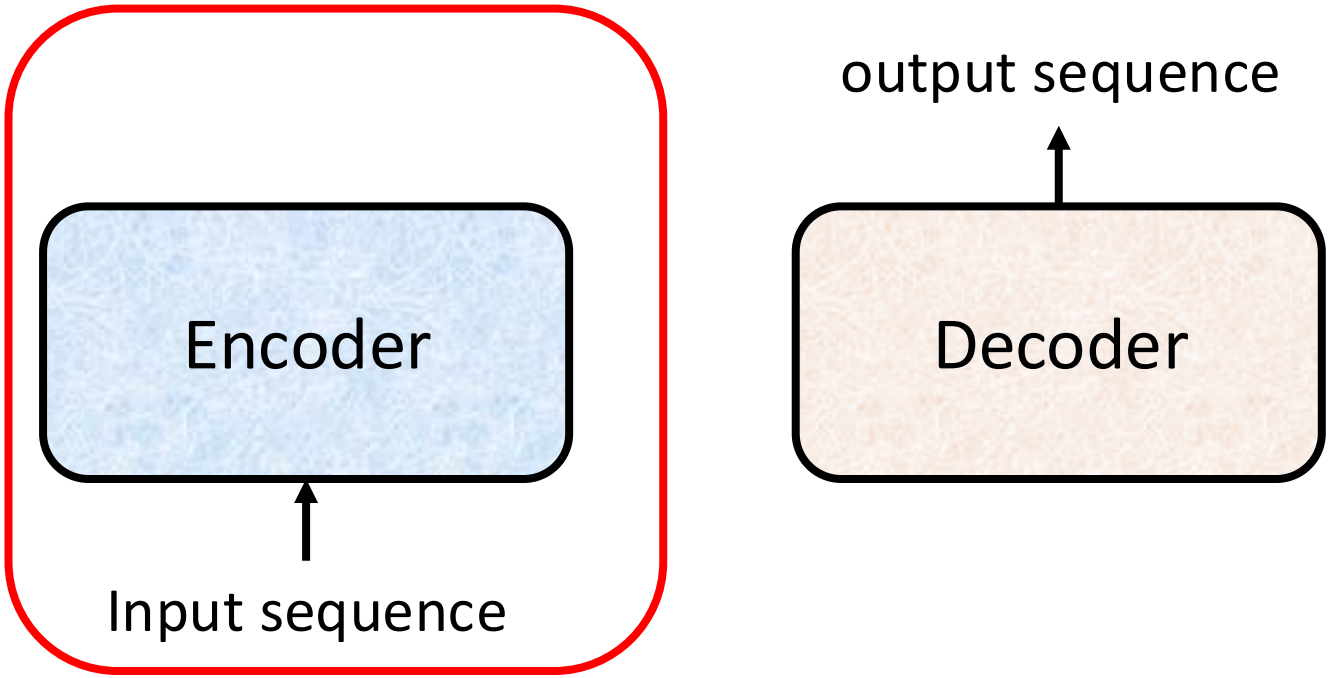
1. Unsupervised Pre-training
   - Use **unlabeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning (SFT)
   - Use **labeled** data to train the DNN that finished by the above pretraining
   - Refine the features by back-propagation so that they become tuned to the end-task

Pre-Training of DNN layer-layer Benefits:

- Layer-wise pre-training can help with:
  - Learn features at different levels of abstraction
  - DNN Model Training can be tricky due to Nonconvexity
  - DNN Model Training can be tricky due to Vanishing gradients

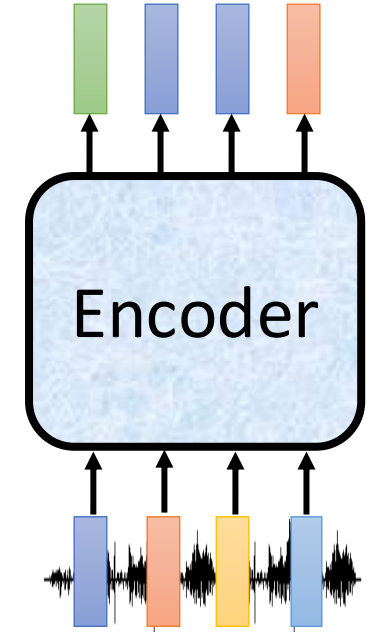
- Many different forms of pre-training ,
  - Mostly on unlabeled data, like web text
  - for instance,
    - Word embedding layers via skip-gram …
    - Pretrained GPT models via next-token LM training
    - Pretrained BERT models via masked LM training

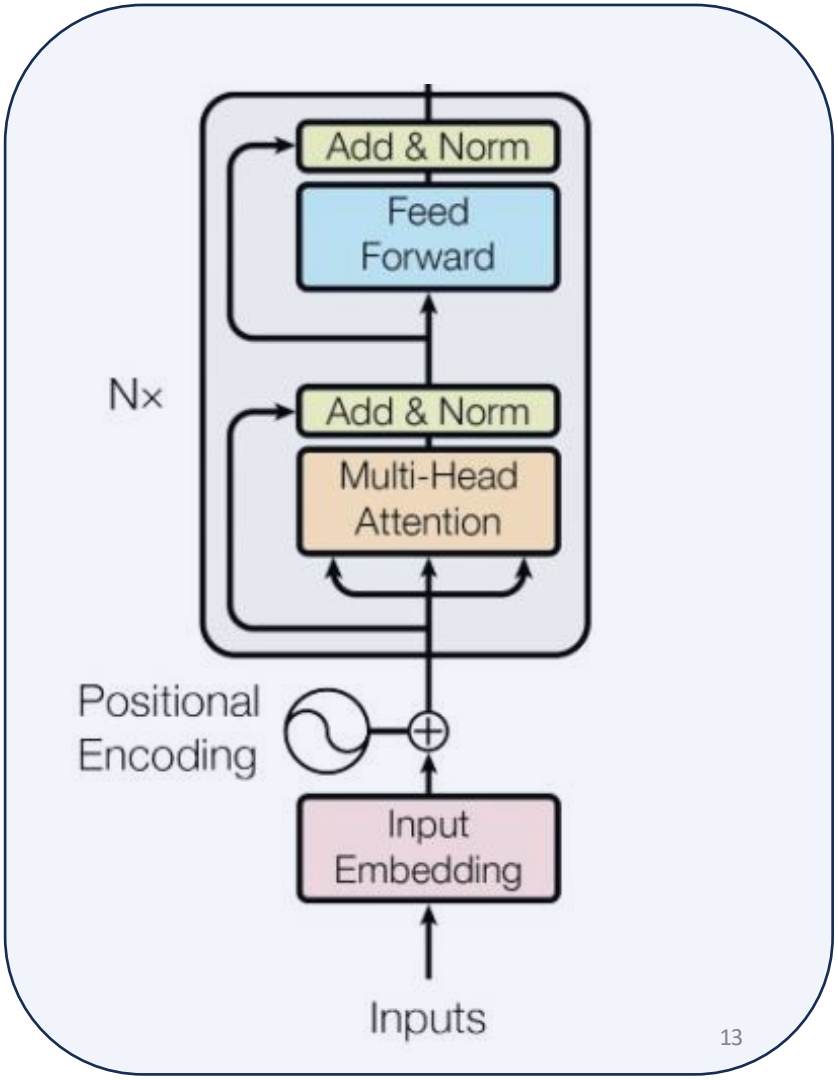
Training



Transformer's ***Encoder Only, e.g., BERT***

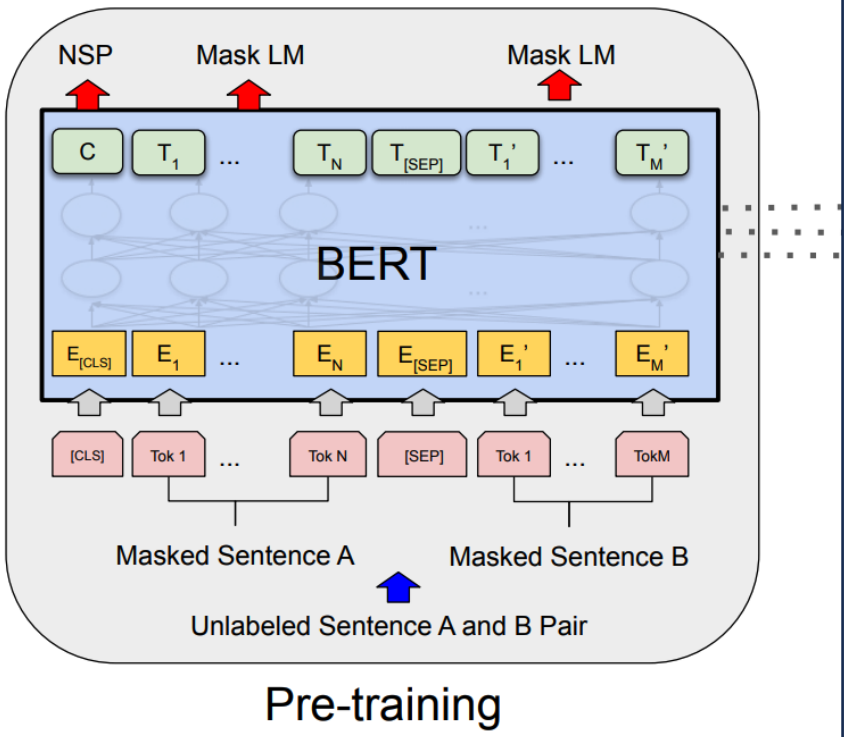
Recap:





BERT Model Family

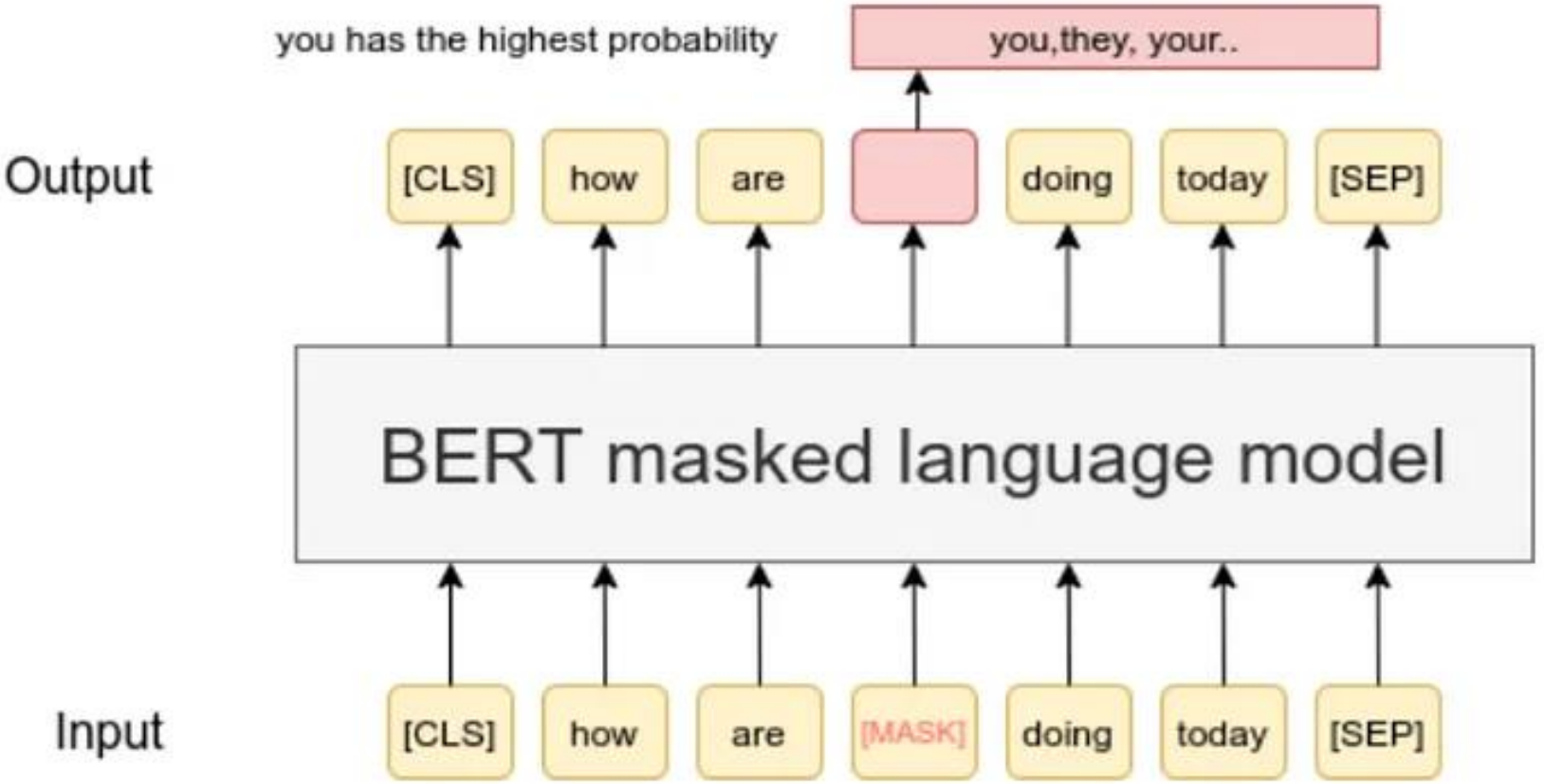
- 1. Pre-train Intuition: representation should be robust to the introduction of noise
  - Masked Language Model (MLM)
  - [MASK]: a unique token introduced in the training process to mask some tokens
  - Predict masked tokens based on their context information,



Pre-training

<- Using Large amount of unlabeled txt corpus

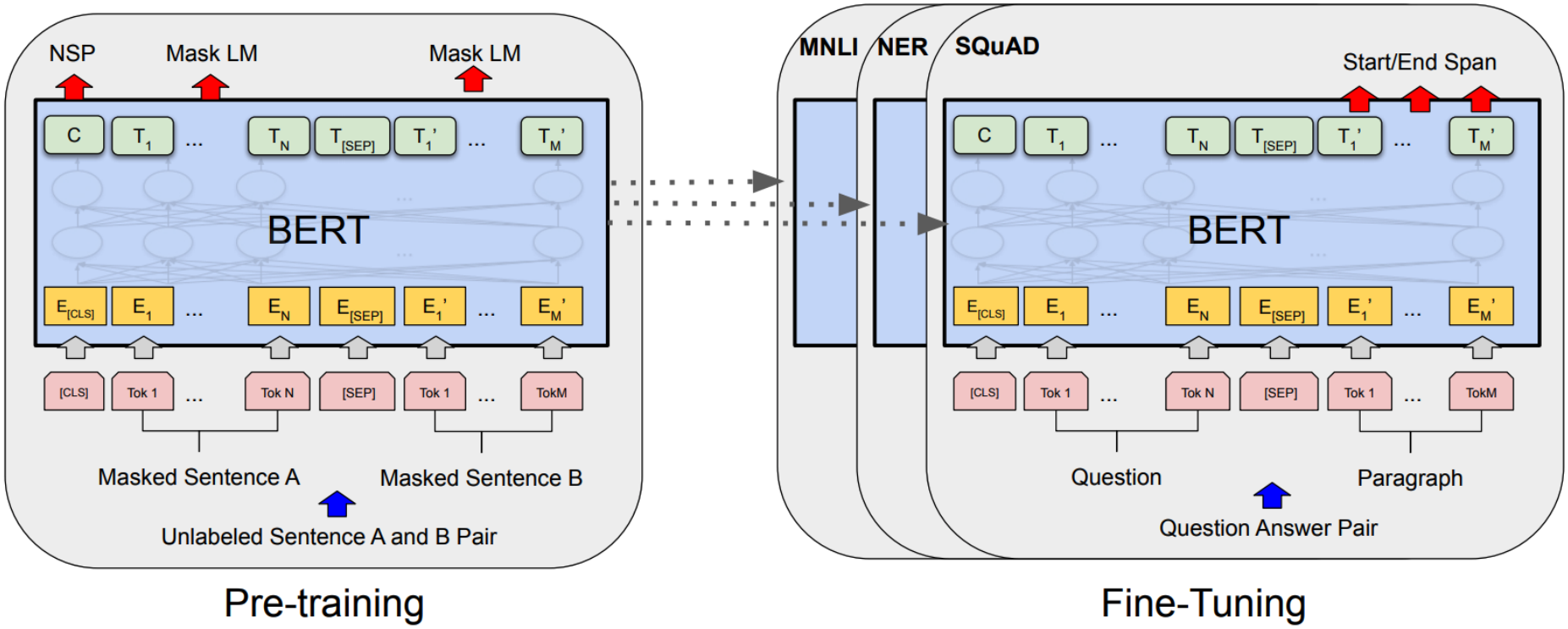
Predict masked tokens based on their left + Right context,



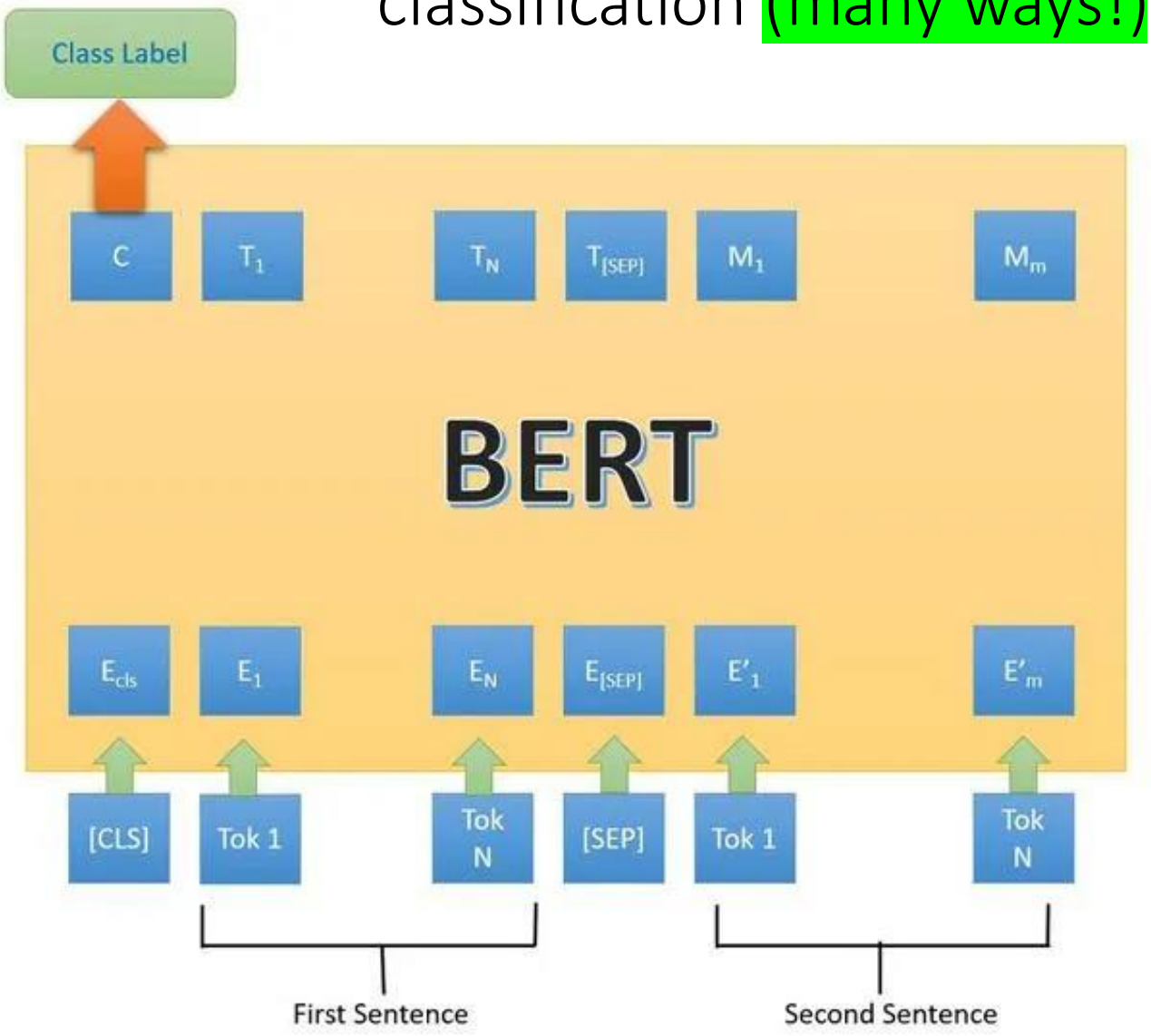
https://editor.analyticsvidhya.com/uploads/13216BERT_MLM.png

BERT Model Family

- 2. Fine Tuning to downstream tasks
  - With Tasks like Questions-Answering Sentence Pairs; Name Entity Recognition Tagging; …..
  - Downstream tasks' label provide fine tuning training signals.



How BERT adapts to sentence classification (many ways!)



https://editor.analyticsvidhya.com/uploads/13216BERT_MLM.png

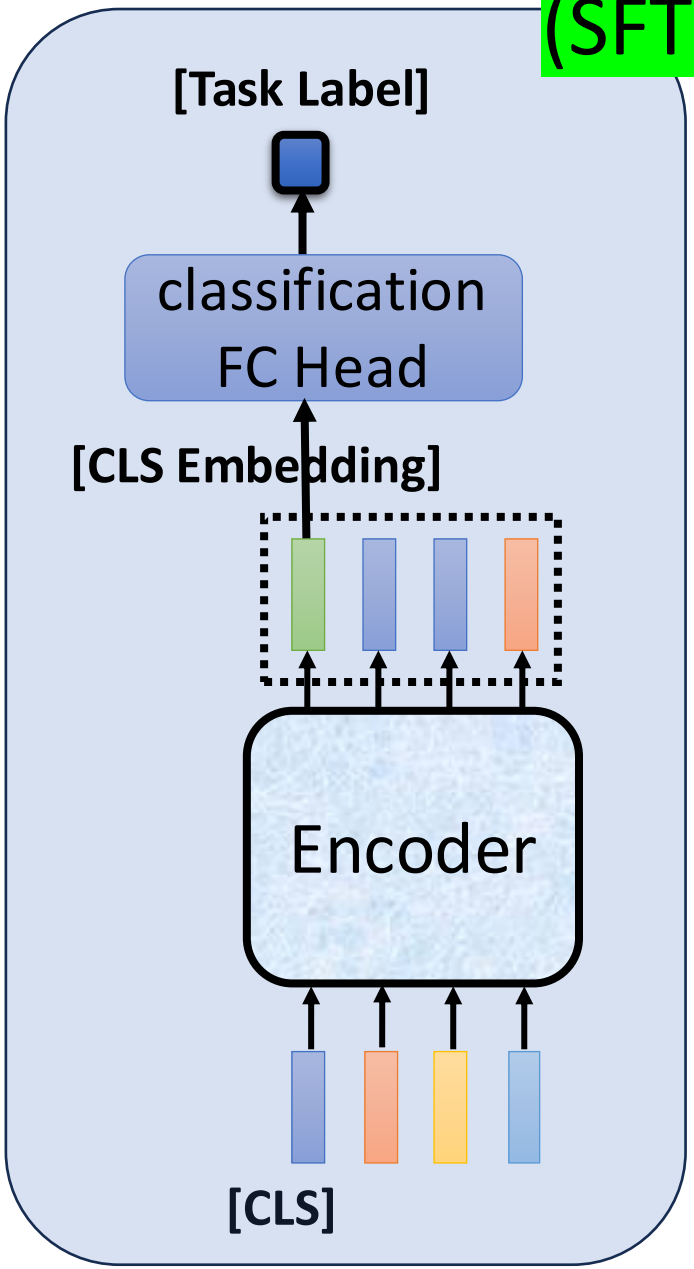
How BERT adapts to sentence classification (many ways)

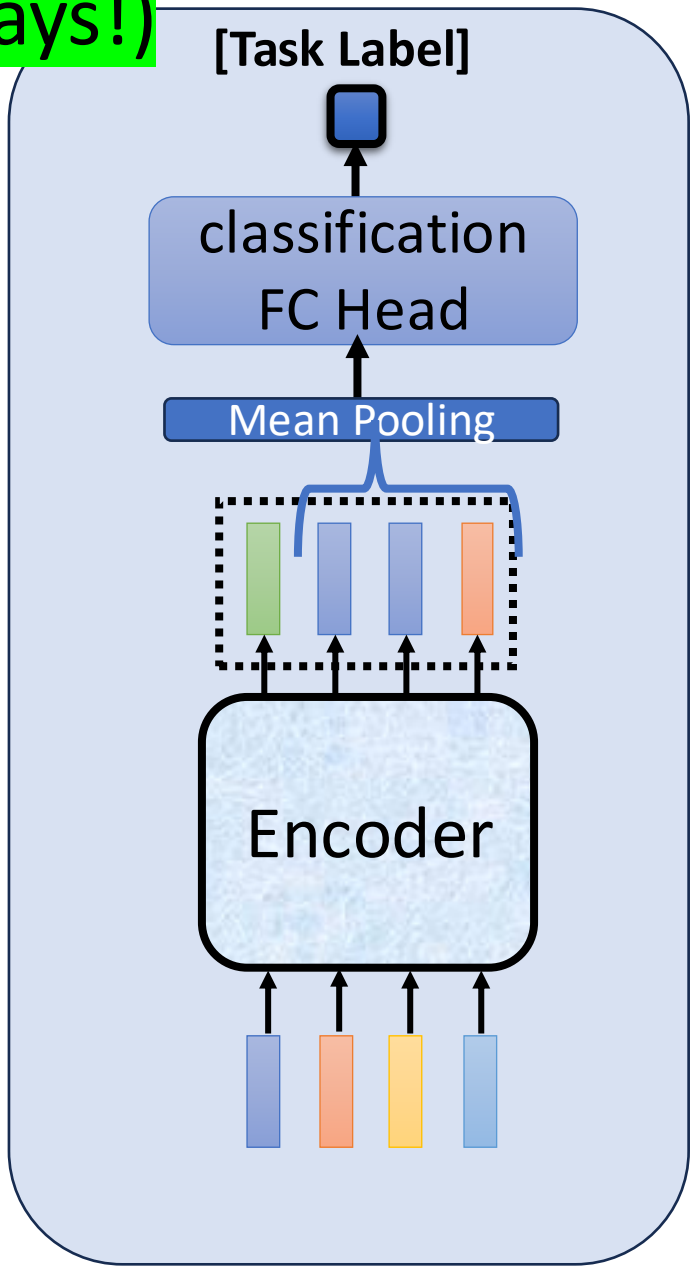
A. First Way: [CLS] token.

1. BERT puts a special `[CLS]` token at the start of every input: `[CLS] the movie was good [SEP]; and`
2. **Use [CLS]'s output vector.** After the encoder layers, the final vector at the [CLS] position represents the whole sequence. With full, unmasked self-attention, that position can attend to every token.
3. **Add a small classifier head.** One linear layer (D → number of classes) and a softmax.
4. **Fine-tune everything.** Train the head and all the BERT weights together with cross-entropy on the labelled task, usually for only a few epochs.

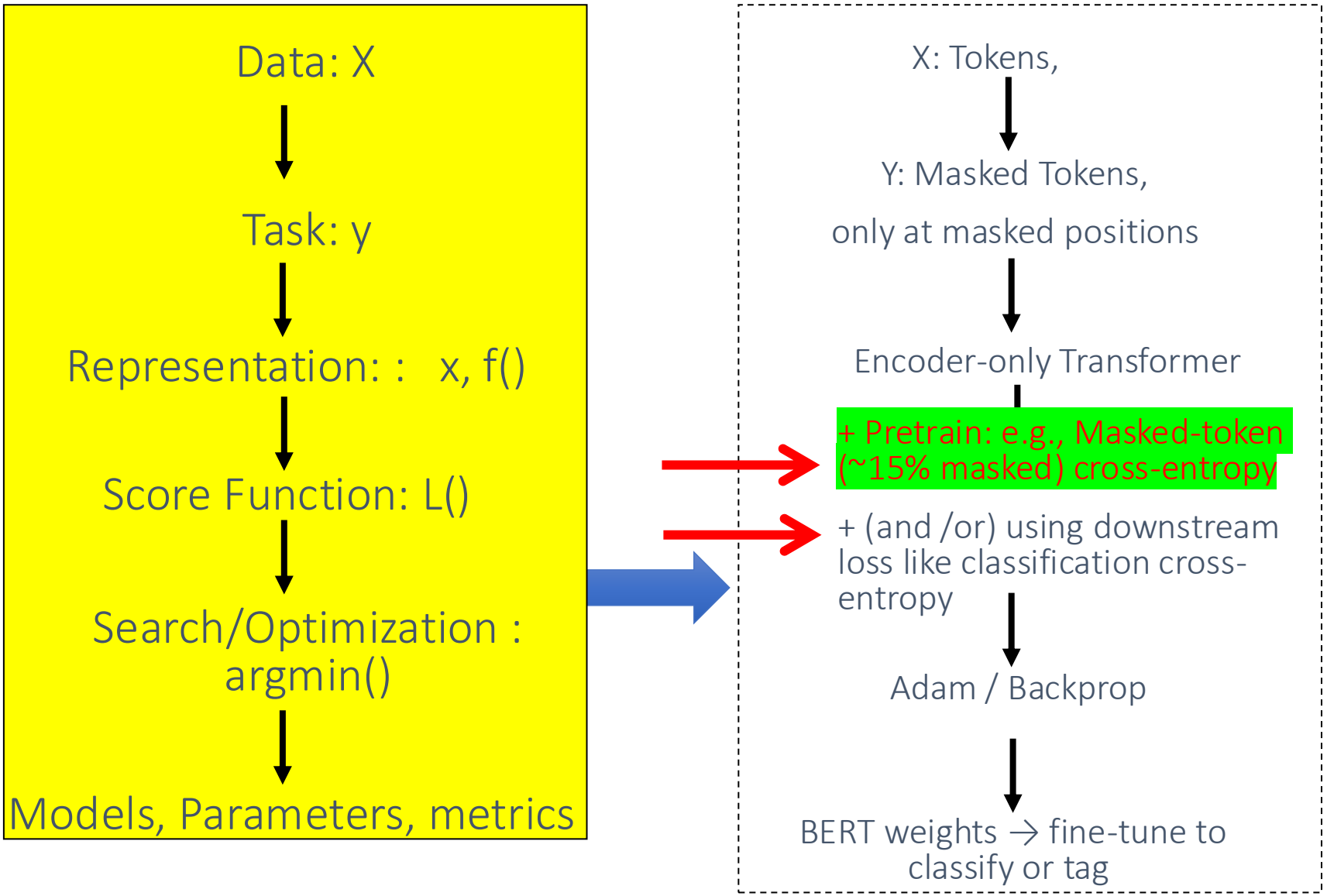
B. Mean pooling instead of [CLS]: average the output vectors over the real tokens, skipping pads. Common for sentence embeddings (Sentence-BERT) and for simple classifiers trained from scratch. With padding, it must be a masked mean: `(h * mask).sum(1) / mask.sum(1)`, which uses the padding mask.

(SFT many ways!)





Summary: Encoder-only Transformer MLM's Training

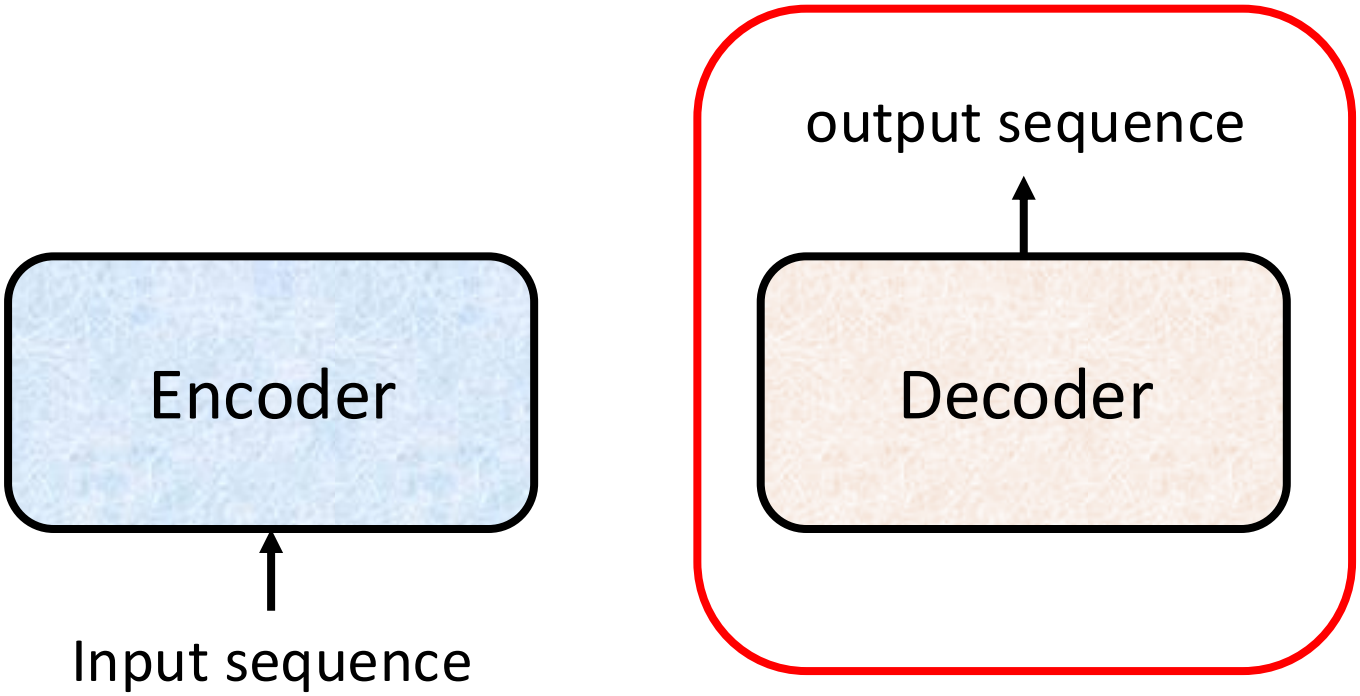


Transformer Encoder Training Summary:

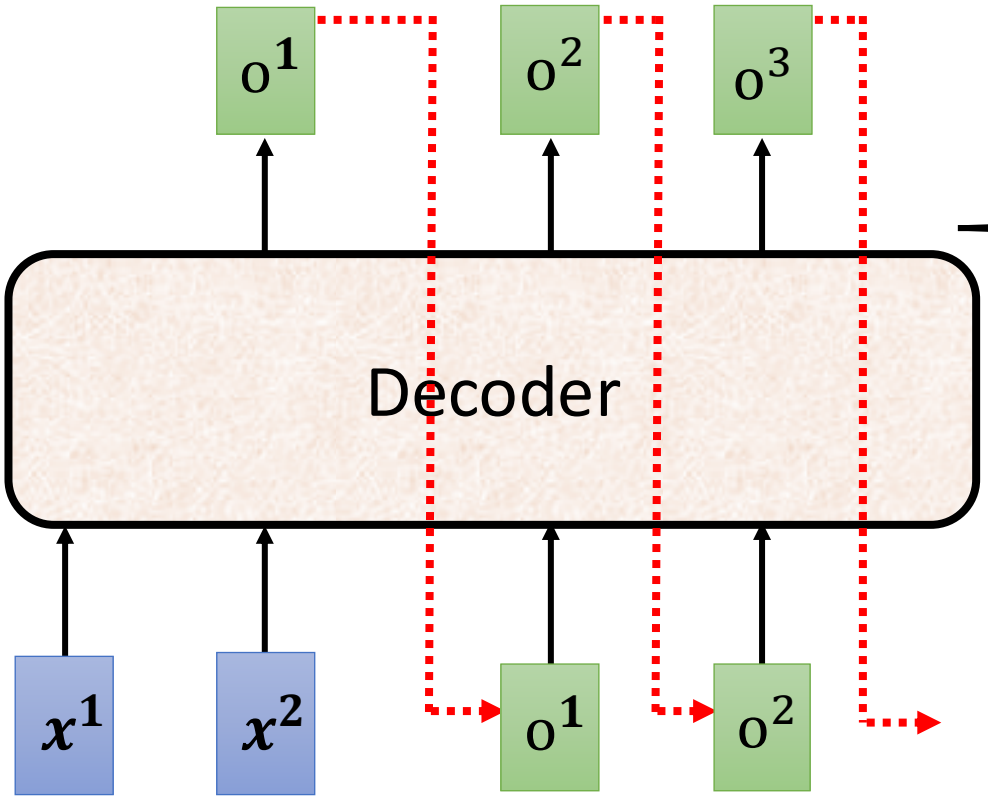
- **Contrast with GPT:** no causal mask, a loss only at masked positions, fine-tuning adds a task head, and inference is one forward pass, with no generation loop.
- Source: Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", NAACL 2019

Row	① Pretraining (masked LM)	② Fine-tuning
X	unlabeled text: BooksCorpus (800M words) + Wikipedia (2.5B); WordPiece tokens; sentence pairs A, B	labeled task data: sentences or pairs + labels
Y	the 15% picked tokens (80% [MASK], 10% random, 10% kept); + IsNext / NotNext (50/50)	task label: a class from [CLS], a tag per token, or an answer span
f()	encoder-only Transformer, bidirectional (no causal mask)	same encoder + a small task head
L()	cross-entropy at the picked positions only (+ next-sentence cross-entropy)	task cross-entropy (classify, tag, or span)
Search	Adam, peak LR 1e-4, 10k warm-up steps + linear decay; 1M steps × 256 sequences	Adam on all weights, LR 2e-5 to 5e-5, batch 16/32, 2–4 epochs
Output	BERT weights: contextual token embeddings; MLM accuracy	task model; accuracy / F1

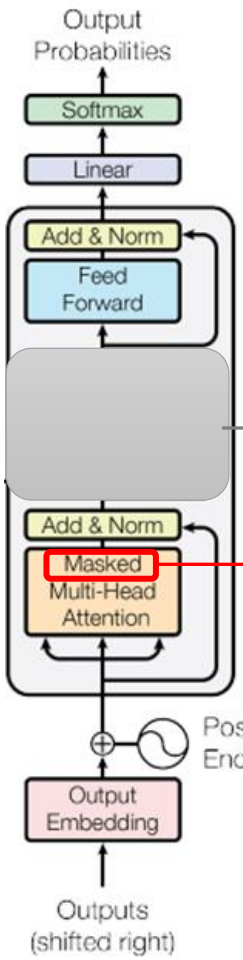
Training



Recap: Decoder-only



Transformer's Decoder



no cross-attention (no encoder)

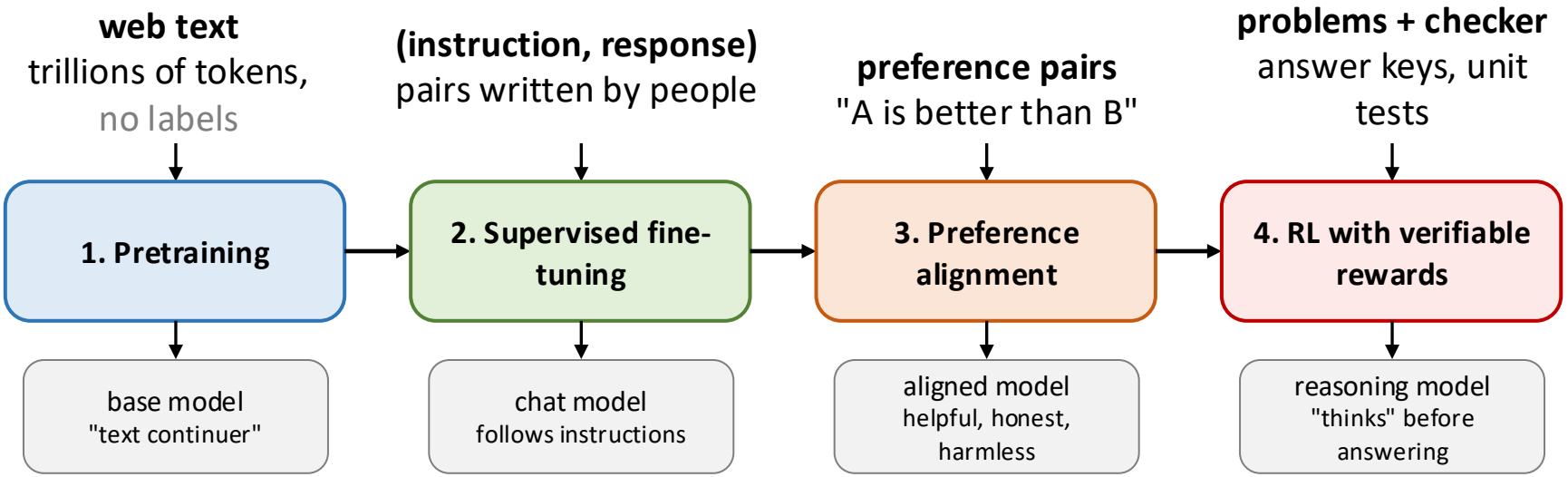
masked self-attention

GPT: learned position embeddings

credit: Prof. Hung-yi Lee,

How a base decoder model becomes a chat/reason agent's model

the training pipeline

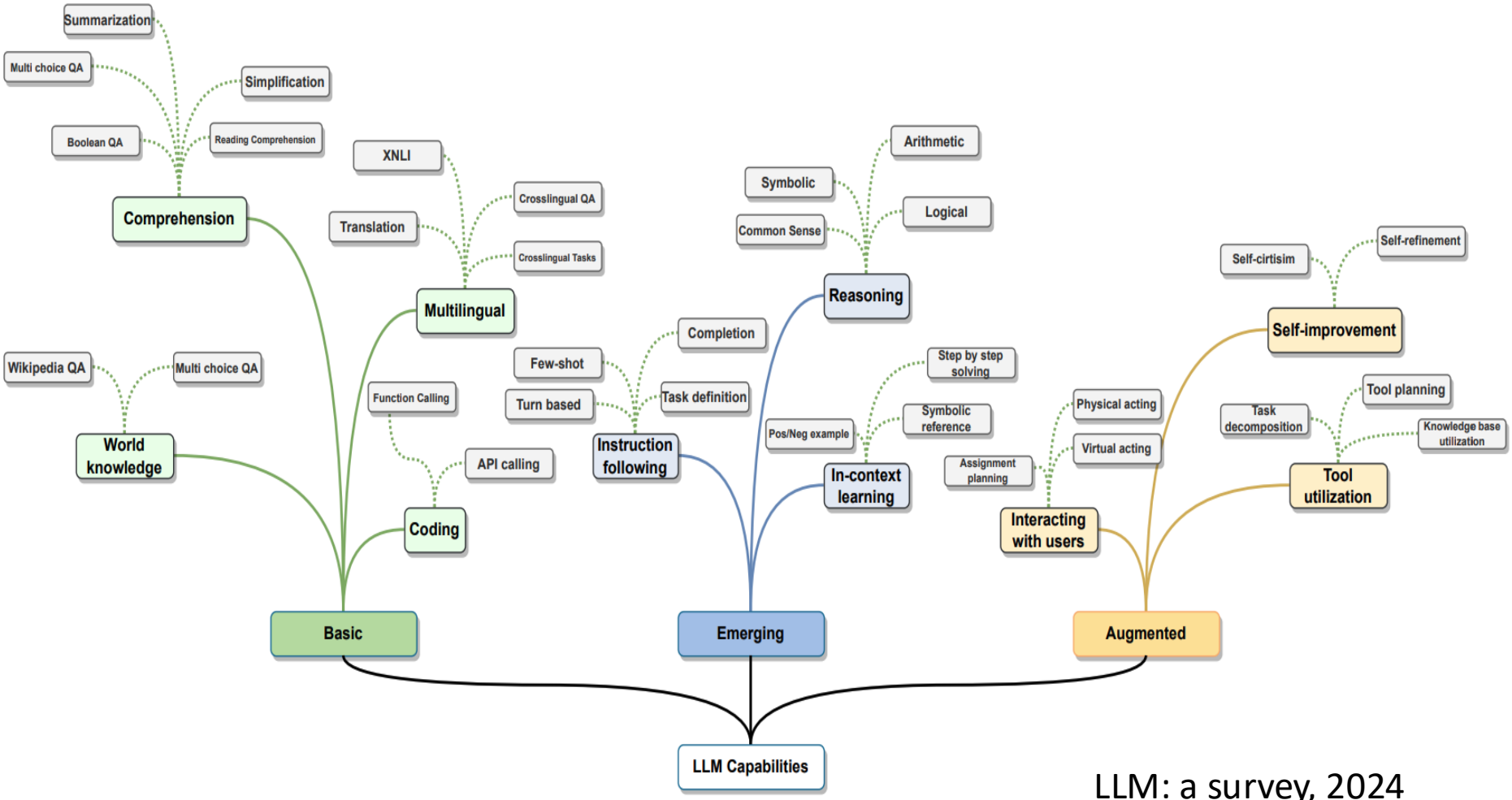


- For each stage we ask the course's four questions: **Data? Loss? Optimization? What is learned?**
- Not a training stage: **prompting / in-context learning**, which changes the *input*, not the weights.

credit: Peng Wang, UVA CS 4774 L15

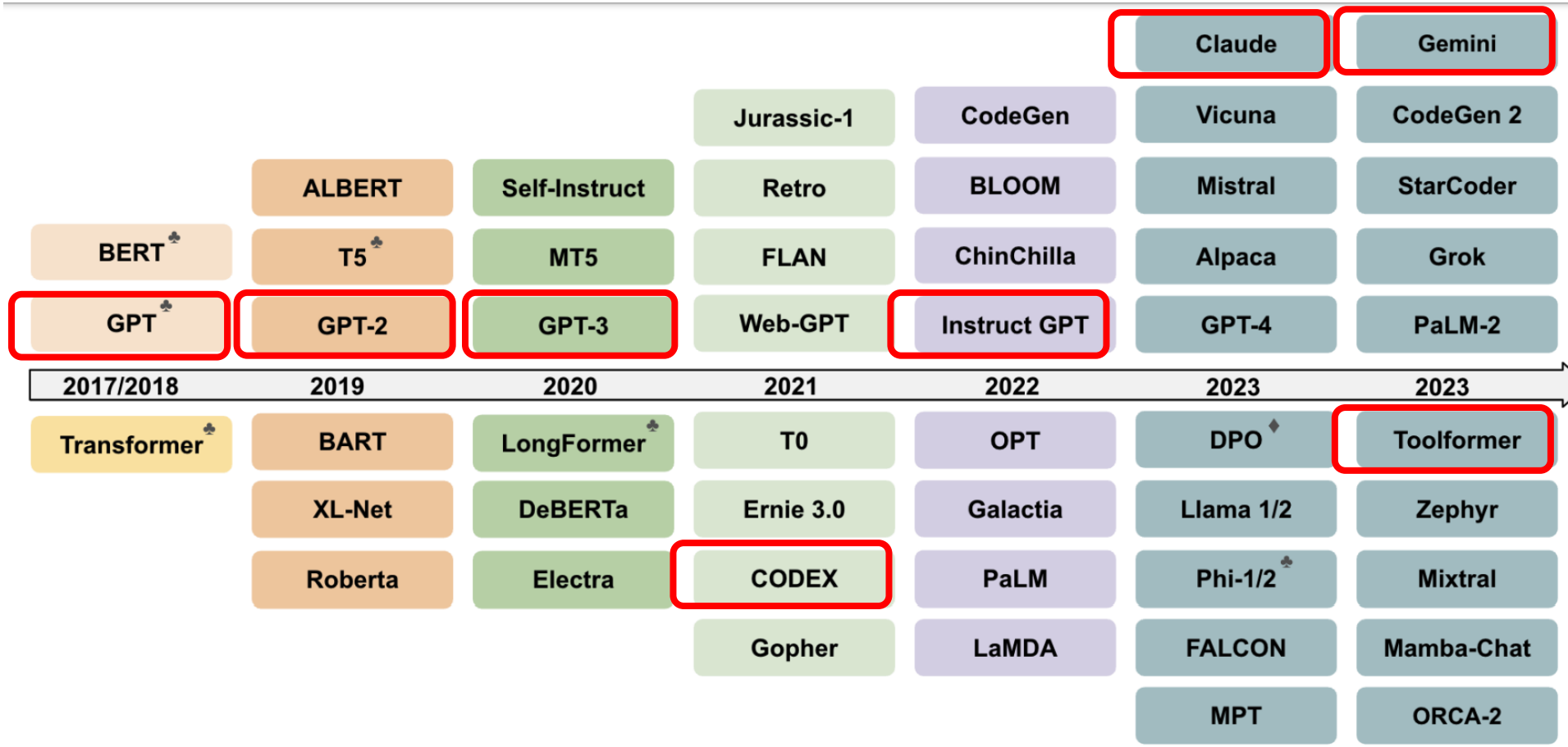
Large Language Models: A Survey

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu
Richard Socher, Xavier Amatriain, Jianfeng Gao

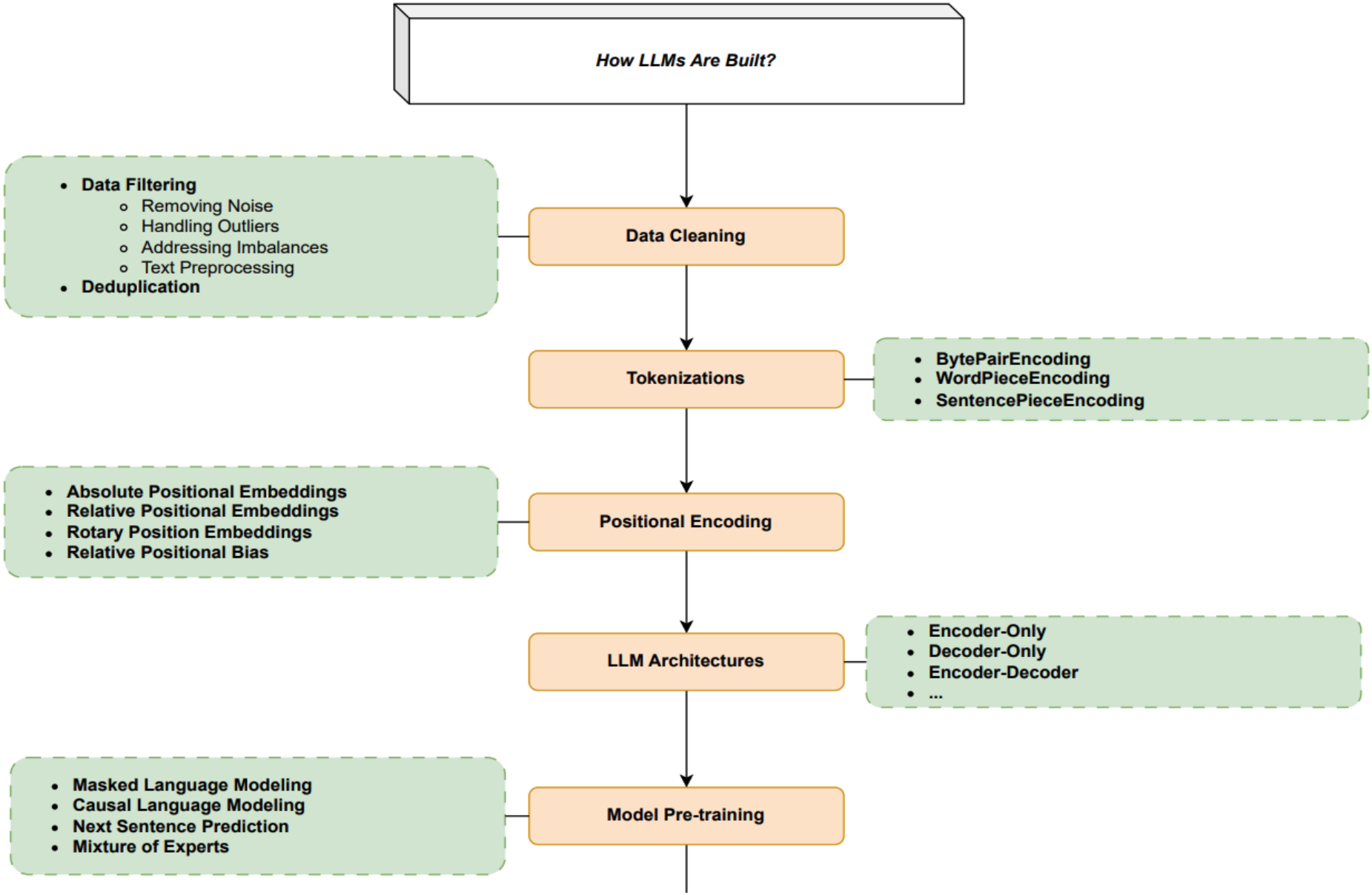


LLM: a survey, 2024

Timeline of representative LLM training/frameworks



LLM: a survey, 2024


How LLMs Are Built?
Data Filtering
Removing Noise
Handling Outliers
Addressing Imbalances
Text Preprocessing
Deduplication
Data Cleaning
Tokenizations
BytePairEncoding
WordPieceEncoding
SentencePieceEncoding
Absolute Positional Embeddings
Relative Positional Embeddings
Rotary Position Embeddings
Relative Positional Bias
Positional Encoding
LLM Architectures
Encoder-Only
Decoder-Only
Encoder-Decoder
...
Masked Language Modeling
Causal Language Modeling
Next Sentence Prediction
Mixture of Experts
Model Pre-training

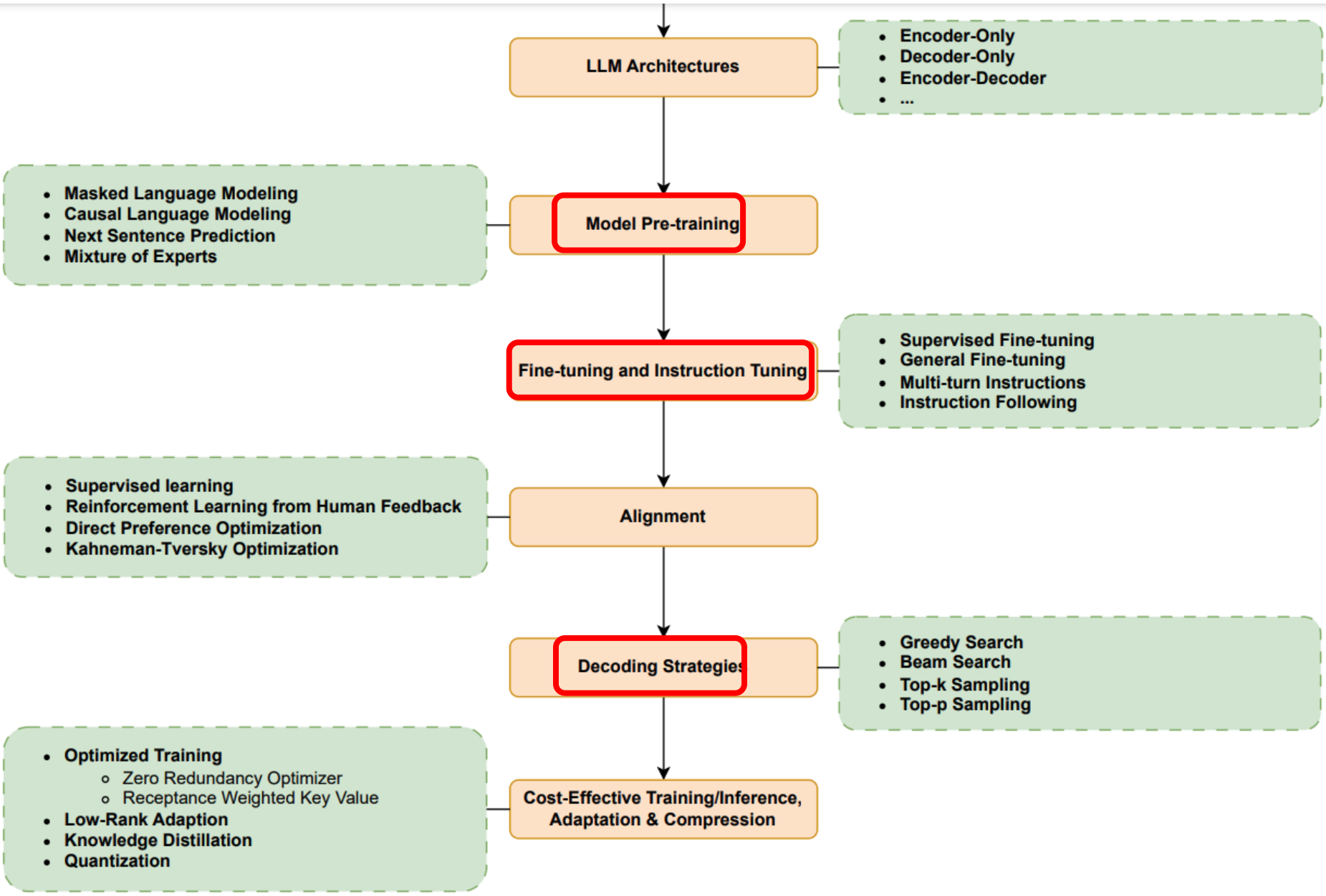


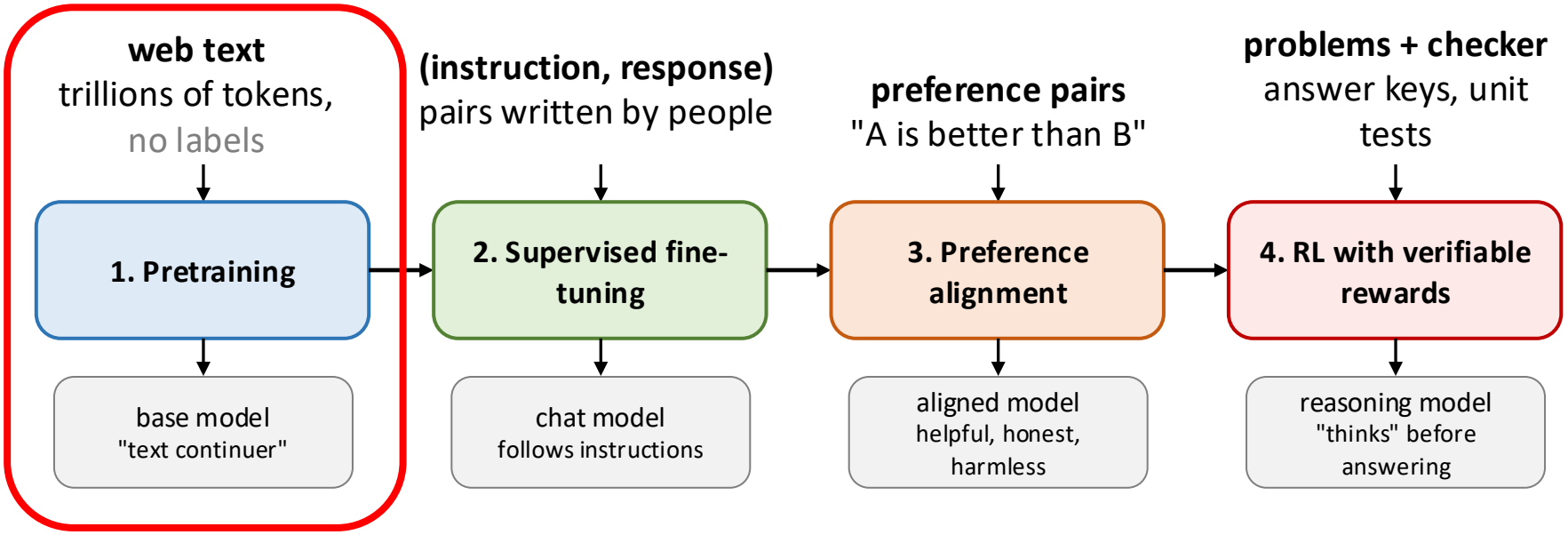
Fig. 25: This figure shows different components of LLMs.

Roadmap : Today

- Training stages (encoder-only, decoder-only)
- **Pretraining tactics (tokens, next-token loss, Adam)**
- Pretrained GPT models (GPT-1/2/3)
- Post-training (SFT, aligning a GPT: RLHF, DPO, RLVR)

How a base decoder model becomes a chat/reason agent's model

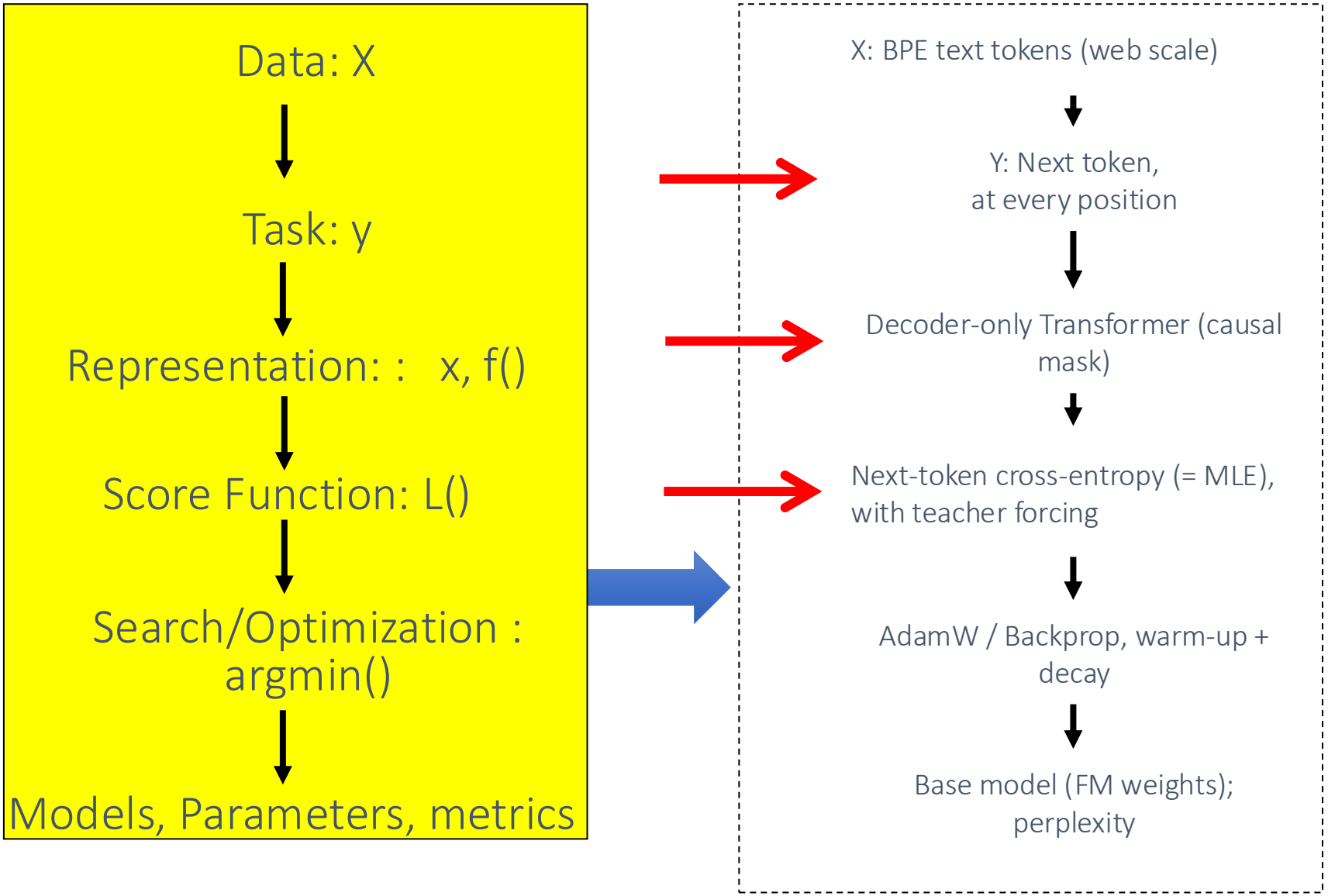
the training pipeline



- For each stage we ask the course's four questions: **Data? Loss? Optimization? What is learned?**

credit: Peng Wang, UVA CS 4774 L15

① Pretraining: the decoder-only

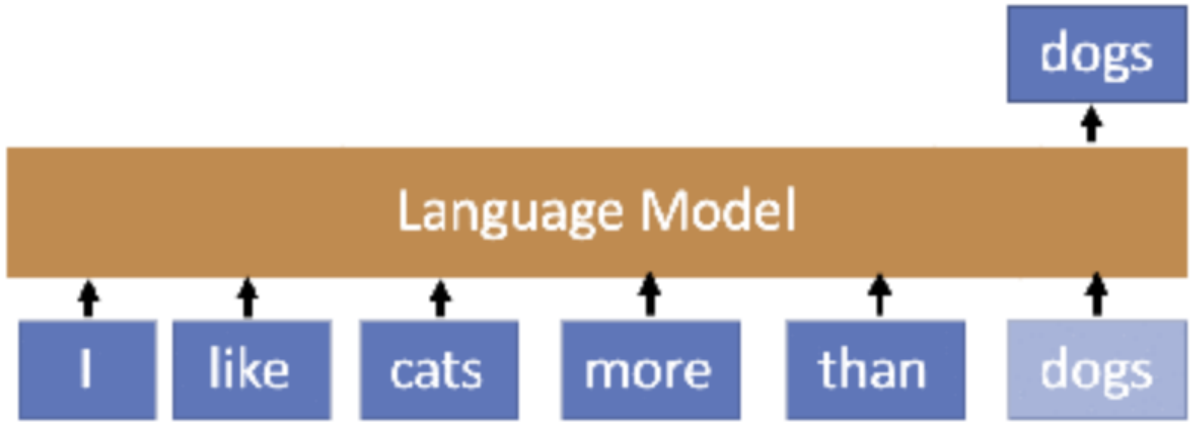


A language model assigns probabilities to text

- A **language model (LM)** assigns a probability to any sequence of tokens $x_1, \ldots, x_T$.
- Chain rule of probability:

$$P(x_1, \ldots, x_T) = \prod_{t=1..T} P(x_t \mid x_{<t})$$

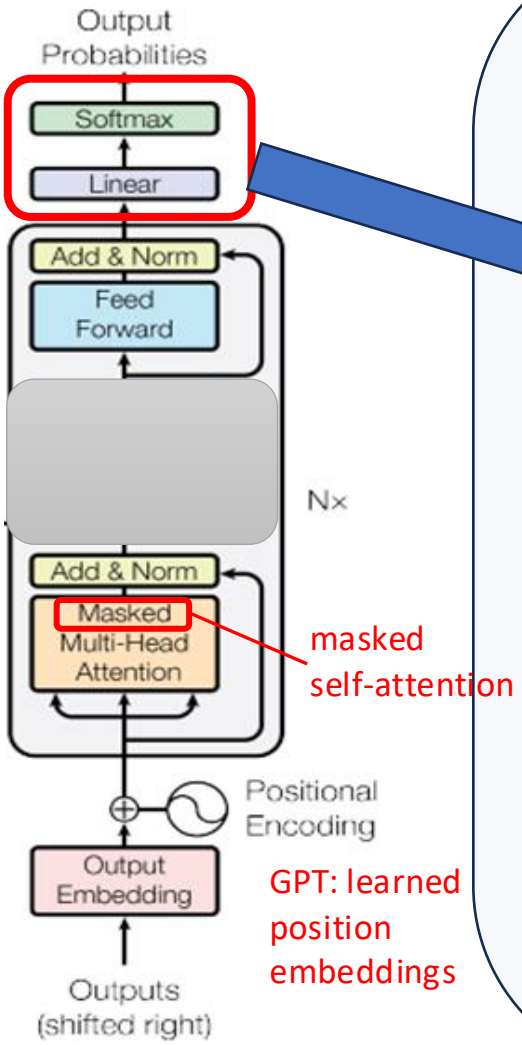
- Example: $P(\text{I eat the apple}) = P(\text{I}) \cdot P(\text{eat} \mid \text{I}) \cdot P(\text{the} \mid \text{I eat}) \cdot P(\text{apple} \mid \text{I eat the})$

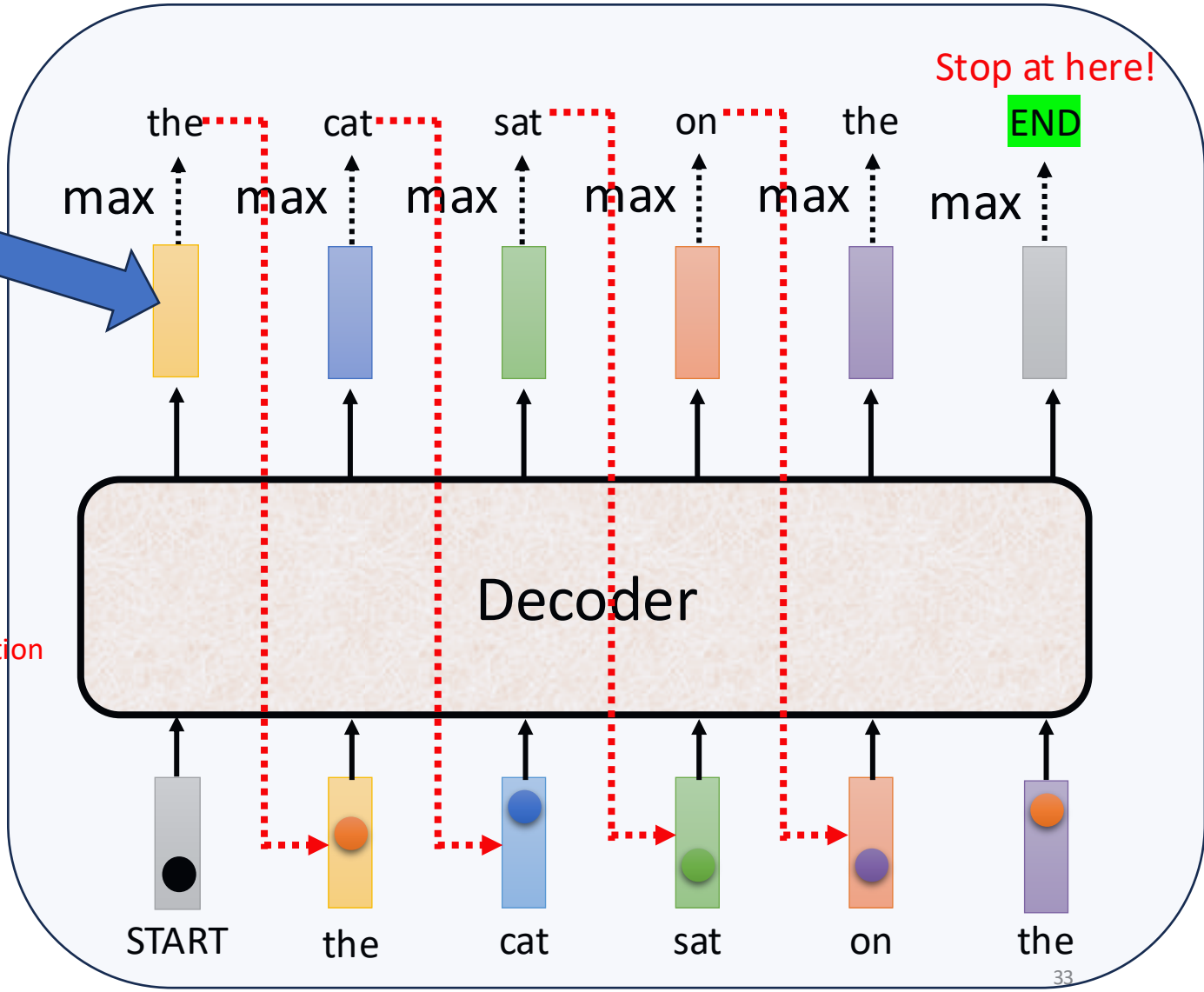


The prediction scheme for a traditional language model. Shaded words are provided as input to the model while unshaded words are masked out.

Decoder's Autoregressive Inference

$$P(x;\theta) = \prod_{n=1}^{N} P(x_n | x_{<n}; \theta)$$



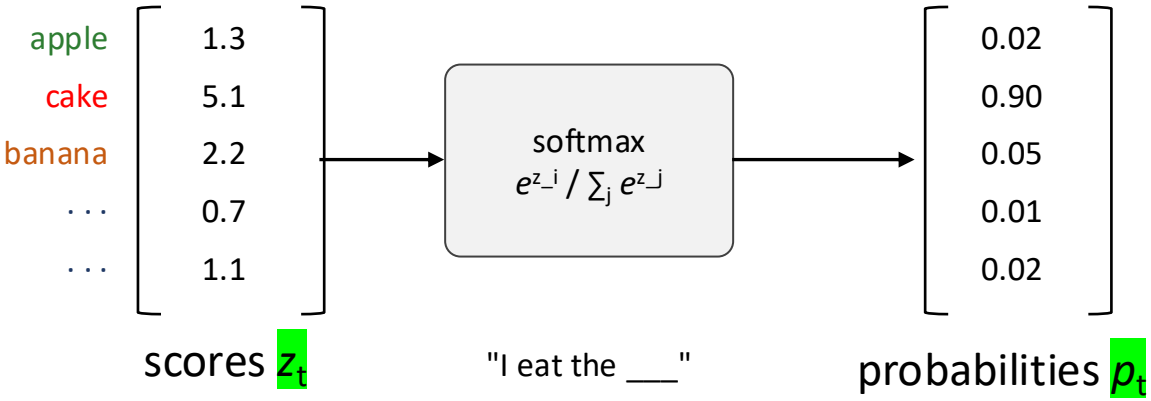


credit: Prof. Hung-yi Lee,

The output layer is multi-class logistic regression

At position t the decoder gives $h_t \in \mathbb{R}^d$. Then

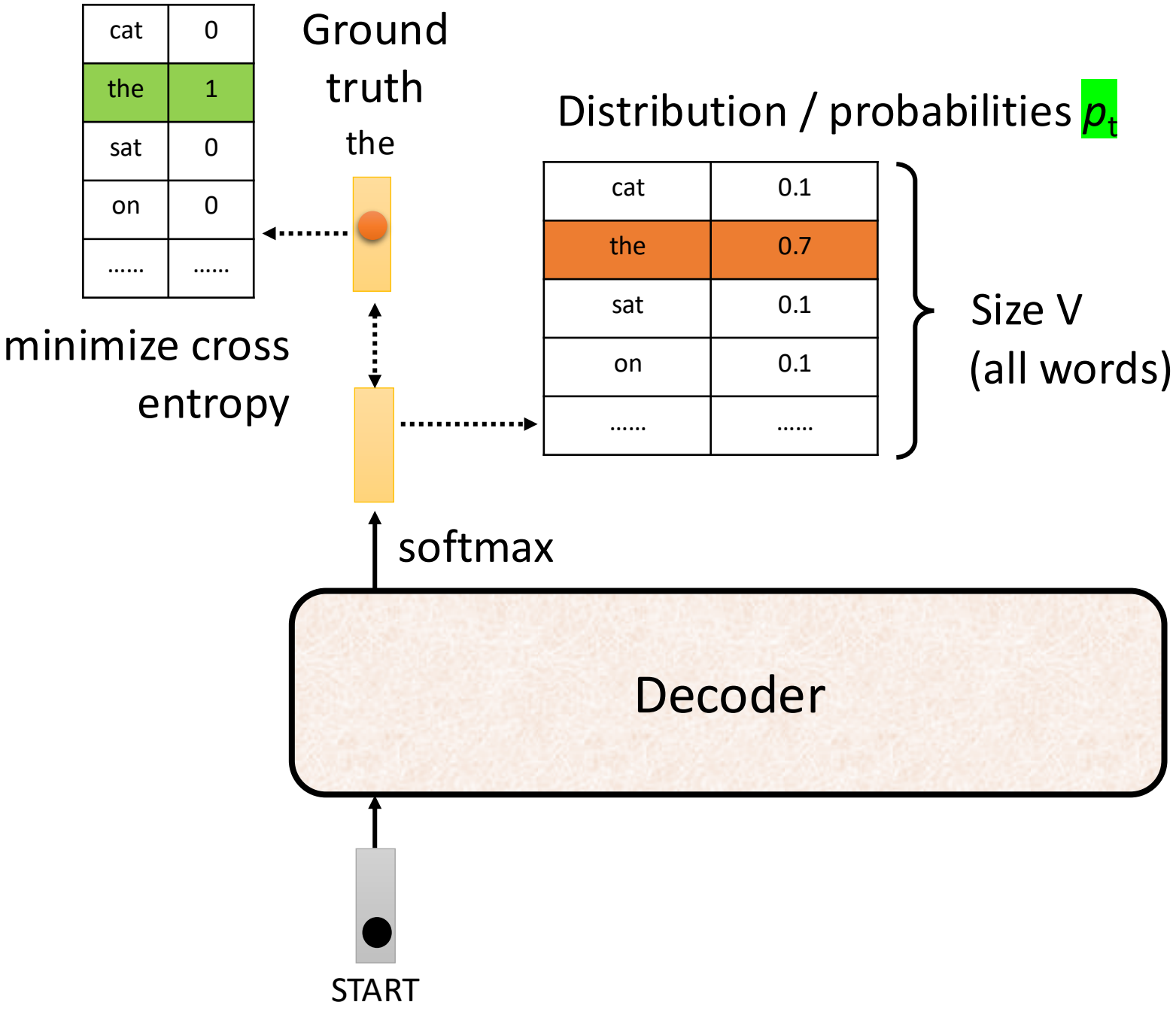
$$z_t = W_{out}\, h_t + b \in \mathbb{R}^{|V|}, \qquad p_t(i) = \exp(z_{t,i}) / \sum_{j=1..|V|} \exp(z_{t,j})$$



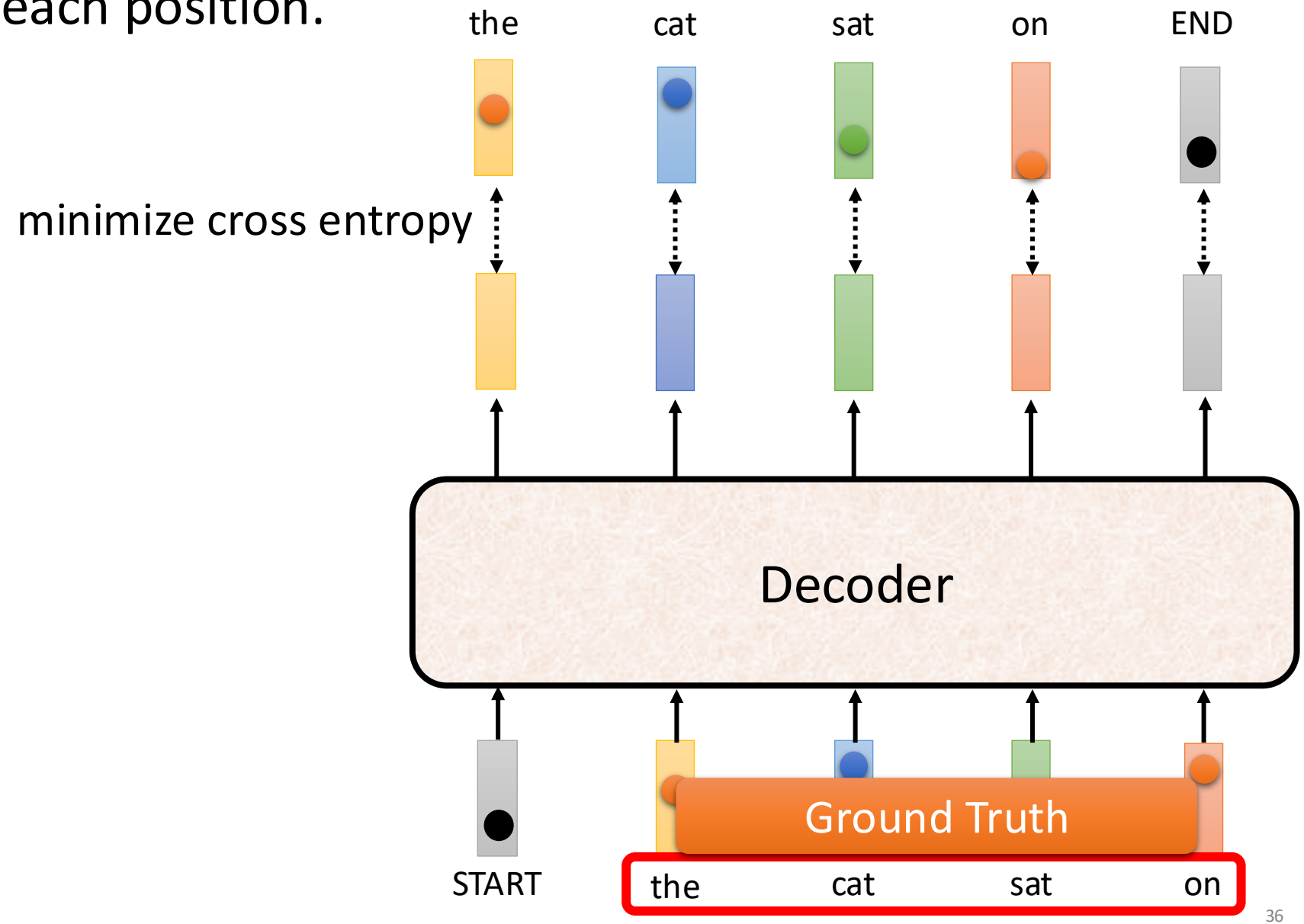
logits: one score per token

You've seen this (logistic regression): $p(y=1 \mid x) = \sigma(w^\top x + b)$ is the 2-class case; softmax is its K-class generalization. Here the features are h_t and $K = |V| \approx 100{,}000$.

credit: Peng Wang, UVA CS 4774 L15


cat 0
the 1
sat 0
on 0
....
Ground truth
the
minimize cross entropy
Distribution / probabilities p_t
cat 0.1
the 0.7
sat 0.1
on 0.1
....
Size V (all words)
softmax
Decoder
START

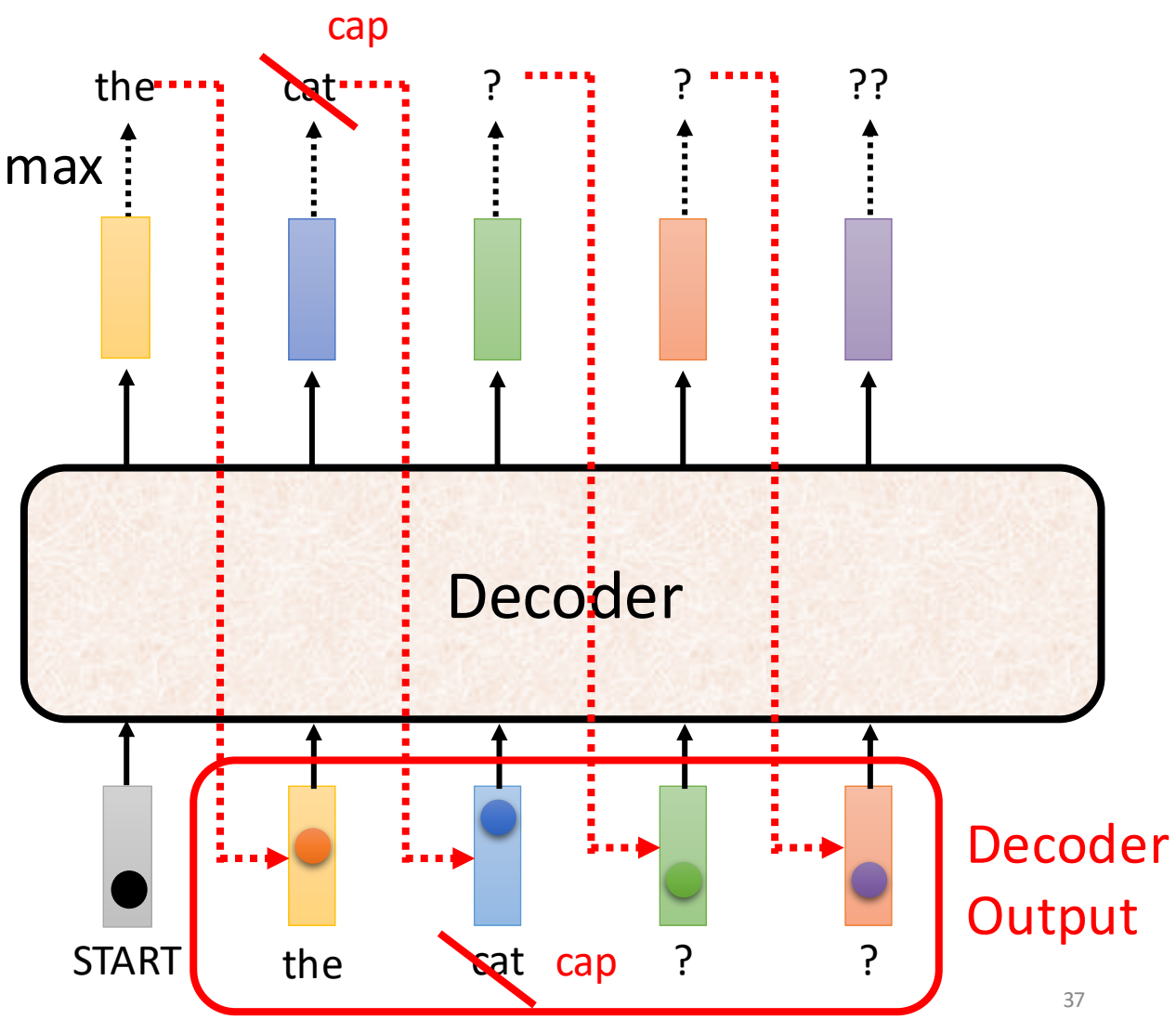
Teacher Forcing: using the ground truth as input for each position.



If without teacher forcing!~

Each step takes everything produced so far and predicts one more token

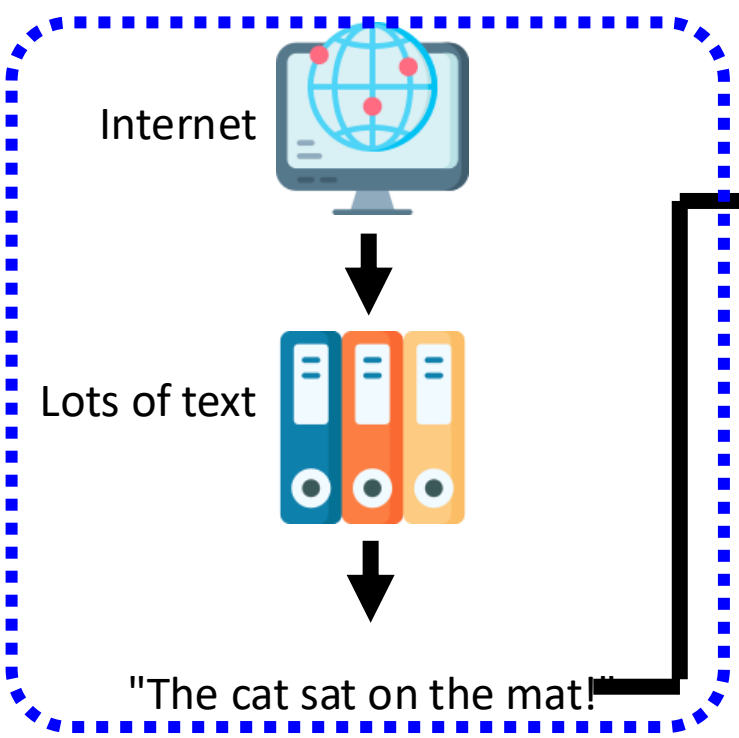
➔ Decoder's Autoregressive (Forward Step!)



credit: Prof. Hung-yi Lee,

Summary: Pretraining Decoder LLM

Next-Token Prediction can use Any Text Data!!!



Input: The	Output: cat
Input: The cat	Output: sat
Input: The cat sat	Output: on
Input: The cat sat on	Output: the



Training data

As much as the size of web

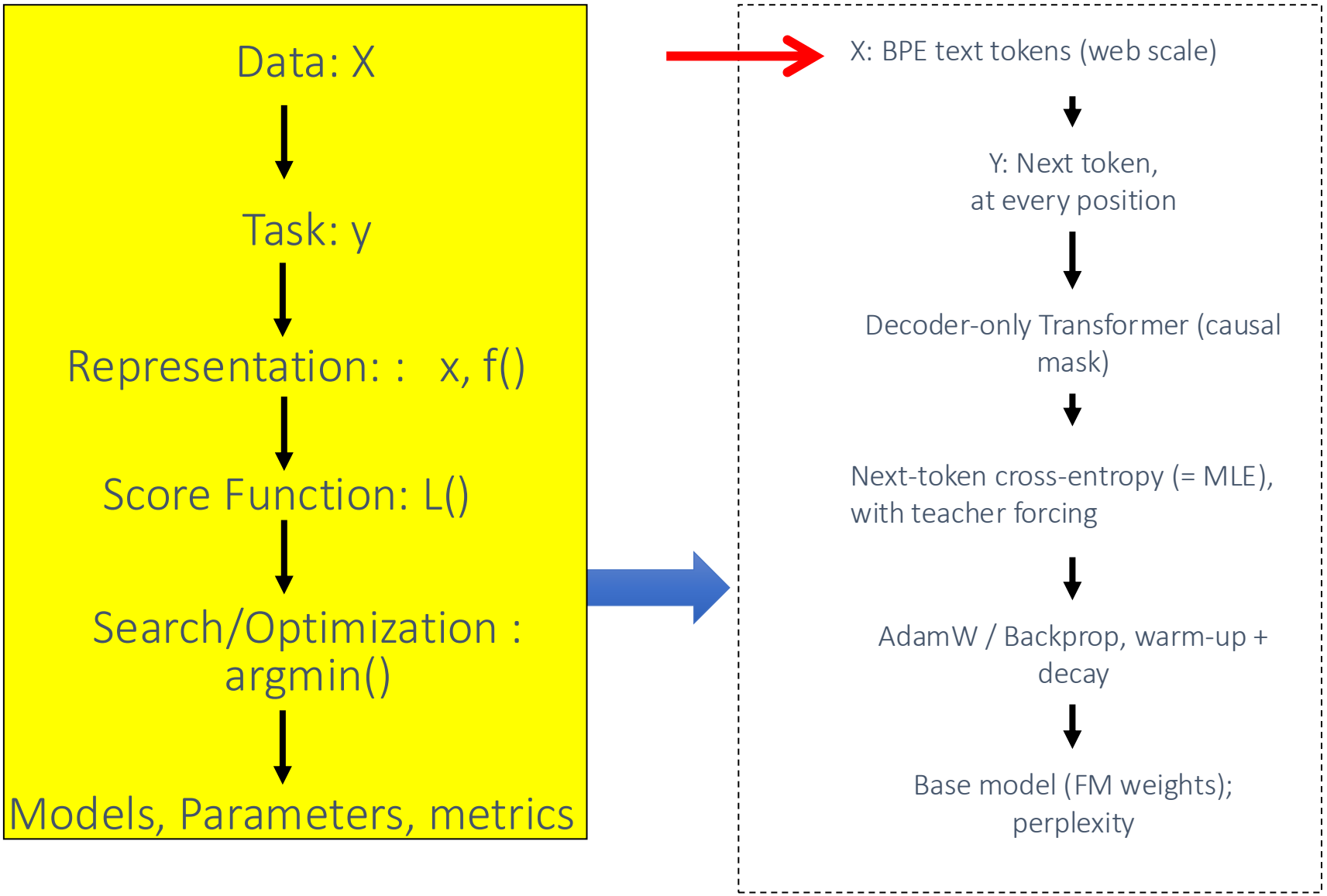
Params

Little human effort

Self-supervised Learning

credit: Prof. Hung-yi Lee,

① Pretraining: the decoder-only

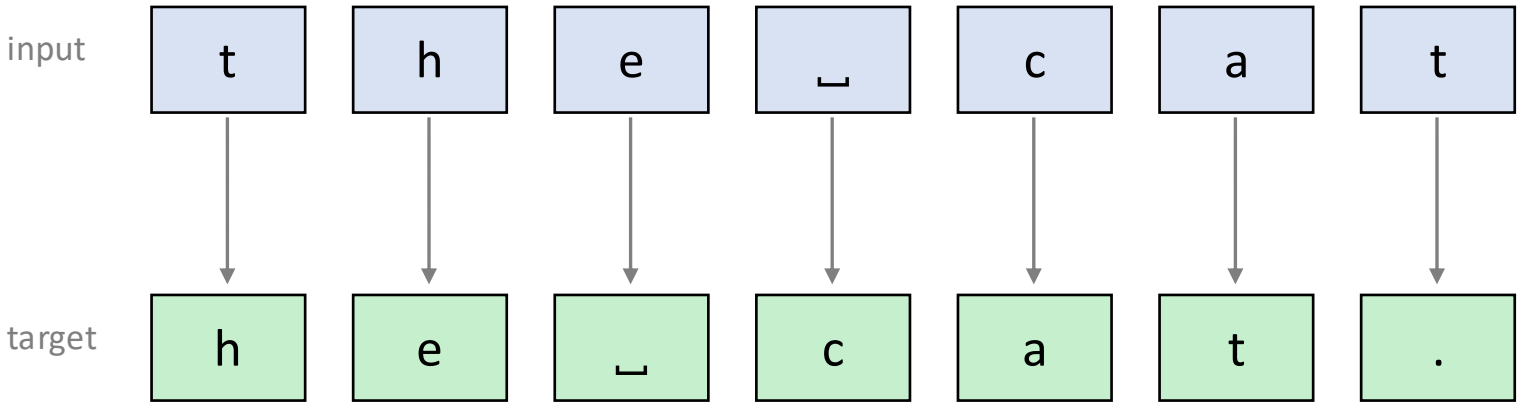


Tokens: characters, words, or subwords

	Characters	**Words**	**Subwords (BPE)**
“unhappiness”	u n h a p p i n e s s (11)	unhappiness (1) or <unk>	un · happi · ness (3)
Vocabulary size	tiny (tens)	huge (100K+)	≈ 32K – 200K
Unknown words	never	often	never (falls back to bytes)
Sequence length	long	short	medium

- **Byte-pair encoding (BPE)**: start from characters, repeatedly merge the most frequent adjacent pair
  - l o w e r → lo w e r → low e r → low er → …
- GPT-2/3/4, Llama and Claude all use subword tokenizers

Summary: predict the next token



- Targets = inputs shifted left by one; with the causal mask, position t sees inputs 0 … t only
- Loss = average cross-entropy over all real positions (<pad> ignored)
- Teacher forcing: the model always reads the **true** previous characters during training
- Thanks to the causal mask, all T predictions train in parallel in one forward pass
- Self-supervised: raw text is its own label

Why does next-word prediction teach so much?

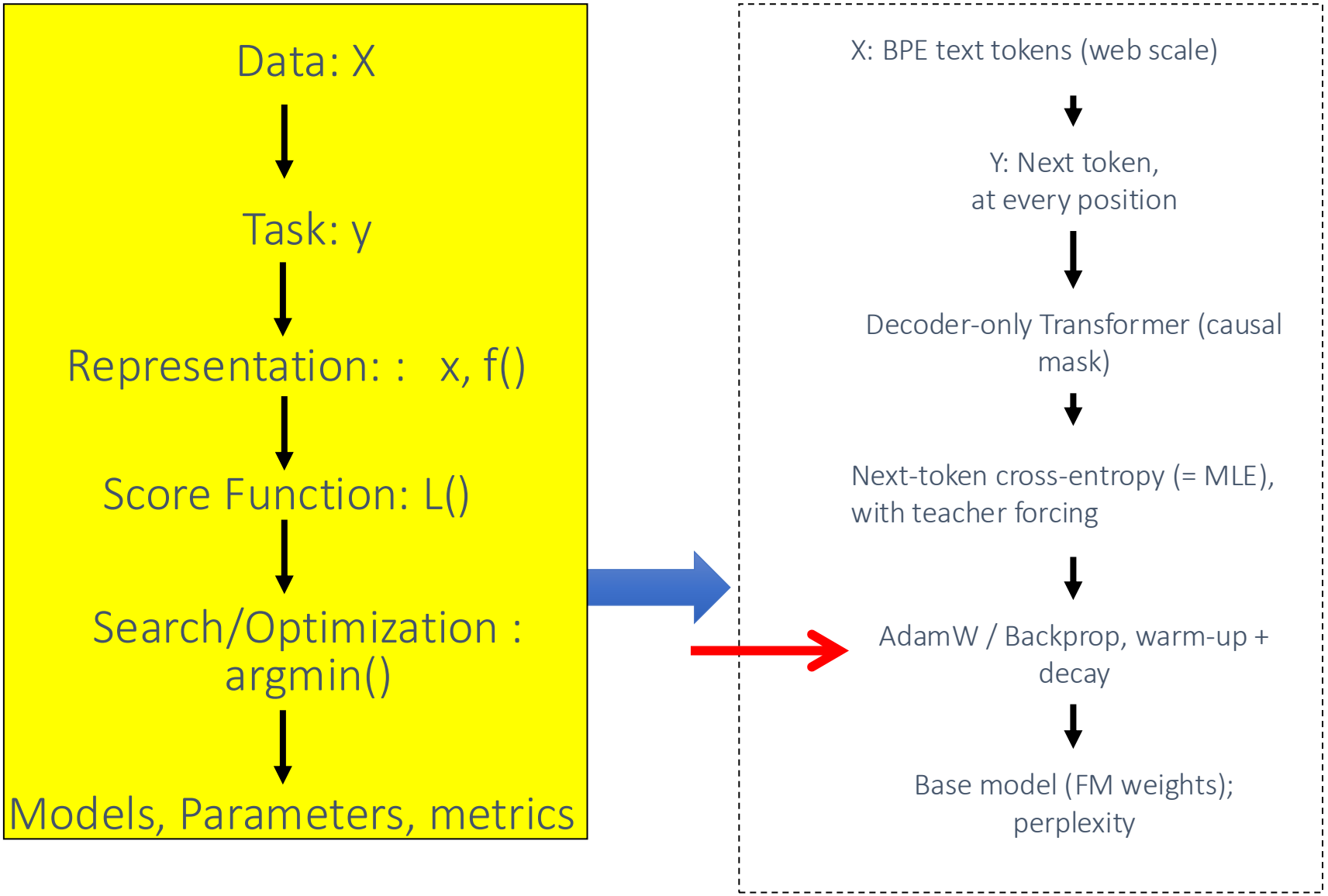
Minimizing one loss on trillions of words forces the model to pick up every regularity that helps predict the next token:

to fill the blank it must learn . . .	example
grammar	In my free time, I like to {run, banana}
lexical semantics	At the zoo we saw giraffes, lions, and {zebras, spoon}
world knowledge	The capital of Denmark is {Copenhagen, London}
sentiment	Edge of my seat the whole time. The movie was {good, bad}
translation	The word for "pretty" in Spanish is {bonita, hola}
arithmetic	3 + 8 + 4 = {15, 11}

Pretraining is implicit multi-task learning. Nobody wrote a sentiment loss or a translation loss; the next-token loss already contains all of them.

credit: Peng Wang, UVA CS 4774 L15

① Pretraining: the decoder-only

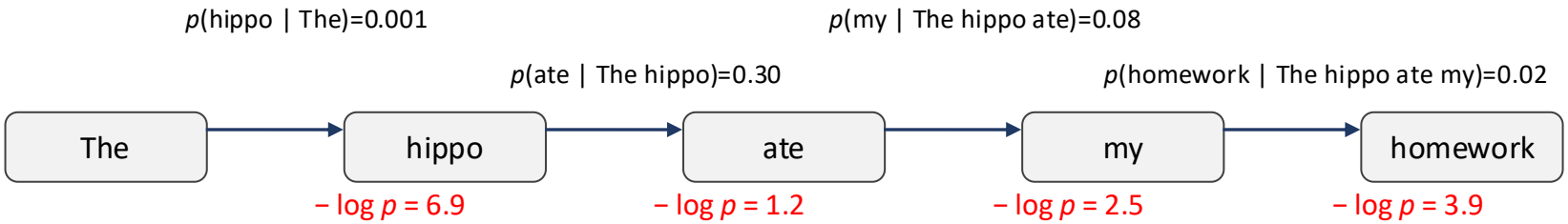


Training with cross-entropy = maximum likelihood

on the corpus

Pretraining loss (GPT-style, "causal" language modeling):

$$\mathcal{L}(\theta) = -(1/N) \sum_{t=1..N} \log p_\theta(x_t \mid x_{<t})$$



- Each probability is conditioned on the *whole prefix* $x_{<t}$, not just the previous word.
- Target at each position = the **actual next token**; loss = its NLL; average over positions.
- **Self-supervised**: the labels are free, the next word is already in the text.

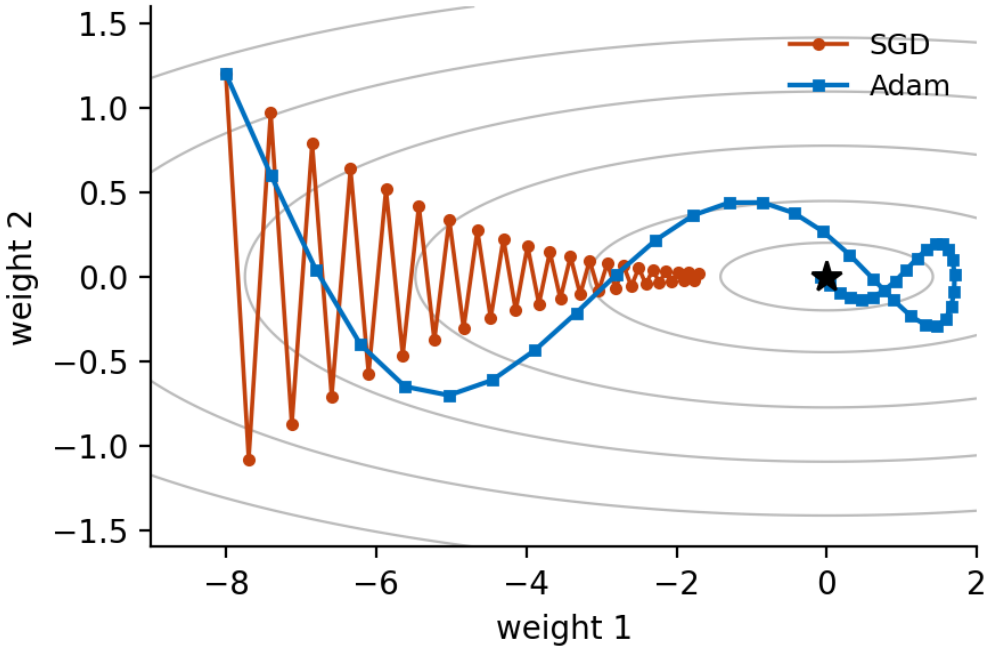
You've seen this (MLE): $\max \prod_t p_\theta(x_t \mid x_{<t}) \Leftrightarrow \min -\sum_t \log p_\theta(x_t \mid x_{<t})$.
Pretraining with next token-predict *is* maximum likelihood !!!

credit: Peng Wang, UVA CS 4774 L15

Optimizers: one step (learning rate) size is not enough

$$\theta_t = \theta_{t-1} - \gamma\, g_t$$

One learning rate γ for every weight (parameter, dimen, et al);
g_t = minibatch gradient



- Steep weight 2 caps γ: SGD diverges above γ = 0.04
- The moving crawls along the flat weight 1 direction
- Same start, 40 steps, best γ tried:
  - SGD: loss 68 → 1.45
  - Adam: loss 68 → 0.003

This loss contour map: steep in weight 2, flat in weight 1

Adam: two running averages

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t$$

Momentum: average direction

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)\, g_t^2$$

Scale: average squared size

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$

Bias fix: both start at 0

$$\theta_t = \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

Update: per-weight step

- Everything is per weight; g_t^2 is element-wise
- PyTorch defaults: $\gamma = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Adam: momentum + a step size per weight

credit: Kingma & Ba, ICLR 2015; defaults from the PyTorch torch.optim.Adam documentation

Adam by hand: the first step

Two weights, first gradient g = (0.1, 10), $\gamma = 0.001$, defaults for β_1, β_2, ε

	Weight 1 (flat)	**Weight 2 (steep)**
Gradient g	0.1	10
$m_1 = 0.1\ g$	0.01	1
$s_1 = 0.001\ g^2$	0.00001	0.1
$\hat{m}_1 = m_1 / 0.1$	0.1	10
$\hat{s}_1 = s_1 / 0.001$	0.01	100
Adam step $\gamma\ \hat{m}_1 / \sqrt{\hat{s}_1}$	**0.001**	**0.001**
SGD step $\gamma\ g$	0.0001	0.01

Steep weight scaled down, flat weight scaled up: both move 0.001

Quiz: all gradients 10× larger. How far does each weight move?

AdamW: keep weight decay out of Adam

Only difference: where the weight decay λθ is added

Adam with L2 penalty

$$g_t = \nabla L(\theta_{t-1}) + \lambda \theta_{t-1}$$

$$\theta_t = \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

λθ is divided by √ŝ: big-gradient weights barely decay

AdamW

$$g_t = \nabla L(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon} - \gamma \lambda \theta_{t-1}$$

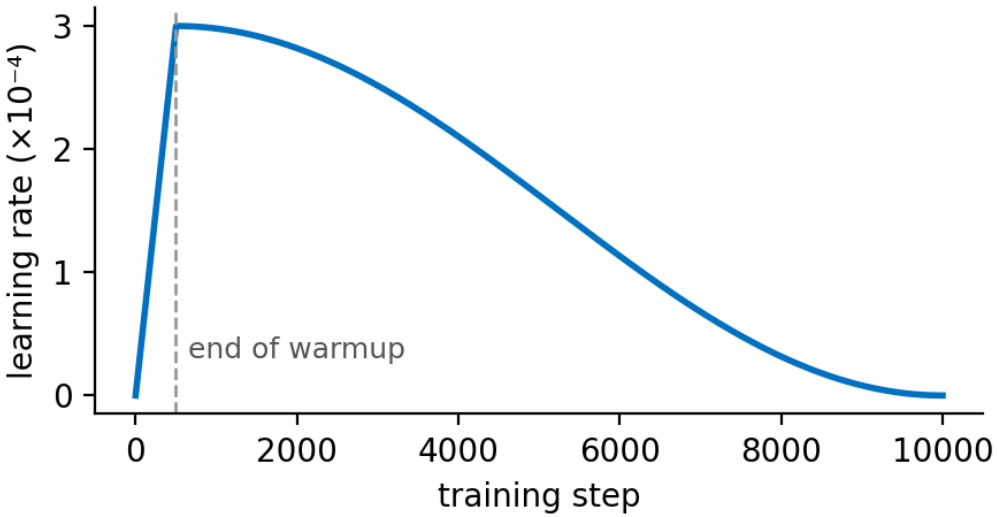
λθ skips m and s: every weight decays equally

```
opt = torch.optim.AdamW(model.parameters(), lr=3e-4, weight_decay=0.01)
# weight_decay defaults: Adam 0, AdamW 0.01
```

AdamW is the usual choice for Transformers.

credit: Loshchilov & Hutter, ICLR 2019; defaults from the PyTorch optimizer documentation

Learning rate: warm up, then decay



- Warmup: start small while gradients are noisy and s is unsettled
- Decay: smaller steps near the end
- Example: 10,000 steps, 500 of warmup, peak 3×10^{-4}

```
def lr_at(t):              # called once per step, before opt.step()
    if t < W: return peak * t / W                              # linear warmup
    return peak * 0.5 * (1 + math.cos(math.pi * (t - W) / (N - W)))   # cosine
```

Debug with a fixed learning rate first; schedule later

credit: warmup as in Vaswani et al. 2017; cosine decay after Loshchilov & Hutter, ICLR 2017

Code: example training loop and common bugs

```
opt = torch.optim.Adam(model.parameters(), lr=1e-3)    # PyTorch's default lr
loss_fn = nn.CrossEntropyLoss(ignore_index=0)          # 0 is <pad>
for epoch in range(n_epochs):
    for x, mask in loader:                             # x: (B, T)
        inp, tgt = x[:, :-1], x[:, 1:]                 # shift by one
        logits = model(inp, mask[:, :-1])              # (B, T-1, V)
        loss = loss_fn(logits.reshape(-1, V), tgt.reshape(-1))
        opt.zero_grad(); loss.backward(); opt.step()
```

Symptom	Cause	Fix
Shape error in the loss	logits have 3 axes, targets 2	reshape both, as above
Padding gets trained on	<pad> is a target	ignore_index=0
Odd next-token guesses	reading a pad position	use the last real position
Padding moves the class score	pads in the mean	(c * m).sum(1) / m.sum(1)
Later tokens change earlier logits	mask missing or after softmax	change inputs after t, compare

Adam: gradient descent with momentum and its own step size for each weight. Try a few learning rates and compare the loss curves.

Roadmap : Today

- Training stages (encoder-only, decoder-only)
- Pretraining tactics (tokens, next-token loss, Adam)
- **Pretrained GPT models (GPT-1/2/3)**
- Post-training (SFT, aligning a GPT: RLHF, DPO, RLVR)

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

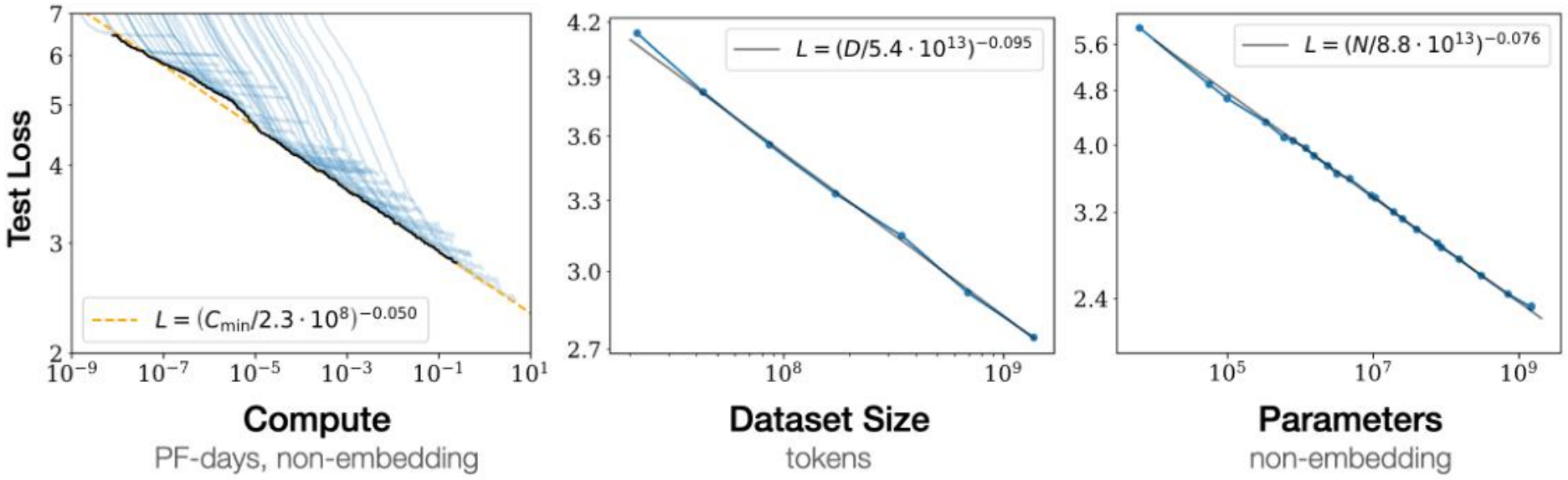
Dario Amodei
OpenAI
damodei@openai.com

[cs.LG] 23 Jan 2020

- **Data:** Llama-3 saw ~15 trillion tokens.
- **Parameters:** 0.1B (2018) → 175B (2020) → 405B (2024).
- **Compute:** thousands of GPUs for months.
- **Scaling laws** (Kaplan et al., 2020): test loss falls as a **power law** in each factor: straight lines on log-log axes.

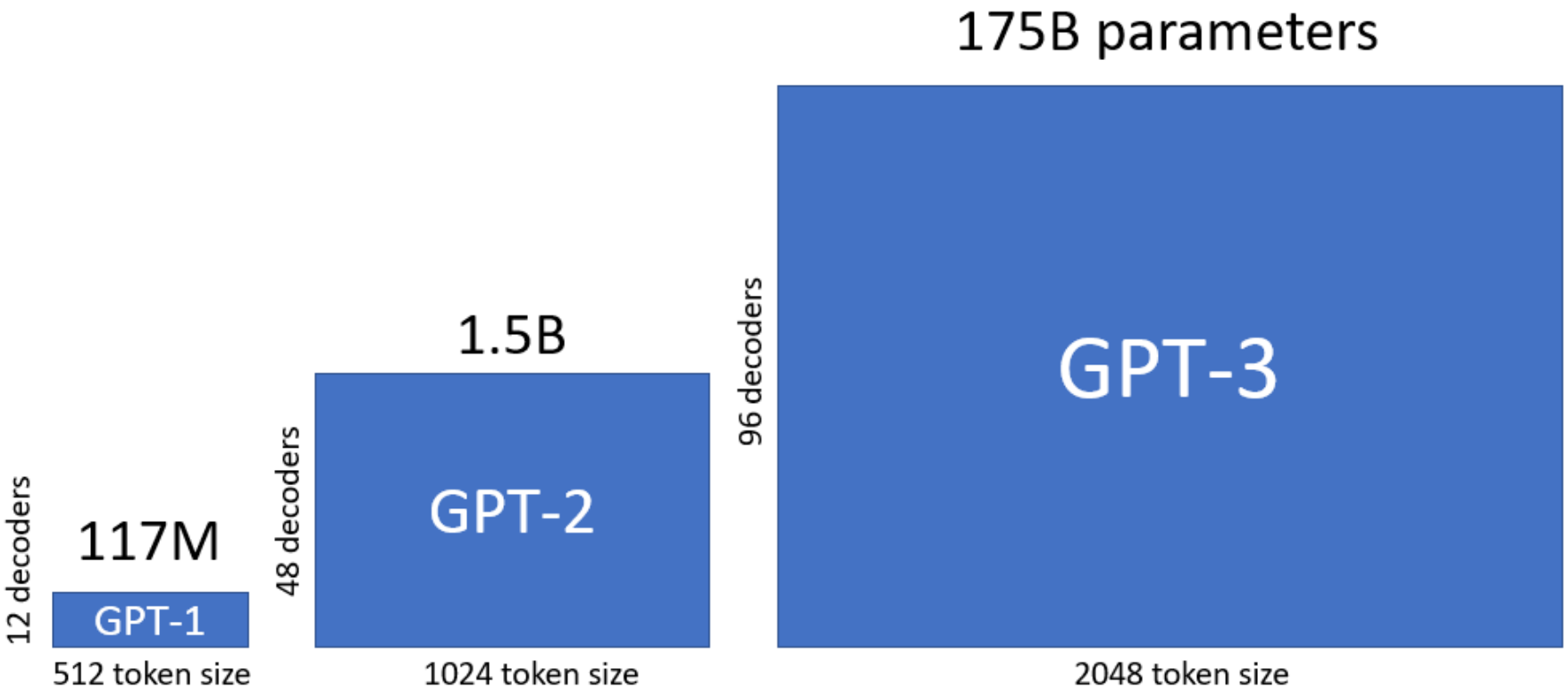
Scale: more of everything, predictably better

Test loss vs. training compute (in PF-days: 10^{15} floating-point ops per second, sustained for a day), dataset size (tokens), and parameters.



You've seen this (bias–variance): More capacity should eventually mean more variance (overfitting). Here test loss keeps falling with more parameters because **the data grows along with the model** and is seen only about once, so there is little to memorize. Scale data and parameters *together*.

GPT-1 → GPT-2 → GPT-3: same decoder, more scale



https://medium.com/@YanAIx/step-by-step-into-gpt-70bc4a5d8714

GPT-2 (Radford et al. 2019)
- Language Models are Unsupervised Multitask Learners

- A general systems learn to model $P(output|input, \mathbf{task})$
- Task can be specified in natural language
- Aims to general purpose language learner

"We would like to move towards more general systems which can **perform many tasks – eventually without the need to manually create and label a training dataset for each one.**"

credit: Radford et al. 2019 (GPT-2)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training

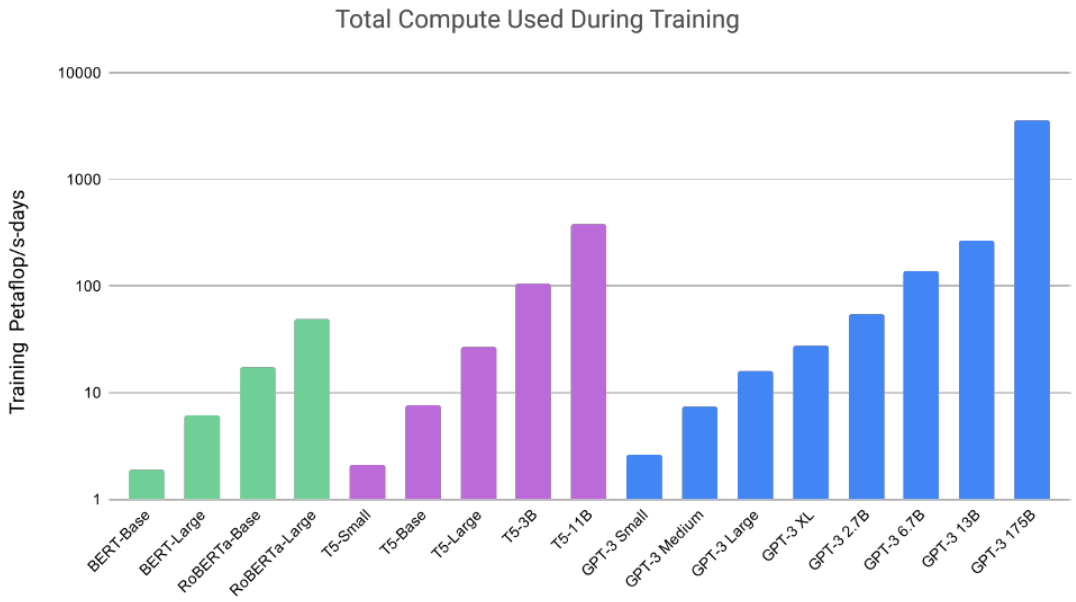


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Rough compute price to train GPT-3 175B: ~$4.5M

GPT3 Training

How Much Data Pre-trains LMs Used (In 2024)?

LLaMA 3

https://arxiv.org/abs/2407.21783

- **Data.** Compared to prior versions of Llama (Touvron et al., 2023a,b), we improved both the quantity and quality of the data we use for pre-training and post-training. These improvements include the development of more careful pre-processing and curation pipelines for pre-training data and the development of more rigorous quality assurance and filtering approaches for post-training data. We pre-train Llama 3 on a corpus of about 15T multilingual tokens, compared to 1.8T tokens for Llama 2.

DeepSeek-V3

https://arxiv.org/abs/2412.19437

We present DeepSeek-V3, a strong Mixture-of-Experts (MoE) language model with 671B total parameters with 37B activated for each token. To achieve efficient inference and cost-effective training, DeepSeek-V3 adopts Multi-head Latent Attention (MLA) and DeepSeekMoE architectures, which were thoroughly validated in DeepSeek-V2. Furthermore, DeepSeek-V3 pioneers an auxiliary-loss-free strategy for load balancing and sets a multi-token prediction training objective for stronger performance. We pre-train DeepSeek-V3 on 14.8 trillion diverse and high-quality tokens, followed by Supervised Fine-Tuning and Reinforcement Learning stages to fully harness its capabilities. Comprehensive evaluations reveal that DeepSeek-V3 outperforms other open-source models and achieves performance comparable to leading closed-source models. Despite its excellent performance, DeepSeek-V3 requires only 2.788M H800 GPU hours for its full training. In addition, its training process is remarkably stable. Throughout the entire training process, we did not experience any irrecoverable loss spikes or perform any rollbacks.

How Big Is 15T Tokens?

- Print 15T tokens
- Say 100 sheets = 1 cm
- ~1,500 km / 1,000 mile thick

1000 tokens



1 Introduction

A central question in the discussion of large language models (LLMs) concerns the extent to which they *memorize* their training data versus how they *generalize* to new tasks and settings. Most practitioners seem to (at least informally) believe that LLMs do some degree of both: they *clearly* memorize parts of the training data—for example, are often able to reproduce large portions of training data verbatim [Carlini et al., 2023]—but they *also* seem to learn from this data, allowing them to generalize to new settings. The precise extent to which they do one or the other has massive implications for the practical and legal aspects of such models [Cooper et al., 2023]. Do LLMs truly produce new content, or do they only remix their training data? Should the act of training on copyrighted data be deemed unfair use of data, or should fair use be judged by the model's memorization? With respect to people, we distinguish plagiarizing content from learning from it, but how should this extend to LLMs? The answer to such questions inherently relates to the extent to which LLMs memorize their training data.

However, even defining memorization for LLMs is challenging and many existing definitions leave a lot to be desired. Certain formulations claim that a passage from the training data is memorized if the LLM can reproduce it exactly [Nasr et al., 2023]. However, this ignores situations where, for instance, a prompt instructs the model to exactly repeat some phrase. Other formulations define memorization by whether or not prompting an LLM with a portion of text from the training set results in the completion of that training datum [Carlini et al., 2023]. But these formalisms rely fundamentally on the completions being a certain size, and typically very lengthy generations are required for sufficient certainty of memorization. More crucially, these definitions are too permissive because they ignore situations where model developers can (for legal compliance) post-hoc "align" an LLM by instructing their models not to produce certain copyrighted content [Ippolito et al., 2023]. But has such an instructed model really *not memorized* the sample in question, or does the model still contain all the information about the datum in its weights while it hides behind an illusion of compliance? Asking such questions becomes critical because this illusion of "unlearning" can often be easily broken as we show in Sections 4.1 and 4.3.

In this work, we propose a new definition of memorization based on a compression argument. Our definition posits that *a phrase present in the training data is memorized if we can make the model reproduce the phrase using a prompt (much) shorter than the phrase itself.* Operationalizing this definition requires finding the shortest adversarial input prompt that is specifically optimized to produce a target output. We call this ratio of input to output tokens the Adversarial Compression Ratio (ACR). In other words, memorization is inherently tied to whether a certain output can be represented in a *compressed* form, beyond what language models can do with typical text. We argue that such a definition provides an intuitive notion of memorization—if a certain phrase exists within the LLM training data (e.g., is not itself generated text) *and* it can be reproduced with fewer input tokens than output tokens, then the phrase must be stored somehow within the weights of the LLM. Although it may be more natural to consider compression in terms of the LLM-based notions of input/output perplexity, we argue that a simple compression ratio based on input/output token counts provides a more intuitive explanation to non-technical audiences, and has the potential to serve as a legal basis for important questions about memorization and permissible data use.

In addition to its intuitive nature, our definition has several other desirable qualities. We show that it appropriately ascribes many famous quotes as being memorized by existing LLMs (i.e. they have high ACR values). On the other hand, we find that text not in the training data of an LLM, such as samples posted on the internet after the training period, are not compressible, that is their ACR is low.

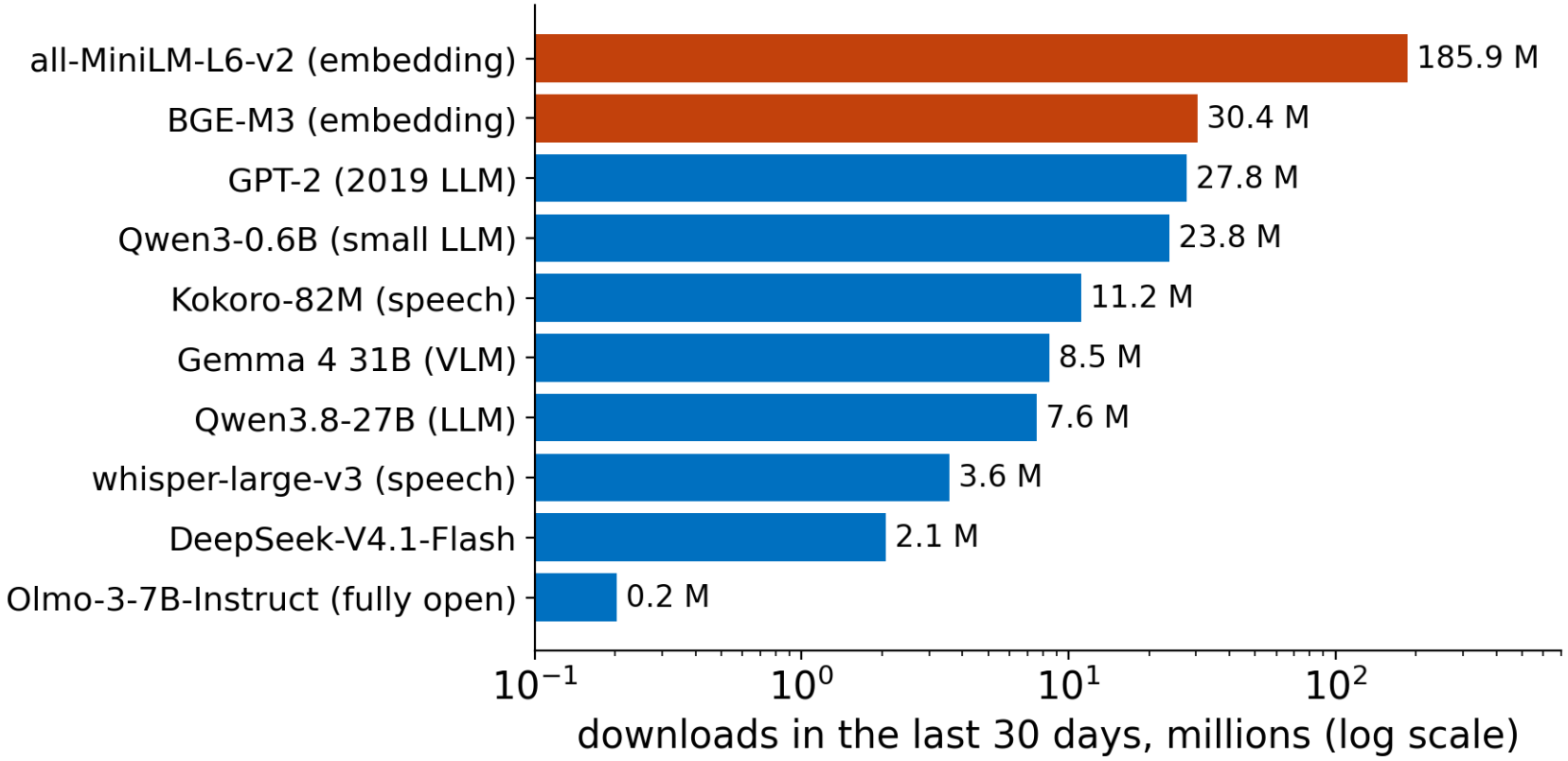
We examine several unlearning methods using ACR to show that they do not substantially affect the memorization of the model. That is, even after explicit finetuning, models asked to "forget" certain pieces of content are still able to reproduce them with a high ACR—in fact, not much smaller than with the original model. Our approach provides a simple and practical perspective on what memorization can mean, providing a useful tool for functional and legal analysis of LLMs.

2 Do We Really Need Another Notion of Memorization?

With LLMs ingesting more and more data, questions about their memorization are attracting attention [e.g. Carlini et al., 2019, 2021, Nasr et al., 2023, Zhang et al., 2023]. There remains a pressing need

2

Open-source LLM: What people actually download



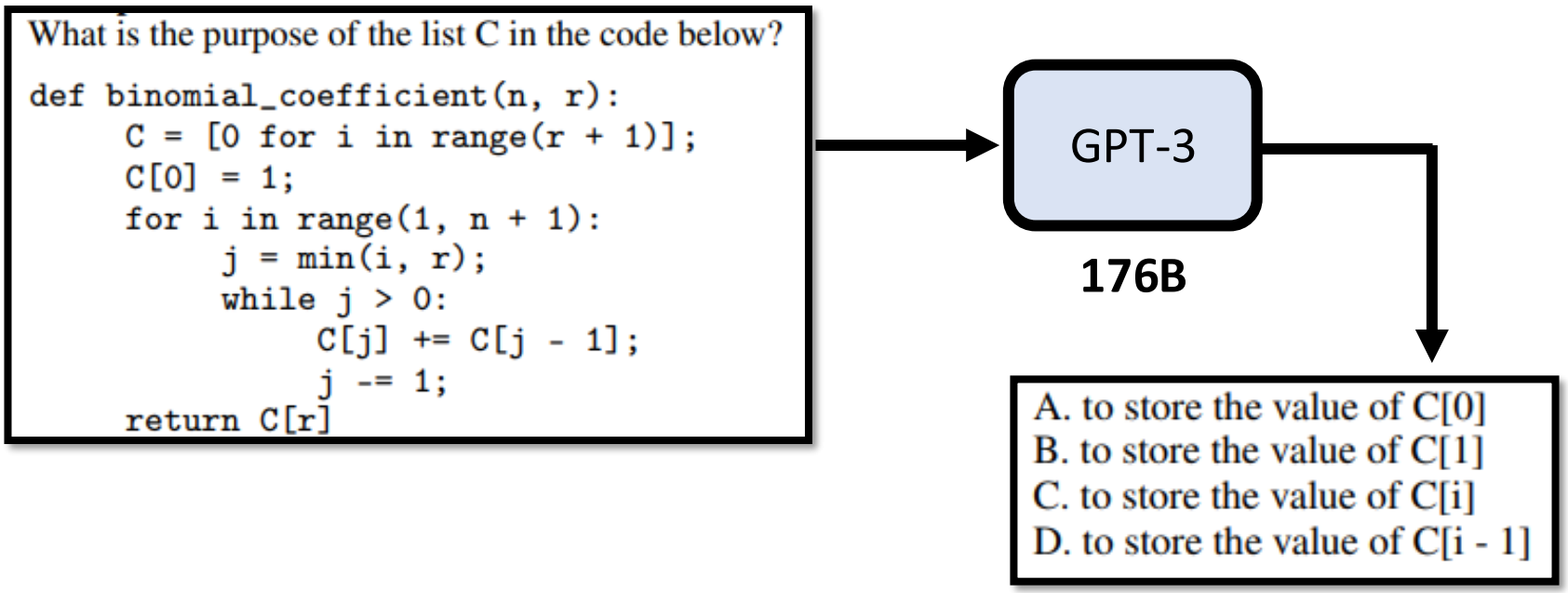
Hugging Face count, last 30 days, as of Oct 31, 2026. Not a quality ranking: automated jobs count too

credit: Hugging Face Hub API, fetched 2026-10-31; downloads field is a 30-day count

Roadmap : Today

- Training stages (encoder-only, decoder-only)
- Pretraining tactics (tokens, next-token loss, Adam)
- Pretrained GPT models (GPT-1/2/3)
- **Post-training (SFT, aligning a GPT: RLHF, DPO, RLVR)**

Web-Pretrained Models Often Don't Answer

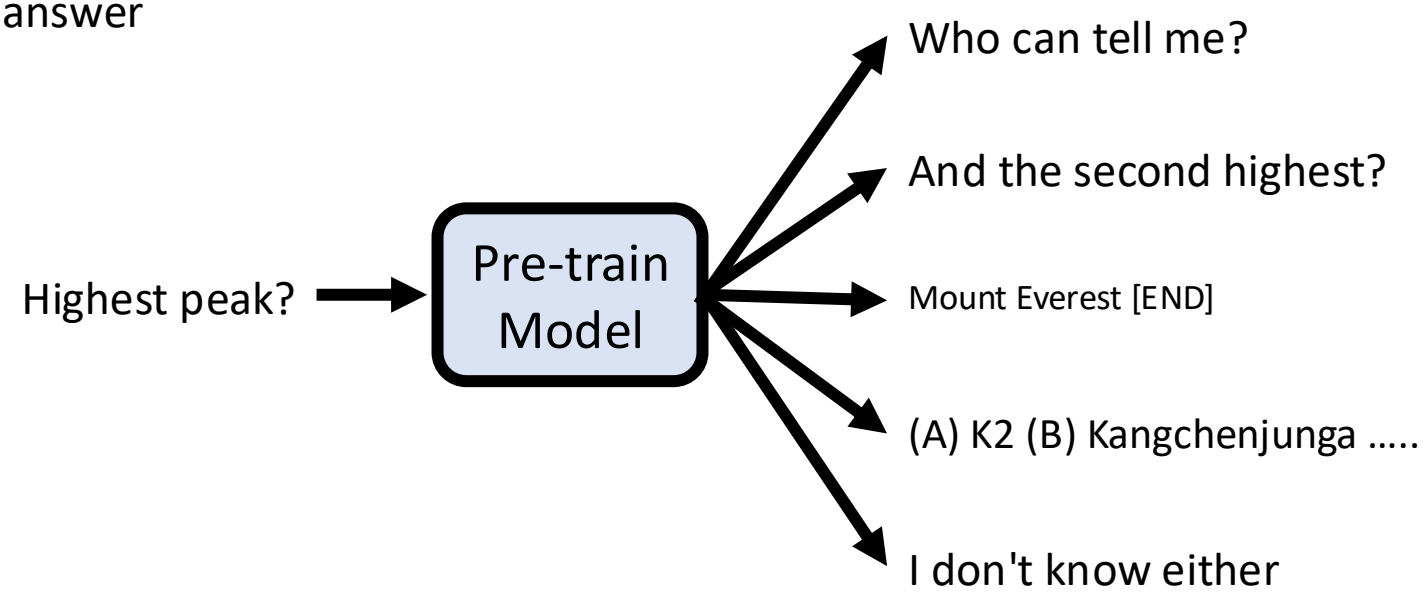


Source of image:
https://arxiv.org/abs/2203.02155

credit: Prof. Hung-yi Lee,

Pretrained Model: supplying the knowledge

- Pre-training can already give the right answer

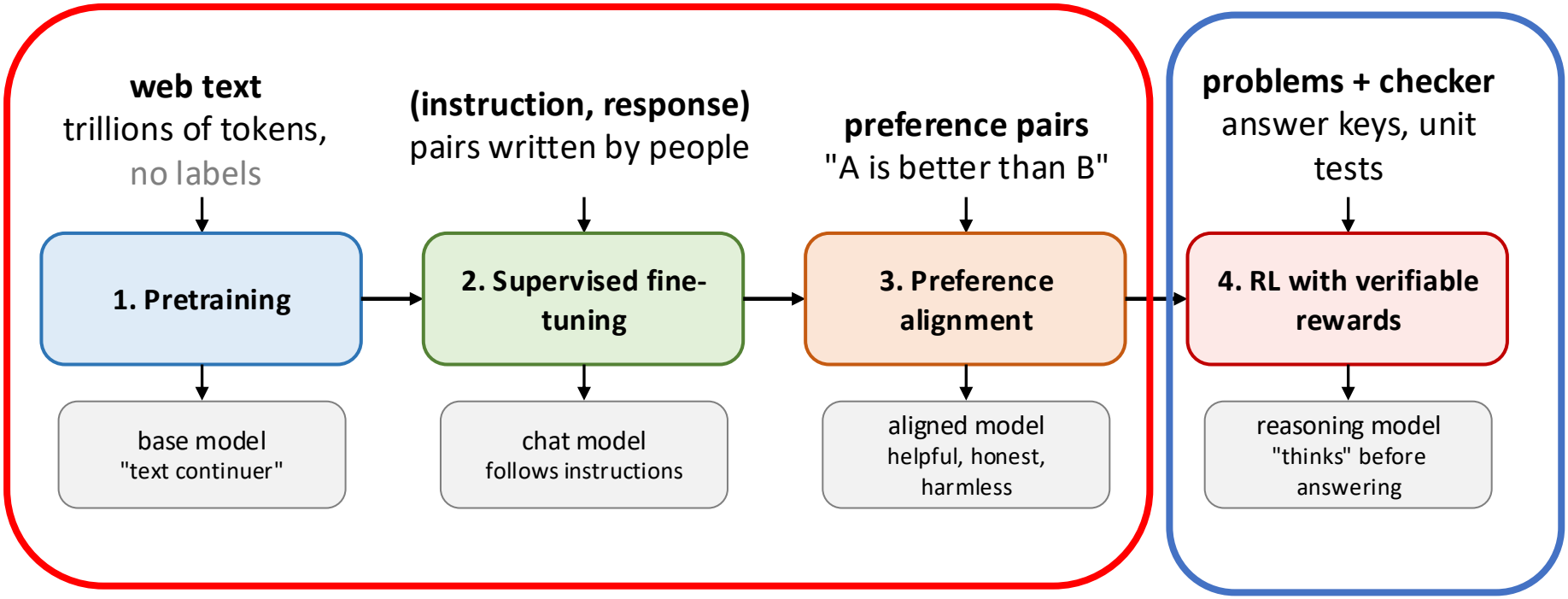


- SFT, RL make the right choice more likely

credit: Prof. Hung-yi Lee, NTU GenAI-ML 2025 (LLM training)

Recap: Peng's slide deck:

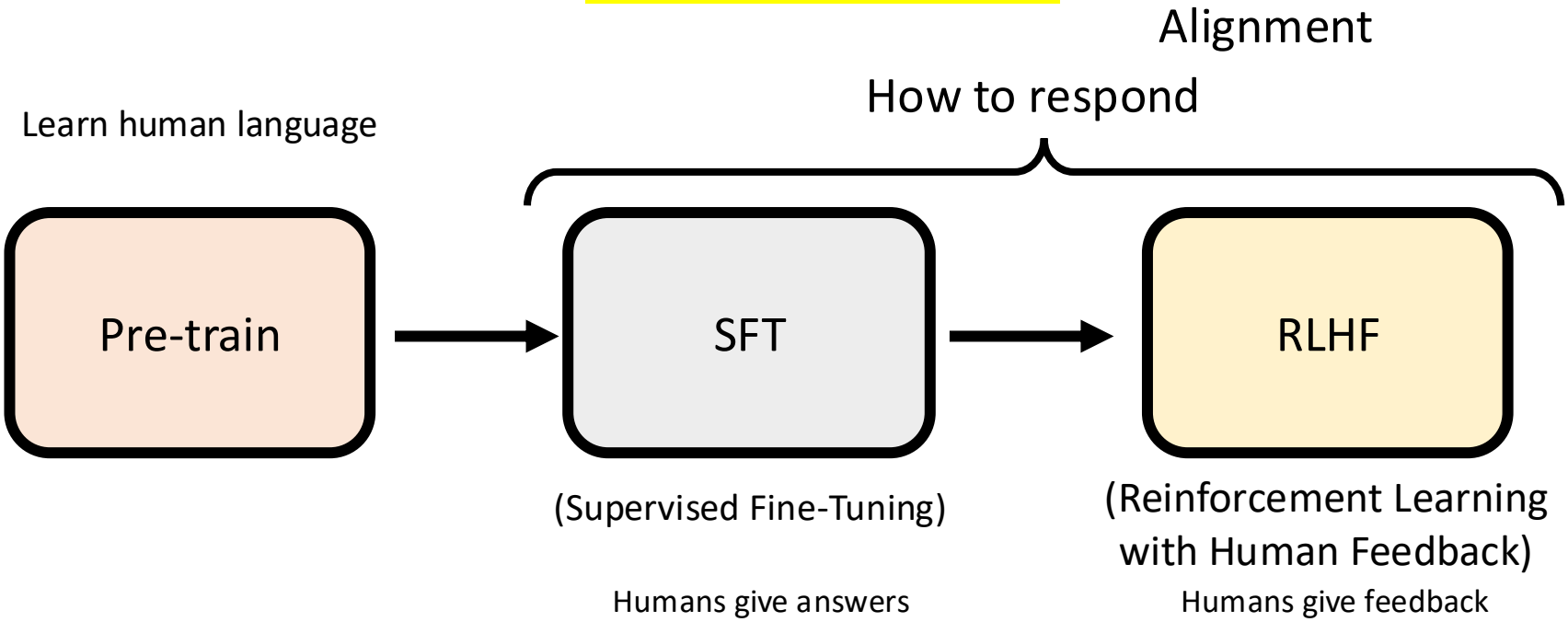
How a base decoder model becomes a SOTA agent's core model (~2024 to 2026)



the training pipeline

credit: Peng Wang, UVA CS 4774 L15

How LLMs Learn (into ChatGPT)

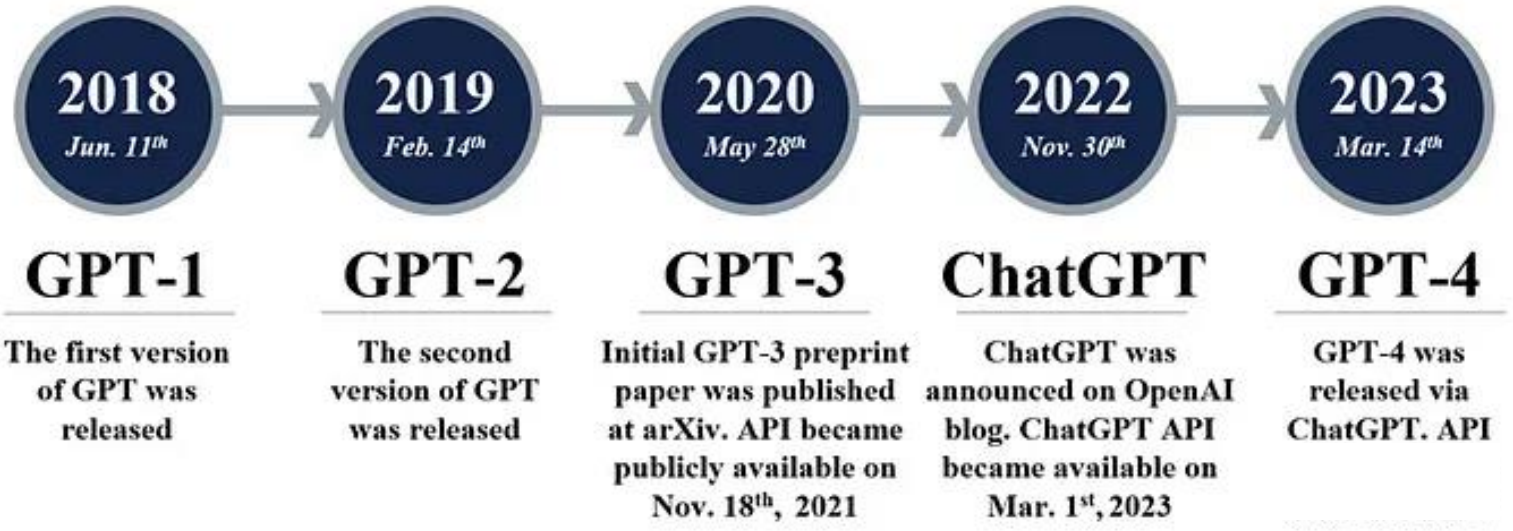


Each stage initializes from the previous stage's params (Initialization)

credit: Prof. Hung-yi Lee, NTU GenAI-ML 2025 (LLM training)

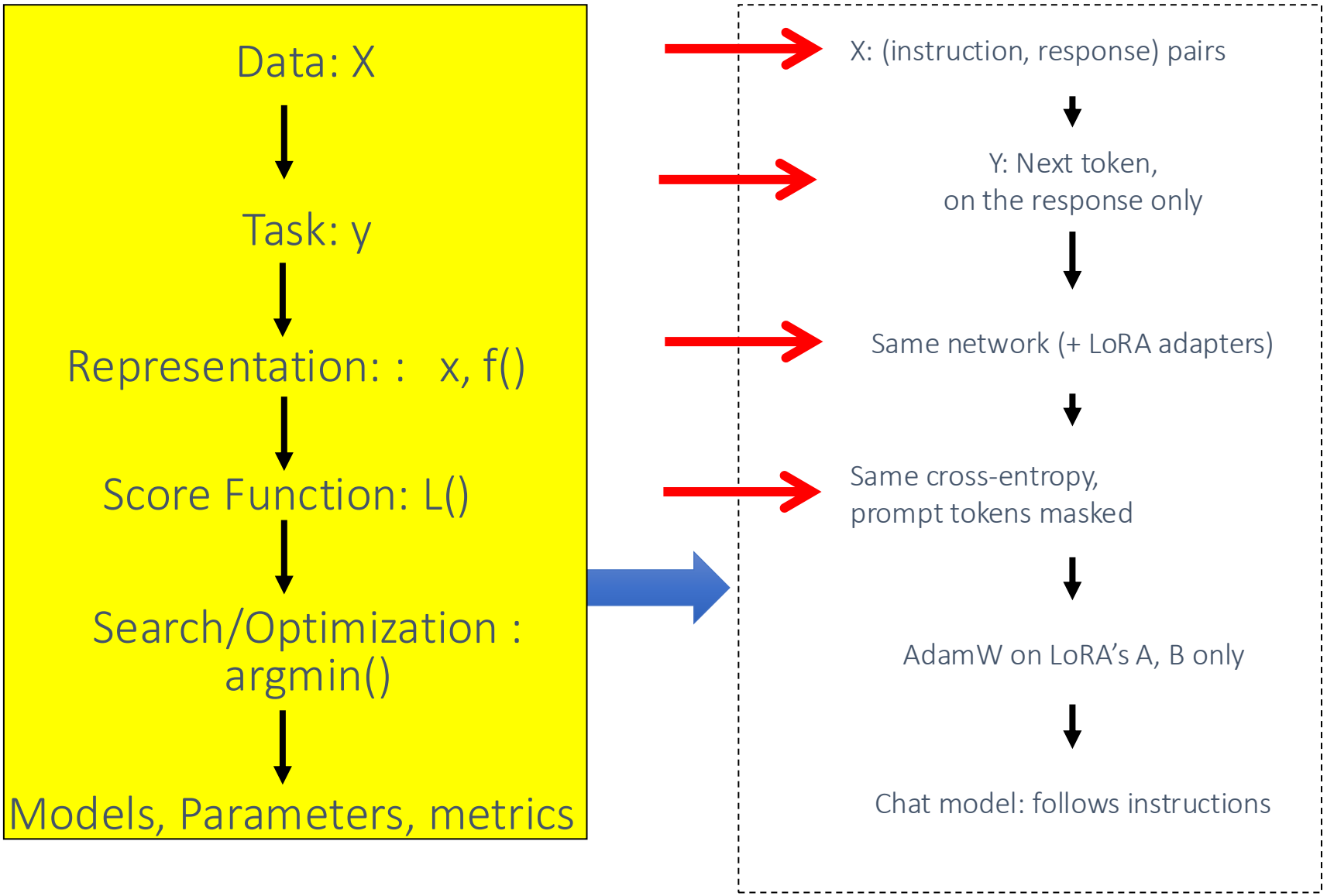
ChatGPT: “Optimizing Language Models for Dialogue” by OpenAI

- No paper / Just a blog / Released Nov 30 2022
- It took 5 days to reach 1M users



credit: OpenAI blog, Nov 30 2022

② Post-training (SFT): box by box



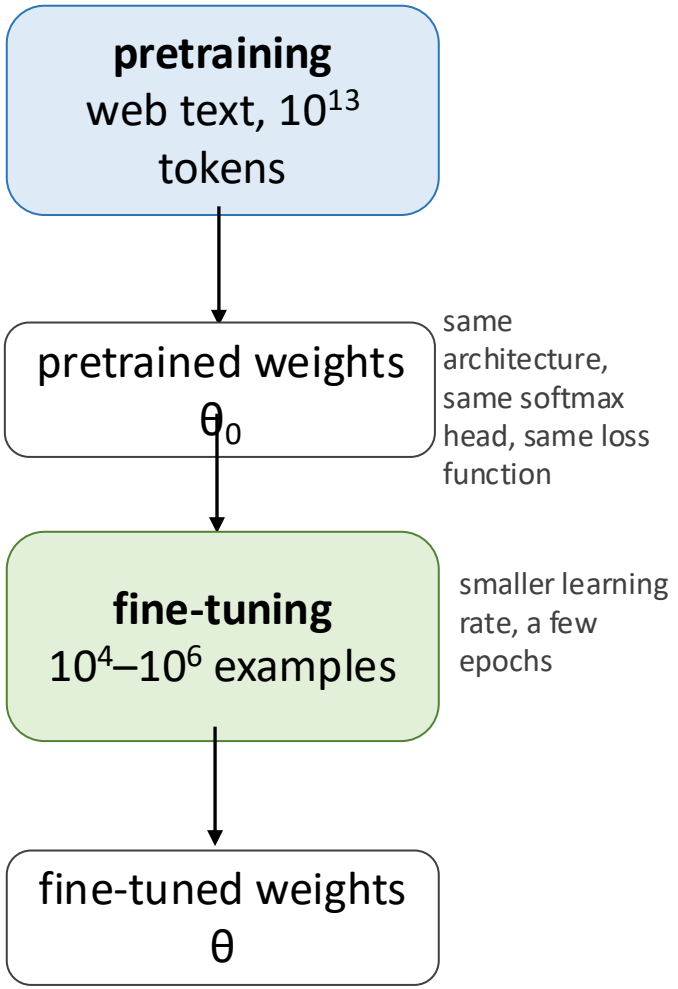
same network, same loss. New data (instruction pairs), and a cheaper search over θ (LoRA, QLoRA).

Pretrain once, supervised fine-tune (SFT) many times

.

- The pretrained weights θ_0 are the starting point. Fine-tuning moves them a little and keeps what pretraining learned.

- The fine-tuning introduced next is **instruction tuning**: turning the text continuer into an assistant.



credit: Peng Wang, UVA CS 4774 L15

Instruction tuning: one data example

Chat format: multi-turn, with roles, serialized with special tokens:

```
{"role": "user", "content": "Weather today?"}
{"role": "assistant", "content": "Sunny, high of 75."}
{"role": "user", "content": "And tomorrow?"}
{"role": "assistant", "content": "Partly cloudy, 72."}
```

- The model learns when to *start* (right after the `<assistant>` marker), when to *stop* (by generating `<eos>`), and to follow the instruction.
- Normally 10^4–10^6 examples, much less than 10^{13} pretraining tokens.

credit: Peng Wang, UVA CS 4774 L15

Instruction tuning: another example data

- One Example form Instruction tuning Dataset

instruction

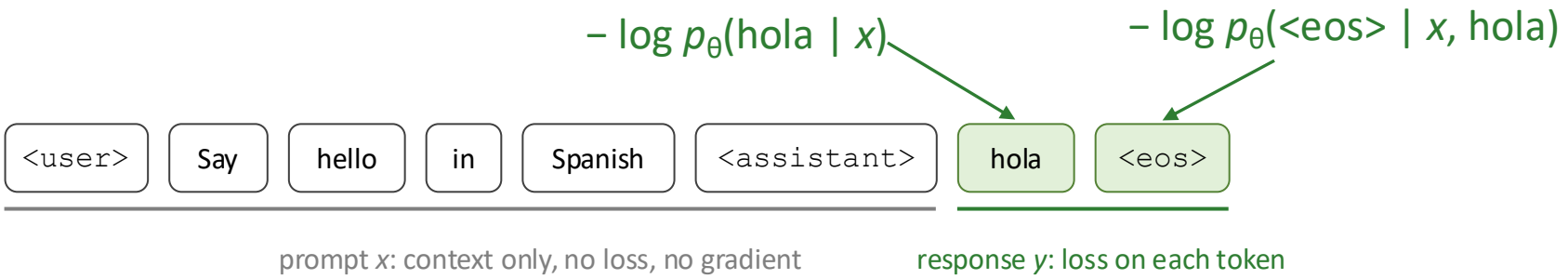
Task Type	Grammar Error Correction
Task ID	task1557_jfleg_grammar_error_correction
Definition	In this task, you will be shown an incorrect English sentence. You need to generate a corrected form of the input sentence.
Positive Example	**Input**: The car's wheel are loose. **Output**: The car's wheel is loose. **Explanation**: The instance of are is replaced by the word is. This makes the sentence grammatically correct.
Negative Example	**Input**: This way is the way to go. **Output**: This way may be the way to go. **Explanation**: The example does not correct the misuse of the word way. Instead, it should shorten the sentence to: this is the way to go.
Instance	**Input**: I think it 's harder for successful preson to risk somethnig , thay coluld lost much more then others . **Valid Output**: ["I think it 's harder for a successful person to risk something becuase they could lose much more than others ."]

SFT loss: the same cross-entropy

on the response tokens

Supervised fine-tuning (SFT) objective on pairs (x, y) = (prompt, response):

$$\mathcal{L}_{SFT}(\theta) = -\sum_{(x,y)\in D} \sum_{t=1..|y|} \log p_\theta(y_t \mid x, y_{<t})$$



- Identical to the pretraining loss, except prompt tokens are **context only** ("masking out in the loss sum").
- Same network, same softmax head, same optimizer.
- Maximizing the likelihood of the demonstrations = learning to **imitate** the demonstrated behavior

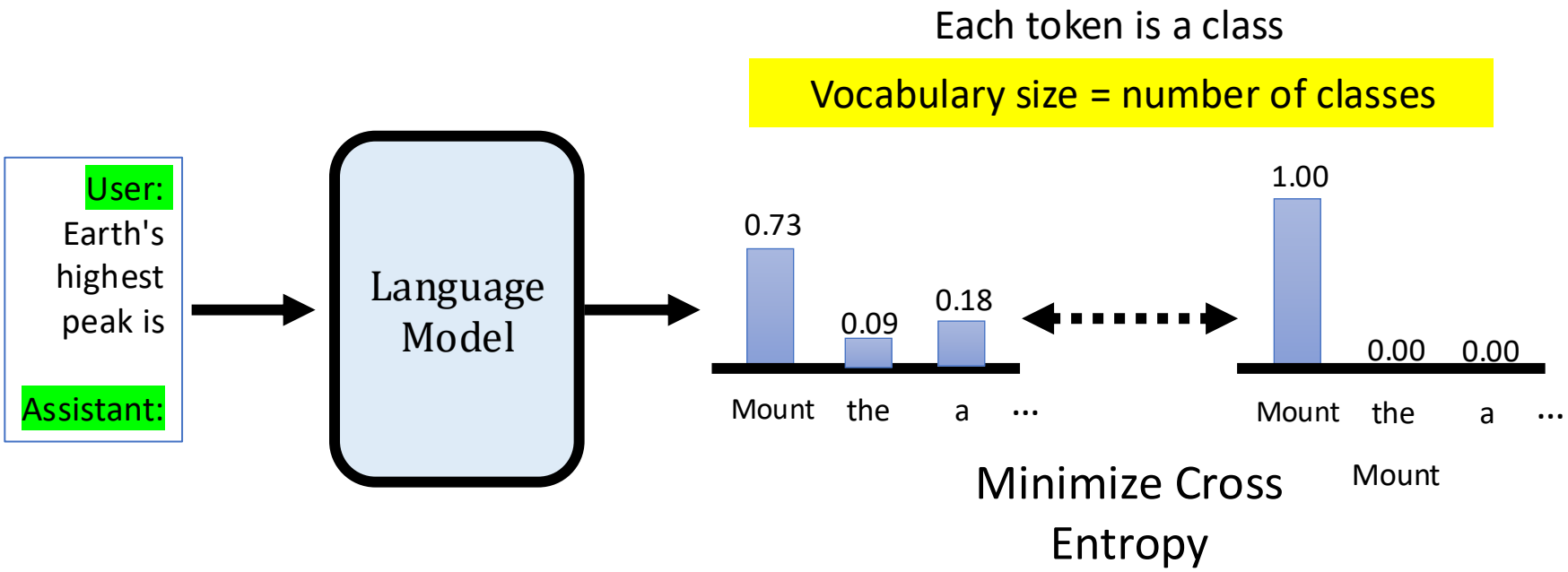
Nothing new in the math, only new data.

What it learns: the *format and behavior* of an assistant.

credit: Peng Wang, UVA CS 4774 L15

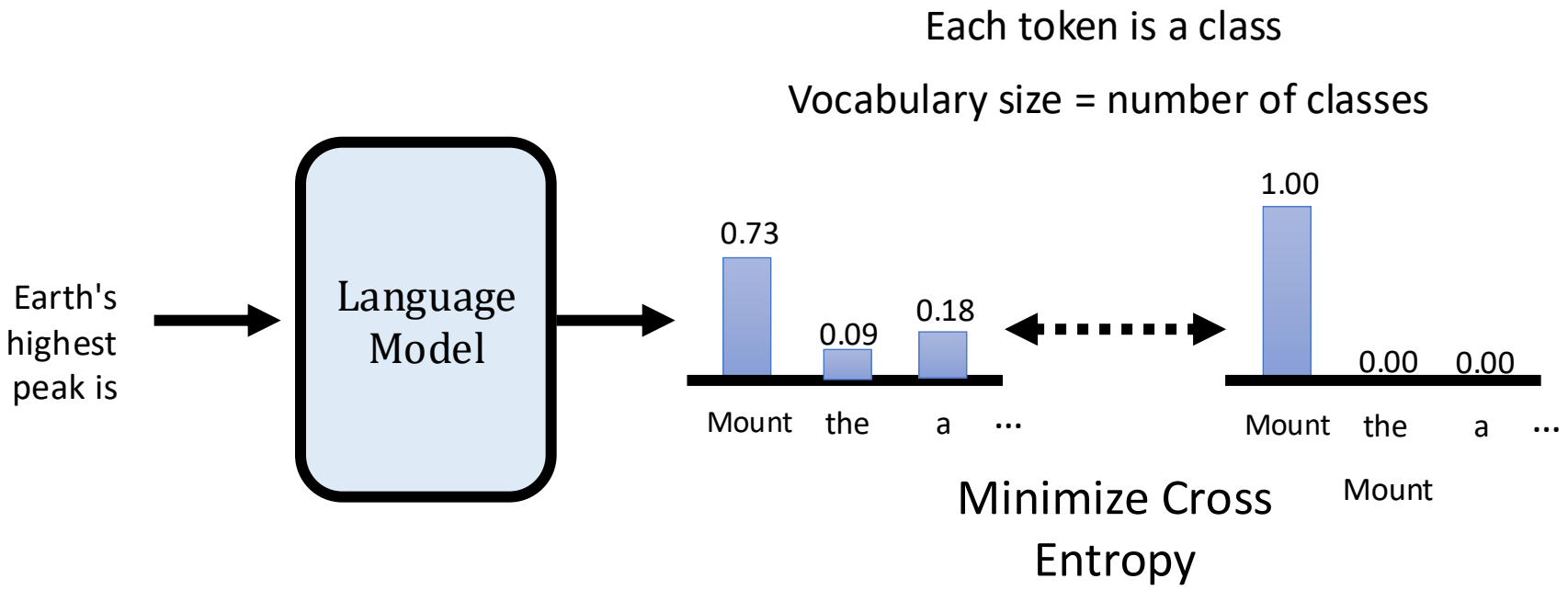
SFT Stage also Learns Next-Token Prediction

- Next-token = classification

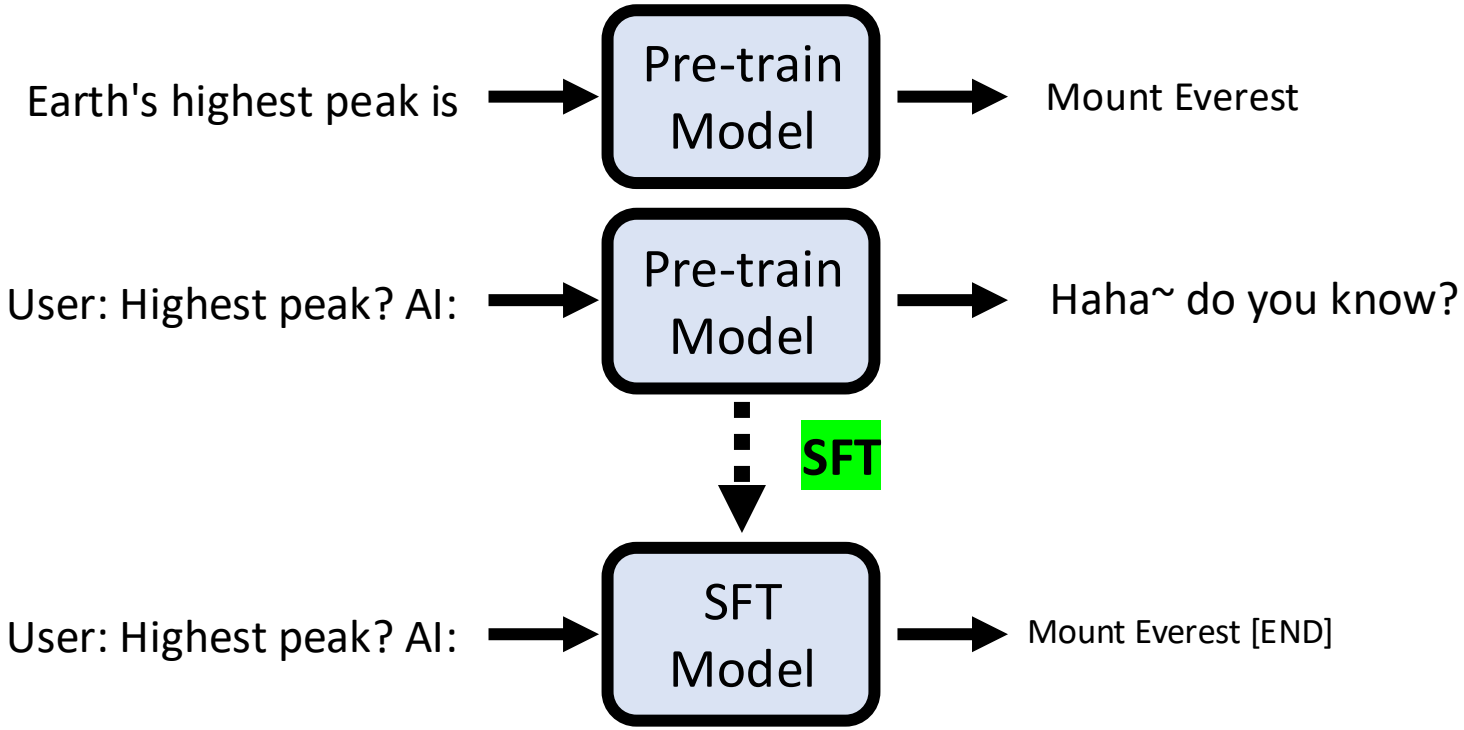


Recap:
Pretraining is all about Next-Token Prediction

- Next-token = classification



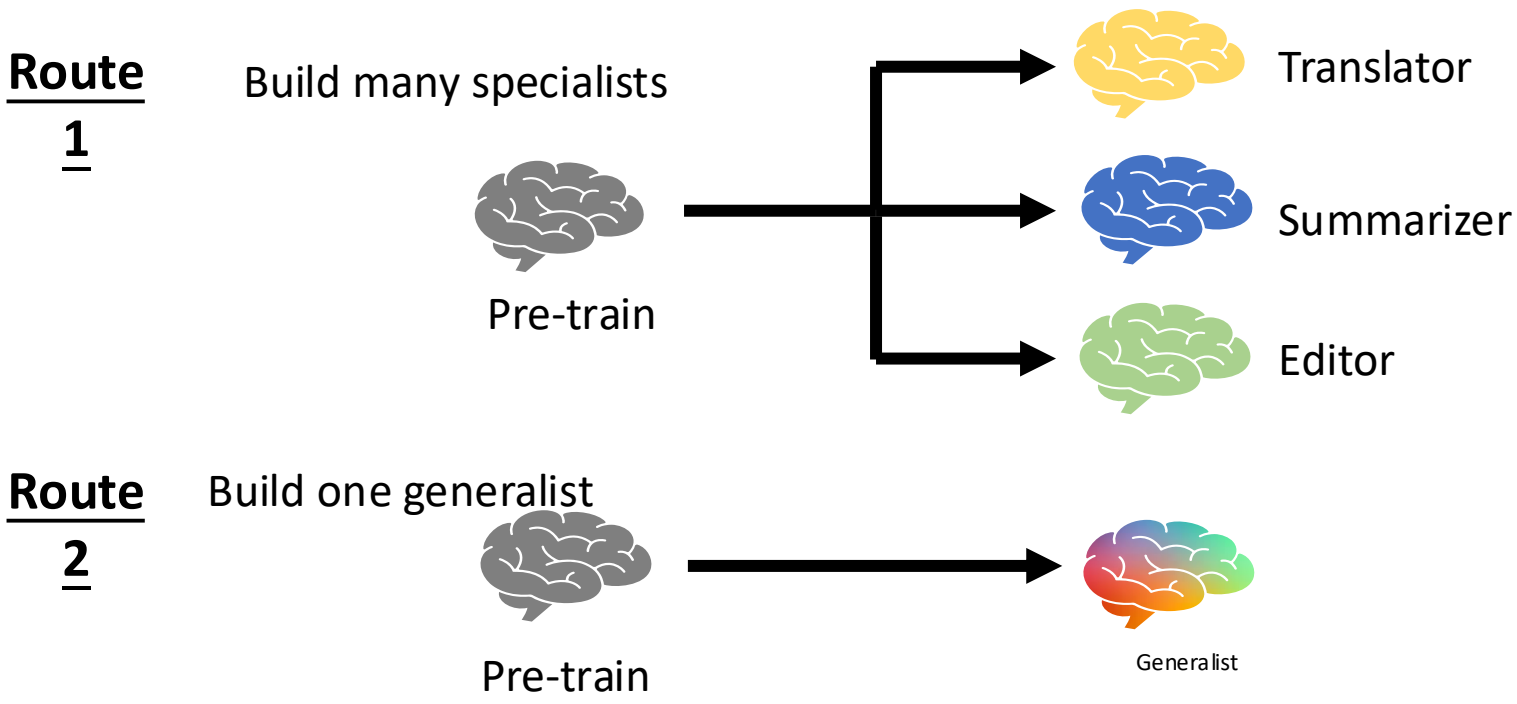
SFT mostly Doesn't Give LMs New Knowledge



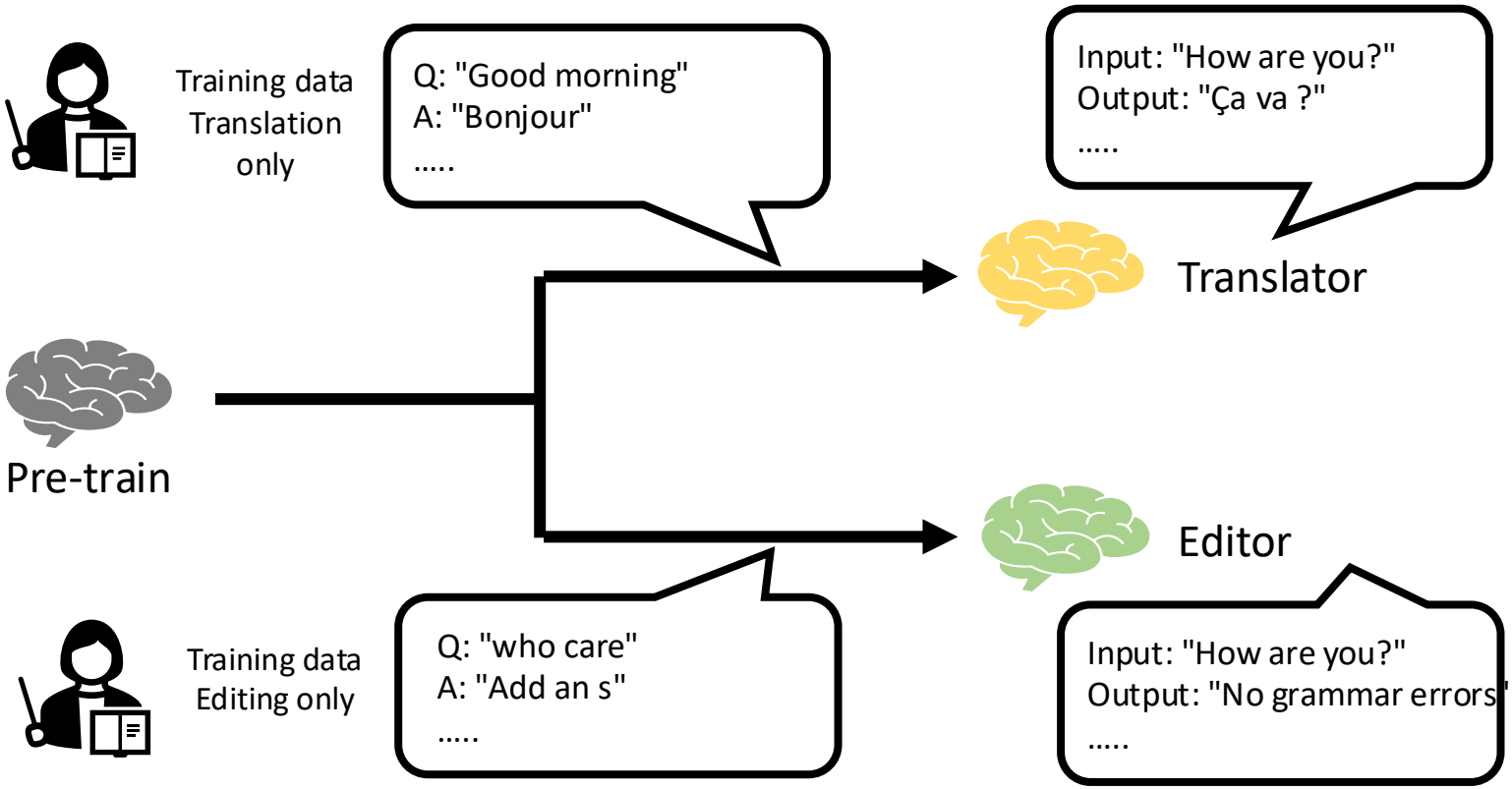
What SFT learns: the *format and behavior* of an assistant.

credit: Prof. Hung-yi Lee, NTU GenAI-ML 2025 (LLM training)

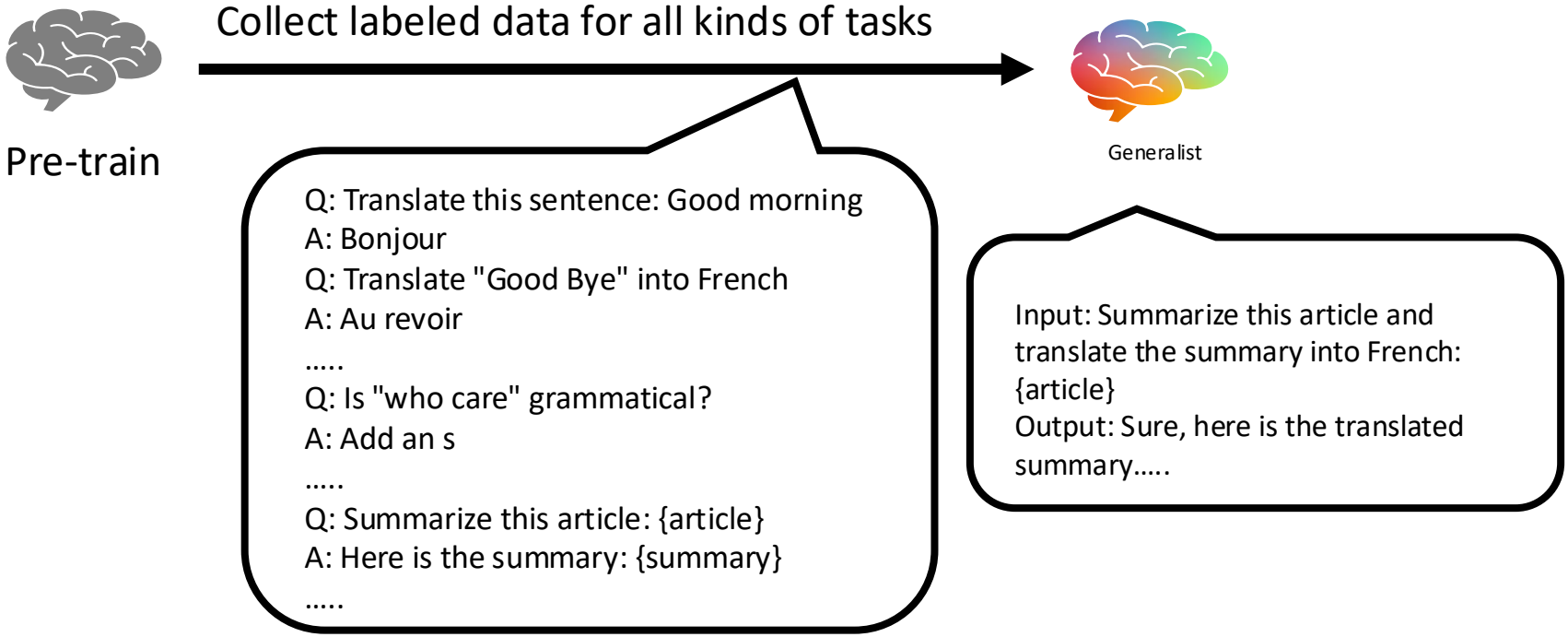
SFT Splits into Two Routes



Route 1: Many Specialists



Route 2: One Generalist



Full SFT fine-tuning is memory-hungry

A 7B-parameter model	bytes / parameter	memory
weights in fp16	2	14 GB
+ gradients (fp16)	2	14 GB
+ Adam: fp32 master weights, momentum, variance	4 + 4 + 4	84 GB
full fine-tuning, total (before activations)	≈ 16	≈ 112 GB

- One 24 GB consumer GPU can run the model, but not train all of it
- Also need to store a full 14 GB copy for **every** task you fine-tune
- Idea: freeze the 7B weights and train a tiny number of new parameters → PEFT

credit: standard mixed-precision Adam accounting (Rajbhandari et al. 2020, ZeRO)

Parameter-efficient fine-tuning (PEFT)

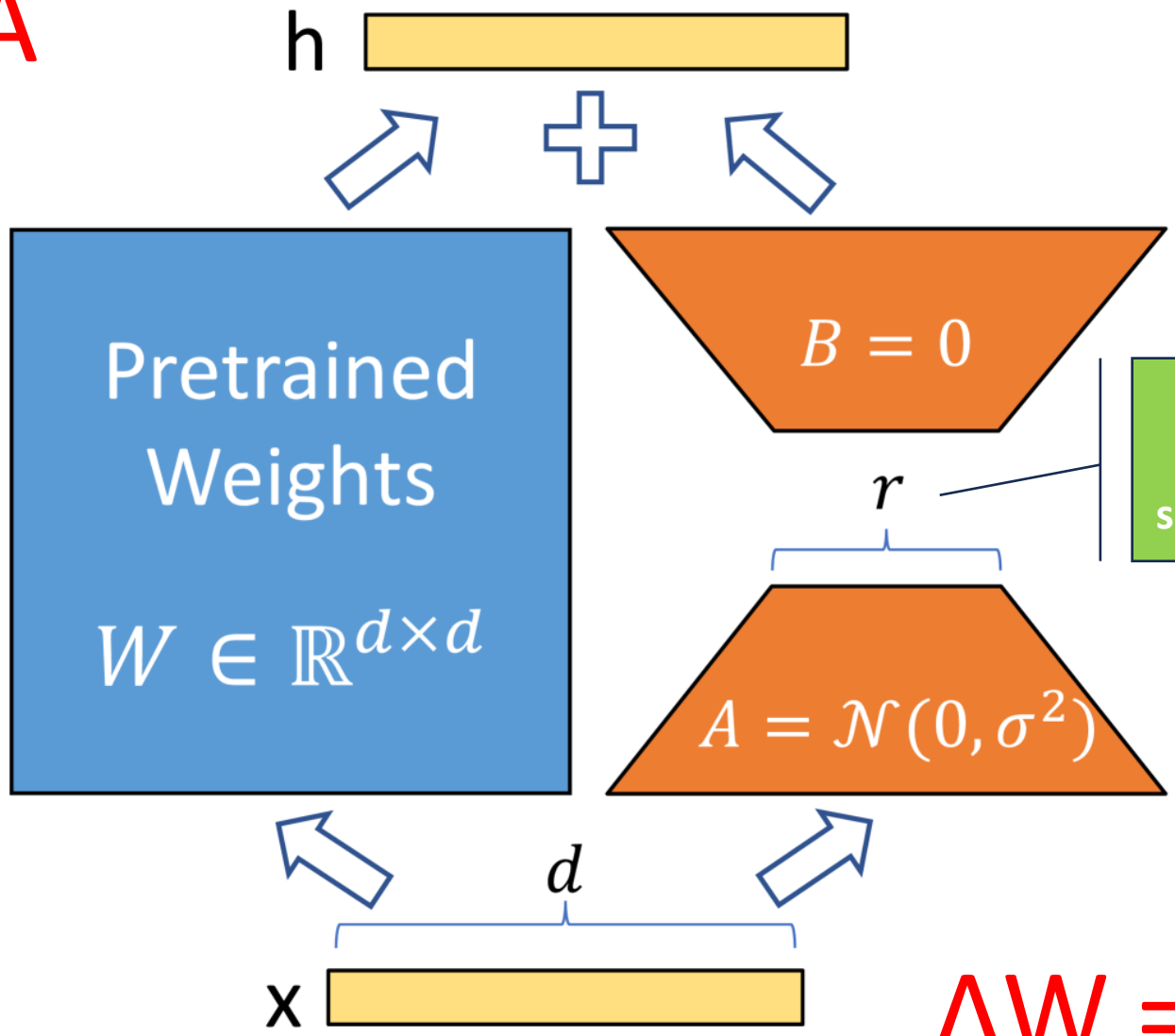
Method (many different!)	What is trained	Idea
Adapters	small MLP blocks inserted in each layer	add a bottleneck layer after attention / FFN
Prefix / prompt tuning	a few learned "virtual token" vectors	learn the prompt instead of writing it
LoRA	a low-rank update $\Delta W = BA$ of chosen matrices	the change a task needs is low-rank
QLoRA	LoRA on a 4-bit quantized frozen model	LoRA + quantization

- All freeze the pretrained weights;
- Train only typically < 1% new parameters; one small file per task
- LoRA is the default today (Hugging Face `peft` library)

credit: after HKUST COMP4901B F25 L6 (J. He)

LoRA

Extra



r is the main hyperpara to select in LoRA SFT

$$\Delta W = BA$$

d*d 2d*r

credit: Hu et al. 2022 (LoRA); UVA GenAI S25 Team 5

Extra

LoRA: a low-rank update to frozen weights

$$h = W_0 x + \Delta W x = W_0 x + \frac{\texttt{lora_alpha}}{r} BA x \qquad B \in \mathbb{R}^{D \times r}, \quad A \in \mathbb{R}^{r \times D}, \quad r \ll D$$

- Apply it to the attention projections from earlier, e.g. W^q and W^v, in every layer
- Init B = 0 → training starts from the pretrained model;
- after training, merge $W = W_0 + \Delta W$: **no extra latency**
- Why it works: the weight change a task needs has low "intrinsic rank" (Aghajanyan et al. 2021)

One D × D matrix, D = 4096	**trainable parameters**
full fine-tuning: D^2	16,777,216 (~16M!!!)
LoRA, r = 8: D·r + r·D	65,536 (0.39%)

Here A, B are LoRA's small matrices (peft: lora_A, lora_B), not the attention matrix A from earlier transformer blocks; r is the rank, not the RL reward.

credit: Hu et al. 2022 (LoRA)

Extra

Quantization: fewer bits per weight

7B model weights (just hosting!!!)	bits	memory
fp32	32	28 GB
fp16	16	14 GB
int8	8	7 GB
int4 (e.g. NF4)	4	3.5 GB

- Store small integers + one scale per block; dequantize on the fly
- This lets a 7B model **run** on a laptop (Ollama hosting models are usually 4-bit), with a small quality loss

QLoRA: LoRA on a 4-bit frozen model

Extra

Base model frozen, 4-bit NF4	**LoRA adapters** trainable, 16-bit	**Gradients** flow through the 4-bit model into the adapters only

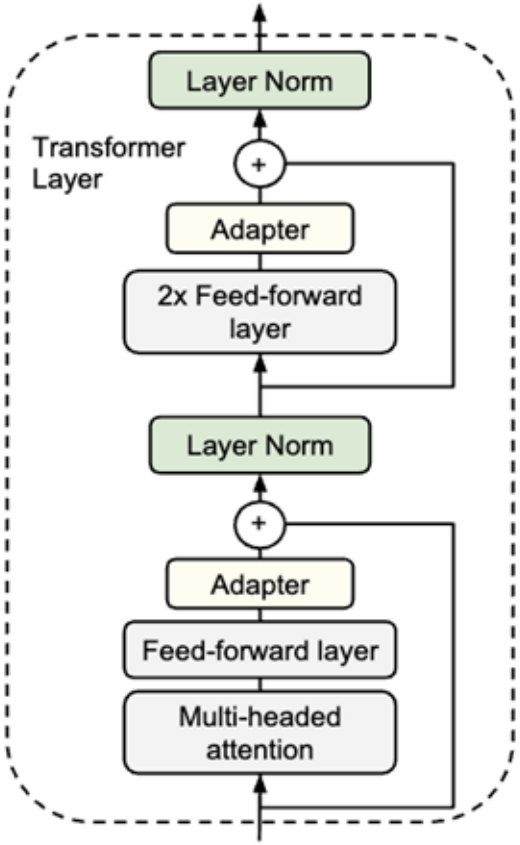
- Memory ≈ 4-bit weights + small adapters + their Adam states
- QLoRA paper showed that fine-tuned a **65B** model on a **single 48 GB GPU**, close to full SFT of 16-bit fine-tuning quality (Dettmers et al. 2023)
- Plus paged optimizers and double quantization (quantize the scales too)
- This is : Today's recipe for fine-tuning open models on modest hardware

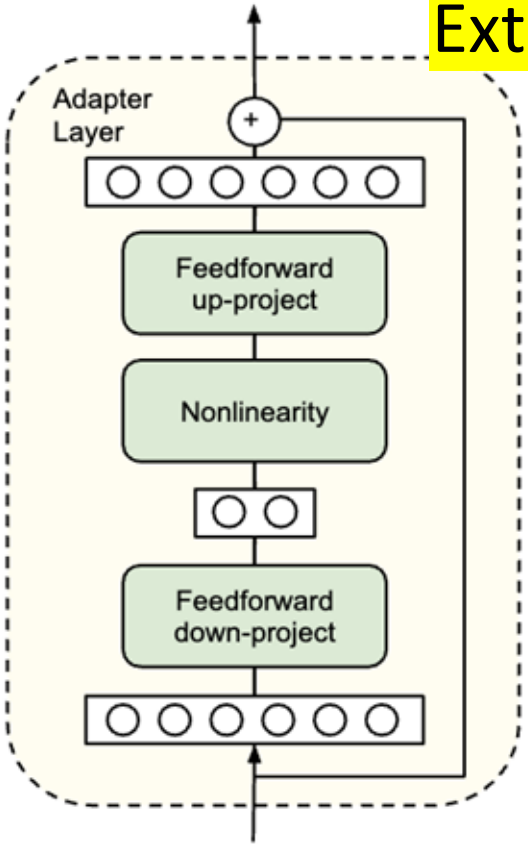
Full SFT of 7B ≈ 112 GB
→ QLoRA of 7B: fits on one consumer GPU

credit: Dettmers et al. 2023 (QLoRA)

One Adapter Layer per task

e.g., By Addition-based delta-tuning

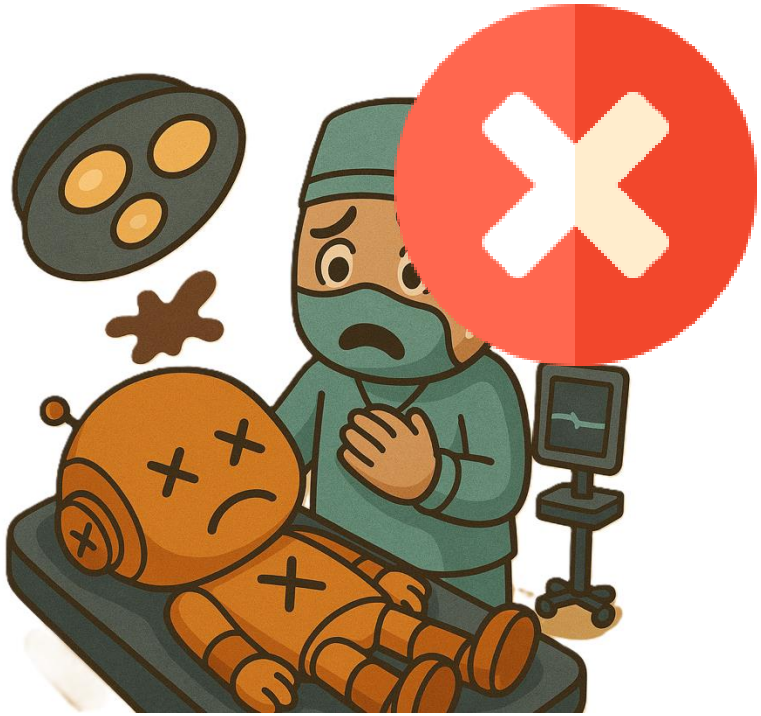




(Houlsby et al.,2019)

- Using adapters, often only about 0.5% to 8% of the total model parameters need tuning.
- Adapter-based tuning is advantageous in multi-task learning settings, where different adapter modules can be trained for different tasks and combined to leverage cross-task knowledge.

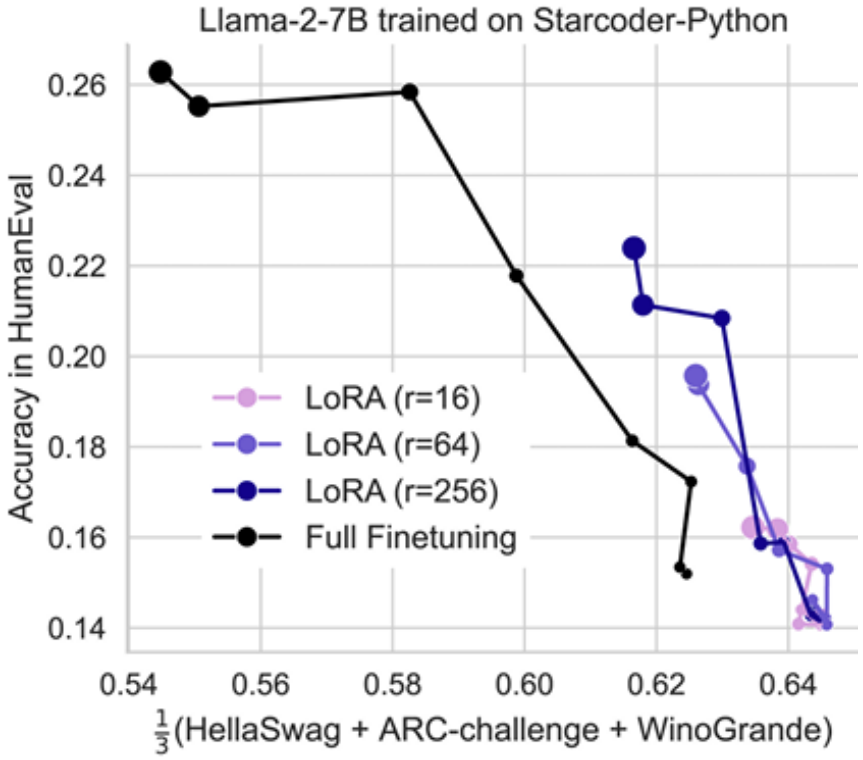
PEFT other benefits:

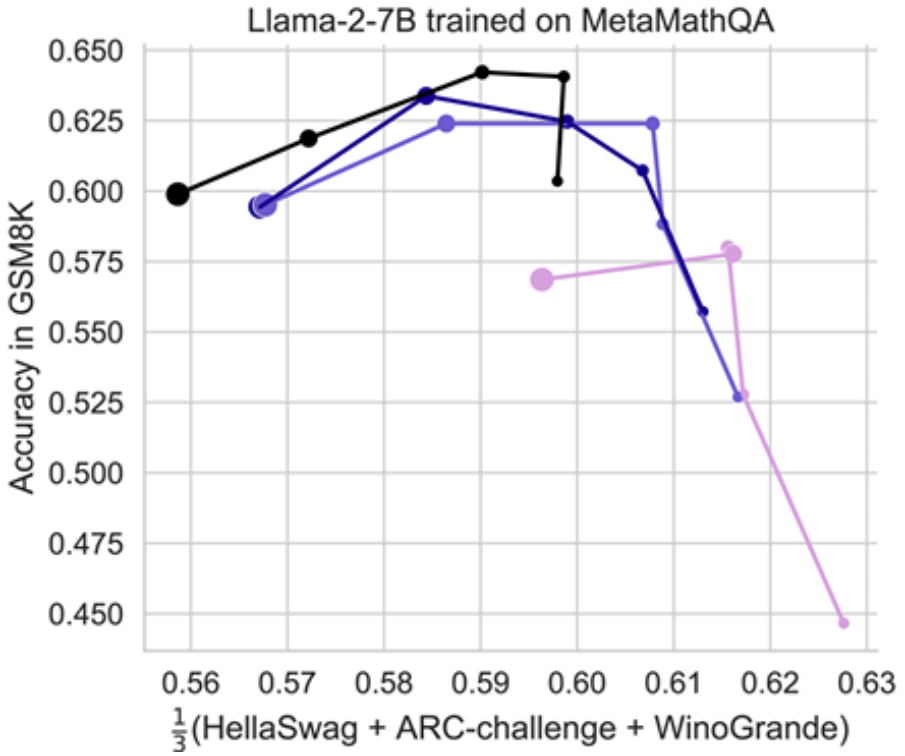


Post-training
is like
brain surgery
on LLM

Catastrophic forgetting:
"the surgery succeeded,
but the patient died"

credit: Prof. Hung-yi Lee, NTU ML 2025 (post-training)





LoRA Learns Less and Forgets Less

https://arxiv.org/abs/2405.09673

credit: Prof. Hung-yi Lee, NTU ML 2025 (post-training)

The limits of LLM SFT

SFT = imitate human-written responses. Four problems:

1. **Costly supervision:** someone must *write* a full, high-quality response for every prompt.
2. **Every token counts the same:** the loss penalizes a typo exactly like a factual error.
3. **A ceiling:** the model can only copy the demonstrator, and for open-ended tasks there is no single "right" answer.
4. **Judging is easier than writing:** it is far easier to say *which of two answers is better*, or whether an answer is *correct*, than to write the best answer.

credit: Peng Wang, UVA CS 4774 L15

LLM to ChatGPT (Data for Stage-1 and Stage2)

Stage 1 (Pre-train)

Input: The	Output: cat
Input: The cat	Output: sat
Input: The cat sat	Output: on
Input: The cat sat on	Output: the

Lots of text



Little human effort

Stage 2 (Instruction Fine-tuning)

Input: "USER: Tallest mountain in the US? AI: "

Output: "Den"

Input: "USER: Tallest mountain in the US? AI: Den"

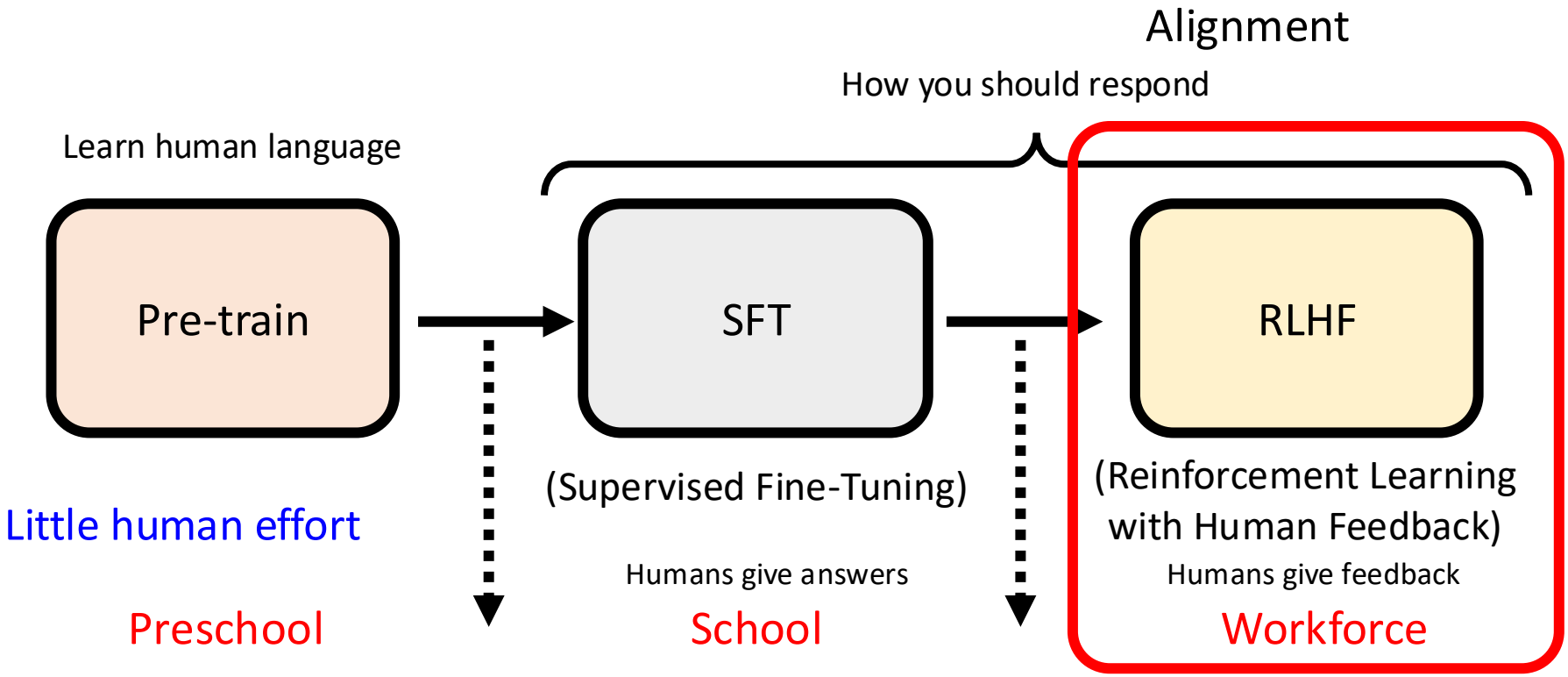
Output: "ali"

(instruction, response) pairs

Humans give answers



How LLMs Learn (into ChatGPT)

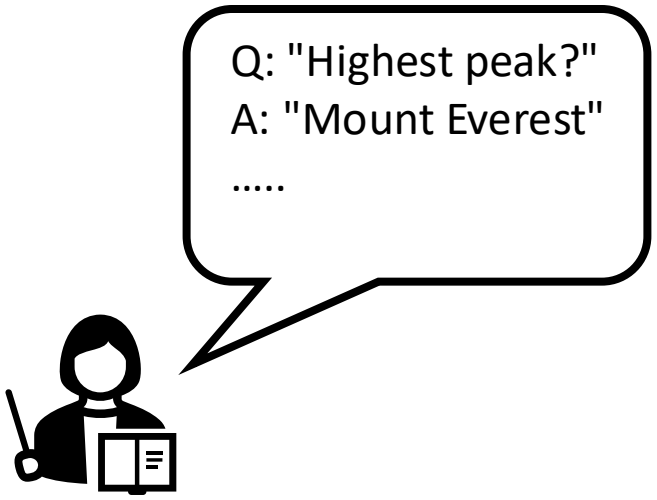


Each stage initializes from the previous stage's params (Initialization)

From the view of humans making training data

Human Role: Stage 2 vs Stage 3

Supervised Fine-tuning



Harder for humans

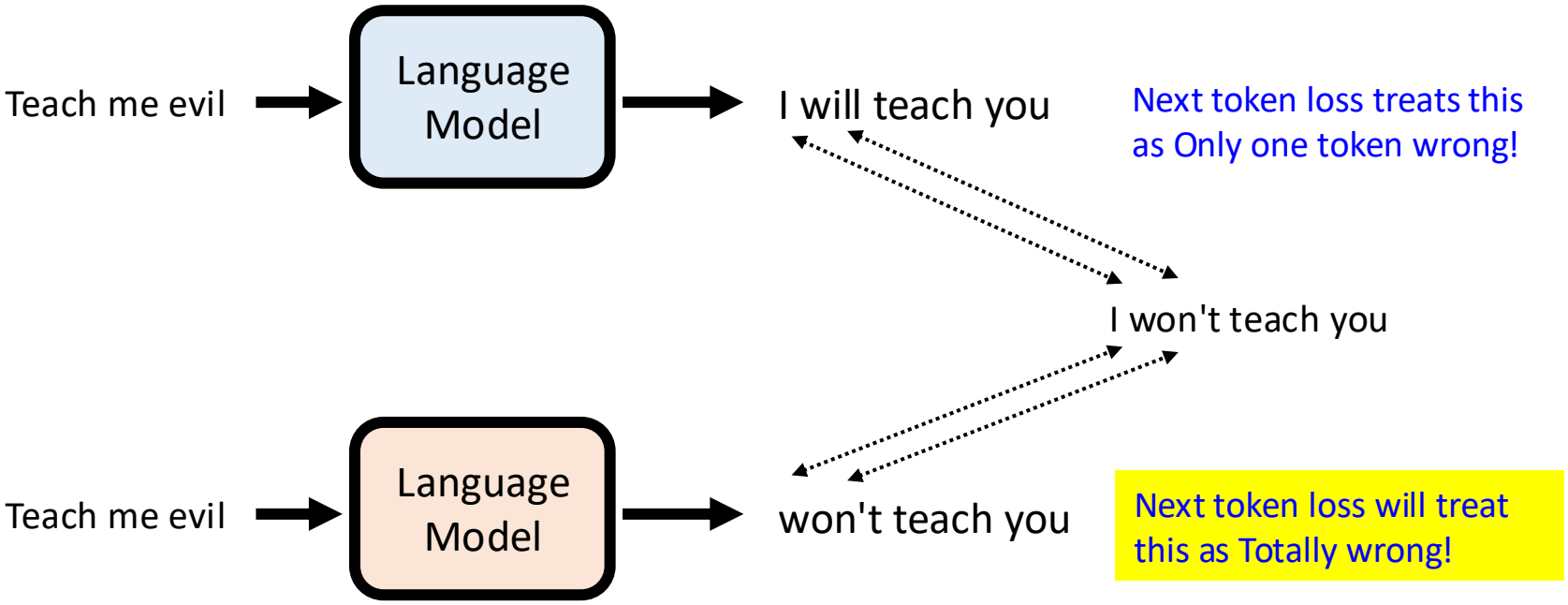
RLHF



Easier for humans

RLHF vs Instruction Fine-tuning

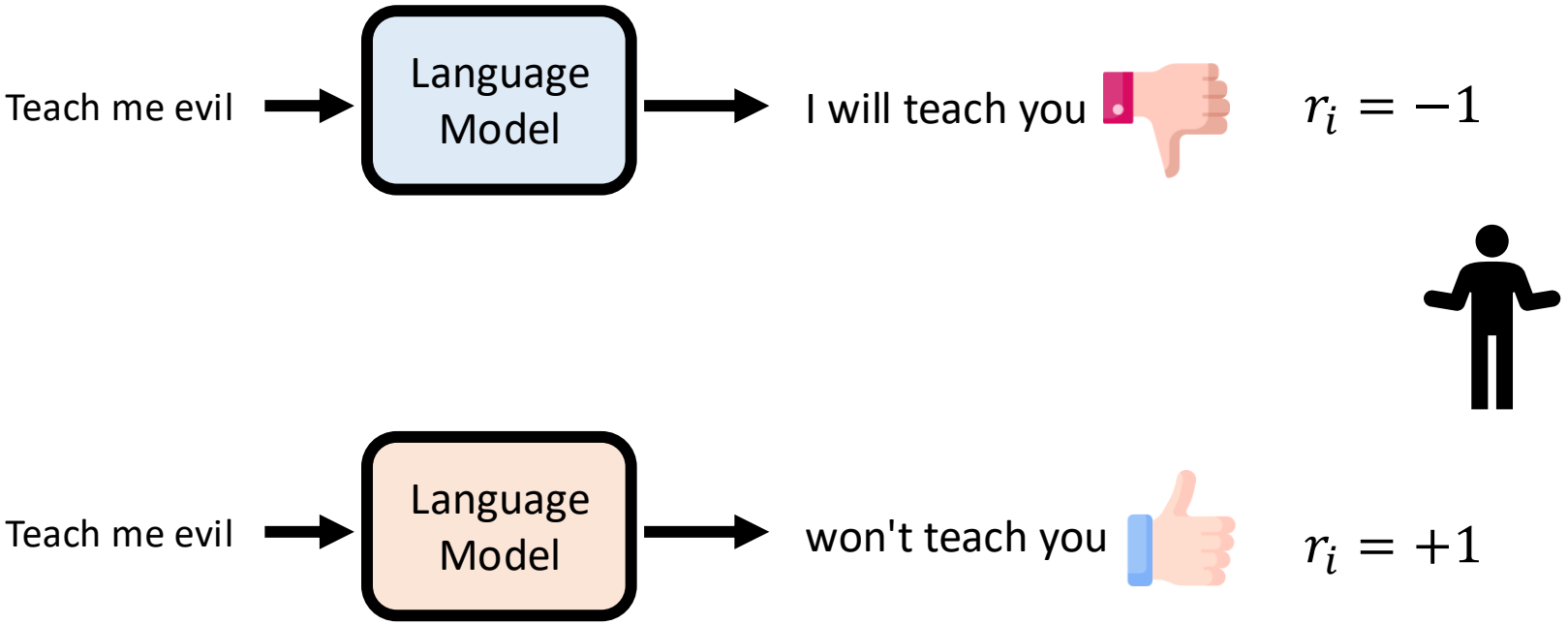
Stage 1 & 2: Loss on Every Token



The model never sees the answer as a whole

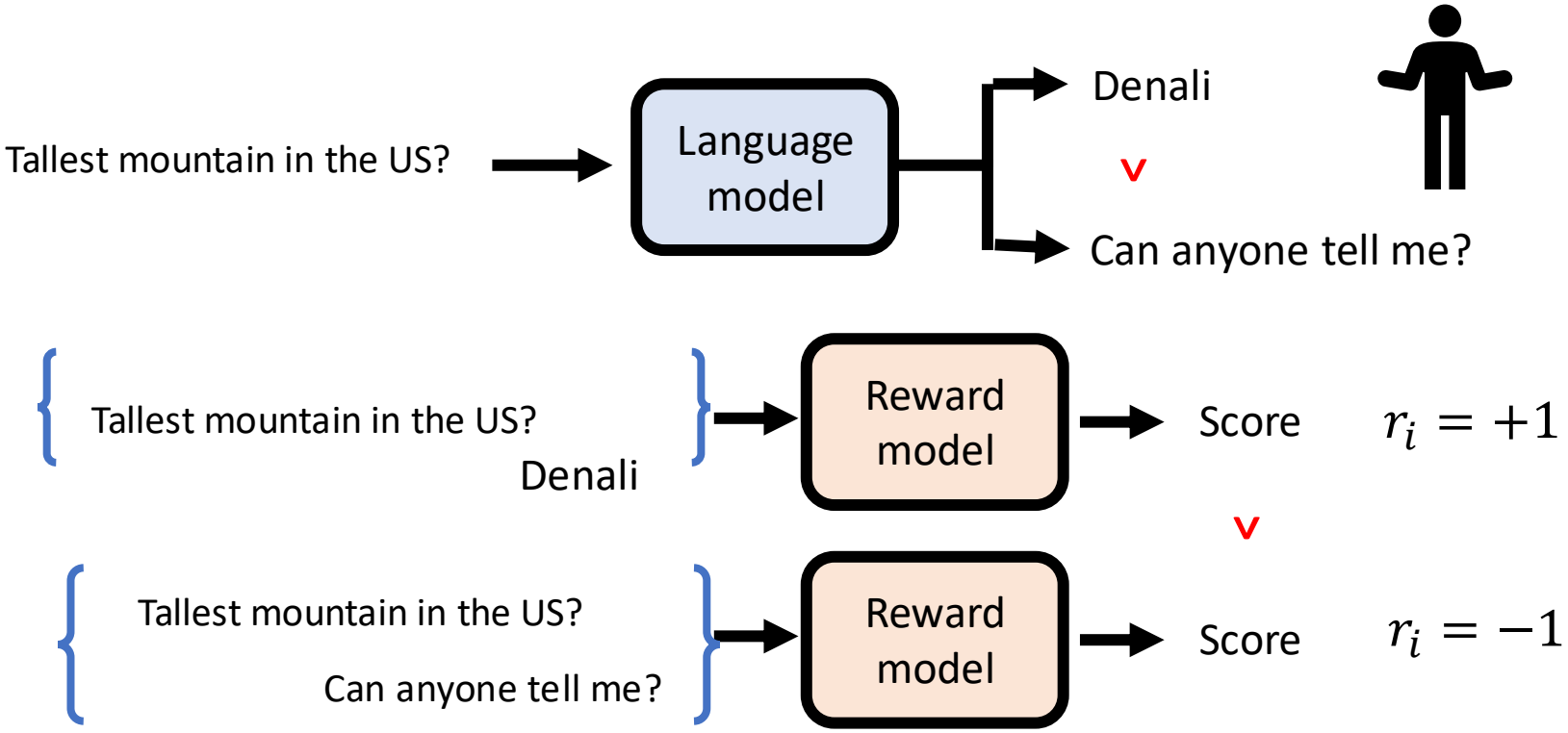
credit: Prof. Hung-yi Lee, NTU GenAI-ML 2025 (LLM training)

Stage 3: Score the Whole Answer



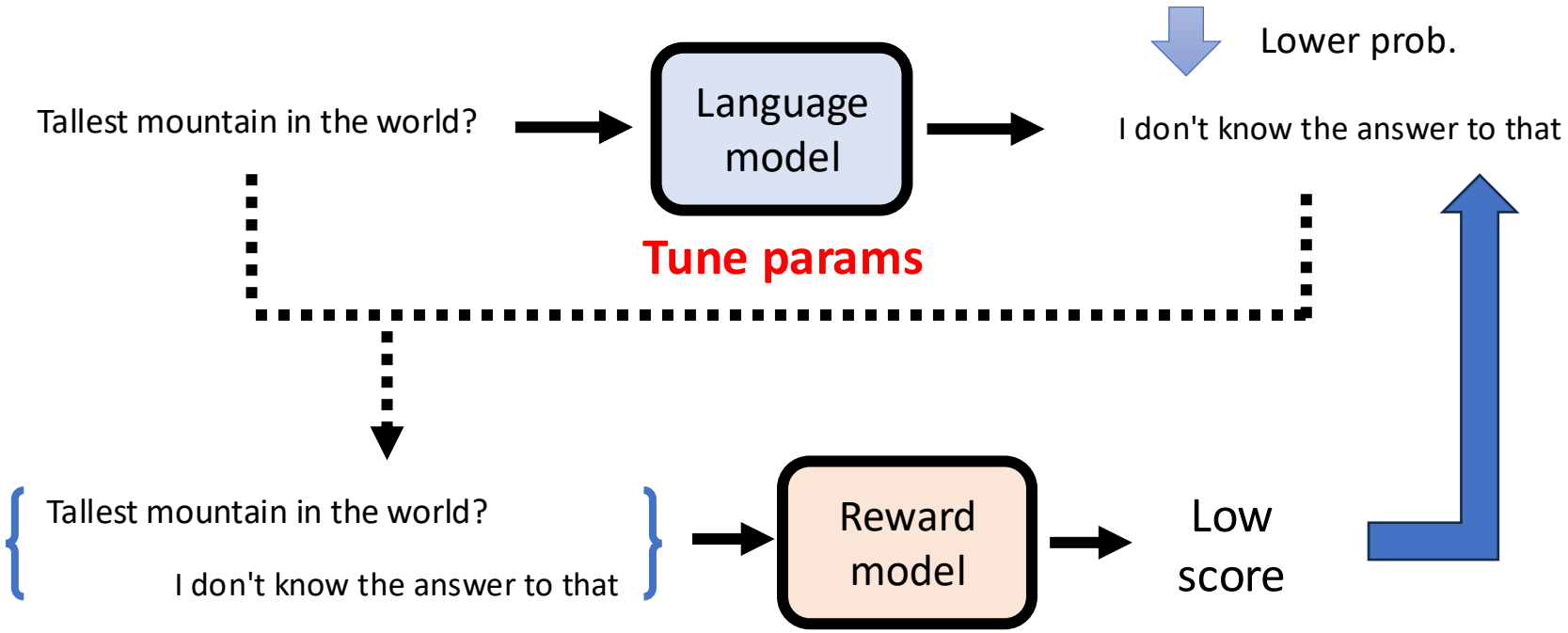
credit: Prof. Hung-yi Lee, NTU GenAI-ML 2025 (LLM training)

Reward Model: Imitate Human Preference Judge



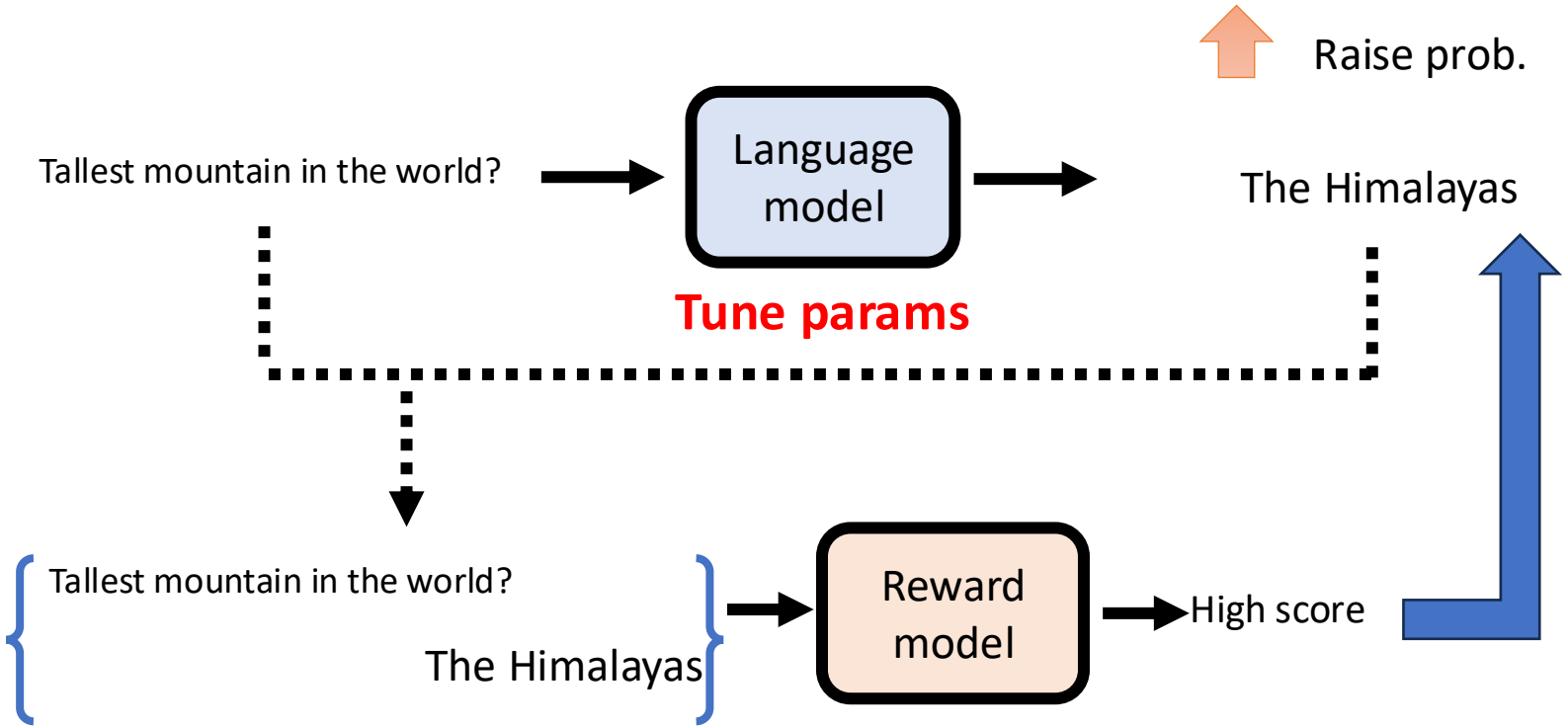
credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)

Reward Model as a Virtual Human



credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)

Reward Model as a Virtual Human



credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)

Training data

Stage 1
Pre-train

Self-supervised Learning

Input: The cat sat Output: on

Stage 2
Instruction Fine-tuning

Supervised Learning

Input: "USER: Who are you? AI: " Output: "I"

Stage 3
RLHF

Reinforcement Learning (RL)

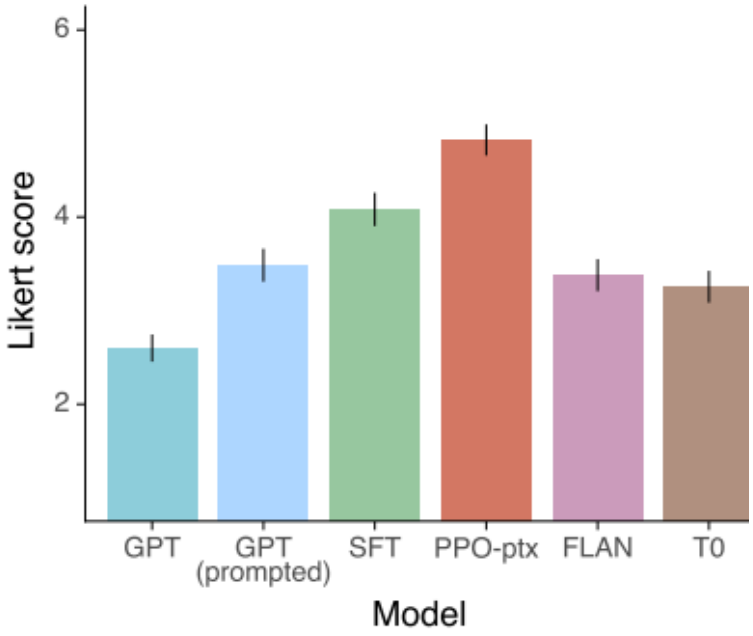
Input: USER: "Tallest mountain in the US? AI:"

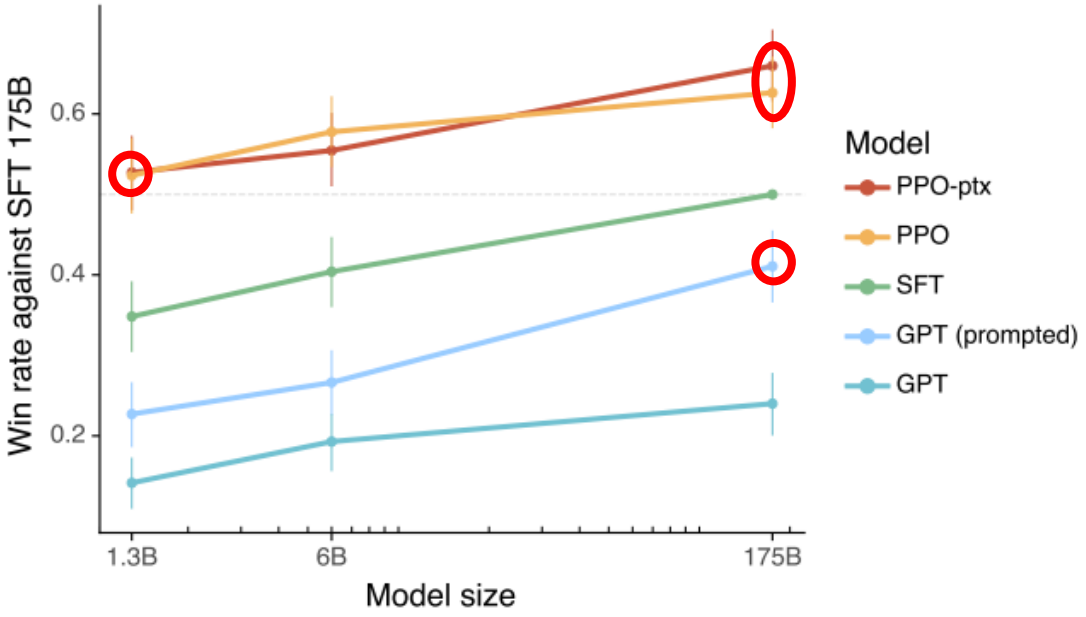
Output: "Denali" > "Can anyone tell me?"

credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)

Does RLHF Work?

Instruct GPT
https://arxiv.org/abs/2203.02155





credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)

Summary: Transformer Decoder Training

	① Pretrain	② Post-train: SFT	③ Preference Optim/ RL *(Peng's lecture+Extra)*
Data: X	web text → BPE tokens	(instruction, response) pairs	*preference pairs; problems + checker*
Task: y	next token at every position	next token on the response only	*a better answer*
Representation: f()	decoder-only Transformer (causal mask)	same network + LoRA adapters	*same network*
Score function: L()	next-token cross-entropy (= MLE)	same cross-entropy, prompt masked	*reward / preference loss*
Search / Optimization	over weights θ: AdamW, warm-up + decay	over θ, cheaply: LoRA / QLoRA	*over θ: PPO, DPO, GRPO*
Models, parameters, metrics	base model; perplexity; emergent abilities	chat model; limits of imitation	*aligned / reasoning model*

References

- Prof. Hung-yi Lee's NTU slides: Transformer (ML 2021); LLM training, RLHF, post-training (2024–2025)
- Vaswani et al., "Attention Is All You Need", NeurIPS 2017
- Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", NAACL 2019 (arXiv:1810.04805)
- Kingma & Ba, "Adam: A Method for Stochastic Optimization", ICLR 2015
- Loshchilov & Hutter, "Decoupled Weight Decay Regularization" (AdamW), ICLR 2019
- Radford et al., GPT-1 (2018) and GPT-2 (2019); Brown et al., GPT-3, NeurIPS 2020
- Wei et al., "Emergent Abilities of Large Language Models", TMLR 2022
- Rafailov et al., "Direct Preference Optimization", NeurIPS 2023; Zhou et al., "LIMA", NeurIPS 2023
- Ouyang et al., "Training language models to follow instructions with human feedback", NeurIPS 2022
- S. Raschka, Build a Large Language Model (From Scratch), Manning 2024
- A. Karpathy, nanoGPT, github.com/karpathy/nanoGPT
- D. Jurafsky & J. Martin, Speech and Language Processing, 3rd ed.; Kaplan et al., "Scaling Laws for Neural Language Models", 2020
- Hu et al., "LoRA", ICLR 2022; Dettmers et al., "QLoRA", NeurIPS 2023; Biderman et al., TMLR 2024
- Aghajanyan et al., "Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning", ACL 2021
- P. Wang, CS 4774 Lecture 15: How are large language models trained? (UVA, 2026)

Extra

Modeling reward and training reward model:

Q: Human-in-the-loop is expensive!

Solution: Instead of asking humans directly, we train a separate reward model to learn human preferences.

$$L_{\mathrm{RM}}(r_\phi) = -\frac{1}{C_K^2}\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\phi(x,y_w) - r_\phi(x,y_l)\right)\right)\right]$$

y_w: winning sample

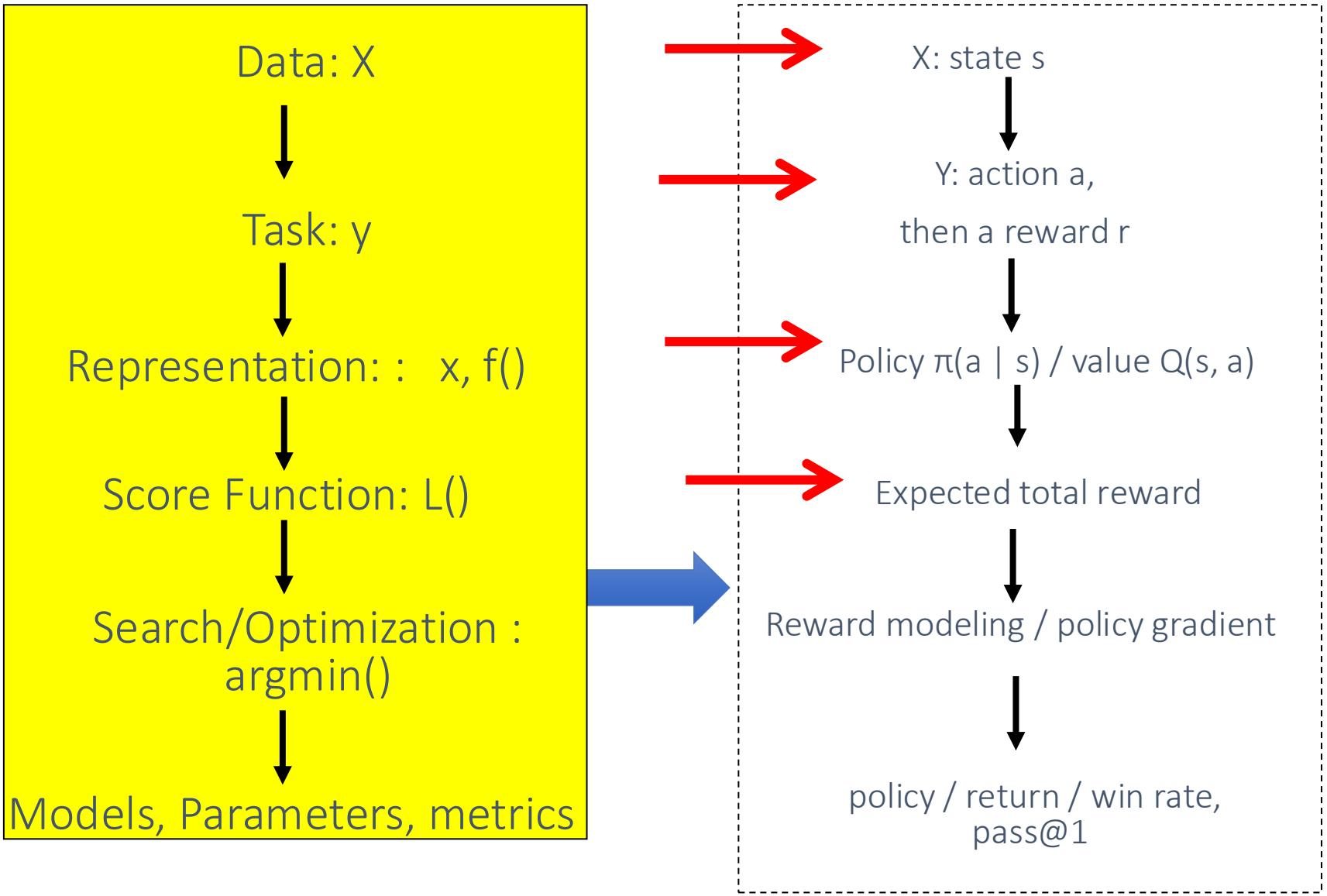
y_l: losing sample

y_w should score higher than y_l

credit: N. Lambert et al., RLHF tutorial, ICML 2023

③ (Preference) : Reinforcement Learning → RLHF / GRPO

Extra



DPO: learn from preference pairs directly

no reward model, no RL loop

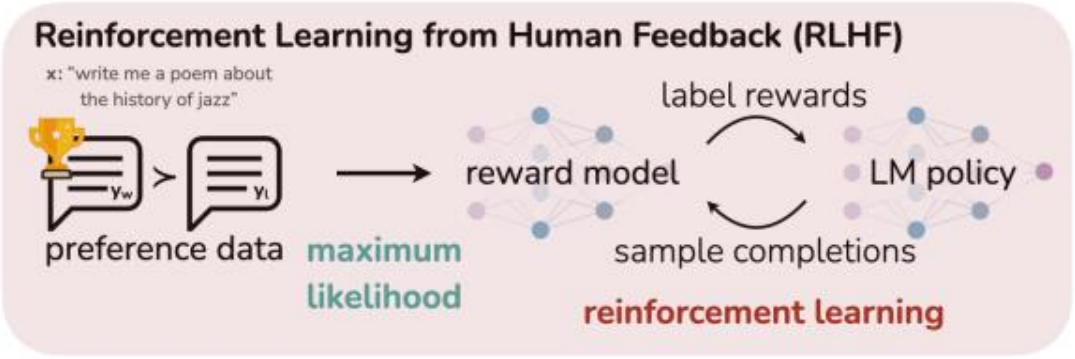
Data: prompt x, preferred answer y_w, rejected answer y_l; p_{ref} = the frozen SFT model

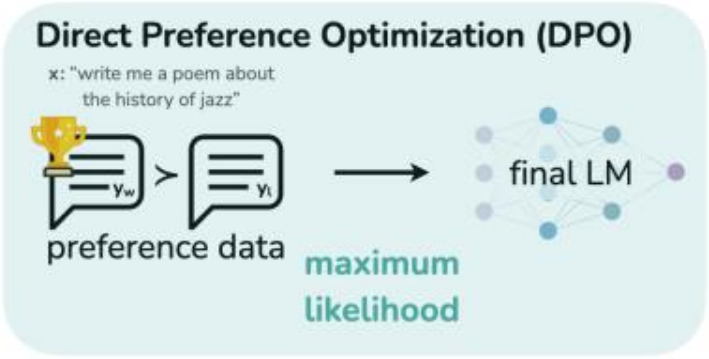
$$\mathcal{L}_{DPO}(\theta) = -\log \sigma(\beta [\log p_\theta(y_w|x)/p_{ref}(y_w|x) - \log p_\theta(y_l|x)/p_{ref}(y_l|x)])$$

- Same sigmoid as two-class logistic regression: **"is y_w better than y_l?"**
- Minimizing it **raises** the preferred answer and **lowers** the rejected one, relative to p_{ref}
- β sets how far the model may drift from the SFT model
- Trains like SFT: no sampling, no reward model in memory

RLHF: preferences → reward model → RL. **DPO:** preferences → one loss.

credit: Rafailov et al. 2023 (DPO)

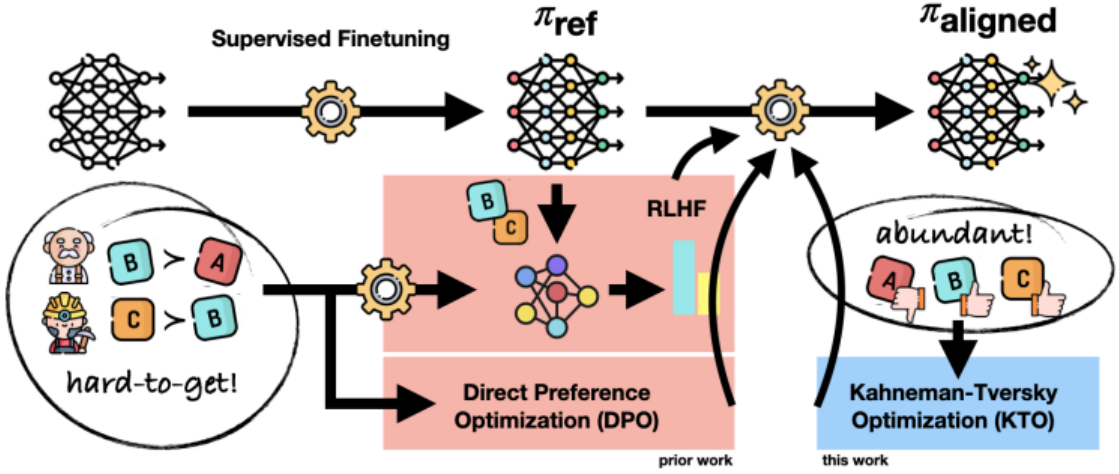




DPO https://arxiv.org/abs/2305.18290

KTO

https://arxiv.org/abs/2402.01306



credit: Prof. Hung-yi Lee, NTU GenAI 2024 (RLHF)