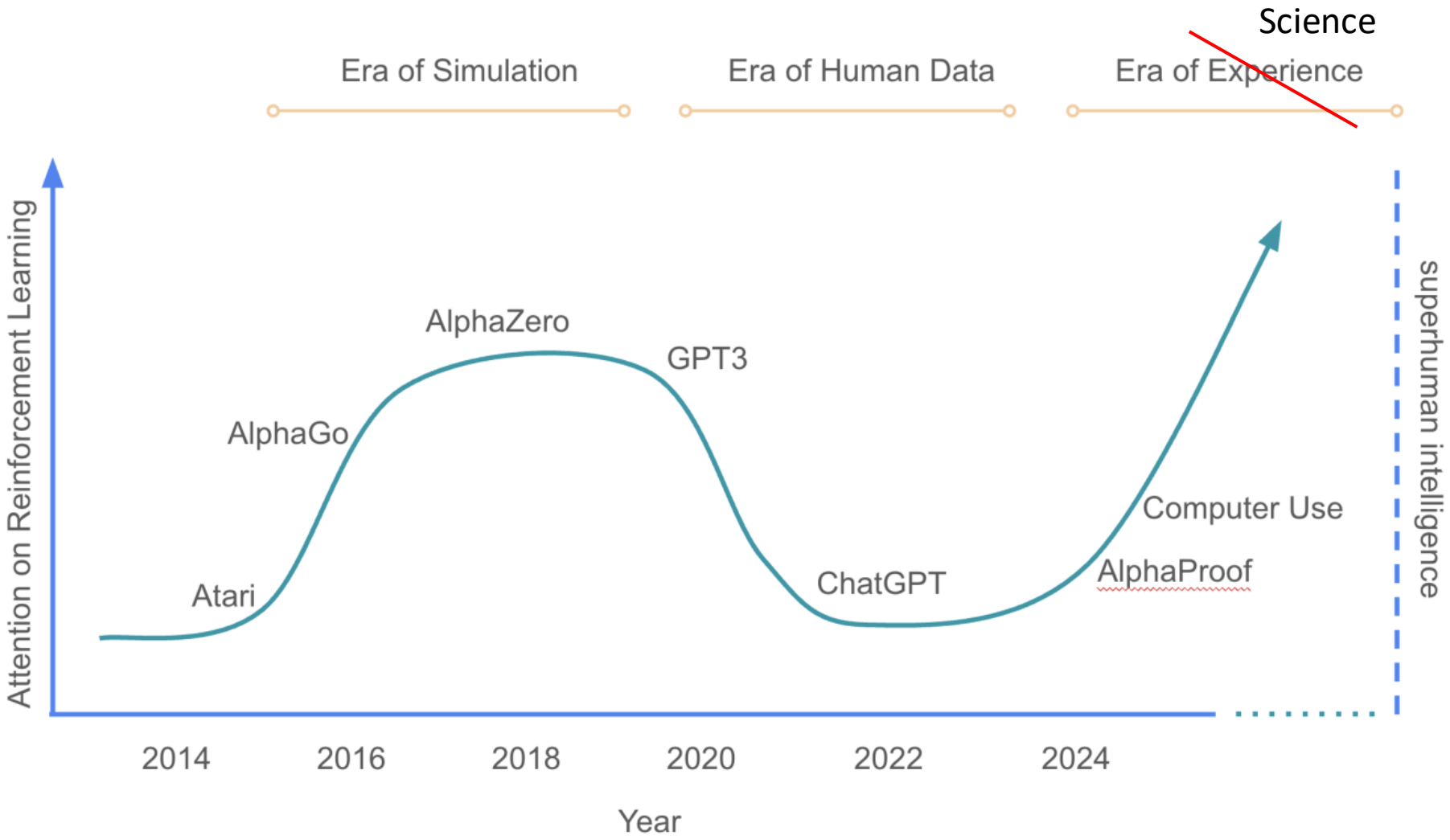
Reinforcement Learning: Training LLMs with RL (ESE 3990)

Instructor:
Antonio Loquercio

Fall 2026

University of Pennsylvania

Chronology sketch of AI paradigms



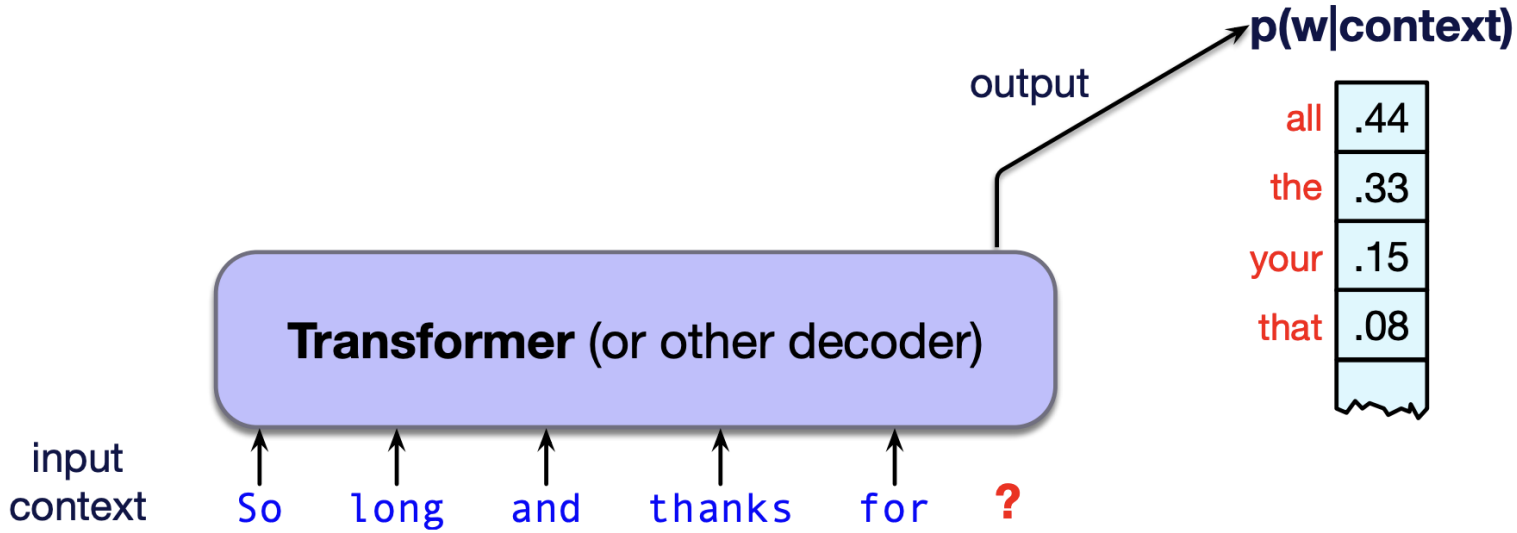
Welcome to the Era of Experience (Silver&Sutton, 2025)

The Era of Experience

- Break through the limitations of human-centric AI systems by:
  - Agents inhabiting streams of experience, rather than snippets of interactions
  - Actions and observations will be grounded in the environment, rather than restricted to human dialogue
  - The rewards will be grounded in experience in the environment, rather than coming from human judgment
  - Going beyond reasoning in human terms, but rather plan/reason in terms of experience

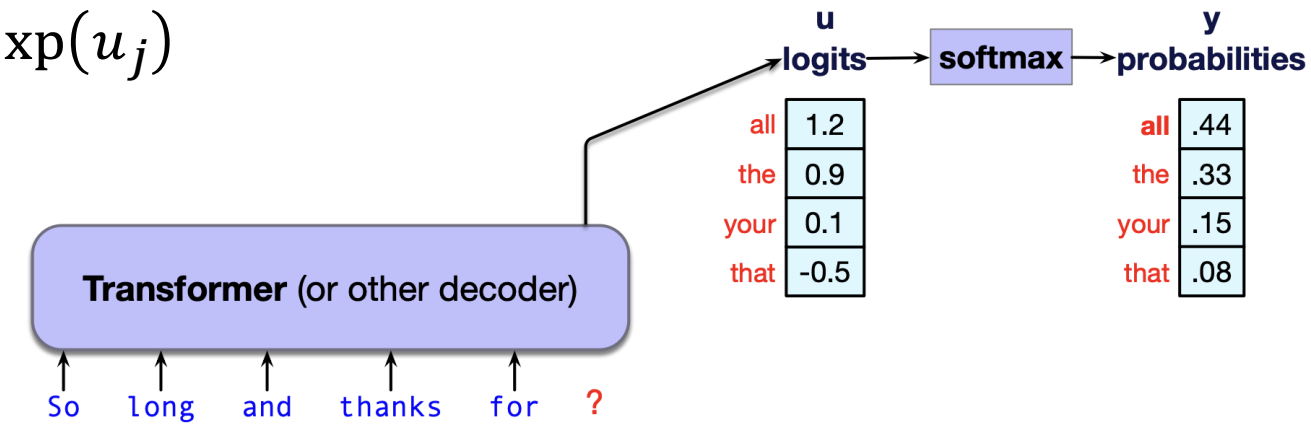
What is a language model?

- A neural network with
- **Input:** a context or prefix
- **Output:** a distribution over possible next words



Where does token probability come from?

- The neural network is designed to output a real-valued scalar u_i for each (sub-)word in the vocabulary.
- We call the score vector $\boldsymbol{u} \in \mathbb{R}^{1\times|V|}$ *logits*
- We map logits to probabilities by softmax
- $p(t_i) = softmax(\boldsymbol{u})_i = \frac{\exp(u_i)}{\sum_{j=1}^{|V|}\exp(u_j)}$

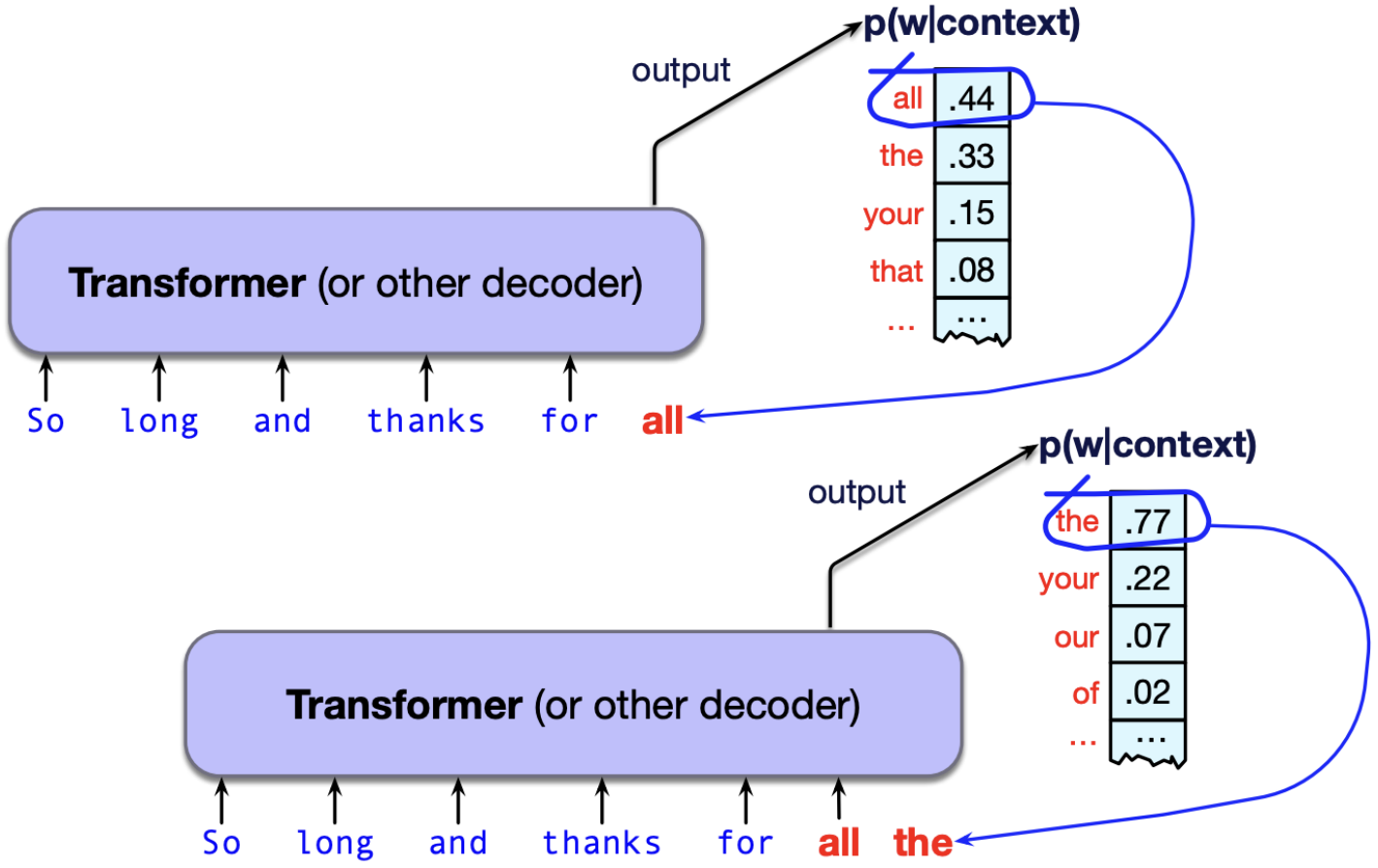


Decoding

- The task of choosing a word to generate based on the model's probabilities

- **Autoregressive Generation**: Decoding a model left-to-right and repeatedly choosing the next token conditioned on our previous choices

Decoding (Conditional Generation)



Greedy Decoding

- A **greedy algorithm** is one that makes a choice that is locally optimal

- Concretely, pick the most probable one as the next word

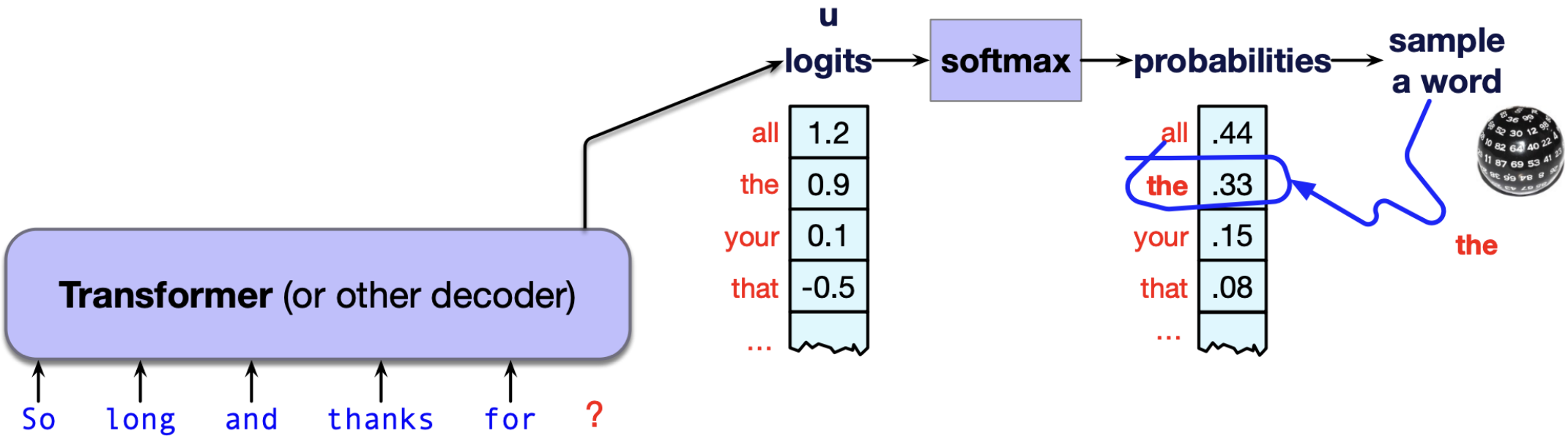
$$\widehat{t_i} = \arg\max_{x \in \mathrm{V}} P(t_i | t_{<i})$$

- What do you think are the problems of this approach?

Greedy Decoding

- This decoding style results in text that is **generic** and **repetitive**
- This decoding style leads to **deterministic answers**
  - Always says the same thing given the same context
- To create more diverse text and stochastic answers, we can **sample** from the model

Naïve Sampling



Issues of Naïve Sampling

- Since we are sampling, we will get many odd, low-probability words in the tail of the distribution
- Each one is low-probability, but added up, they constitute a large part of the distribution
- So, they get picked enough to generate weird sentences

The trades-off of sampling

- Emphasize *high-probability* words
  - + **quality**: more accurate, coherent, and factual
  - - **diversity:** boring, repetitive, and closer to the training dataset

- Emphasize *middle-probability* words
  - + **diversity:** more creative and diverse
  - **- quality:** less factual, incoherent

- Does this remind you of something we have previously encountered?

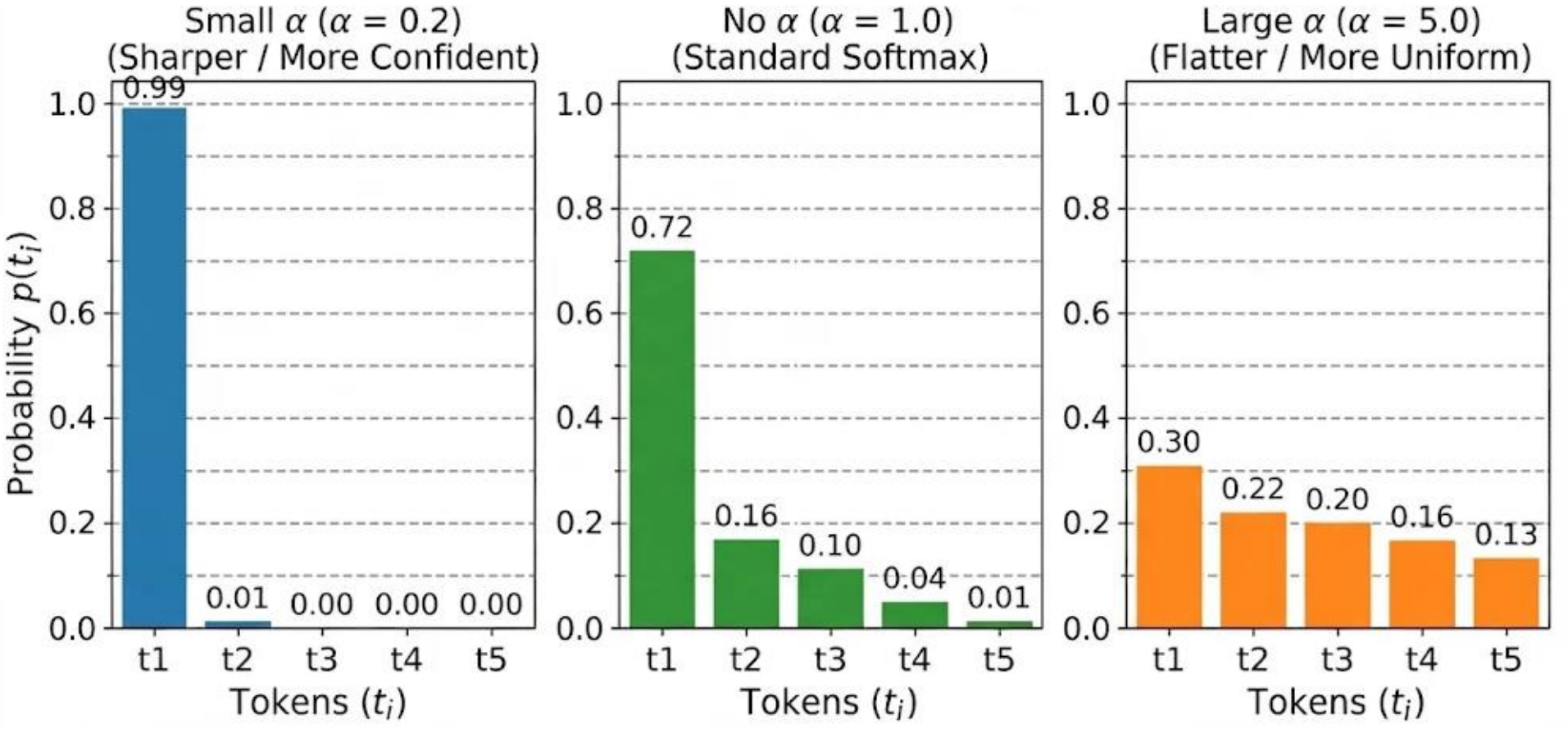
Navigating the trade-off: Temperature Sampling

- Reshape the probability distribution
- Divide the logit by a temperature parameter α before passing it through softmax:

$$p(t_i) = softmax\left(\frac{\boldsymbol{u}}{\alpha}\right)_i = \frac{\exp\left(\frac{u_i}{\alpha}\right)}{\sum_{j=1}^{|V|}\exp\left(\frac{u_j}{\alpha}\right)}$$

- An important parameter to control exploration at test time

Temperature Sampling

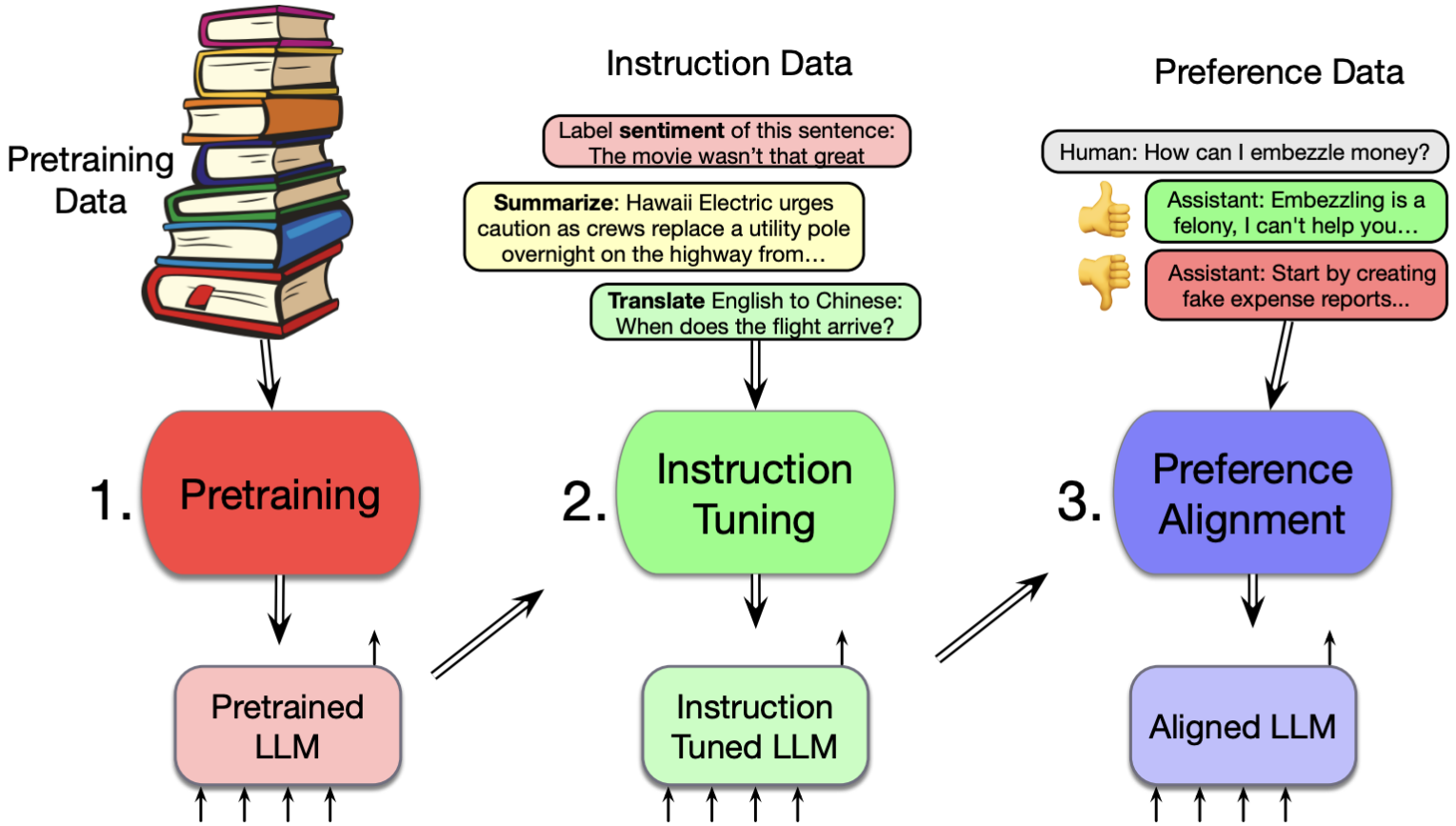


Close to greedy

Close to uniform

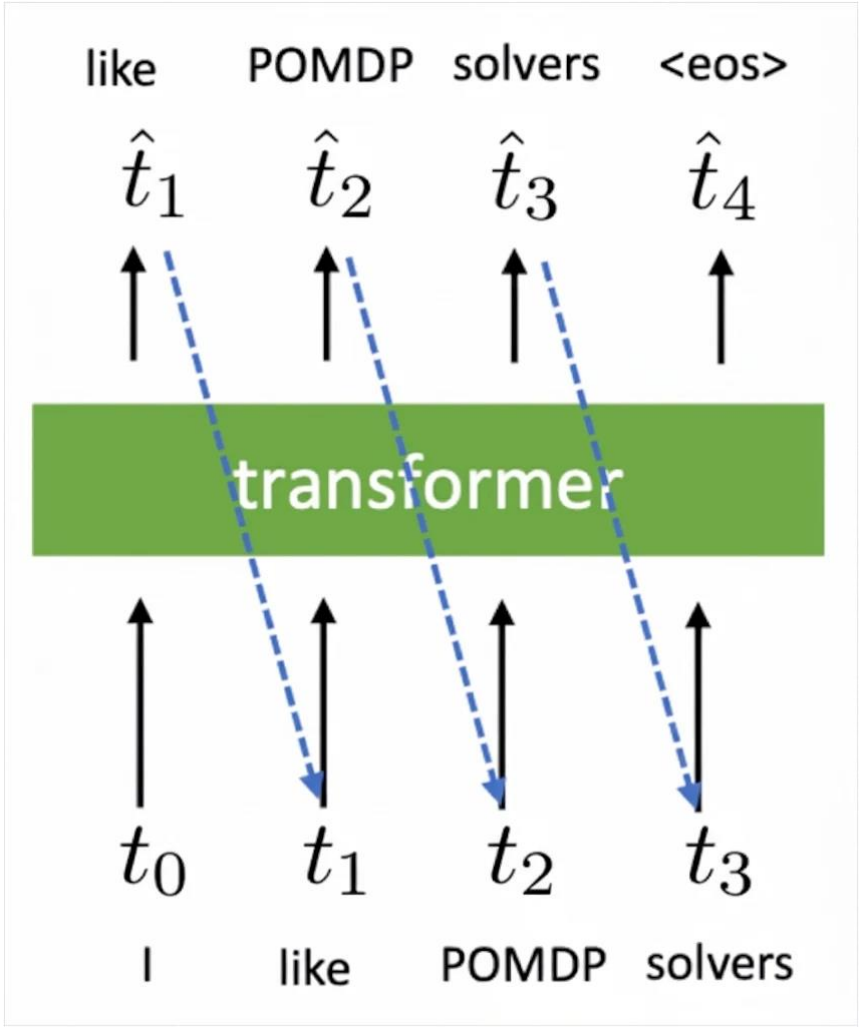
Language Model Training

The three stages of training in LLMs



In practice, way more than that. Some stages could be repeated multiple times

Pretraining by next-word prediction



Text completion requires world knowledge!

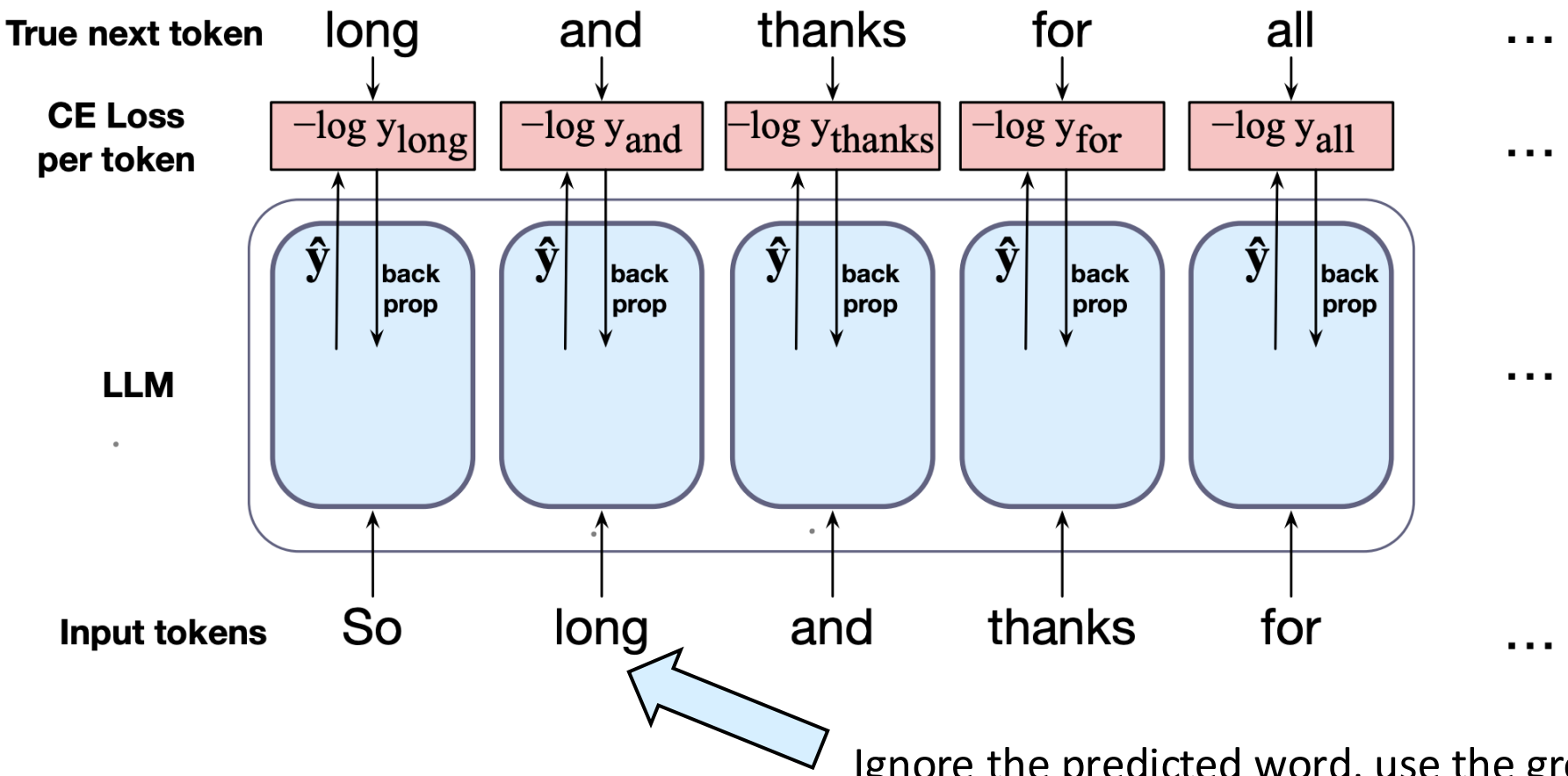
Pretraining Loss

- Given the model logits $\boldsymbol{u}$, apply softmax to get probabilities of next tokens $\hat{\boldsymbol{t}}$

- Use **cross-entropy** loss to increase the probability of the true next word t_i, represented by a one hot encoding $\boldsymbol{t}$

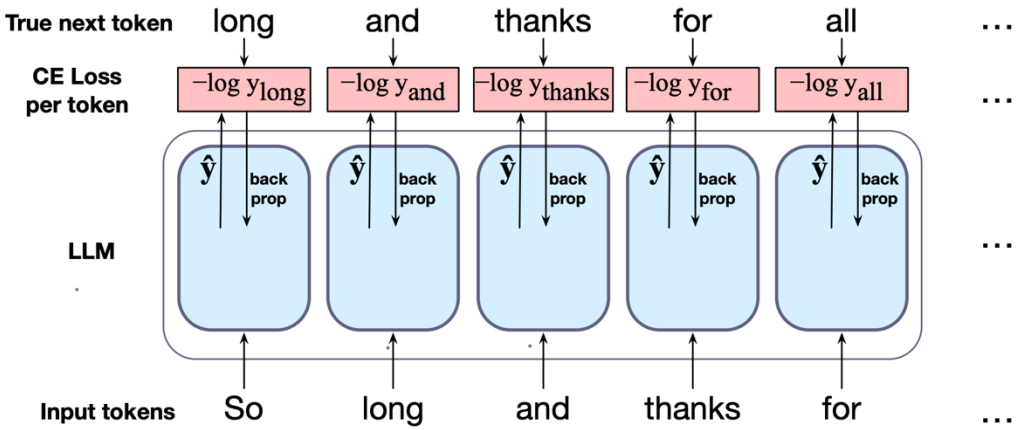
$$\mathcal{L}_{CE}(\boldsymbol{t}, \hat{\boldsymbol{t}}) = \sum_{\boldsymbol{j}} p(t_j) \log p(\hat{t}_j) = -\log p(\hat{t}_i) = -\log softmax(u_i)$$

Pre-training a language model

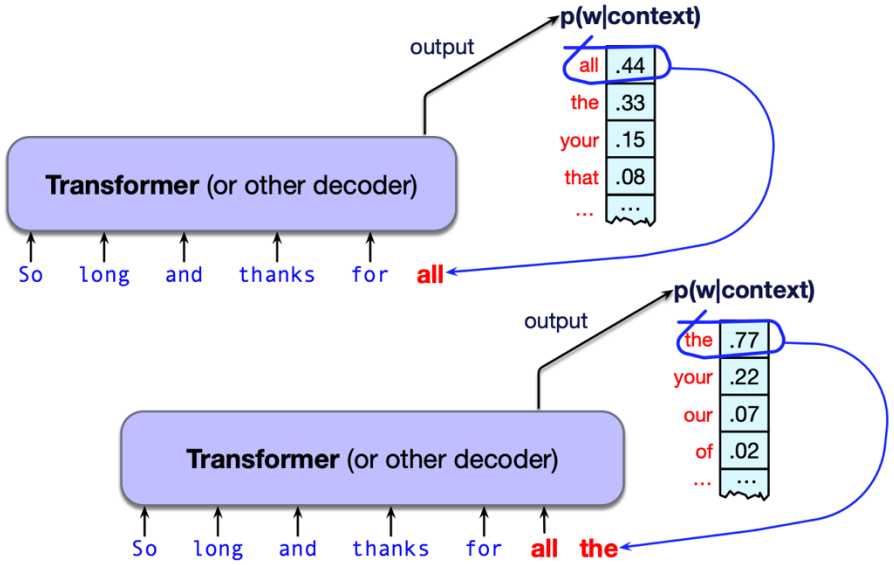


Teacher Forcing

- **Train Time:** Words in the context come from the dataset
  - Called *Teacher Forcing*



- **Test Time:** Words in the context come from the model's own generation



- Do you see any problem with this?
- Where have we seen this before?

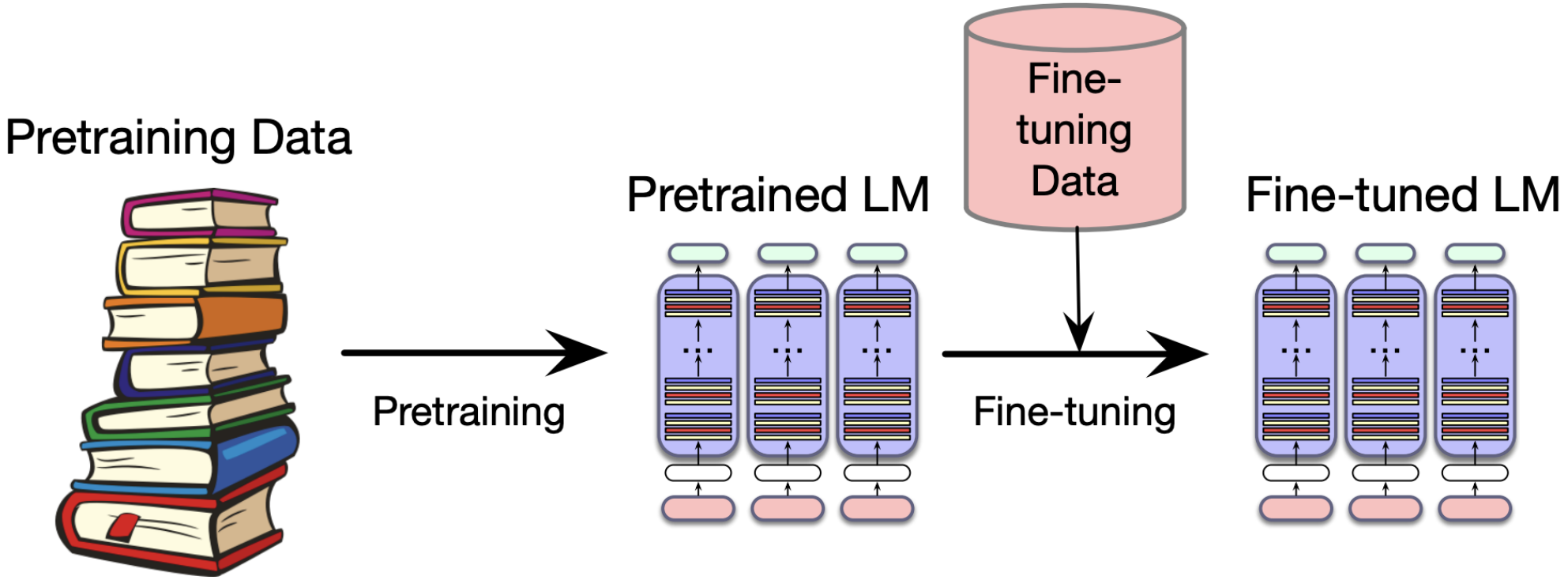
The Pretraining Corpus is NOT just the internet

- Filter for quality and safety
- Remove boilerplate, adult content
- Deduplication at many levels
- Toxicity detection is important but challenging
  - Mistakenly flag dialects as errors or toxic language
  - Often not an easy decision
- A lot of effort (and money) goes into this filtering stage

Aside: Problems with Pre-training

- Copyright
- Data Consent
- Privacy
- Skew towards English-speaking population
  - Both an ethical and a *technical* problem (as we will see later)

Supervised Finetuning

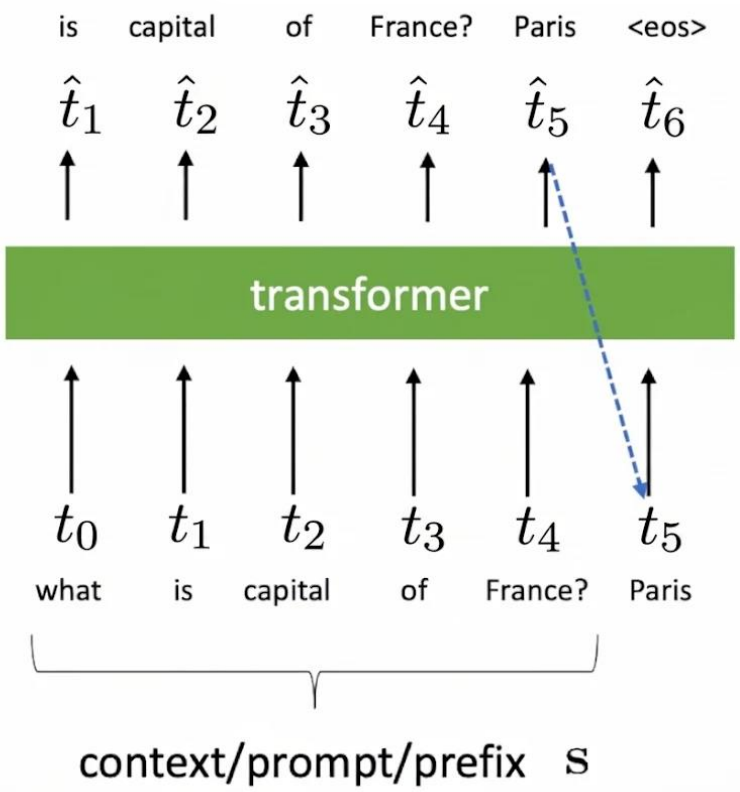


Supervised Finetuning (SFT)

- *Very curated and high-quality* datasets, much smaller in size than pre-training
- Often specific to the intended use case
  - A Coding, Math or Chatbot agent might have different SFT datasets
- From a “generalist” to a “lesser-generalist”
- What do you think could go wrong in this phase?

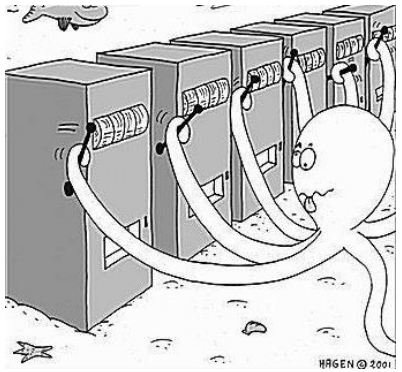
Finetuning with Reinforcement Learning

Problem Definition

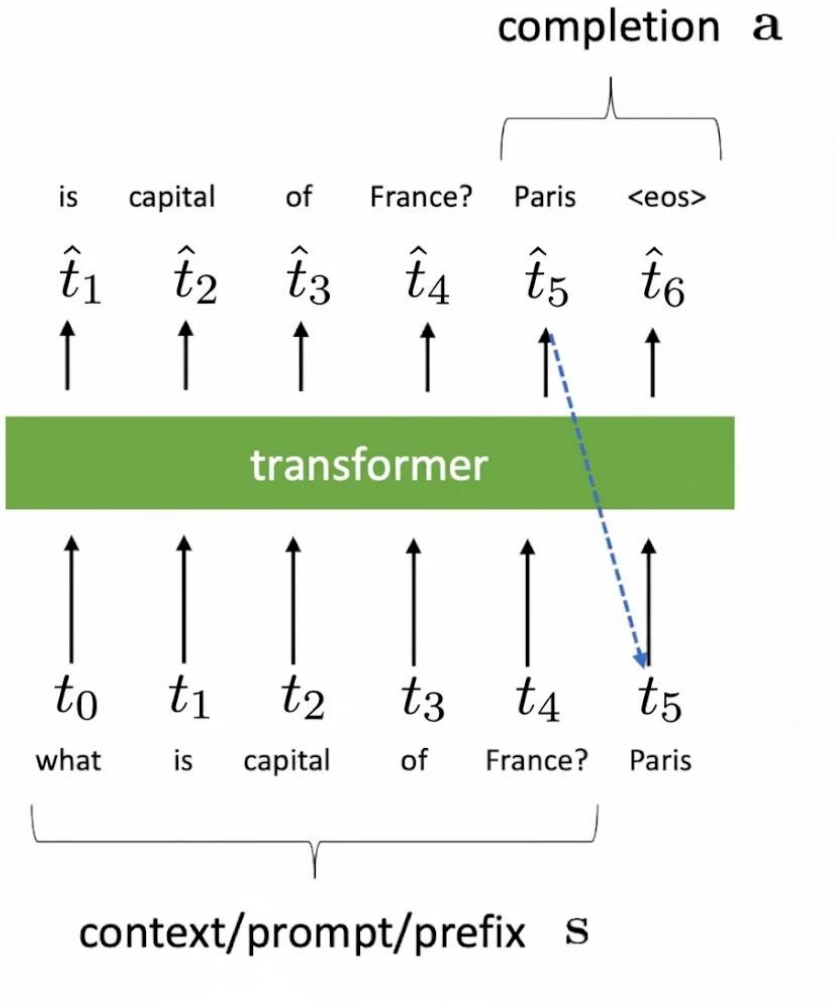


- $p(a|s) = p(t_{5:6}|t_{1:4}) = p(t_5|t_{1:4})p(t_6|t_{1:4}, t_5)$
- We can define $p(a|s)$ as $\pi_\theta(a|s)$
  - The language model is a policy, the completion an action
- $r(s, a)$: How much do I like the completion?
- One-step RL Problem (Bandits)

$$\mathbb{E}_{a\sim\pi_\theta(a|s)}[r(s, a)]$$



Equivalent Problem Definition



- Each new token is first an action, then a state

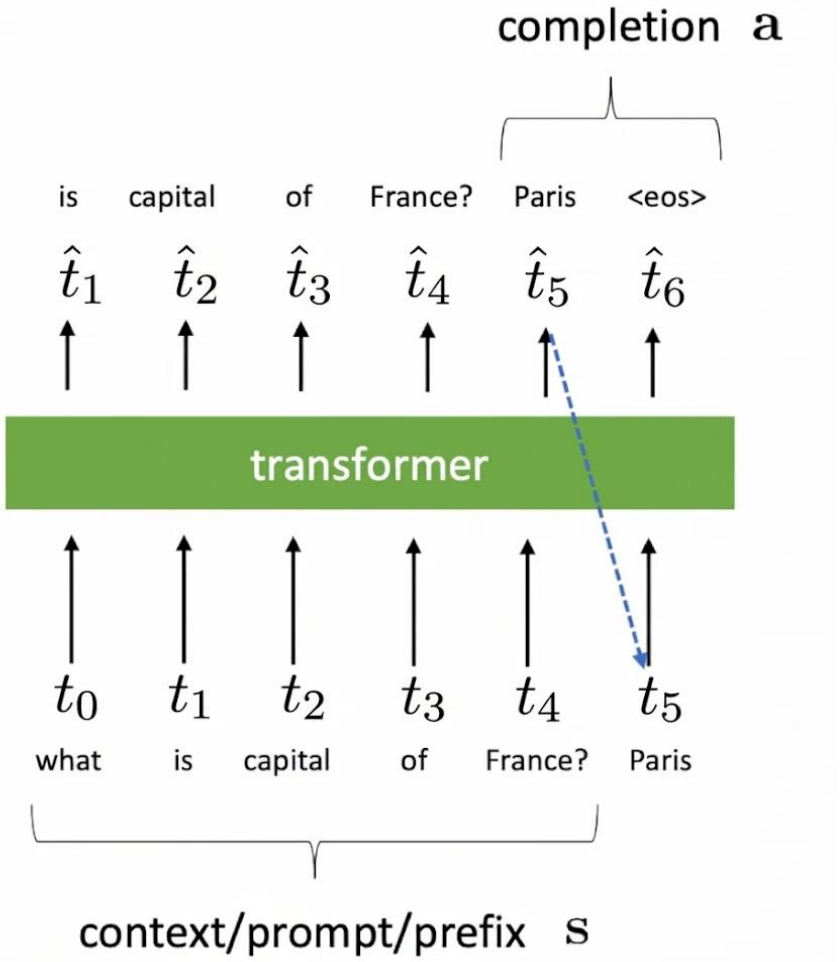
$$p(t_5|t_{1:4})p(t_6|t_{1:4}, t_5) = \pi_\theta(a_1|s_1)\pi_\theta(a_2|s_2)$$

- $r(s_t, a_t)$: How much do I like the sequence of words *up to now*?

- Multi-Step RL Problem

$$\mathbb{E}_{\tau \sim p_\pi(\tau)}\left[\sum_t \cancel{\gamma^t} r(s_t, a_t)\right]$$

Equivalent Problem Definition



- Multi-Step RL Problem

$$\mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Captures the case with intermediate rewards
- Captures the case where the state contains additional words not generated by the LLM
  - Additional human inputs (chatbot), Computer Use
- Makes more sense for value functions

Reinforce for Language Models

$$\nabla_\theta \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_t r(s_t, a_t) \right] = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_t \nabla_\theta \log \pi_\theta(a_t | s_t) A_t \right]$$

- Where A_t is the advantage at time t:

$$A_t = \sum_{t'}^{T} r_{t'} - V(s_t)$$

Reinforce for Language Models: Algorithm

- Initialize θ with the result of SFT
- Until tired (of spending money)
  - Sample a batch of *N* answers from the prefix s_0
  - Evaluate $r\left(s_t^i, a_t^i\right)$ for each answer i and new word a_t^i
  - Evaluate advantage $A_t^i = \sum_{t'=t}^{T} r\left(s_{t'}^i, a_{t'}^i\right) - V\left(s_t^i\right)$
  - $\hat{g} = \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta\left(a_t^i \middle| s_t^i\right) A_t^i$
  - $\theta \leftarrow \theta + \alpha \hat{g}$

An Example

- Binary (Verifiable) Reward:

$$r(s_t, a_t) = \begin{cases} 1 & \text{if } \mathrm{a_t} \text{ is eos and the answer } a_{o:t} \text{ is correct} \\ 0 & \text{otherwise} \end{cases}$$

- $R(\tau) = \sum_{t=0}^{T} r_t$ is either 0 or 1
- $V(s_t) \propto p(correct\ answer)$
- Let's look at $A_t = \sum_{t'}^{T} r_{t'} - V(s_t)$

- What is the sign of A_t if the answer is correct? What if it's wrong?
- If A_t is positive, is there any $t' \neq t$ such that $A_{t'}$ is negative?

An Example

- Binary (Verifiable) Reward:

$$r(s_t, a_t) = \begin{cases} 1 & \text{if } a_t \text{ is eos and the answer } a_{o:t} \text{ is correct} \\ 0 & \text{otherwise} \end{cases}$$

- $A_t = \sum_{t'}^{T} r_{t'} - V(s_t)$

- Positive if the final answer was correct.
  - *Increase* the probability of every token in the answer, according to how important they were according to V.
- Negative if the final answer was wrong.
  - *Decrease* the probability of every token in the answer, according to how important they were according to V.

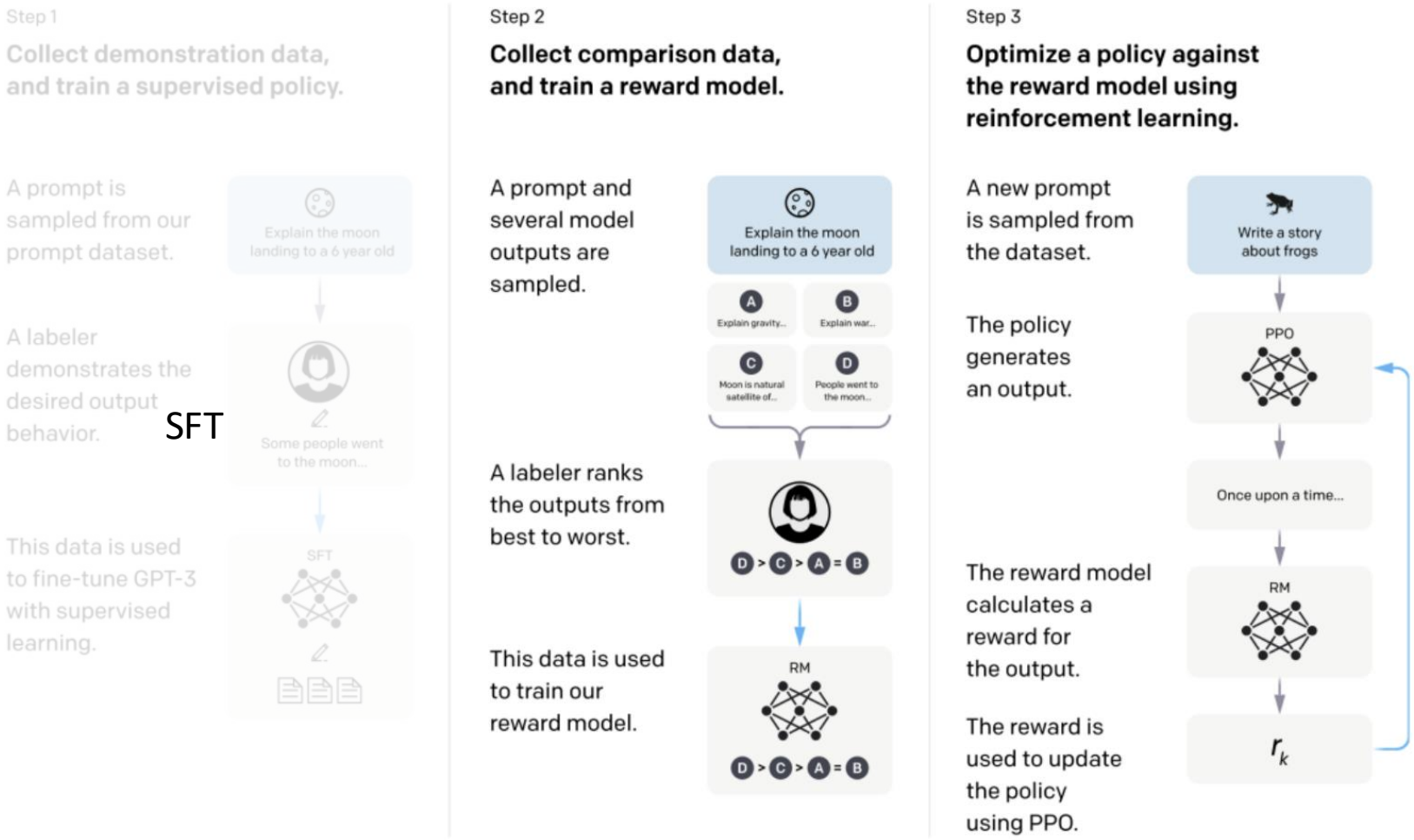
Reinforce for Language Models: Algorithm

- Initialize θ with the result of SFT
- Until tired (of spending money)
  - Sample a batch of *N* answers from the prefix s_0
  - Evaluate $r(s_t^i, a_t^i)$ for each answer i and new word a_t^i what reward?
  - Evaluate advantage $A_t^i = \sum_{t'}^{T} r(s_{t'}^i, a_{t'}^i) - V(s_t^i)$ what baseline?
  - $\hat{g} = \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) A_t^i$ can we do more steps for each generation (e.g. PPO)?
  - $\theta \leftarrow \theta + \alpha \hat{g}$
- How to prevent forgetting too much from the initial model?

What Reward?

- For questions with yes/no answers that can be independently validated (such as “2+2=?”), the reward is trivial
  - Just run the external verifier on the answer
- For open-ended questions, we can’t use external verifiers
- The most popular reward in this case is *human preferences*
- Does this mean we need a human in the loop for training?

Learning Reward Models from Humans



OpenAI

Learning a Reward from Preferences

Optometrist Algorithm (Bradley-Terry) 

- Initialize reward model $r(\tau; w)$ with parameters w
- Sample 2 (or more) trajectories τ_i, τ_j from the language model
- Ask a person to choose preference(s) $\tau_i \succ \tau_j$
- Use preferences to improve the reward model $r(\tau; w)$

- Is there anything weird in the algorithm above?

Bradley-Terry Model

- How to improve the reward model?

- We use a probabilistic representation of preferences:

$$p(\tau_i \succ \tau_j; w) = \frac{\exp(\sum_t r(s_t^i, a_t^i; w))}{\exp(\sum_t r(s_t^i, a_t^i; w)) + \exp(\sum_t r(s_t^j, a_t^j; w))}$$

- If the human said that $\tau_i \succ \tau_j$, we should increase that probability

Bradley-Terry Model Update

- $p(\tau_i \succ \tau_j; w) = \frac{\exp(\sum_t r(s_t^i, a_t^i; w))}{\exp(\sum_t r(s_t^i, a_t^i; w)) + \exp(\sum_t r(s_t^j, a_t^j; w))}$
- $p(\tau_i \succ \tau_j; w) = \frac{1}{1 + \exp\left(\sum_t r(s_t^j, a_t^j; w) - \sum_t r(s_t^i, a_t^i; w)\right)}$
- $p(\tau_i \succ \tau_j; w) = sigmoid(R(\tau_i; w) - R(\tau_j; w))$

Sigmoid Function

$$y = \frac{1}{1 + e^{-x}}$$

- $\log\left(p(\tau_i \succ \tau_j; w)\right)$ is a differentiable function!
- Update w with gradient ascent:

$$w \leftarrow w + \alpha \nabla_w \log\left(p(\tau_i \succ \tau_j; w)\right)$$

Learning a Reward from Preferences

Optometrist Algorithm (Bradley-Terry) 

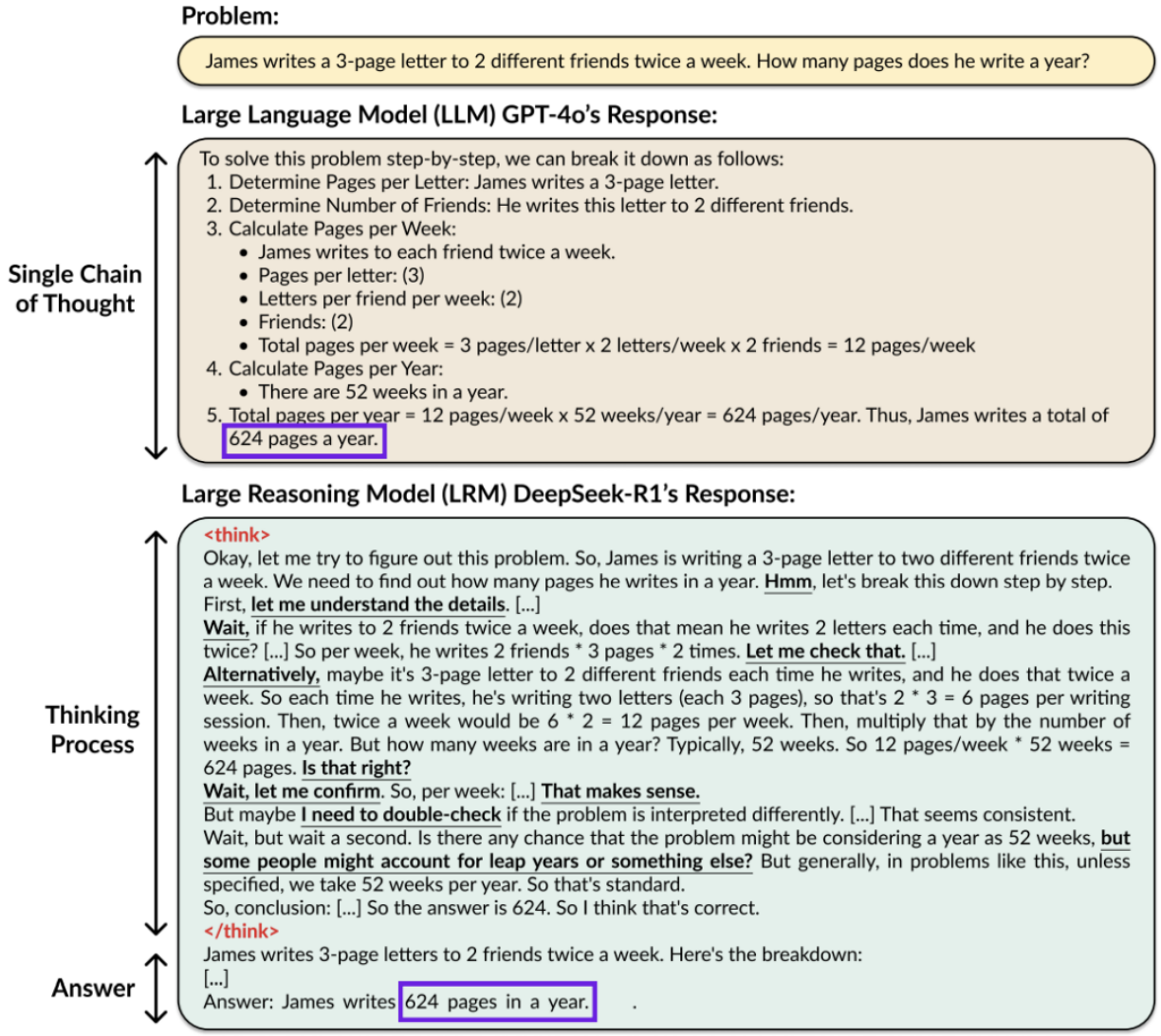
- Initialize reward model $r(\tau; w)$ with parameters w
- Sample 2 (or more) trajectories τ_i, τ_j from the model
- Ask a person to choose preference(s) $\tau_i \succ \tau_j$
- Use preferences to improve the reward model $r(\tau; w)$:

$$w \leftarrow w + \alpha \nabla_w \log\big(p(\tau_i \succ \tau_j; w)\big)$$

- Does this problem remind you of something?

Reward Sparsity

- **Verifier Reward**
  - Only at the end
  - The reward model could be another language model!



Marjanović et al. '25

Reward Sparsity

- **Verifier Reward**
  - Only at the end
  - The reward model could be another language model!
- **Process Reward**
  - The reward model evaluates each step of generation, indicating whether it is correct
  - Dense(r)

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

This problem is about factoring a polynomial over the integers, which can be a tricky task.

One possible strategy is to look for rational roots of the polynomial, using the rational root theorem.

However, that would involve checking a lot of possible factors of -4, and there is no guarantee that the polynomial has any rational roots at all.

Another possible strategy is to use some algebraic tricks to simplify the polynomial or reduce its degree.

I notice that the polynomial has only even powers of x, so I can try to substitute $y = x^2$ and see what happens.

Then the polynomial becomes $y^4 + 3y^2 - 4$, which is a quadratic equation in y^2.

I can try to factor this quadratic equation by looking for two numbers that multiply to -4 and add to 3.

Those numbers are 4 and -1, so I can write $y^4 + 3y^2 - 4 = (y^2 + 4)(y^2 - 1)$.

Now I can substitute back $x^2 = y$ and get $x^8 + 3x^4 - 4 = (x^4 + 4)(x^4 - 1)$.

This looks promising, because I have reduced the degree of the factors from 8 to 4.

I can try to factor each of these factors further, using the difference of squares formula.

I get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x^2 - 1)$.

I can apply the difference of squares formula again to the last factor and get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x + 1)(x - 1)$.

Now I have factored the polynomial completely into monic linear and quadratic factors with integer coefficients.

Reinforce for Language Models: Algorithm

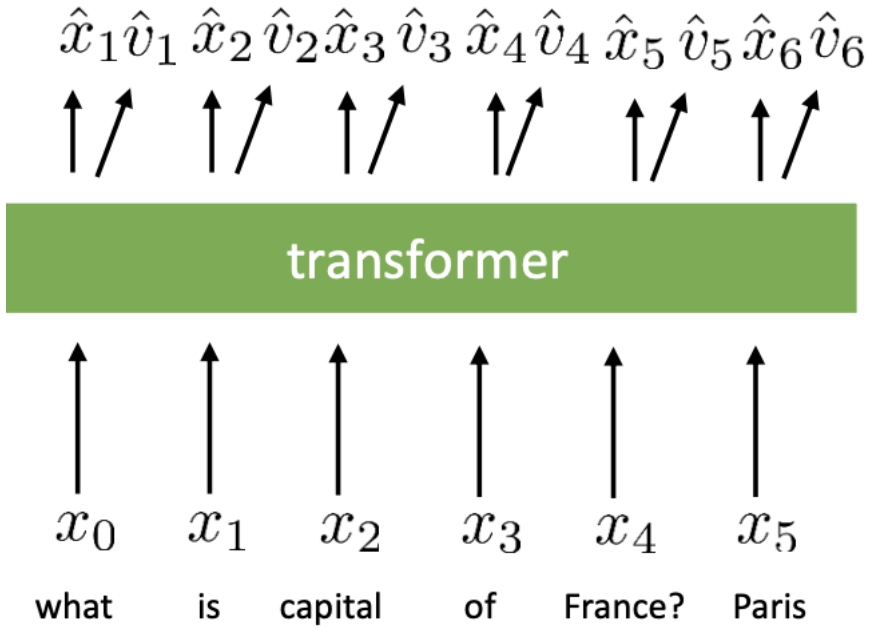
- Initialize θ with the result of SFT

- Until tired (of spending money)
  - Sample a batch of *N* answers from the prefix s_0
  - Evaluate $r(s_t^i, a_t^i)$ for each answer i and new word a_t^i
  - Evaluate advantage $A_t^i = \sum_{t'}^{T} r(s_{t'}^i, a_{t'}^i) - V(s_t^i)$ what baseline?
  - $\hat{g} = \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) A_t^i$
  - $\theta \leftarrow \theta + \alpha \hat{g}$

Value Function baselines

- $TD(\lambda)$ is the most commonly used method to learn value functions

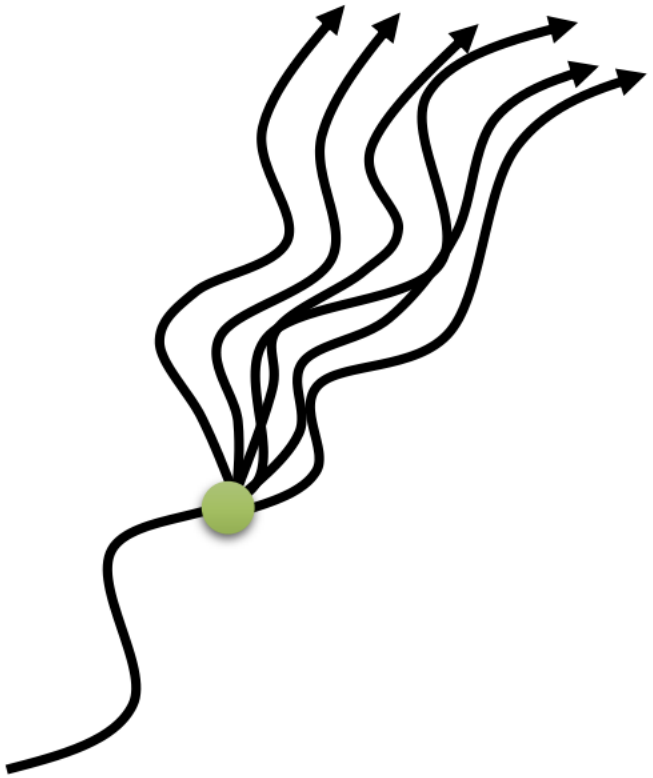
- Either add a new head or make a copy of the entire network



Baselines Without Value Function

- For each prompt s
- Sample K full answers from the model
- Answers' average reward = value function

$$V(s) = \frac{1}{K}\sum_{k}\sum_{t} r\left(s_t^k, a_t^k\right)$$



- Have we seen this before?
- One less model, one less headache!
- Particularly useful if model generations are fast!
- Idea popularized by DeepSeekMath (GRPO)

Reinforce for Language Models: Algorithm

- Initialize θ with the result of SFT

- Until tired (of spending money)
  - Sample a batch of *N* answers from the prefix s_0
  - Evaluate $r(s_t^i, a_t^i)$ for each answer i and new word a_t^i
  - Evaluate advantage $A_t^i = \sum_{t'}^{T} r(s_{t'}^i, a_{t'}^i) - V(s_t^i)$
  - $\hat{g} = \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) A_t^i$ can we do more steps for each generation (e.g. PPO)?
  - $\theta \leftarrow \theta + \alpha \hat{g}$

- How to prevent forgetting too much from the initial model?

Squeezing More From Each Generation

- Generating an answer (or multiple ones) can be expensive
- We can use an off-policy method to do more backpropagation steps, given the same rollouts
- Off-Policy Policy-Gradient Methods like PPO are the most popular
  - Why?

PPO for Language Models:

- Until tired (of spending money)
  - Sample a batch of *N* answers from the prefix s_0
  - Evaluate $r(s_t^i, a_t^i)$ for each answer i and new word a_t^i
  - Evaluate advantage $A_t^i = \sum_{t'}^{T} r(s_{t'}^i, a_{t'}^i) - V(s_t^i)$
  - $\theta^{new} \leftarrow \theta$
  - K
    - $\hat{g} = \nabla_\theta \mathcal{L}_{CLIP}(\theta^{new}, \theta)$
    - $\theta^{new} \leftarrow \theta^{new} + \alpha \hat{g}$
  - $\theta \leftarrow \theta^{new}$

Reference Model Regularization

- Often, we don't want the model $\pi_\theta(a|s)$ to drift too much from some reference model $\pi_{ref}(a|s)$
- Otherwise, the model could start to produce "nonsense text"
- Or exploit errors in the reward model

- In practice, we add a policy constraint term to the reward

$$\bar{r}(s_t, a_t) = r(s_t, a_t) - \beta D_{KL}\left(\pi_\theta(a_t|s_t), \pi_{ref}(a_t|s_t)\right)$$

- We could potentially use a closed-source model as a reference!

RL with Bradley-Terry Rewards Needs no RL

- Bandit formulation where the action $a \sim \pi_\theta(a|s)$ is the whole answer.
- Cost Function:

$$J(\theta) = \mathbb{E}_{s\sim \mathrm{D}, a\sim\pi_\theta(s)}[r(s,a)] - \beta D_{KL}\left(\pi_\theta(a_t|s_t), \pi_{ref}(a_t|s_t)\right)$$

- This constrained RL problem can be solved analytically

$$\pi^*(a|s) = \frac{1}{Z(s)}\pi_{ref}(a|s)e^{\frac{r(s,a)}{\beta}}$$

- We can solve for $r(s,a)$ in terms of π^* and π_{ref}
- Substituting π^* with π_θ I get a reward model $r(s,a;\theta)$

RL with Bradley-Terry Rewards Needs no RL

- Assume that you have a dataset D_p of preferences $p(a_i \succ a_j)$
- Let's learn a reward model $r(s, a; \theta)$ on D_p
- For $a_i, a_j \sim D_p$
  - $\theta \leftarrow \theta + \alpha \nabla_\theta \log\big(p(\tau_i \succ \tau_j; \theta)\big)$
  - $\theta \leftarrow \theta + \alpha \nabla_\theta \log\big(sigmoid\left(r(a_i; \theta) - r(a_j; \theta)\right)\big)$



- At convergence, $\pi_\theta = \pi^*$. Without any explicit RL!
- This is called DPO (Direct Preference Optimization).

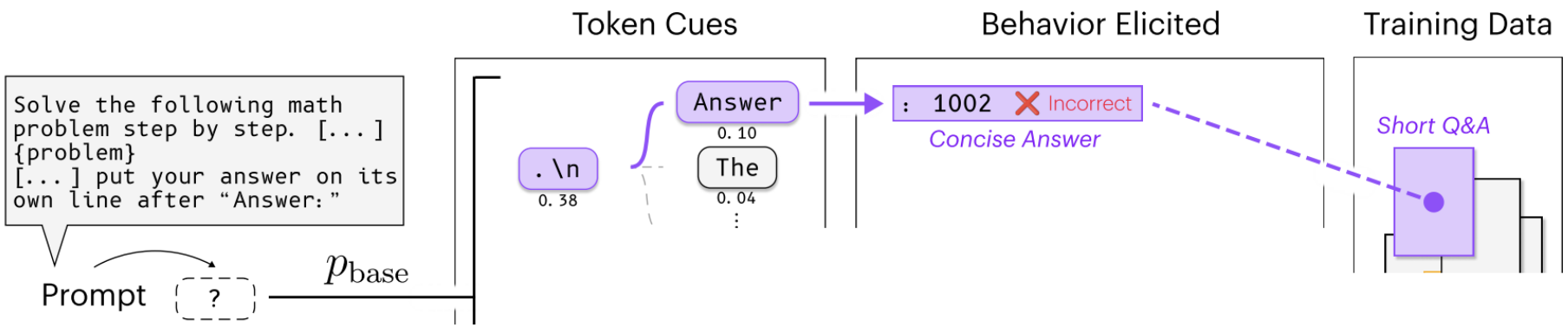
Summary: RL for LLMs

- In the RL stage, we increase the probability of answers that a verifier or a human likes and decrease the probability of the others.

- A lot of technical details for doing this in practice at scale
  - But the main ideas behind each technique are quite intuitive
  - Often entails doing multiple steps (SFT -> DPO -> RL ->SFT …)

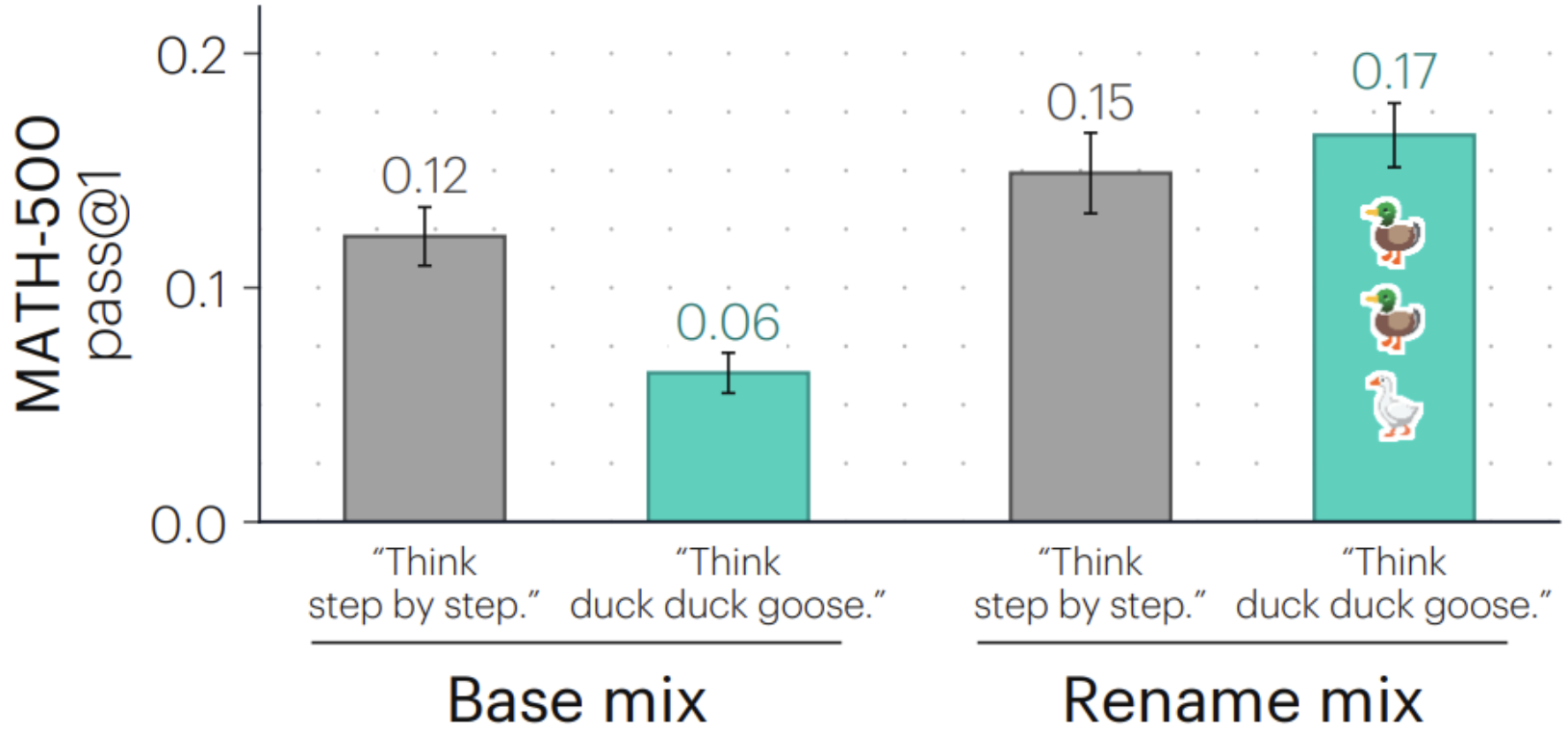
- But is RL giving the LLM new capabilities, or is it mainly learning to retrieve more effectively from the pre-training dataset?
  - Hotly debated topic

Evidence for the Retrieval Hypothesis



Base Models Can Reason by Taking a Cue from Training Data, Wang et al. 25

Learning to Think Duck Duck Goose



- Replacing “step by step” in the training dataset with “duck duck goose”
- Reasoning traces are now enabled by it

Base Models Can Reason by Taking a Cue from Training Data, Wang et al. 25

Conclusion

- LLM+RL is a very exciting topic to do research on
- In my opinion, the era of “experience” has yet to start
  - We might still be in the era of “human data”
- Unclear if new technical ideas are needed
- Or it is just a matter of setting the right environment/incentives
- What a time to be a researcher!