



Lecture 06: LLM Generation & Post-training

Xiang Ren
USC CSCI 662 Advanced NLP
Spring 2026

Announcements

Announcements + Logistics

- Project proposal grades WIP, delayed a bit, expected Thu
- Project midterm update presentations: March 11
 - Project midterm report due: March 25

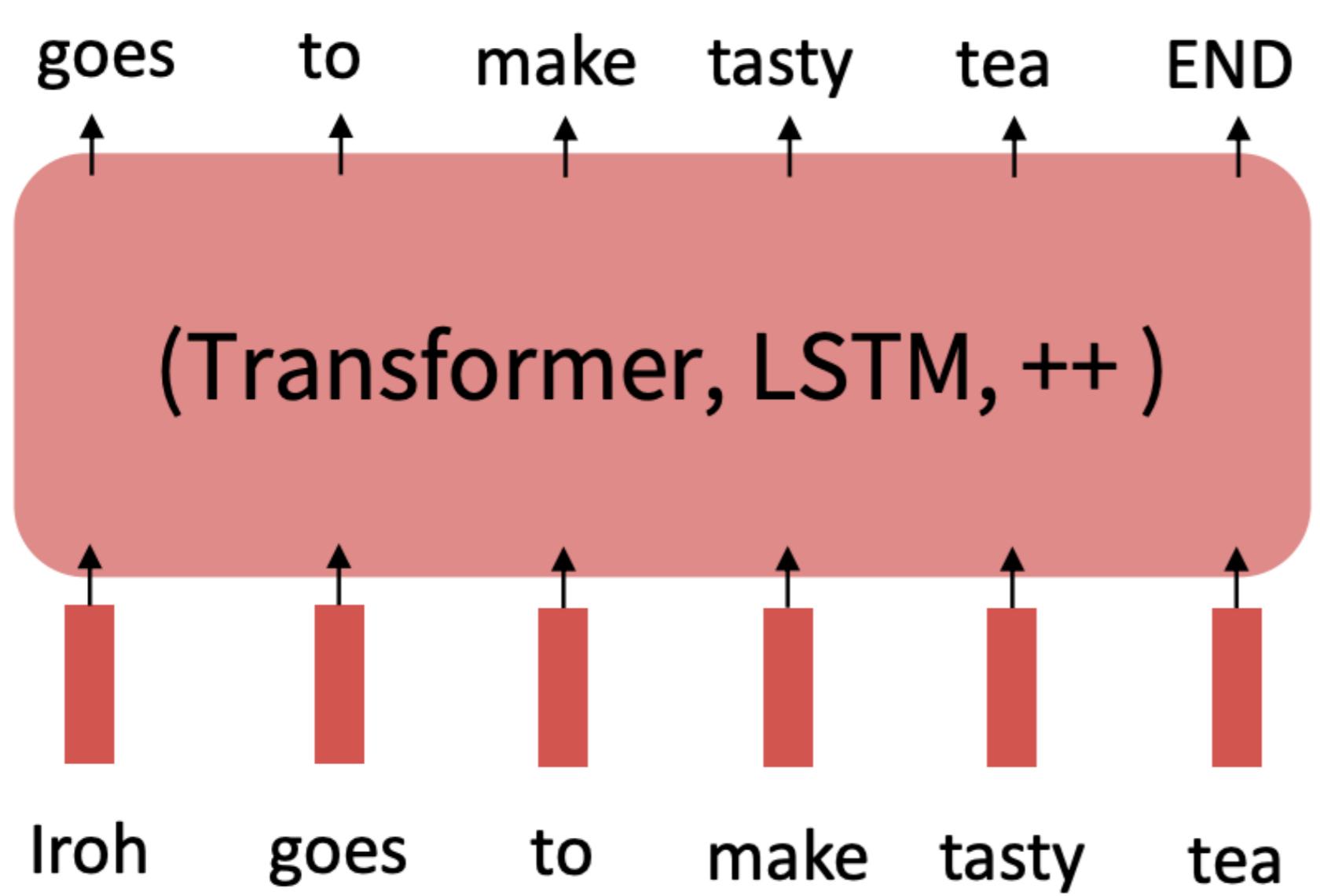
Recap: LLM Pre-training & NLG

The Pretraining / Finetuning Paradigm

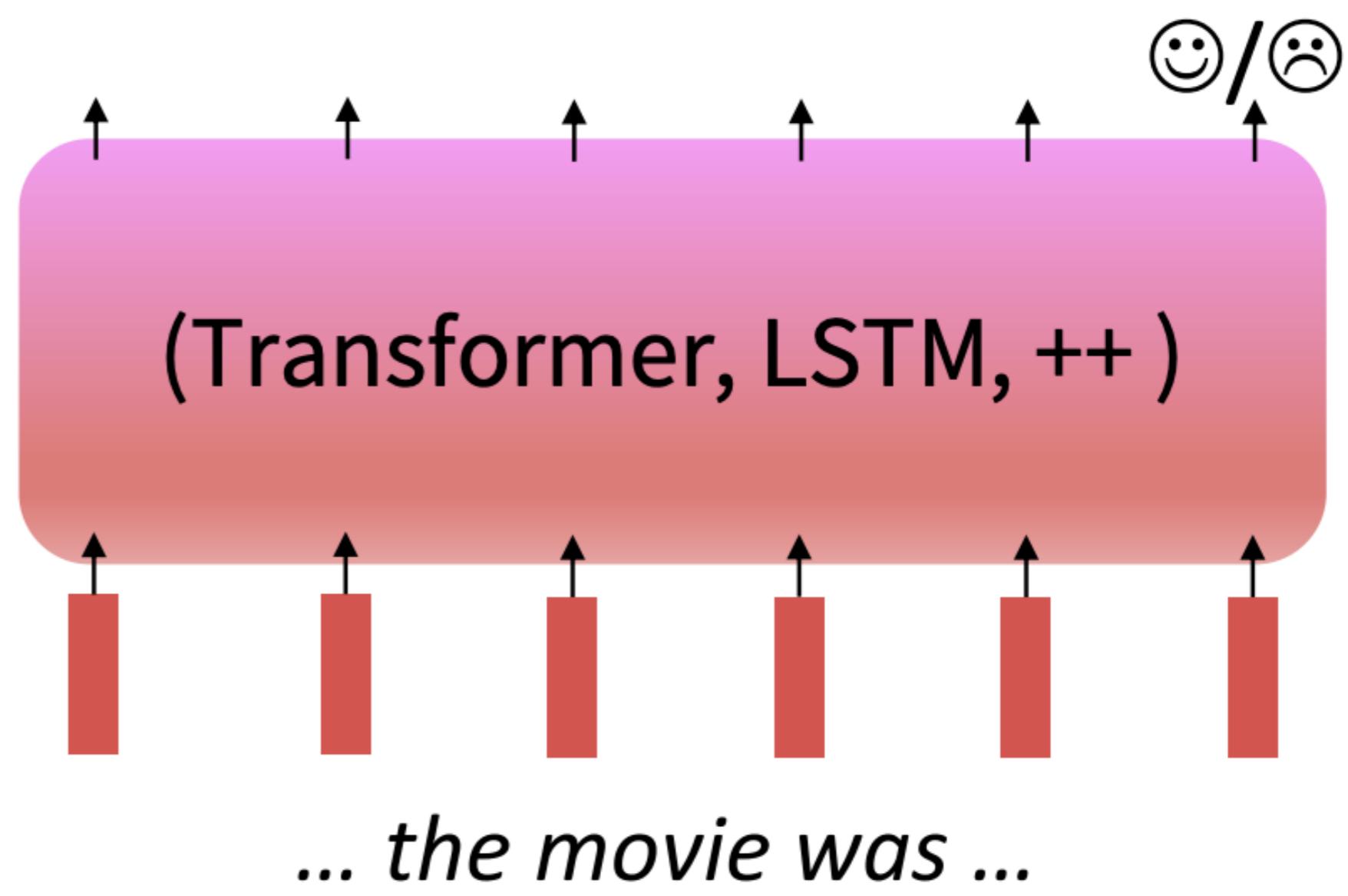
- Pretraining can improve NLP applications by serving as parameter initialization.

Key idea: “Pretrain once, finetune many times.”

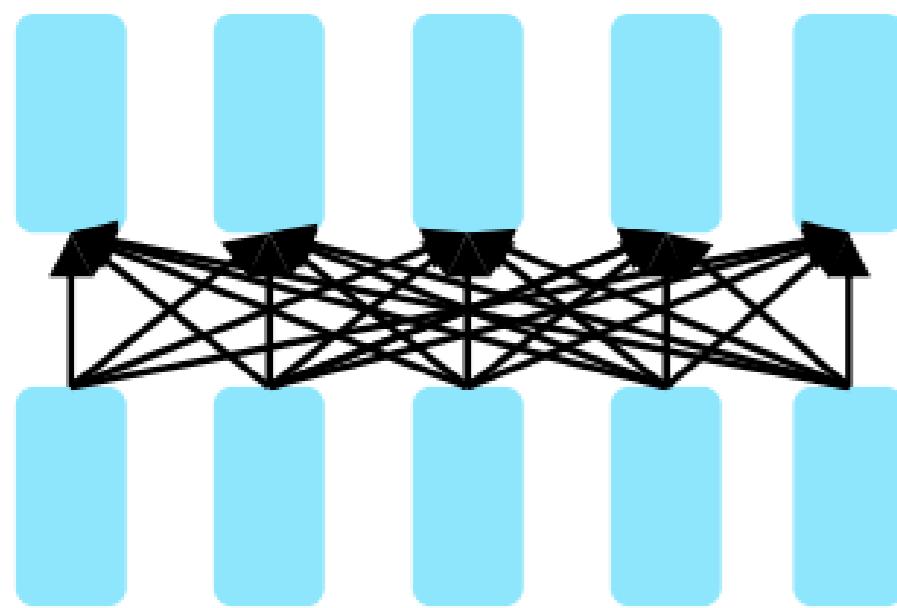
Step 1: Pretrain (on language corpora)
Lots of text; learn general things!



Step 2: Finetune (on your task data)
Not many labels; adapt to the task!

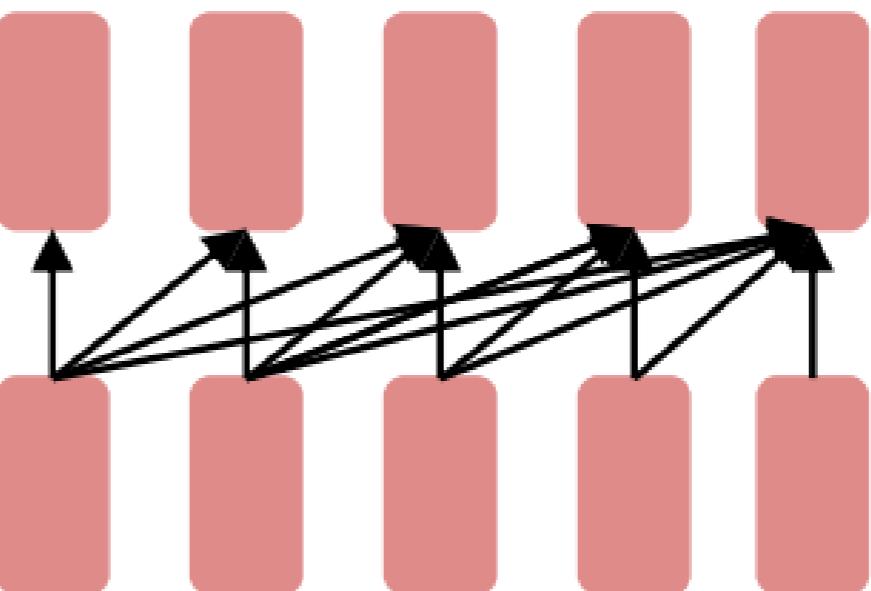


Pretraining for three types of architectures



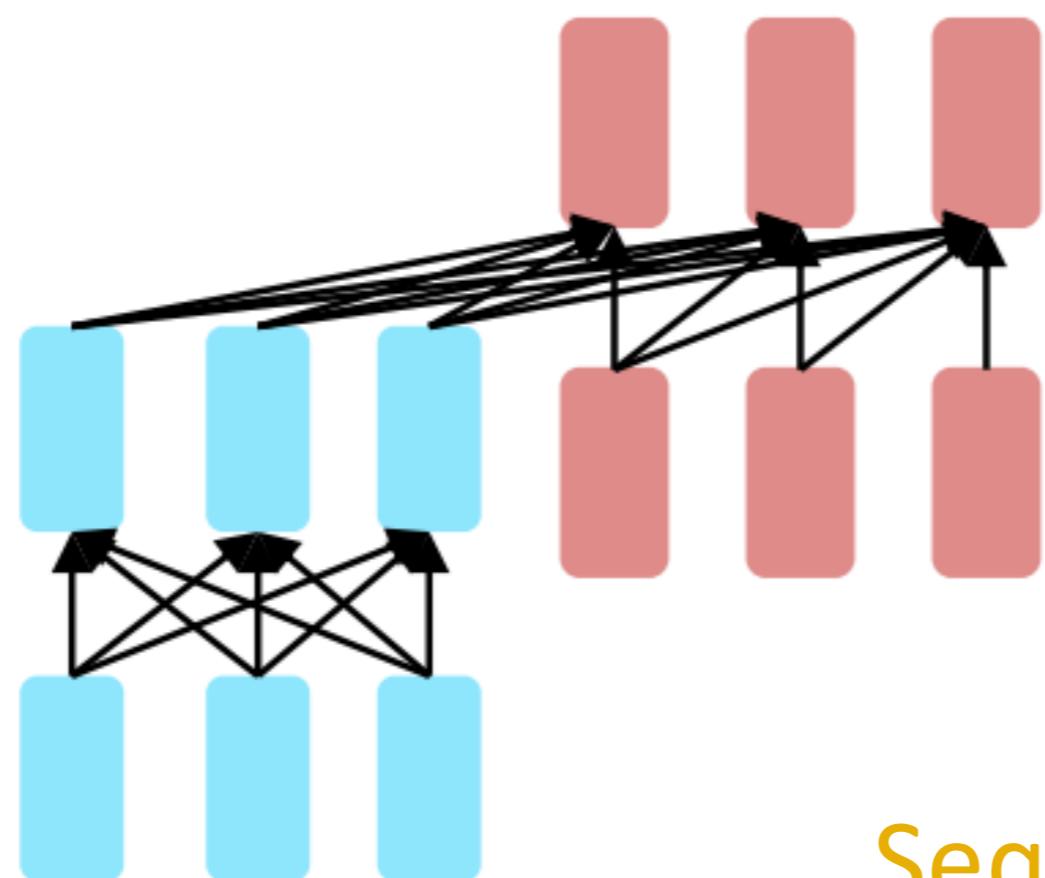
Encoders

Bidirectional Context



Decoders

Language Models



**Encoder-
Decoders**

Sequence-to-sequence

Generative Pretrained Transformer (GPT)

- 2018's GPT was a big success in pretraining a decoder!
 - Transformer decoder with 12 layers, 117M parameters.
 - 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.
 - The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"



OpenAI

[Radford et al., 2018]

GPT-3 and beyond

- Markedly improved generation capabilities by greatly increasing data and model scale
- Solving all tasks through generation
- Even obviating the need to fine-tune!

More in future weeks!



Pretraining Encoders: Bidirectional Context

I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____

Universal Studios Theme Park is located in _____, California

Problem: Input
Reconstruction

'Cause darling i'm a _____ dressed like a daydream

Bidirectional context is important to reconstruct the input!

BERT: Training Details

- Two models were released:
 - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
 - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
 - Trained on:
 - BooksCorpus (800 million words)
 - English Wikipedia (2,500 million words)
 - Pretraining is expensive and impractical on a single GPU.
 - BERT was pretrained with 64 TPU chips for a total of 4 days.
 - (TPUs are special tensor operation acceleration hardware)
 - Finetuning is practical and common on a single GPU
 - “Pretrain once, finetune many times.”



T5: A Pretrained Encoder-Decoder Model

- Raffel et al., 2018 built T5, which uses a span corruption pretraining objective

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

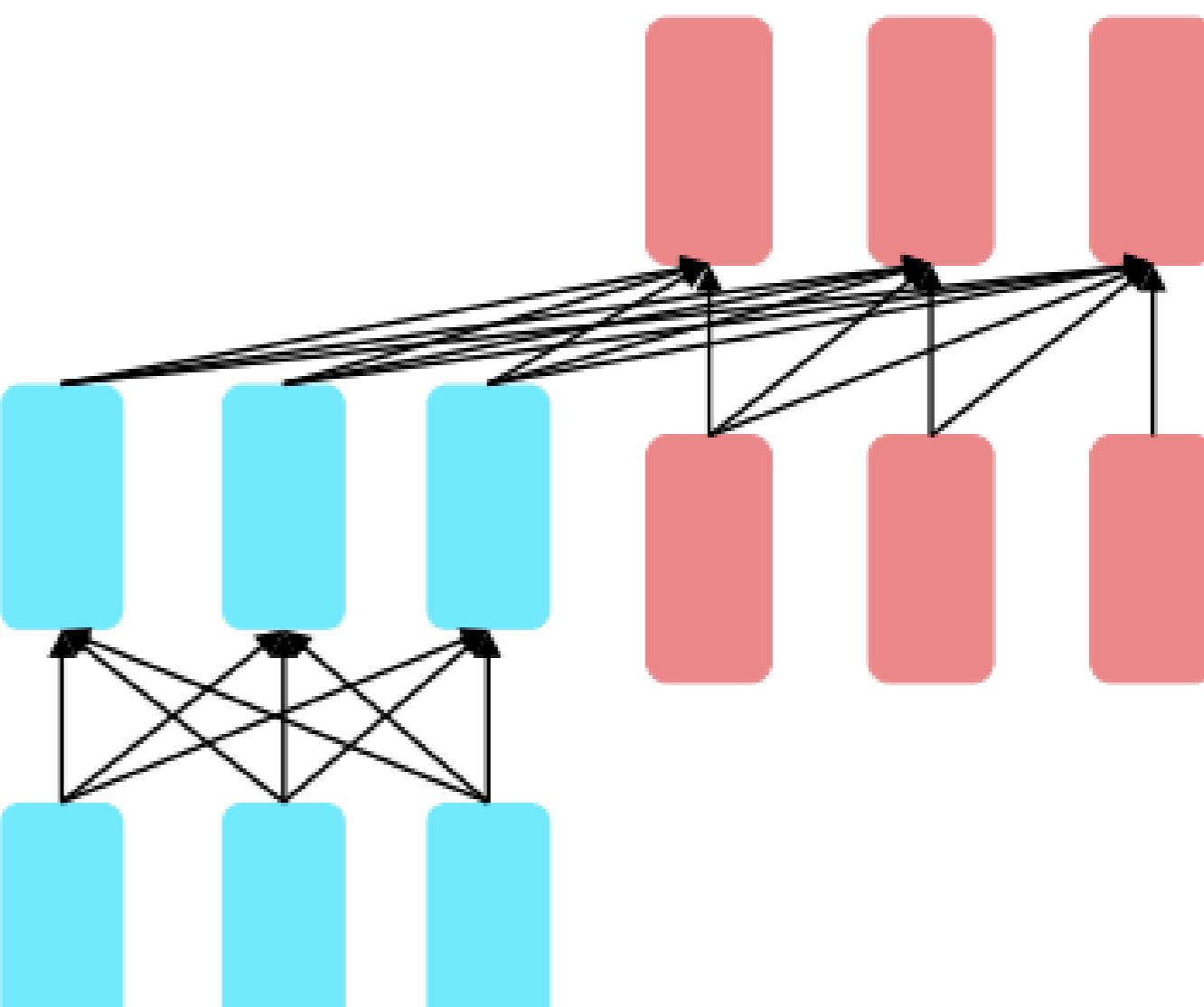
Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

Still uses an objective that looks like language modeling at the decoder side.

Targets

<X> for inviting <Y> last <Z>

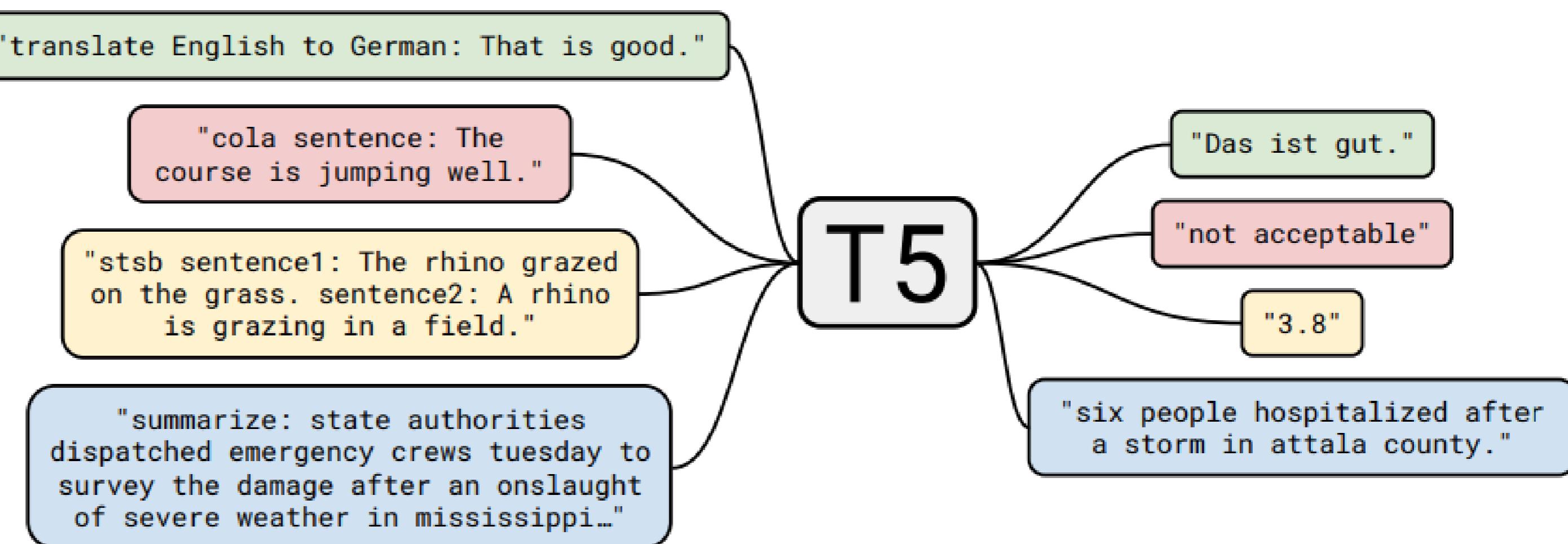


Inputs

Thank you <X> me to your party <Y> week.

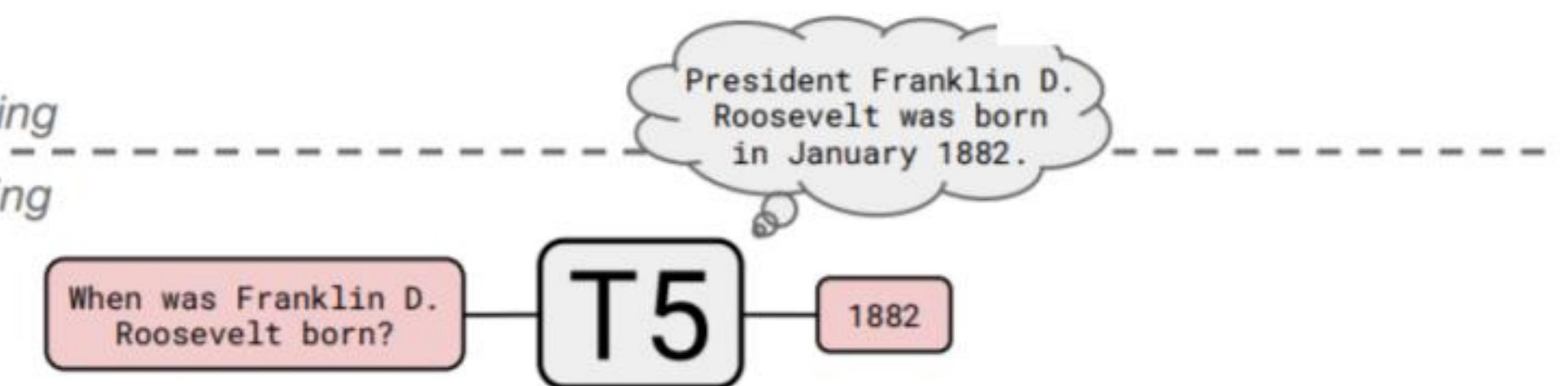
T5: Task Preparation

T5 can be finetuned to answer a wide range of tasks, where the input and output are expressed as a sequence of tokens



Pre-training

Fine-tuning

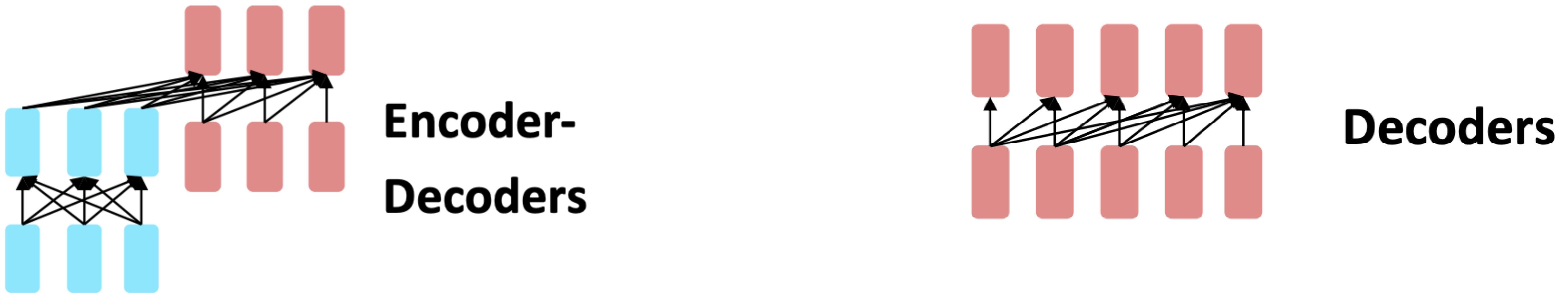


Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- In the worst case, words are split into as many subwords as they have characters.

	word	→	vocab mapping	embedding
Common words	hat	→	hat	
	learn	→	learn	
Variations	taaaaasty	→	taa## aaa## sty	
	laern	→	la## ern##	
misspellings				
novel items	Transformerify	→	Transformer## ify	

Broad Spectrum of NLG Tasks

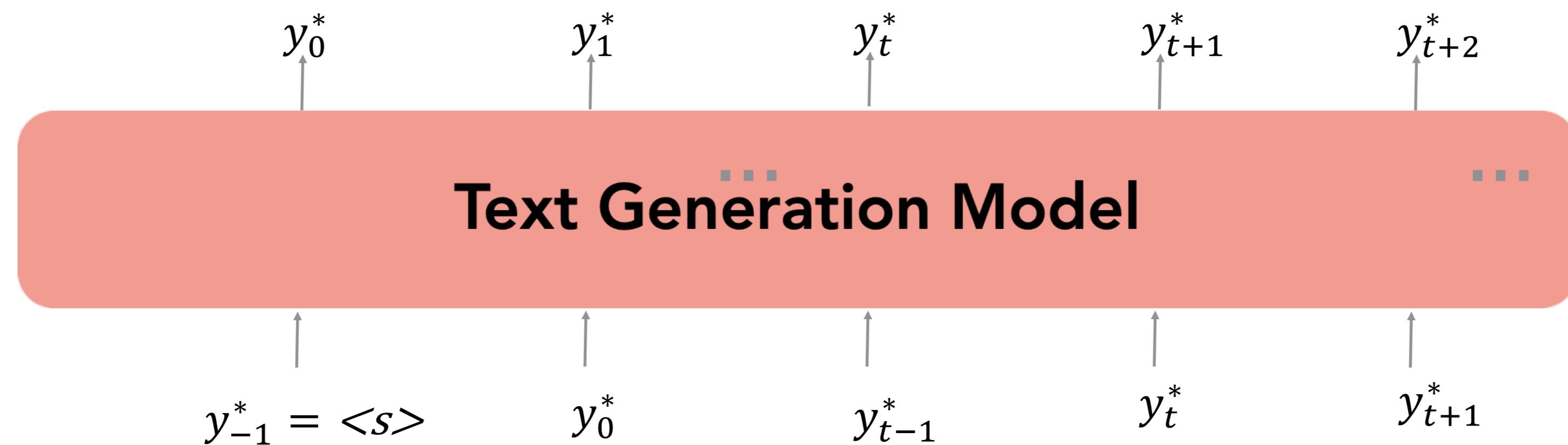


Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}$

$$\mathcal{L} = - \sum_{t=1}^T \log P(y_t | y_{<t}) = - \sum_{t=1}^T \log \frac{\exp(S_{y_t|y_{<t}})}{\sum_{v \in V} \exp(S_{v|y_{<t}})}$$

- Classification task at each time step trying to predict the actual word y_t^* in the training data
- “Teacher forcing” (reset at each time step to the ground truth)



Teacher Forcing

- Strategy for training decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1} , rather than allowing it to rely on the (possibly erroneous) decoder output y_t
- Runs the risk of exposure bias!
 - During training, our model's inputs are gold context tokens from real, human-generated texts
 - At generation time, our model's inputs are previously-decoded tokens

Language Generation: Inference

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t | y_{<t}))$$

Inference / Decoding Algorithm

- The “obvious” decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

$$\hat{y}_t = \operatorname{argmax}_{w \in V} (P(y_t = w | y_{<t}))$$

Classic Inference Algorithms: Greedy and Beam Search

Decoding

- Generation from a language model is also called decoding
 - Think encoder-decoder
 - Also called inference
- Strategy so far: Take argmax on each step of the decoder to produce the most probable word on each step
- This is called greedy decoding
 - Greedy Strategy: we are not looking ahead, we are making the hastiest decision given all the information we have

Greedy Decoding: Issues

- Greedy decoding has no wiggle room for errors!
 - Input: the green witch arrived
 - Output: llego
 - Output: llego la
 - Output: llego la **verde**
- How to fix this?
 - Need a lookahead strategy / longer-term planning

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$\begin{aligned} P(y|x) &= P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots, P(y_T|y_1, \dots, y_{T-1}, x) \\ &= \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, x) \end{aligned}$$

- We could try co

- This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size
- This $O(V^T)$ complexity is far too expensive!

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
 - k is the beam size (in practice around 5 to 10, in NMT)
 - A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam Search Decoding: Example

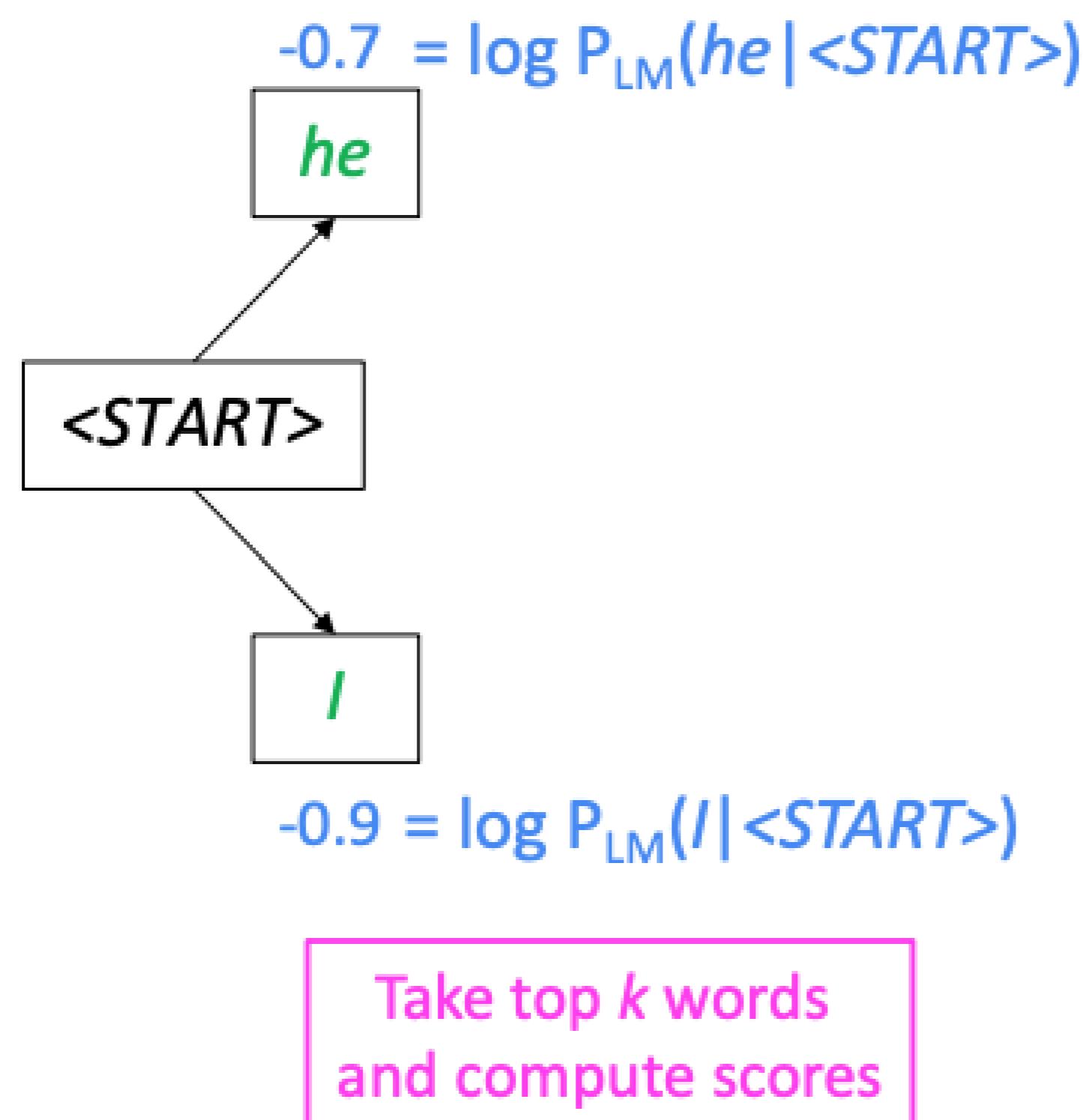
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$

<START>

Calculate prob
dist of next word

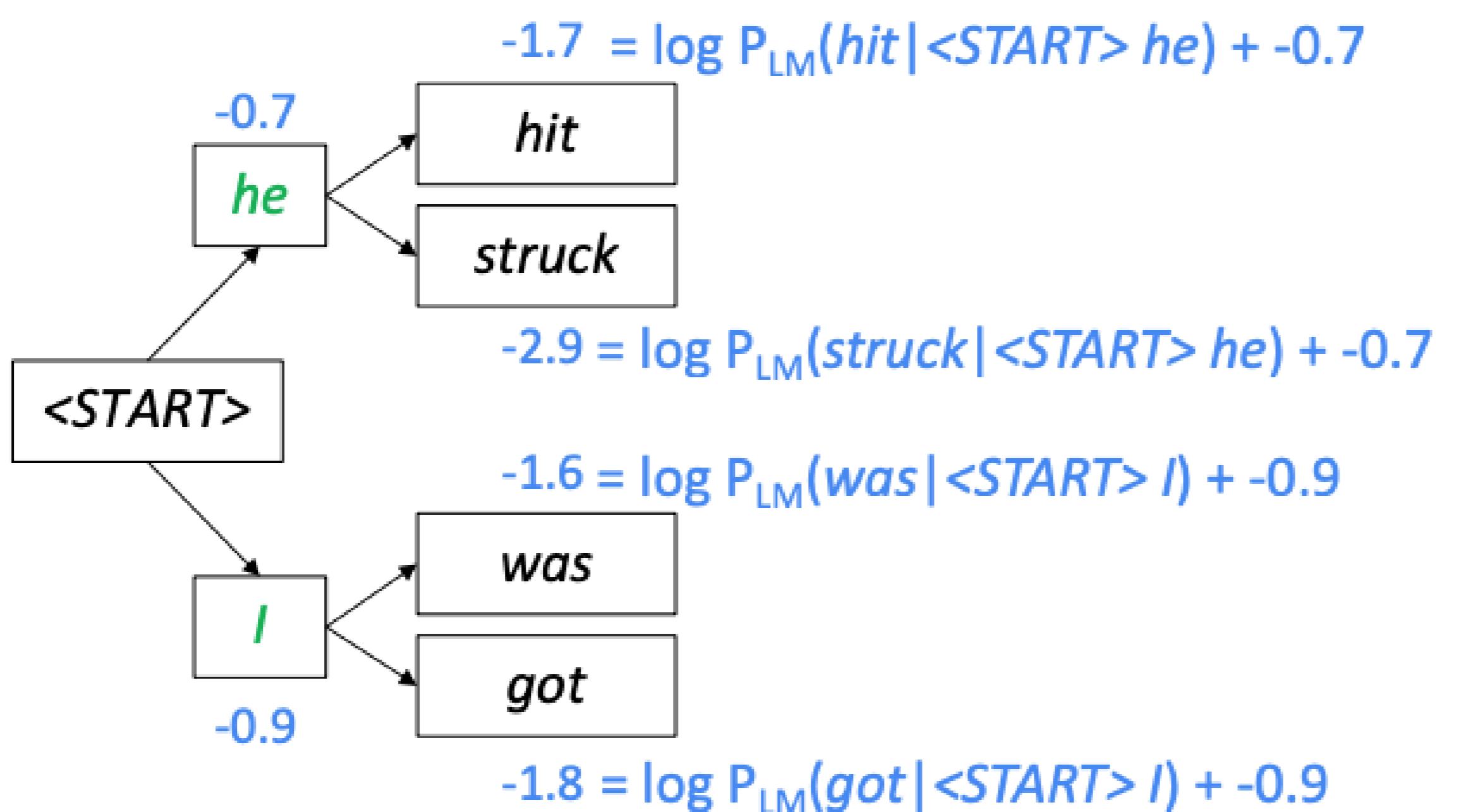
Beam Search Decoding: Example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Beam Search Decoding: Example

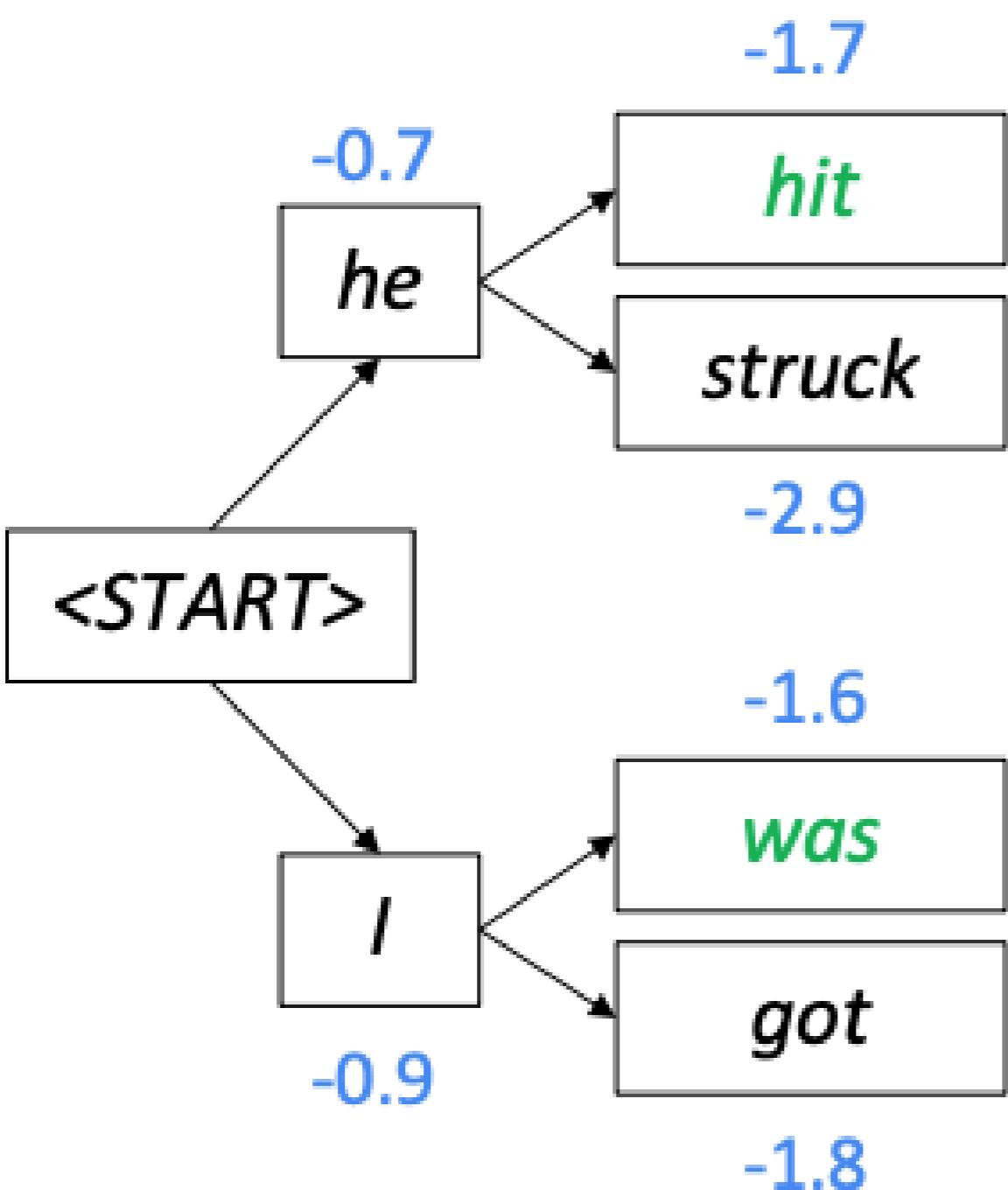
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



For each of the k hypotheses, find
top k next words and calculate scores

Beam Search Decoding: Example

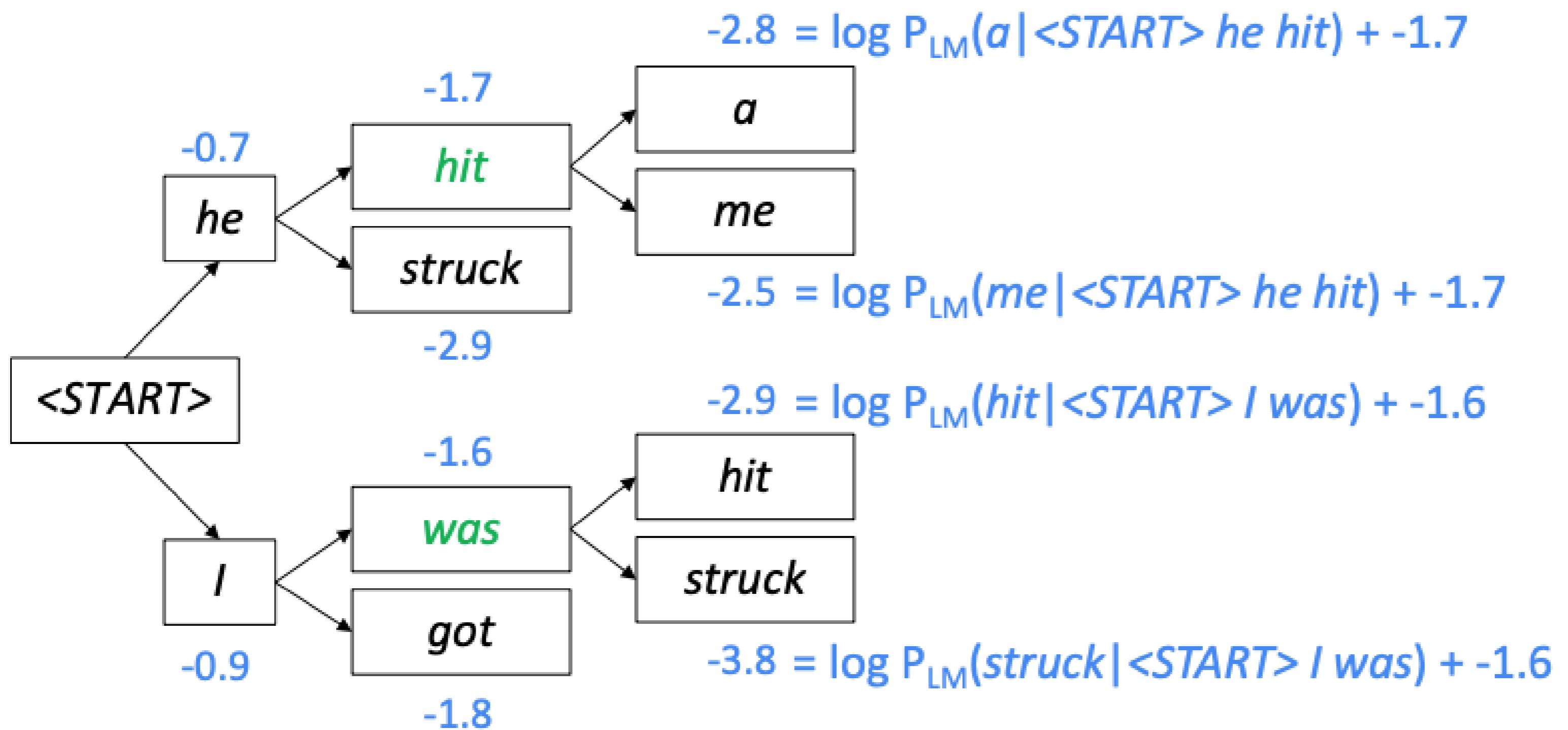
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: Example

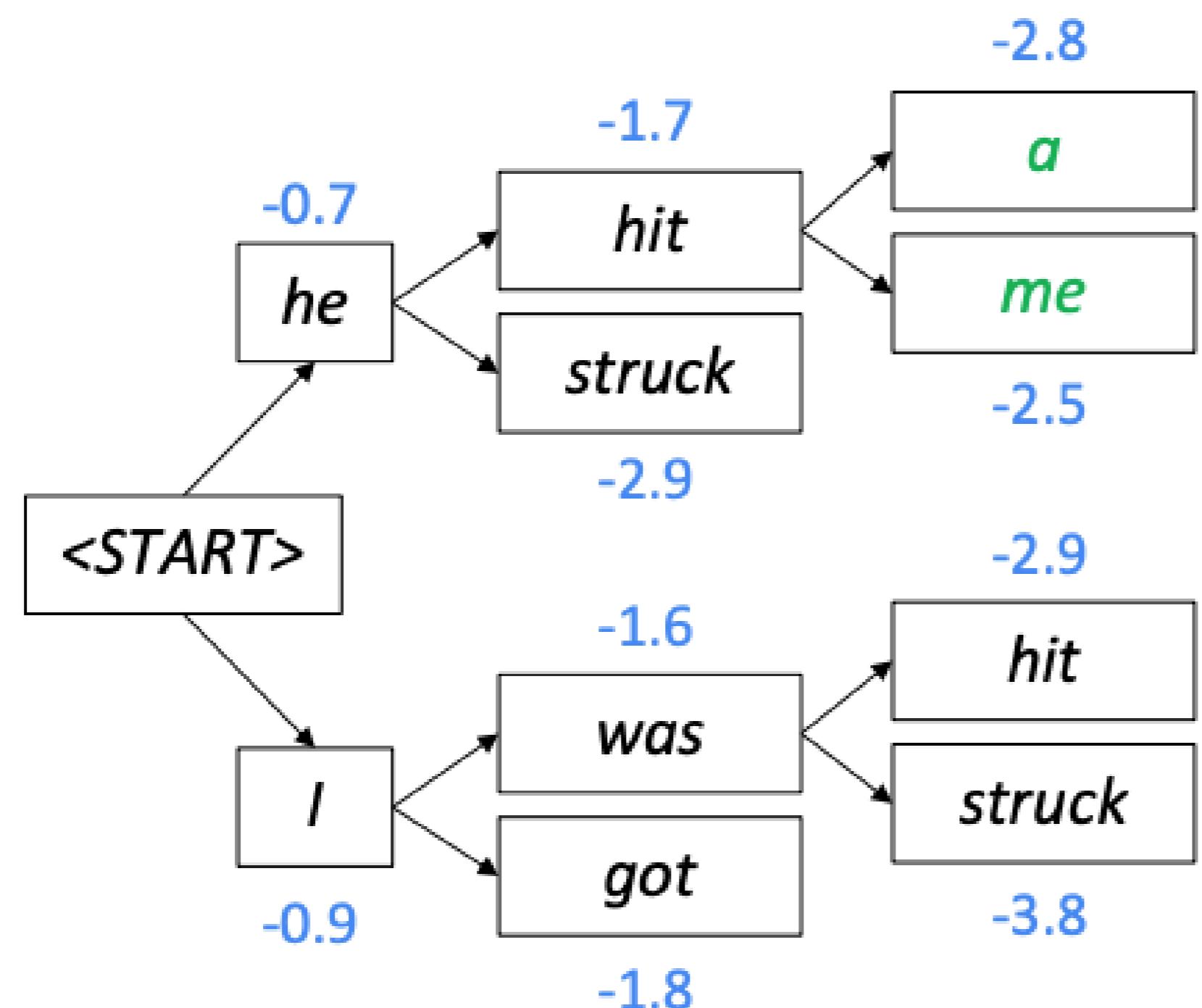
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



For each of the k hypotheses, find
top k next words and calculate scores

Beam Search Decoding: Example

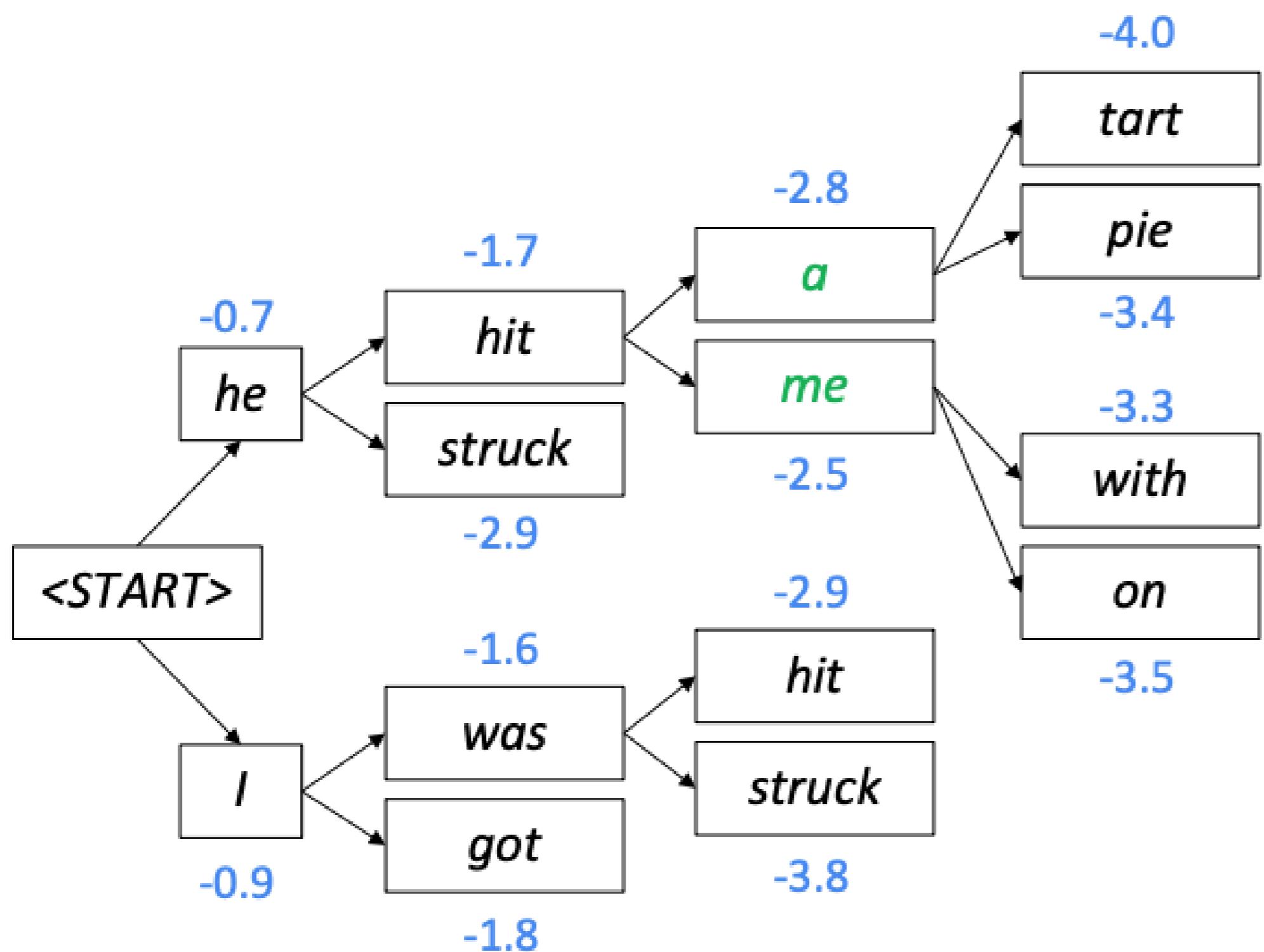
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: Example

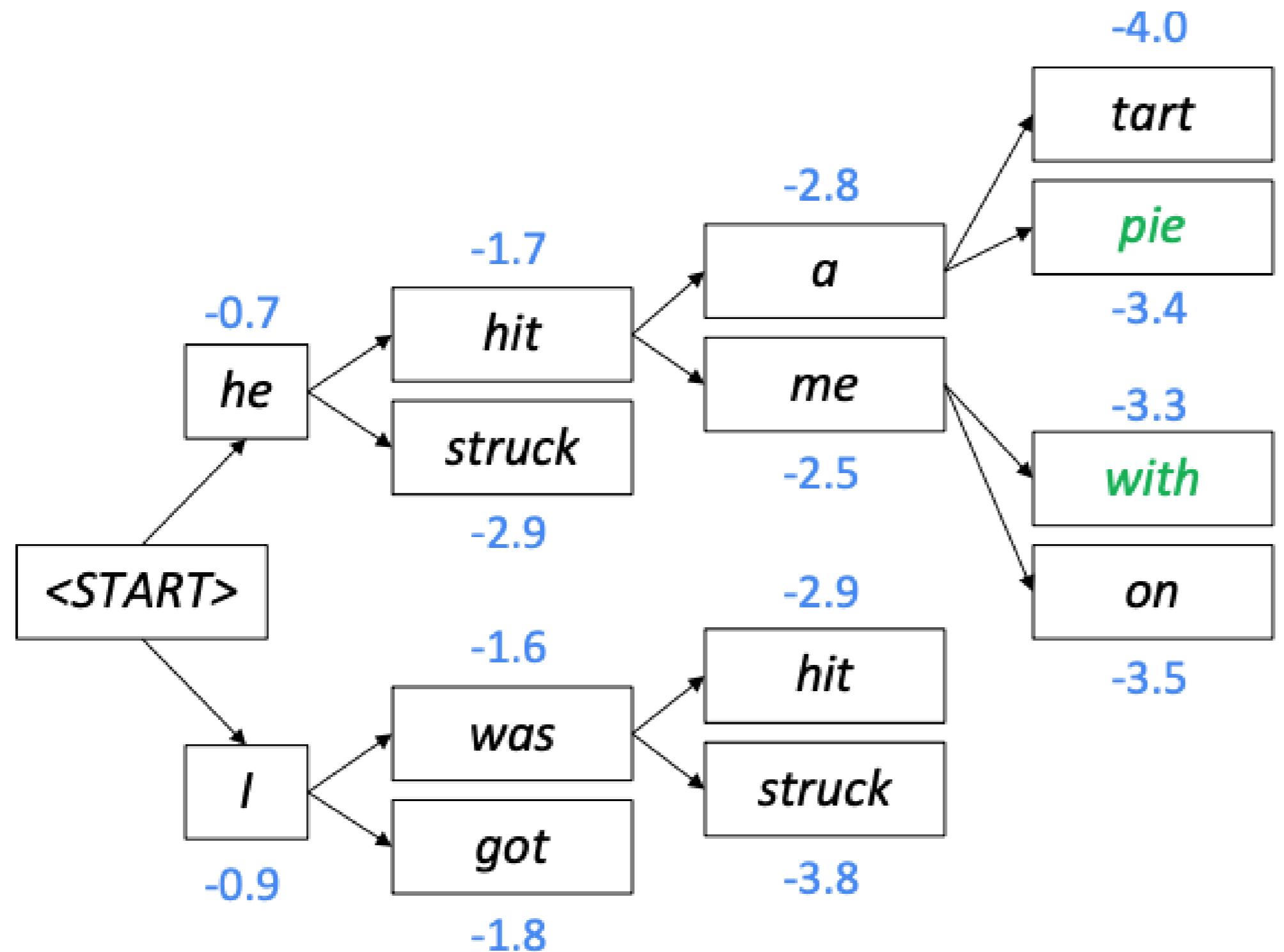
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



For each of the k hypotheses, find top k next words and calculate scores

Beam Search Decoding: Example

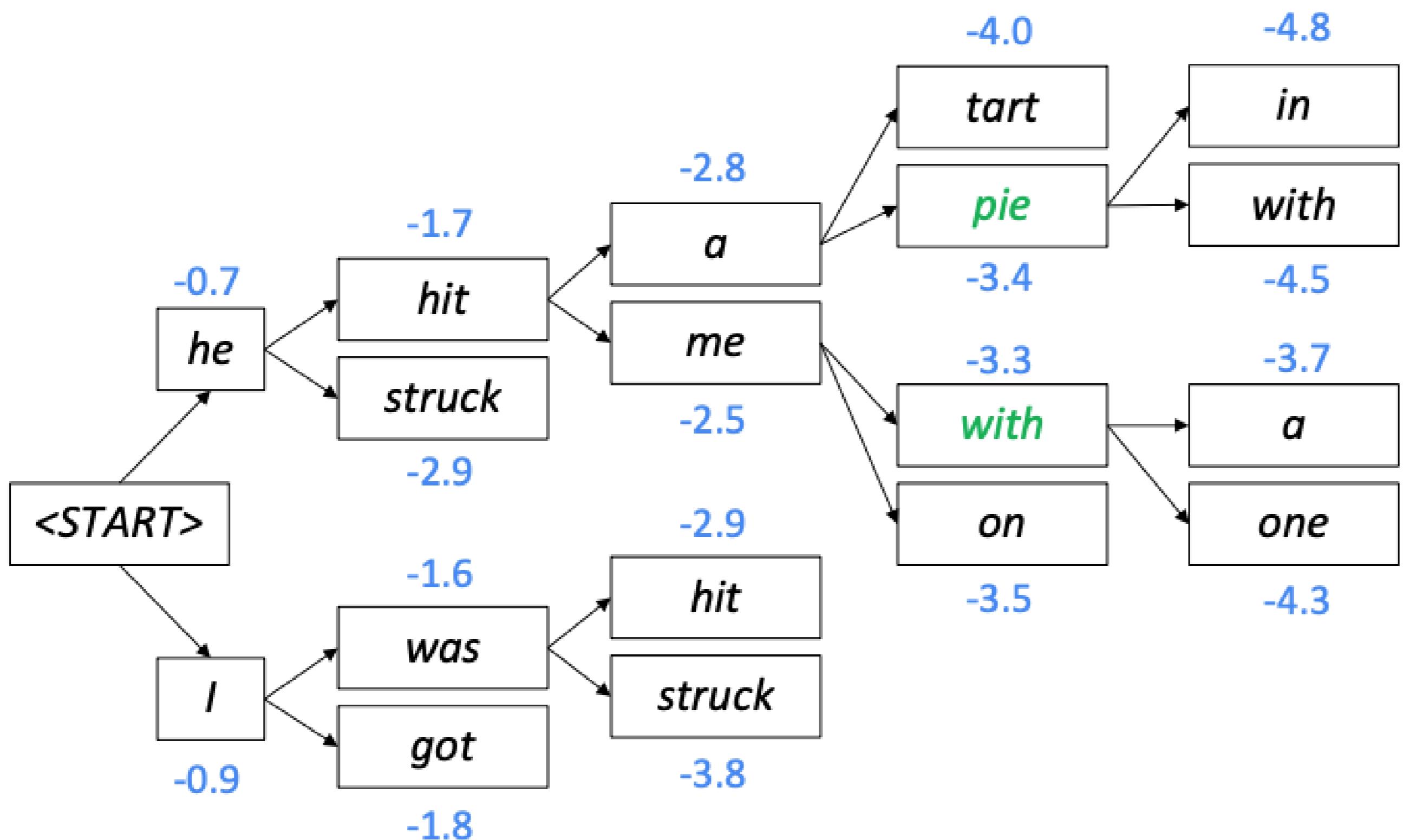
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: Example

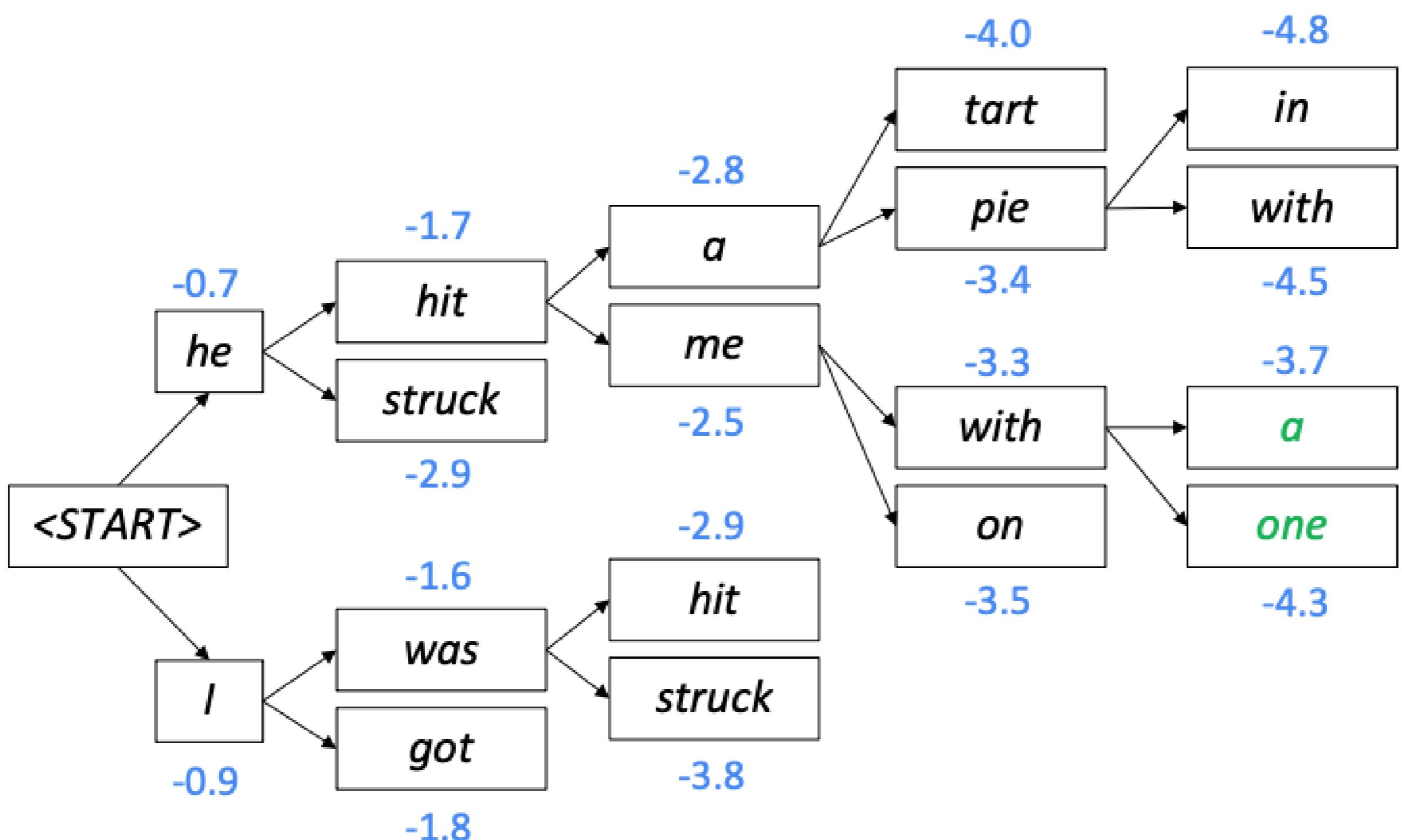
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



For each of the k hypotheses, find top k next words and calculate scores

Beam Search Decoding: Example

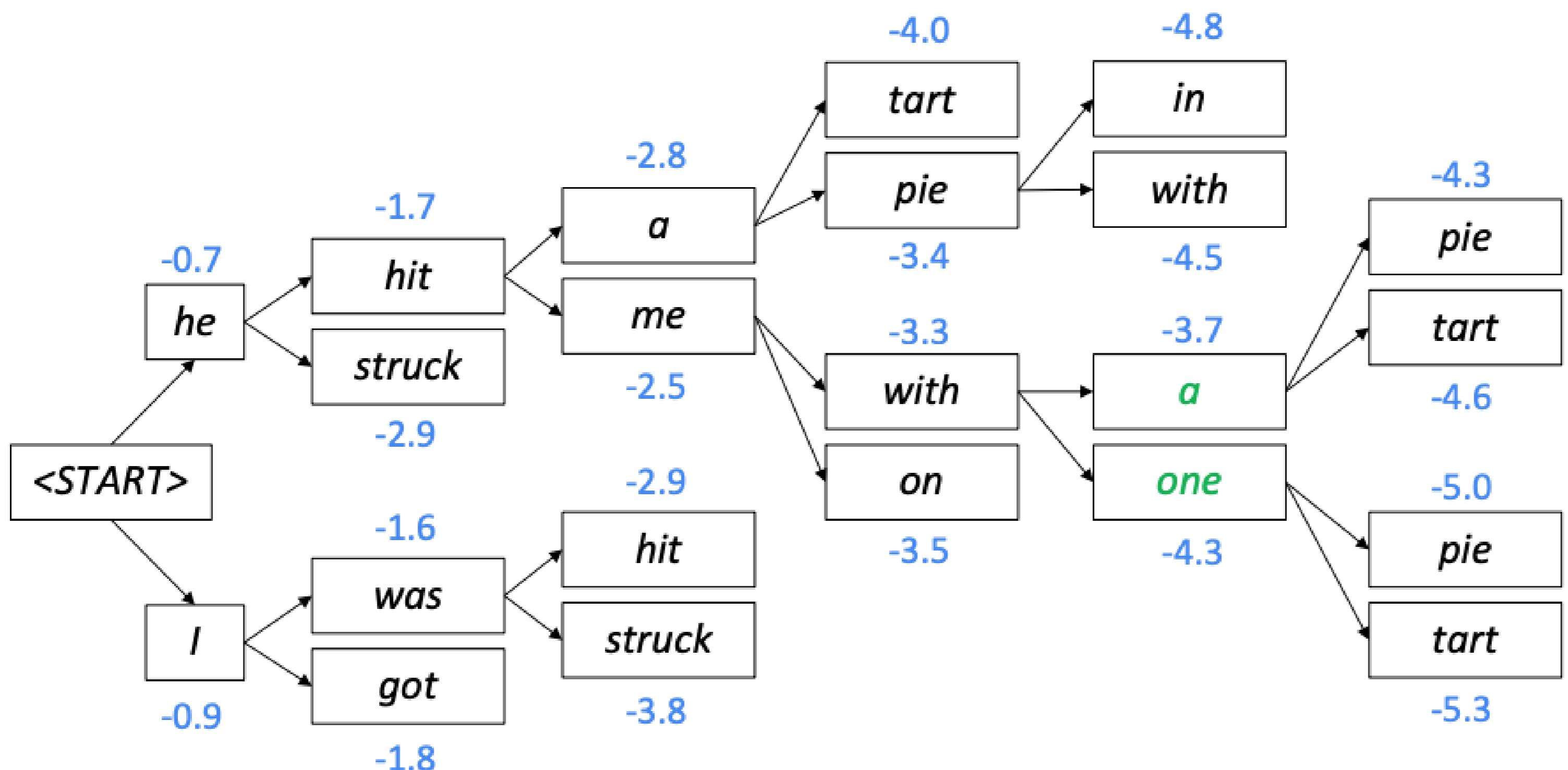
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: Example

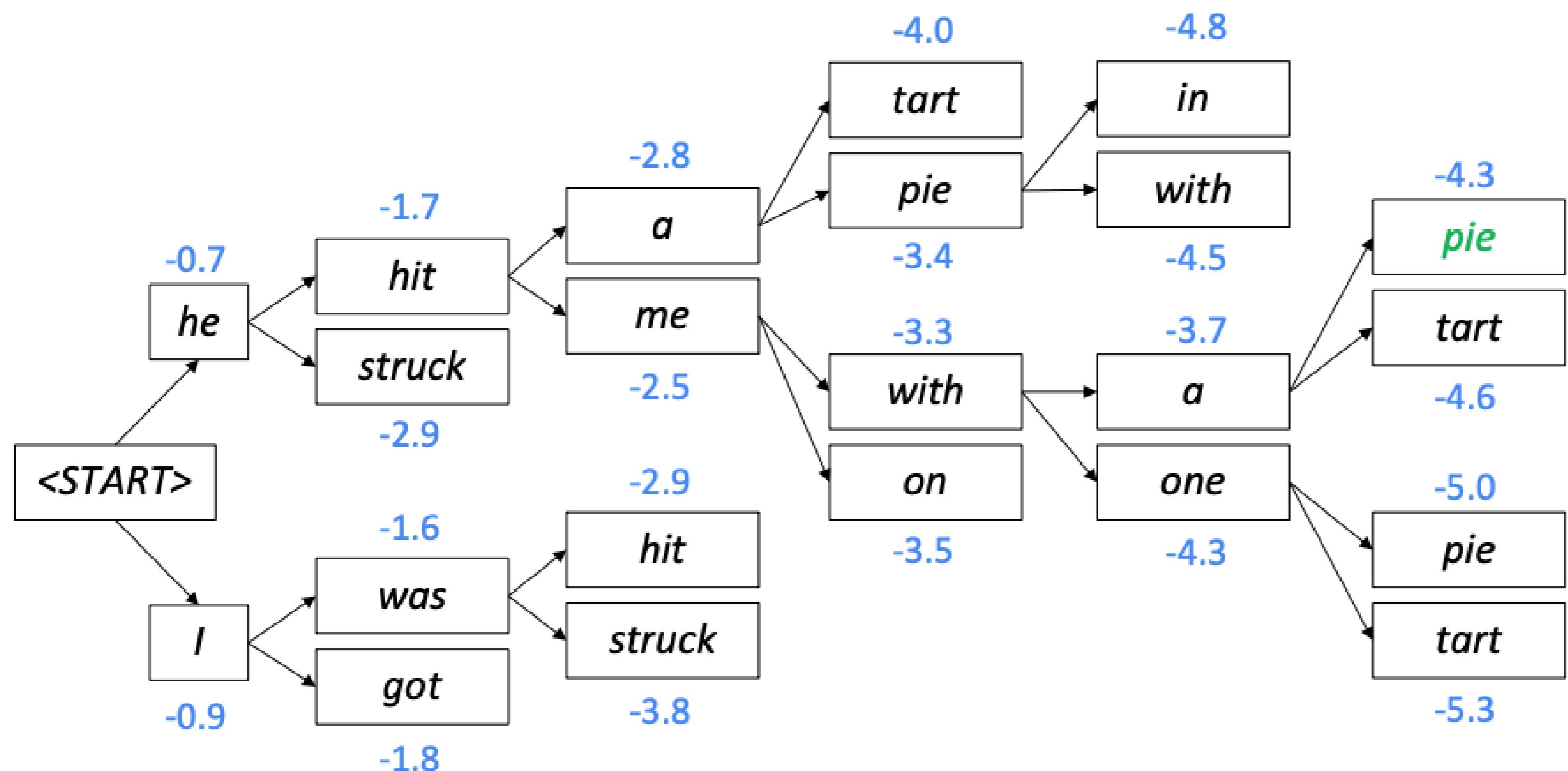
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



For each of the k hypotheses, find top k next words and calculate scores

Beam Search Decoding: Example

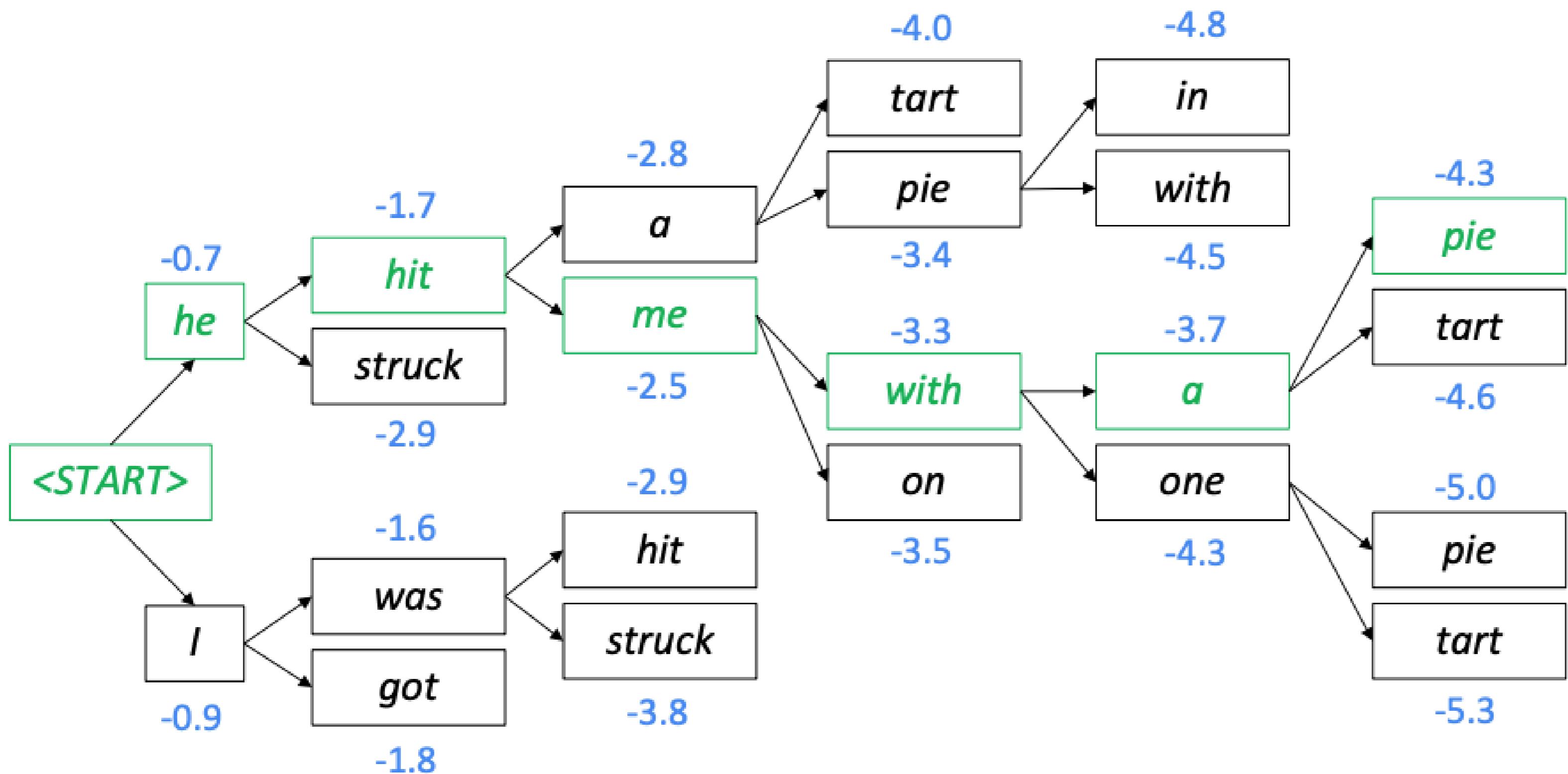
Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam Search Decoding: Example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Slide credit: Chris Manning

Beam Search Decoding: Stopping Criterion

- Greedy Decoding is done until the model produces an </s> token
 - For e.g. <s> he hit me with a pie </s>
- In Beam Search Decoding, different hypotheses may produce </s> tokens at different time steps
 - When a hypothesis produces </s>, that hypothesis is complete.
 - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
 - We reach time step T (where T is some pre-defined cutoff), or
 - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis y_1, \dots, y_t on our list has a score

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score
- Fix: Normalize by length. Use this to select top one instead

$$\frac{1}{t} \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

But this is expensive!

Issues with Maximization-Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive
- Another key issue:

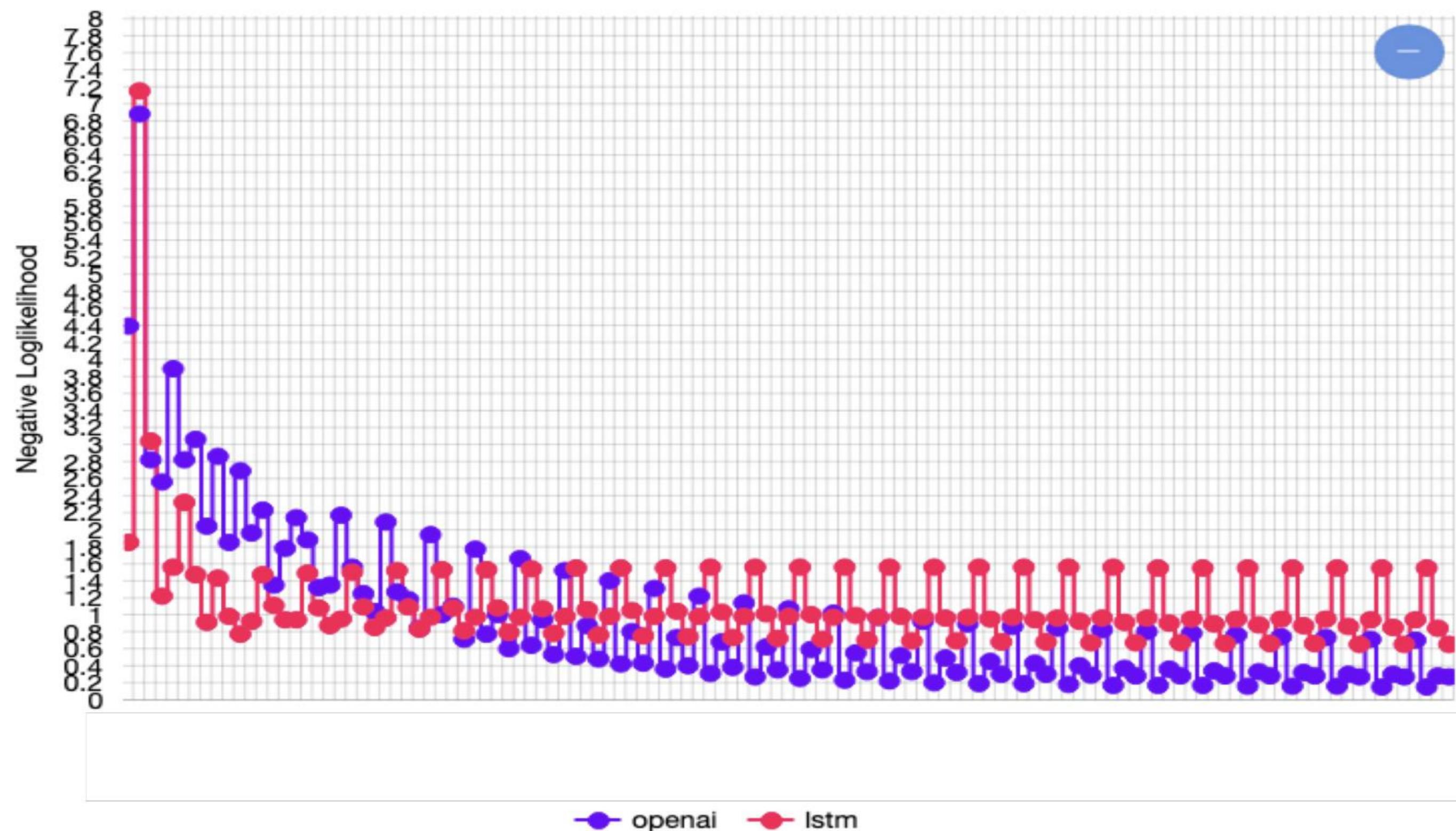
Generation can be bland or repetitive
(also called degenerate)

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México...)**

Degenerate Outputs

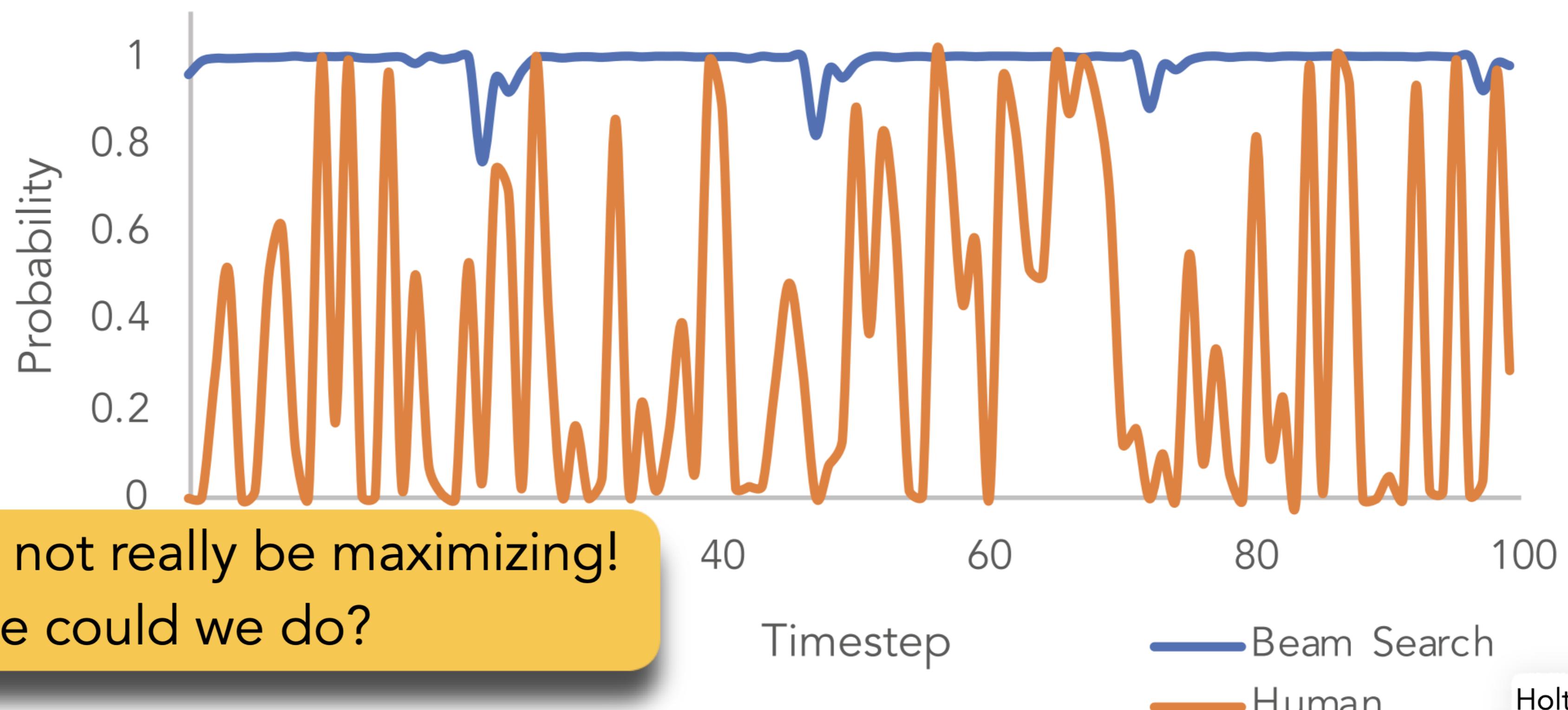
I'm tired. I'm tired.



Scale doesn't solve this problem: even a 175 billion parameter LM still repeats when we decode for the most likely string.

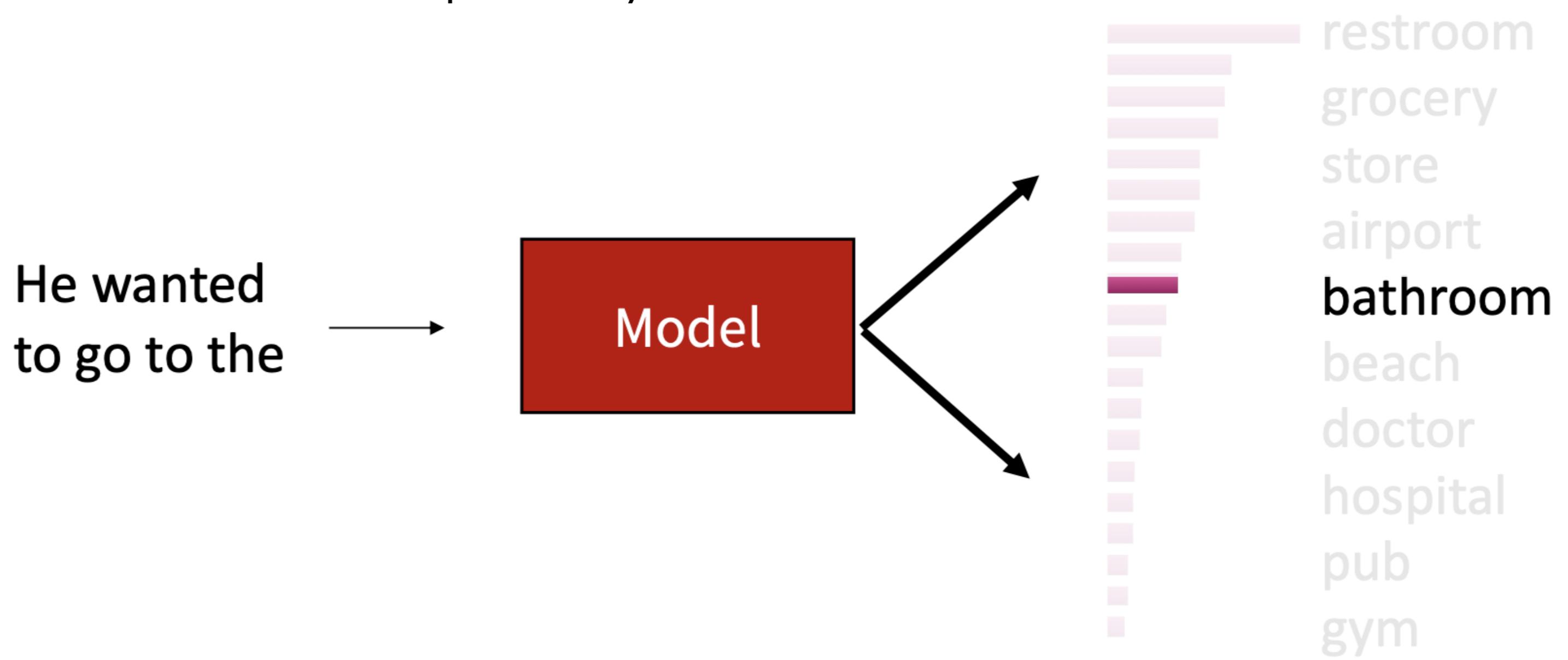
Why does repetition happen?

- Probability amplification due to maximization based decoding
- Generation fails to match the uncertainty distribution for human written text



Solution: Don't Maximize, Pick a Sample

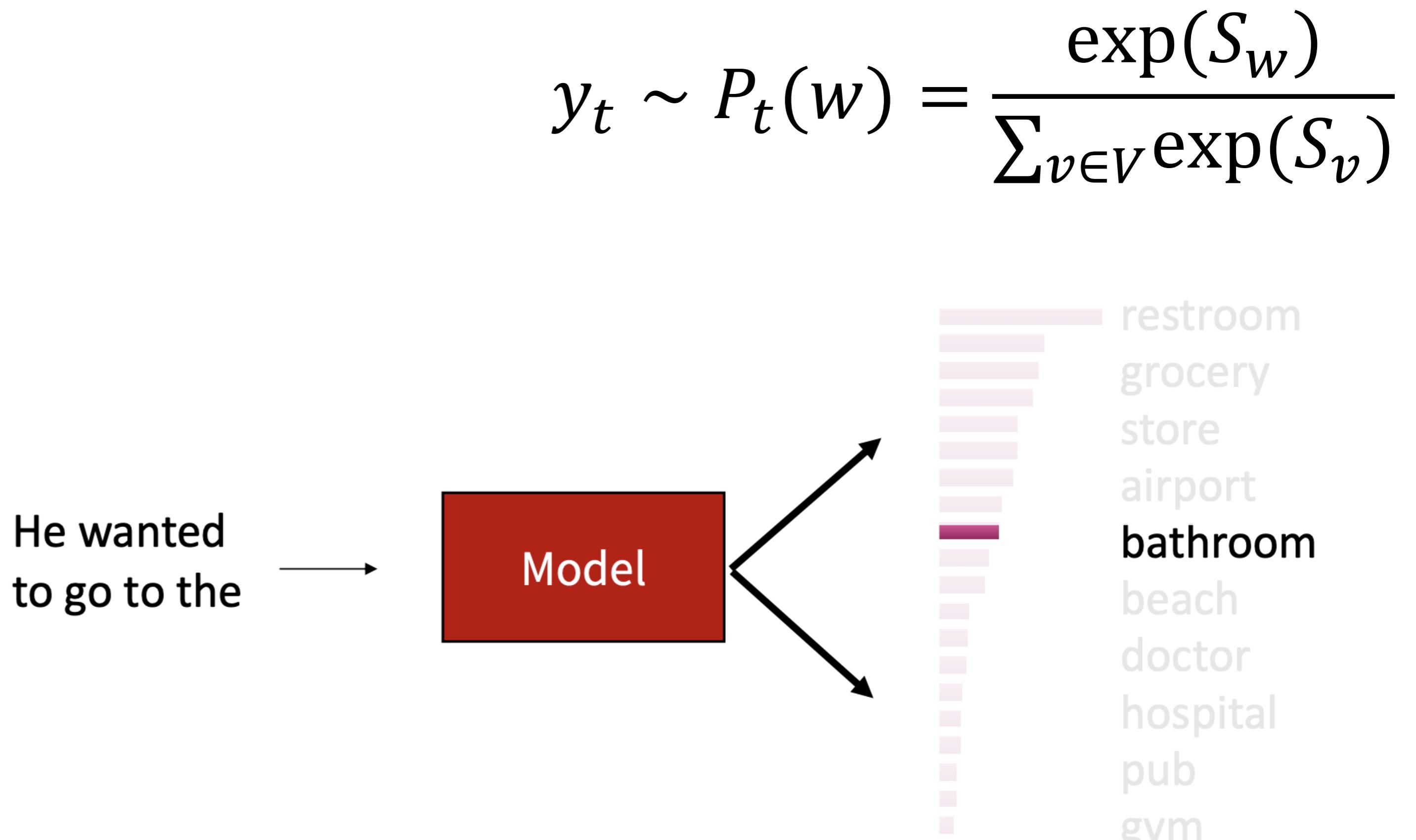
- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution
 - Respects the probabilities, without going just for the maximum probability option
 - Or else, you would get something meaningless
 - Many good options which are not the maximum probability!



Modern Generation: Sampling

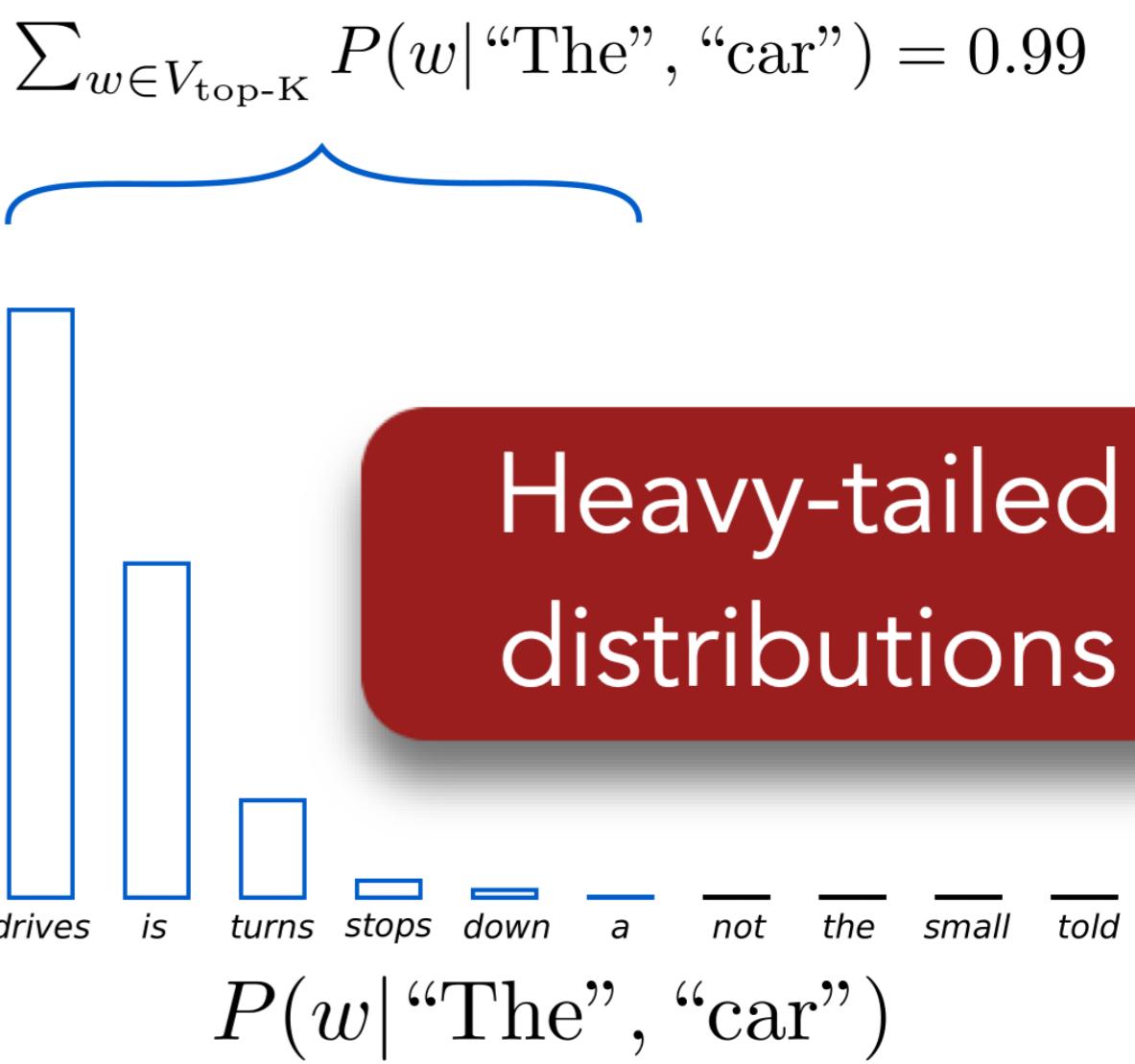
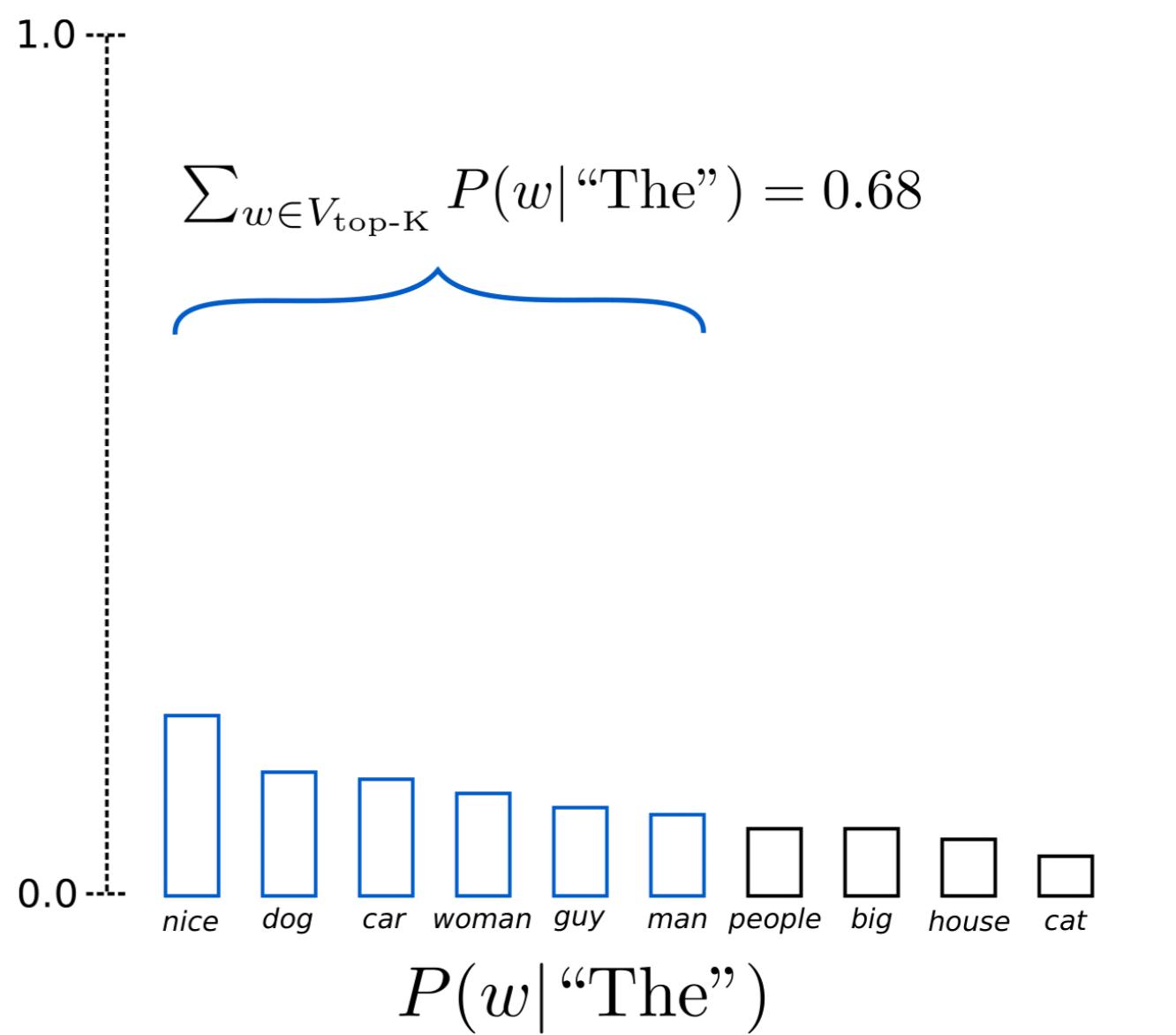
Pure Sampling

- Sample directly from P_t
- Still has access to the entire vocabulary
- But if the model distributions are of low quality, generations will be of low quality as well
- Often results in ill-formed generations
 - No guarantee of fluency



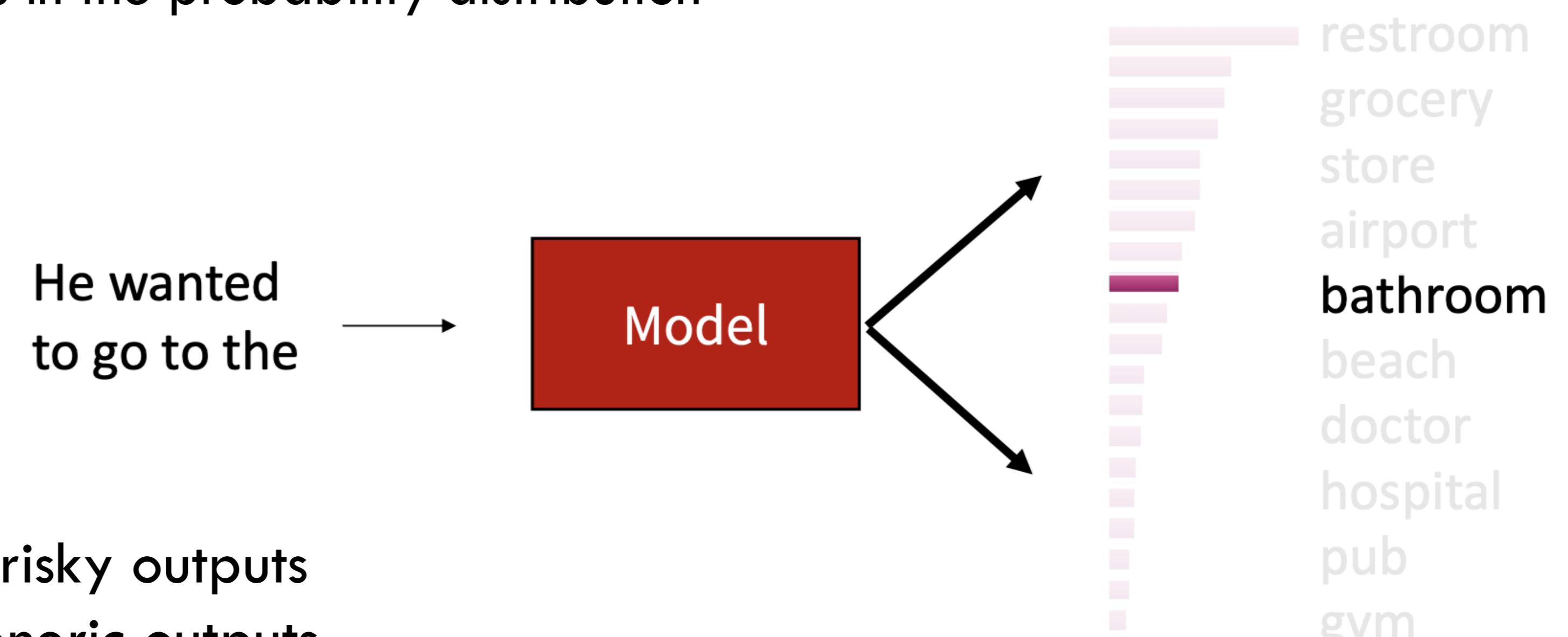
Top- K Sampling

- Problem: Pure sampling makes every token in the vocabulary an option
 - Many tokens are probably really wrong in the current context. Yet, we give them individually a tiny chance to be selected.
 - the tail of the distribution could be very long and in aggregate have considerable mass
- Solution: Top- K sampling
 - Only sample from the top K tokens in the probability distribution



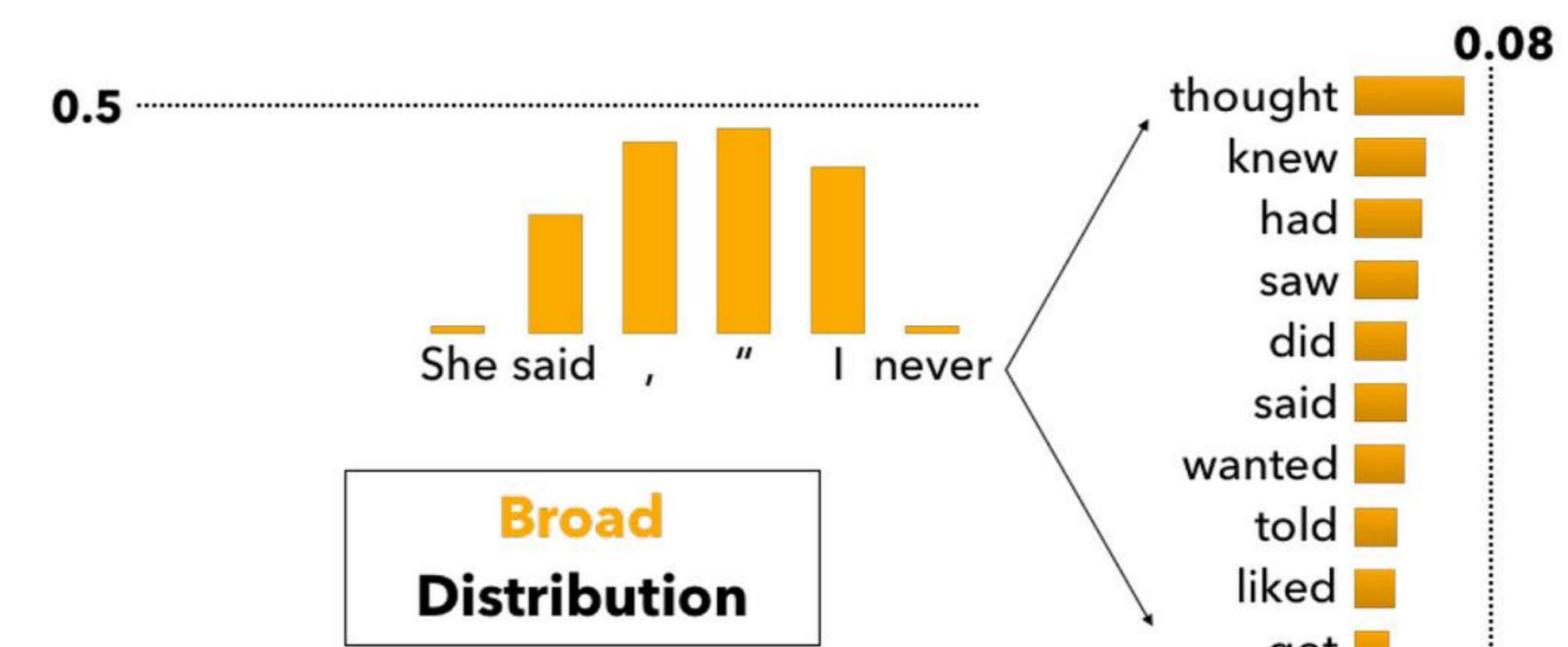
Top- K Sampling: Value of K

- Solution: Top- K sampling
 - Only sample from the top K tokens in the probability distribution
 - Common values are $K = 50$
- Increase K yields more diverse, but risky outputs
- Decrease K yields more safe but generic outputs



Top- K Sampling: Issues

Top- K sampling can cut off too quickly



Top- K sampling can also cut off too slowly!

We can do better than having one-size-fits-all: a fixed K for all contexts

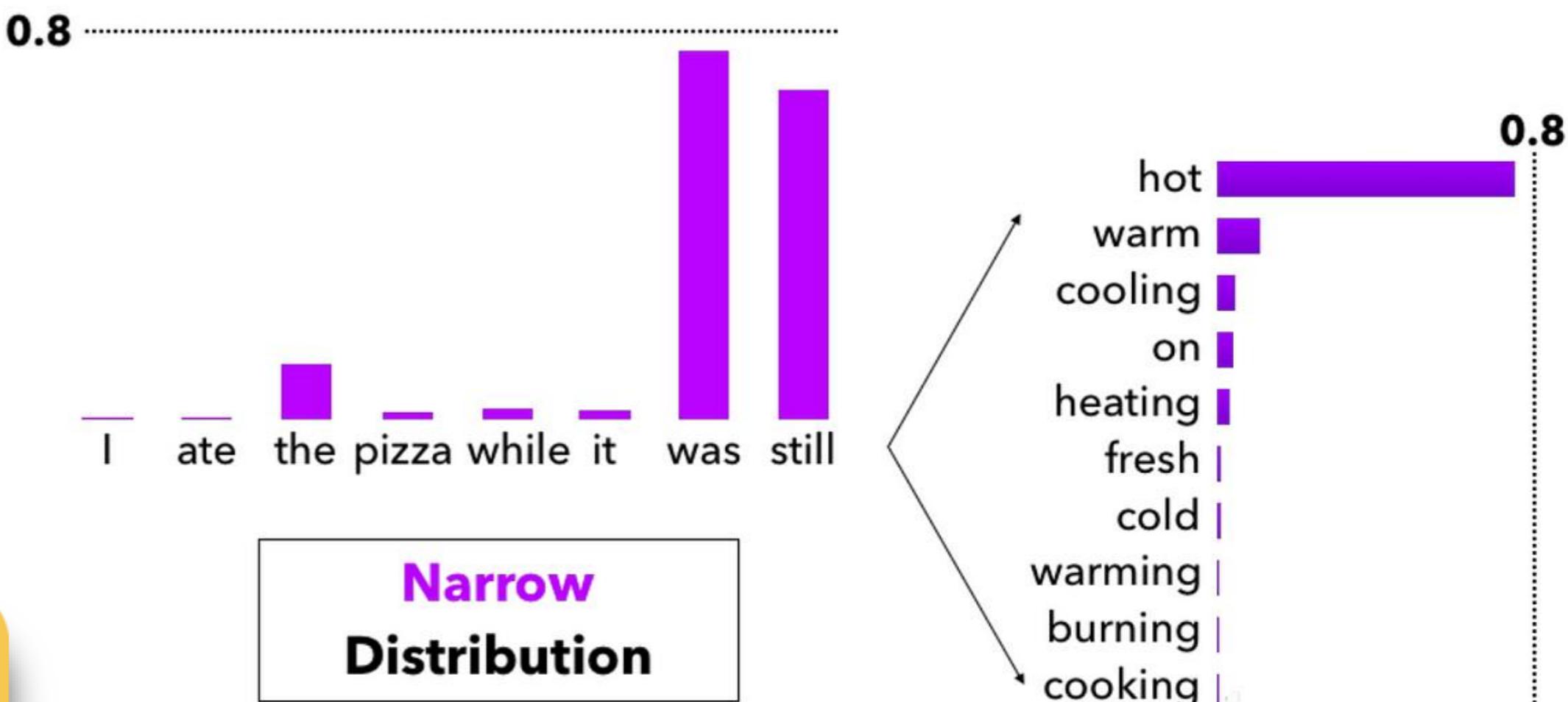


Image Source: Holtzmann et al., 2019

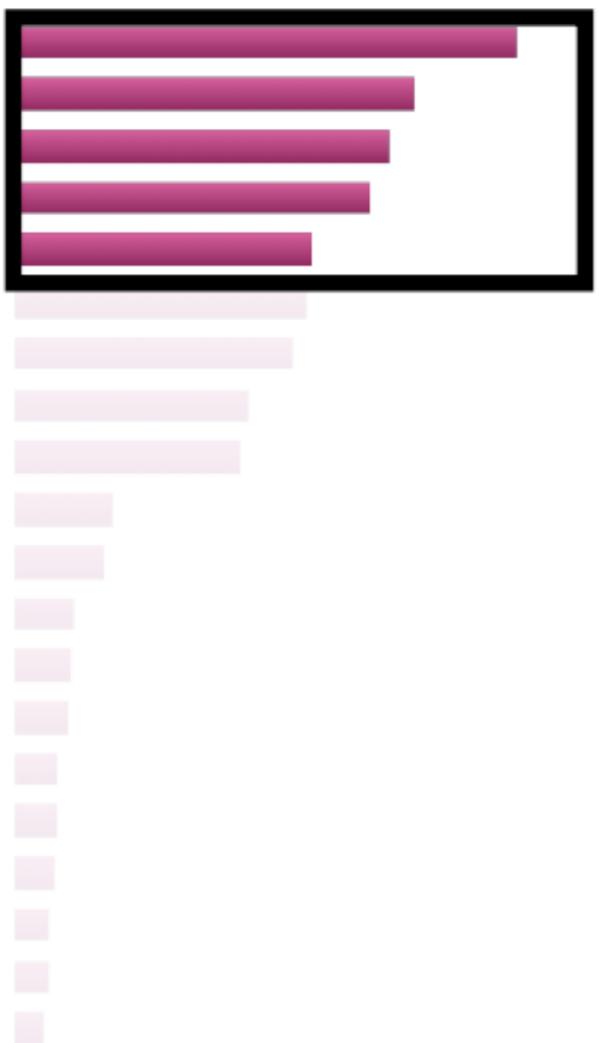
Modern Decoding: Nucleus Sampling

- Problem: The probability distributions we sample from are dynamic
 - When the distribution P_t is flatter, a limited K removes many viable options
 - When the distribution P_t is peakier, a high K allows for too many options to have a chance of being selected
- Solution: Nucleus Sampling / Top- P sampling
 - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
 - Varies K depending on the uniformity of P_t

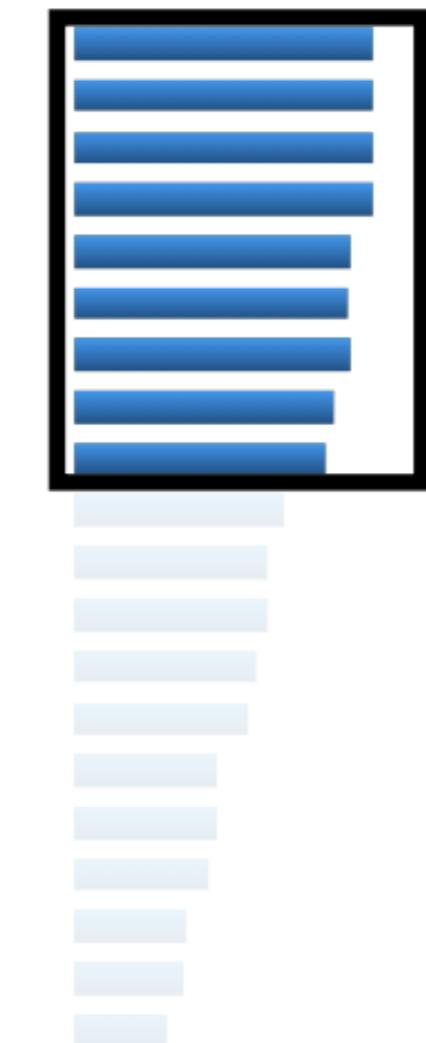
Nucleus (Top- P) Sampling

- Solution: Top- P sampling
 - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
 - Varies K depending on the uniformity of P_t

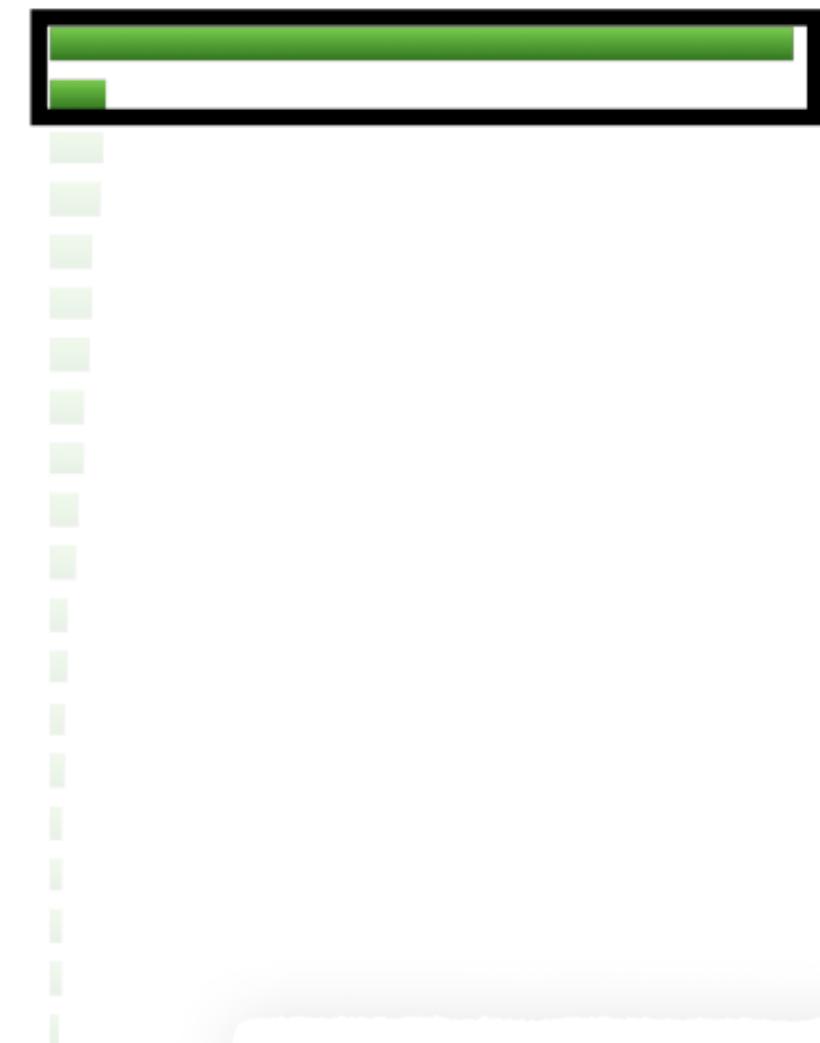
$$P_t^1(y_t = w | \{y\}_{<t})$$



$$P_t^2(y_t = w | \{y\}_{<t})$$

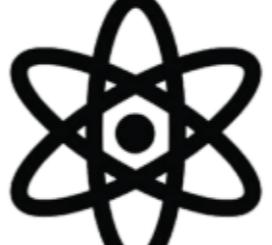


$$P_t^3(y_t = w | \{y\}_{<t})$$



Comparing different decoding algorithms

- Generate text to continue a given context
 - Open-ended generation
- Same decoding algorithms are also useful for close-ended generation tasks

	WebText	So what's new in my life? 09/11/18 - Just got back from vacation.
	Beam Search, $b=16$	Just got back from vacation. 09/10/18 - Just got back from vacation. Just got back from vacation. 09/09/18 - Just got back from vacation. Just got back from vacation. 09/08/18 - Just got back from vacation. Just got back from vacation.
	Pure Sampling	01/05/19 - Visited The Subaru Canada Heritage15th Star for my vehicle research. That was a wonderful experience. Thanks again for your knowledge and support. 05/23/19 - I was thrilled to receive my Subaru Legacy character! It fits me perfectly and makes me feel like a kid again! 06/15/19 - Just regained my licence for my different coloured 2016 WRX / STI...
	Sampling, $t=0.9$	Just got back from vacation. 09/07/18 - Happy 9th birthday to me! Happy 9th birthday to me! 09/01/18 - Sofwidden also! Sofwidden also! 08/25/18 - Happy birthday to me! Happy birthday to me! 08/17/18 - Night shift! □□Night shift! □□ 08/17/18 - Lucky me! Lucky me!
	Top-k, $k=640$	Just got back from vacation. 09/03/18 - After more than 40 years, I might have an apartment in Virginia Beach. After more than 40 years, I might have an apartment in Virginia Beach. 08/20/18 - Going for a hike at Mount Eerie in Northeast Virginia Spring Going for a hike at Mount Eerie in Northeast Virginia Spring
	Top-k, $k=640, t=0.7$	Just got back from vacation. 09/08/18 - I've gotten really sick. - I've gotten really sick. 09/07/18 - My wife and I are getting married in February. - My wife and I are getting married in February. 09/06/18 - I'm so excited to go back to college this fall. - I'm so excited to go back to college this fall.
	Nucleus, $p=0.95$	Just got back from vacation. 07/12/18 - Happy birthday to Swingu, who is nearly 5 years old. I would like to say hi to him on the road as well as when I ride with him. You cannot go to work without feeling physically sick or psychologically exhausted because you can barely breathe. Even if you ride on rollercoaster even once, it is easy to recover from the physical side of it.
	WebText	I just got back from a much needed and really great nine day vacation to my remote Arizona property. It was a really restful and relaxing visit. I got a lot accomplished while I was there, but still found time to just goof off and have fun too. I got to do some astronomy, even though the weather was pretty cloudy most of the time. Here is a 50 minute exposure of M101. It turned out pretty good.

Temperature Scaling

$$P(y_t = w) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$

- Recall: On timestep t , the model computes a prob distribution P_t by applying the softmax function to a vector of scores $S \in \mathbb{R}^{|V|}$
- We can apply a temperature hyperparameter τ to the softmax to rebalance P_t

$$P(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{v \in V} \exp(S_v/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
 - More diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
 - Less diverse output (probability is concentrated on top words)

Temperature is a hyperparameter for decoding: It can be tuned for both beam search and sampling.

Modern Decoding: Takeaways

- Natural language distributions are very peaky but the softmax function assigns probabilities to all tokens in the vocabulary
- Hence we need approaches to truncate / modify the softmax distribution
 - Ancestral, Top-k, Top-p (Nucleus), Temperature

Post-training of LLM

Three Stages of Training LLMs

Stage 1

- Pre-training on large corpus of text
- Produces a Base Language Model
- Continued Pre-training for domain adaptation (optional; sometimes called mid-training, stage 1.5)

Stage 2

- Post-training for Task Adaptation
- Seq2Seq Instruction Tuning (Supervised Finetuning, different meaning than BERT-style classification tasks)

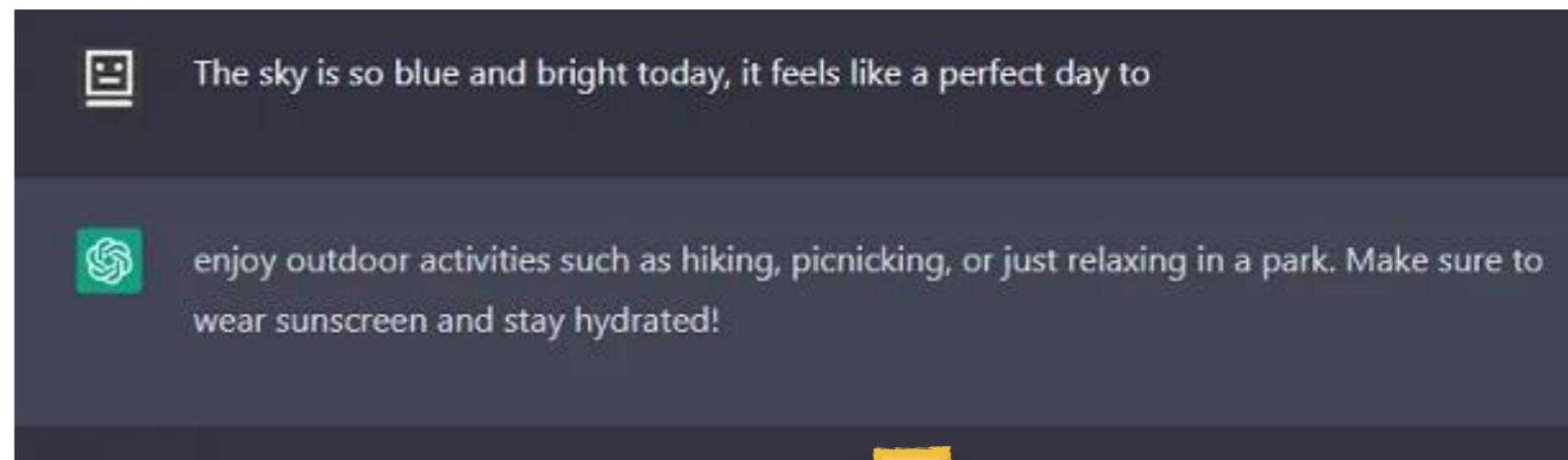
Stage 3

- Post-training for Preference Alignment
 - Either Reinforcement Learning with Human Feedback : Train a supervised classifier (reward model) on human demonstrations to provide feedback to LM. Reinforcement learning to maximize rewards given by reward model
 - Or, train with a preferred and a dispreferred generation (more popular now)

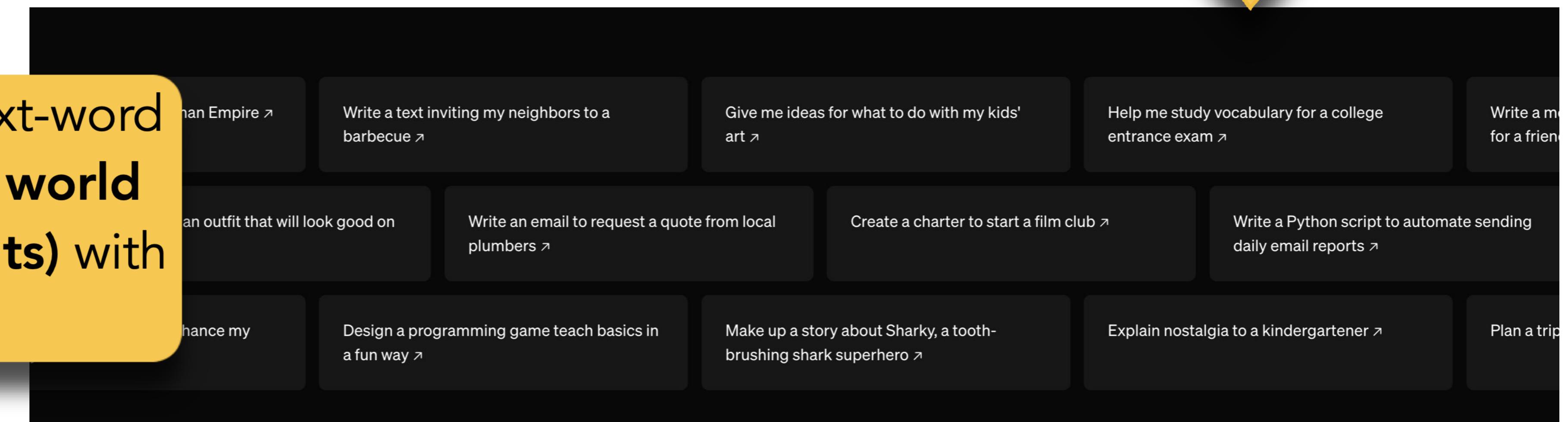
Inference Prompting with Instructions and Demonstrations (also called examples, shots)

Pre-training and Post-training

- The term “training” is no longer specific enough :)
- Pre-training: Self-supervised, standard next token prediction
- Post-training: Supervision (sequence to sequence)
- Instruction-Tuning: Supervision is not necessarily via labels, but sequence pairs.
Labels in standard NLP benchmarks can be converted into sequence pairs
- Preference-Tuning: Collects human judgments / preferences as rewards, or a pair of preferred / dispreferred generations



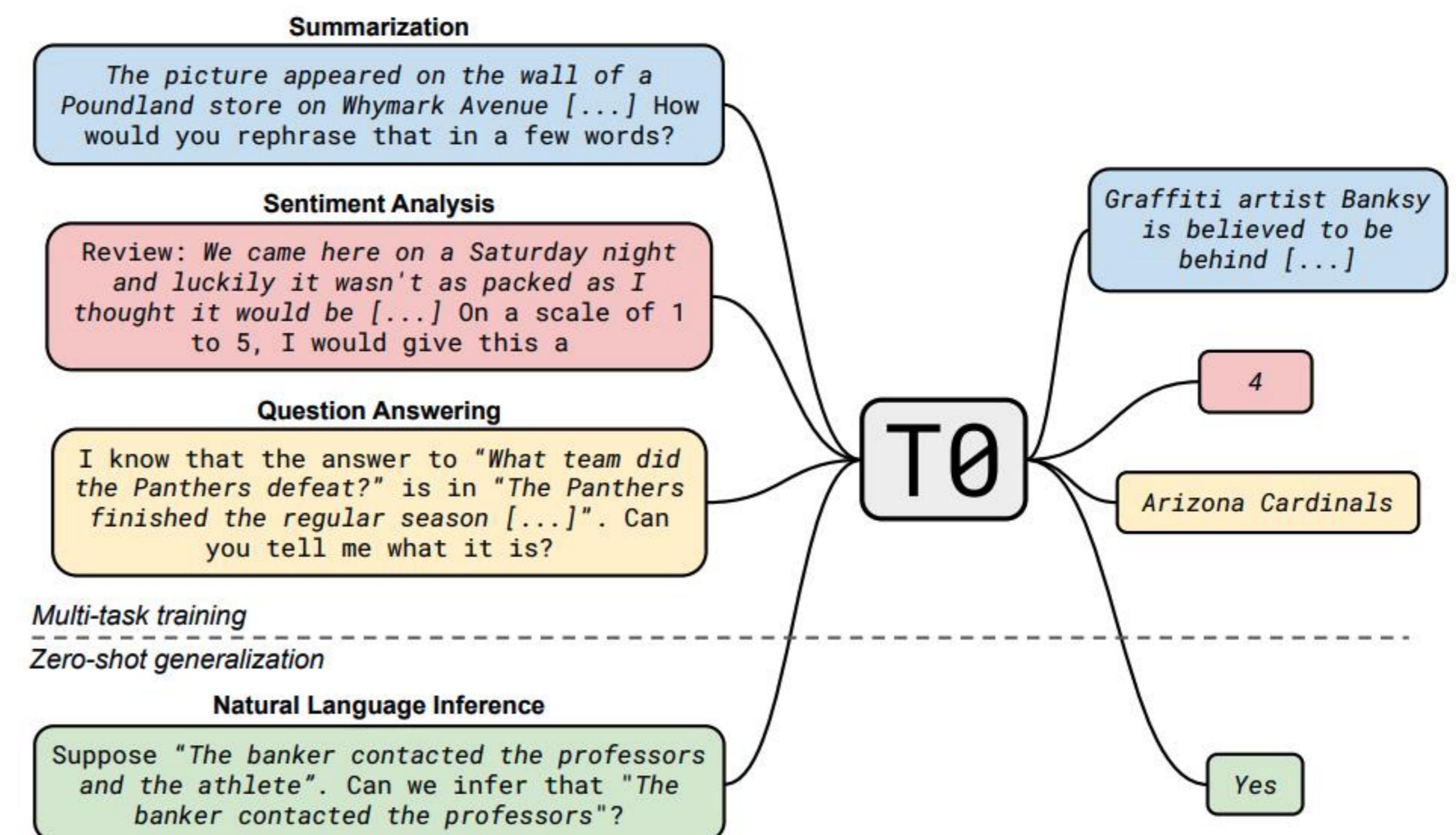
Post-training converts next-word completion models into **world models (agents, assistants)** with many capabilities!



Instruction-Tuning LLMs for Task Adaptation

Instruction Tuning

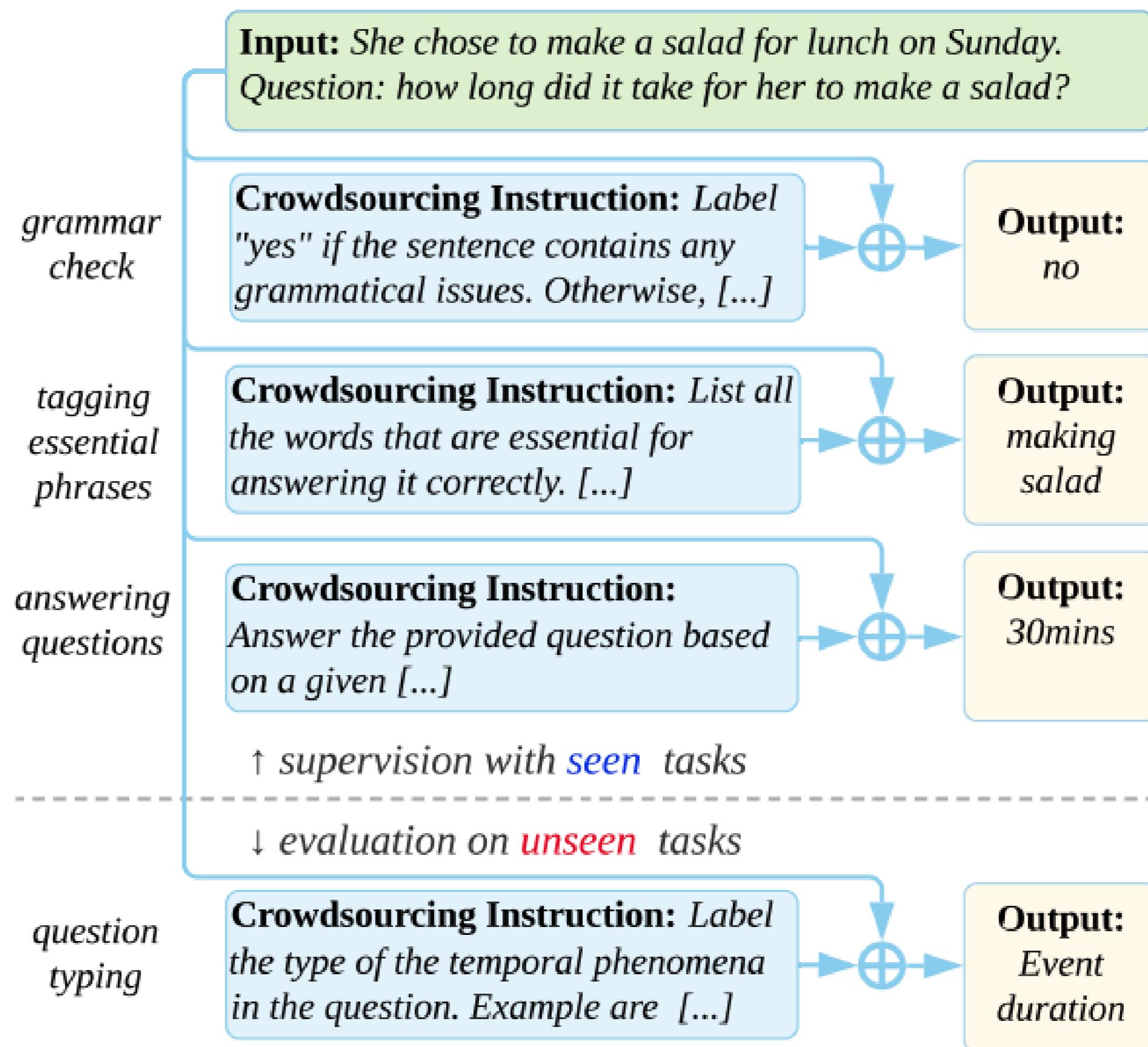
- Pretraining:
 - Train a model to continue a given context
- Instruction Tuning:
 - Train a model to follow varied instructions
 - Needed because the vast majority of pretraining is done on data which are not in the form of instructions
 - Fine-tuned (using the next-token-prediction objective) on a dataset of instructions together with correct responses



"Multitask Prompted Training Enables Zero-Shot Task Generalization" (Sahn et al., 2022)

Instruction Tuning and Task Generalization

- During instruction tuning, the model learns to follow instructions of given tasks
- At test time, it generalizes to follow instructions on unseen tasks!



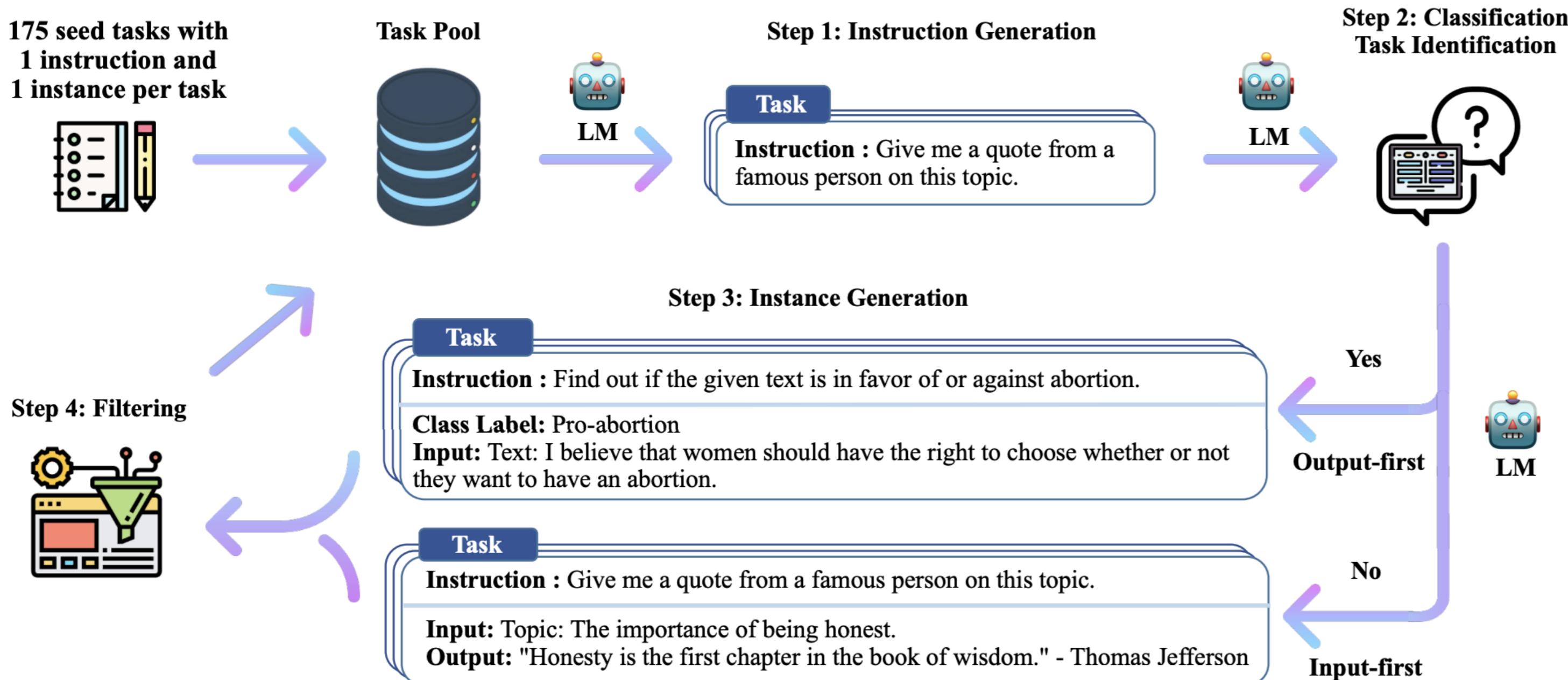
"Cross-Task Generalization via Natural Language Crowdsourcing Instructions" (Mishra et al., 2022)

Instruction Tuning Data

More data (instructions)
→ better model

Resource →	SUP-NATINST (this work)	NATINST (Mishra et al., 2022b)	CROSSFIT (Ye et al., 2021)	PROMPTSOURCE (Bach et al., 2022)	FLAN (Wei et al., 2022)	INSTRUCTGPT (Ouyang et al., 2022)
Has task instructions?	✓	✓	✗	✓	✓	✓
Has negative examples?	✓	✓	✗	✗	✗	✗
Has non-English tasks?	✓	✗	✗	✗	✓	✓
Is public?	✓	✓	✓	✓	✓	✗
Number of tasks	1616	61	269	176	62	—
Number of instructions	1616	61	—	2052	620	14378
Number of annotated tasks types	76	6	13	13*	12	10
Avg. task definition length (words)	56.6	134.4	—	24.8	8.2	—

Super-NaturalInstructions. (Wang et al., 2022)



Diverse data (instructions)
→ better model

"Self-Instruct: Aligning Language Models with Self-Generated Instructions" (Wang et al., 2023)

Instruction Tuning Data

- Instruction tuning datasets are often created by repurposing standard NLP datasets for tasks like question answering or machine translation
- Often synthesized!
 - Prompting existing LLMs
- More variety in the instruction templates → better models!

Release	Collection	Model	Model Details			Data Collection & Training Details				Methods
			Base	Size	Public?	Prompt Types	Tasks in Flan	# Exs	Methods	
2020 05	UnifiedQA	UnifiedQA	RoBERTa	110-340M	P	ZS	46 / 46	750k		
2021 04	CrossFit	BART-CrossFit	BART	140M	NP	FS	115 / 159	71M		
2021 04	Natural Inst v1.0	Gen. BART	BART	140M	NP	ZS / FS	61 / 61	620k	+ Detailed k-shot Prompts	
2021 09	Flan 2021	Flan-LaMDA	LaMDA	137B	NP	ZS / FS	62 / 62	4.4M	+ Template Variety	
2021 10	P3	T0, T0+, T0++	T5-LM	3-11B	P	ZS	62 / 62	12M	+ Template Variety + Input Inversion	
2021 10	MetalCL	MetalCL	GPT-2	770M	P	FS	100 / 142	3.5M	+ Input Inversion + Noisy Channel Opt	
2021 11	ExMix	ExT5	T5	220M-11B	NP	ZS	72 / 107	500k	+ With Pretraining	
2022 04	Super-Natural Inst.	Tk-Instruct	T5-LM, mT5	11-13B	P	ZS / FS	1556 / 1613	5M	+ Detailed k-shot Prompts + Multilingual	
2022 10	GLM	GLM-130B	GLM	130B	P	FS	65 / 77	12M	+ With Pretraining + Bilingual (en, zh-cn)	
2022 11	xP3	BLOOMz, mT0	BLOOM, mT5	13-176B	P	ZS	53 / 71	81M	+ Massively Multilingual	
2022 12	Unnatural Inst. [†]	T5-LM-Unnat. Inst.	T5-LM	11B	NP	ZS	~20 / 117	64k	+ Synthetic Data	
2022 12	Self-Instruct [†]	GPT-3 Self Inst.	GPT-3	175B	NP	ZS	Unknown	82k	+ Synthetic Data + Knowledge Distillation	
2022 12	OPT-IML Bench [†]	OPT-IML	OPT	30-175B	P	ZS + FS CoT	~2067 / 2207	18M	+ Template Variety + Input Inversion + Multilingual	
2022 10	Flan 2022 (ours)	Flan-T5, Flan-PaLM	T5-LM, PaLM	10M-540B	P NP	ZS + FS CoT	1836	15M	+ Template Variety + Input Inversion + Multilingual	

"The Flan Collection: Designing Data and Methods for Effective Instruction Tuning" (Longpre et al., 2023)

Instruction Tuning: Masking Instructions

- We're still using decoder-only models (the same as we used in pre-training)
- How to update these models for an encoder-decoder like behavior?
- The instruction itself is masked, so the model does not generate instructions

In the loss function, mask out the tokens corresponding to prompt

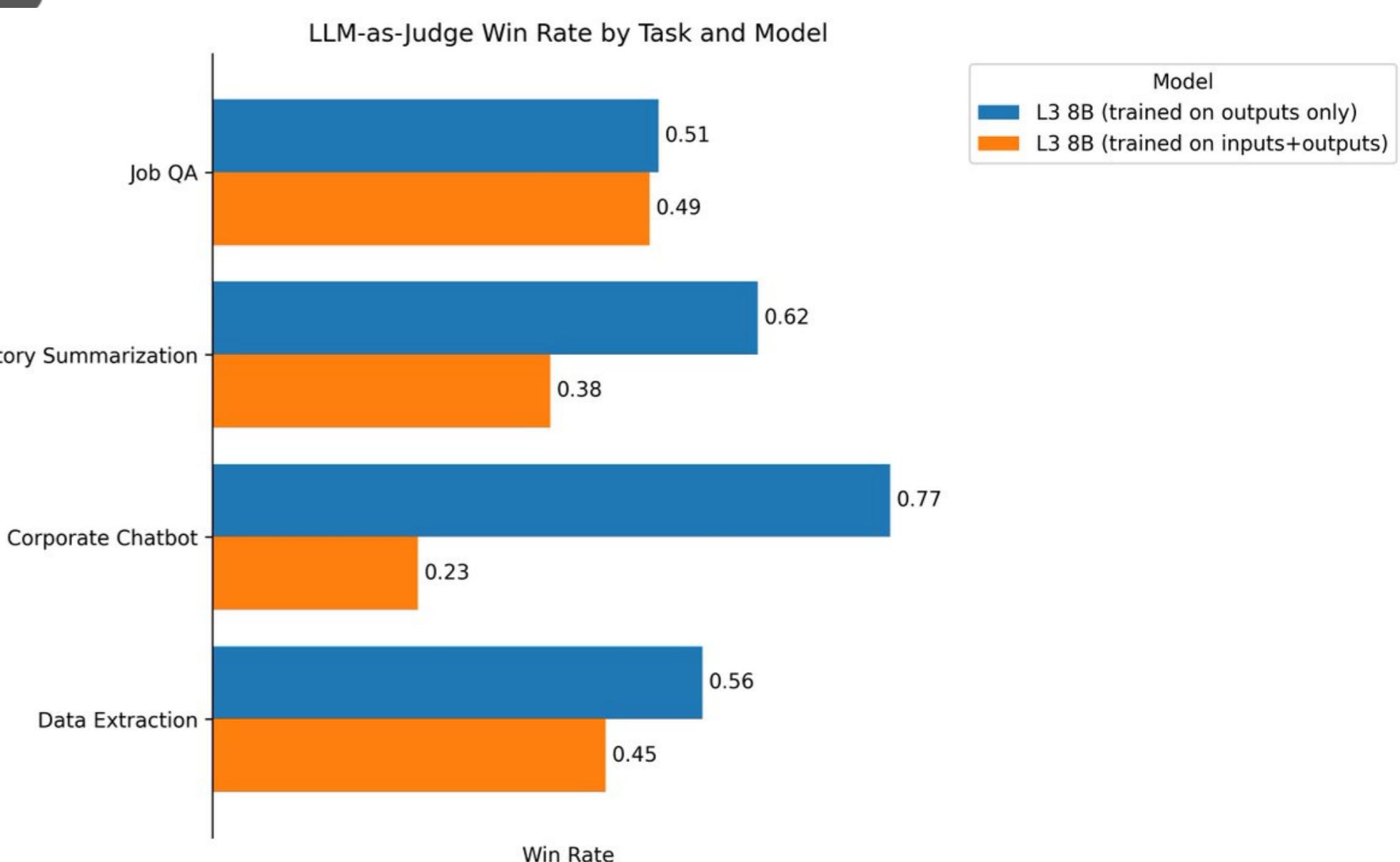
Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:
Rewrite the following sentence using passive voice.

Input:
The team achieved great results.

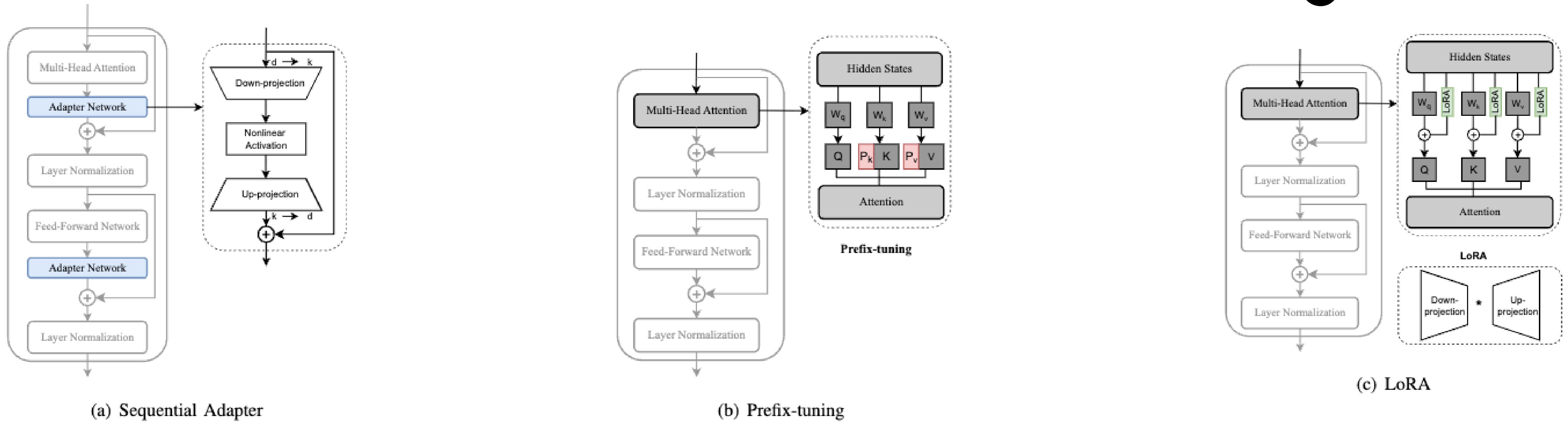
Response:
Great results were achieved by the team.

Calculate the loss only over the tokens corresponding to the response text



An illustration of input masking where the highlighted text is still fed to the LLM, but it is not used when calculating the loss during training.

Parameter-Efficient Fine-tuning



- Fine-tuning can be very difficult and expensive with LLMs
 - enormous numbers of parameters to train; think 70B parameters, and their gradients!!
 - each pass of batch gradient descent has to backpropagate through many many huge layers.
- Alternative: allow a model to be finetuned without changing all the parameters.
 - parameter-efficient fine tuning or PEFT,
 - efficiently select a subset of parameters to update when finetuning and keep the rest frozen
- Examples: Adapters, Prefix-Tuning, LoRA or Low Rank Adaptation

LoRA: Low-Rank Adaptation

- Instead of updating weight matrices in attention layers during finetuning, Low-Rank Adaptation eases this by updating a low-rank approximation of the matrix : matrices which are far smaller
- LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture
- Gradient updates are reparametrized as $\mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{B} \cdot \mathbf{A}$
 - Where \mathbf{B} and \mathbf{A} are low-rank matrices, the only matrices to be updated
- LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times for GPT-3 175B
- Usually comes at a (small) cost to performance

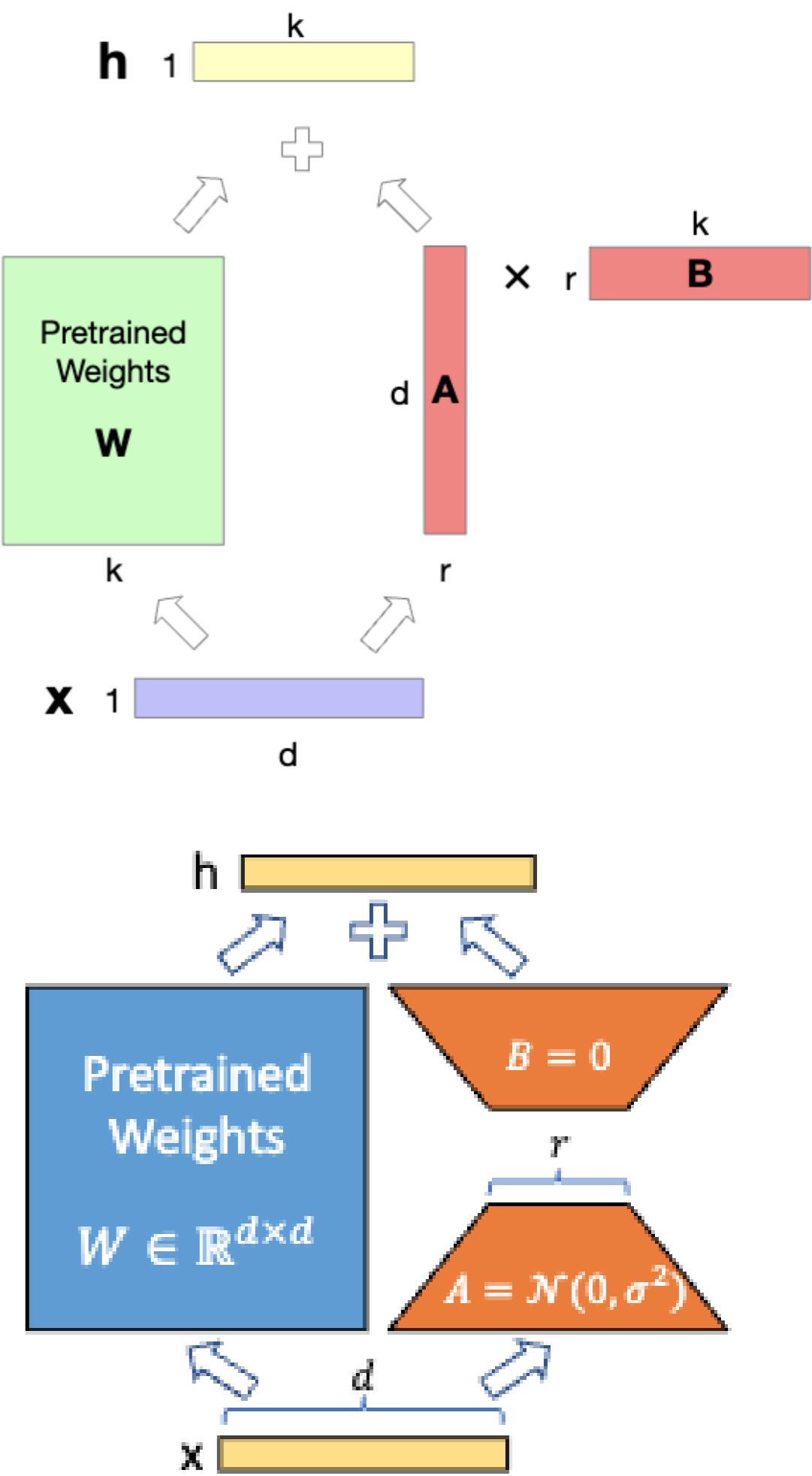


Figure 1: Our reparametrization. We only train A and B .

Interacting with LLMs: Prompting

Interfacing with Large Language Models

- Once trained, language models can be very powerful
 - Most tasks in NLP can be formatted as sequence completion tasks
 - How to interface with a language model to extract relevant information?
- Prompting (or In-Context / Few-Shot Learning): the ability to do many tasks with no gradient updates and no / a few examples, by simply:
 - Specifying the right sequence prediction problem
 - You can get interesting zero-shot behavior if you're creative enough with how you specify your task!

Basic Prompt Templates

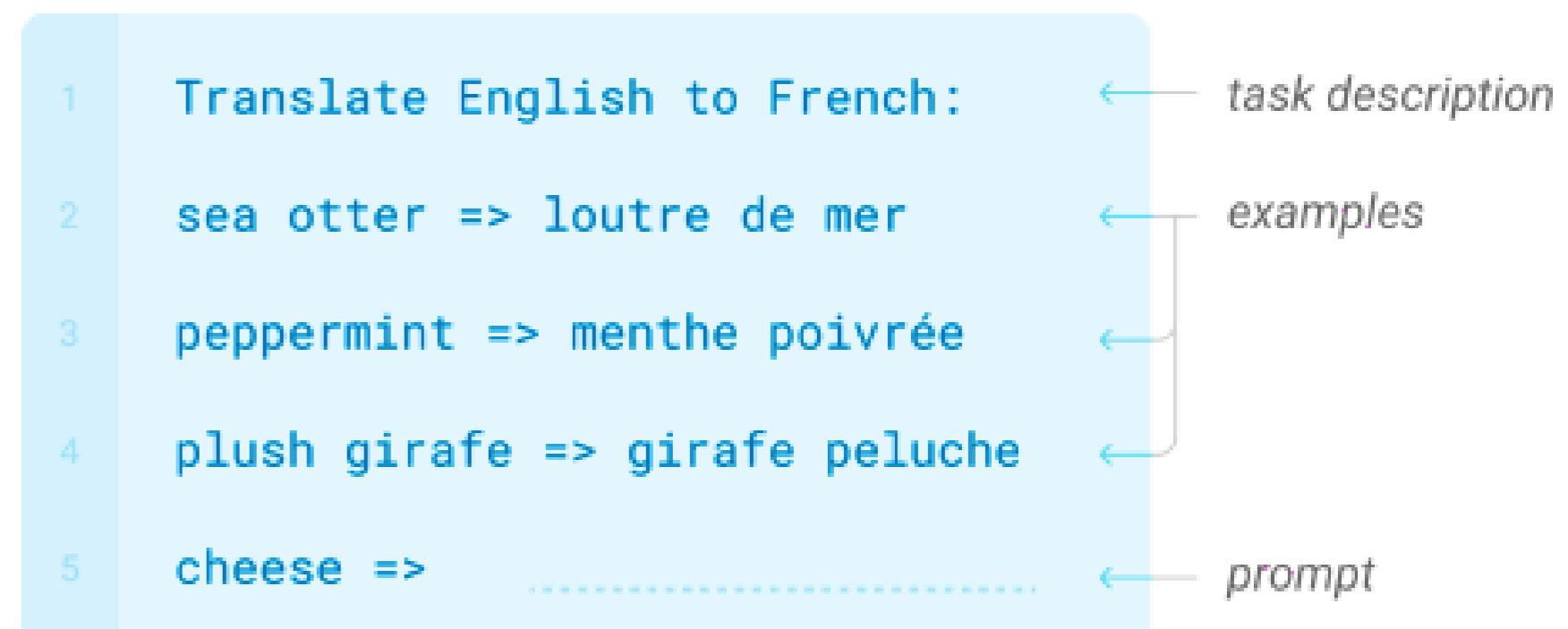
Summarization	{input} ; tldr;
Translation	{input} ; translate to French:
Sentiment	{input}; Overall, it was
Fine-Grained-Sentiment	{input}; What aspects were important in this review?

Prompting

- Interface to a language model: prompts in natural language
- Very large language models seem to perform some kind of “learning” without gradient updates!!! “Learn” simply from examples you provide within their contexts
 - Sometimes called in-context learning
 - Misnomer: no learning (parameter update) actually happens during prompting
 - Can be zero shot (without examples) or few-shot (with a few examples)
 - Typically <10
 - With powerful LLMs, 0-shot approaches are all you need!



Zero-Shot



Few-Shot

“Language Models are Few-Shot Learners” (Brown et al., 2020)

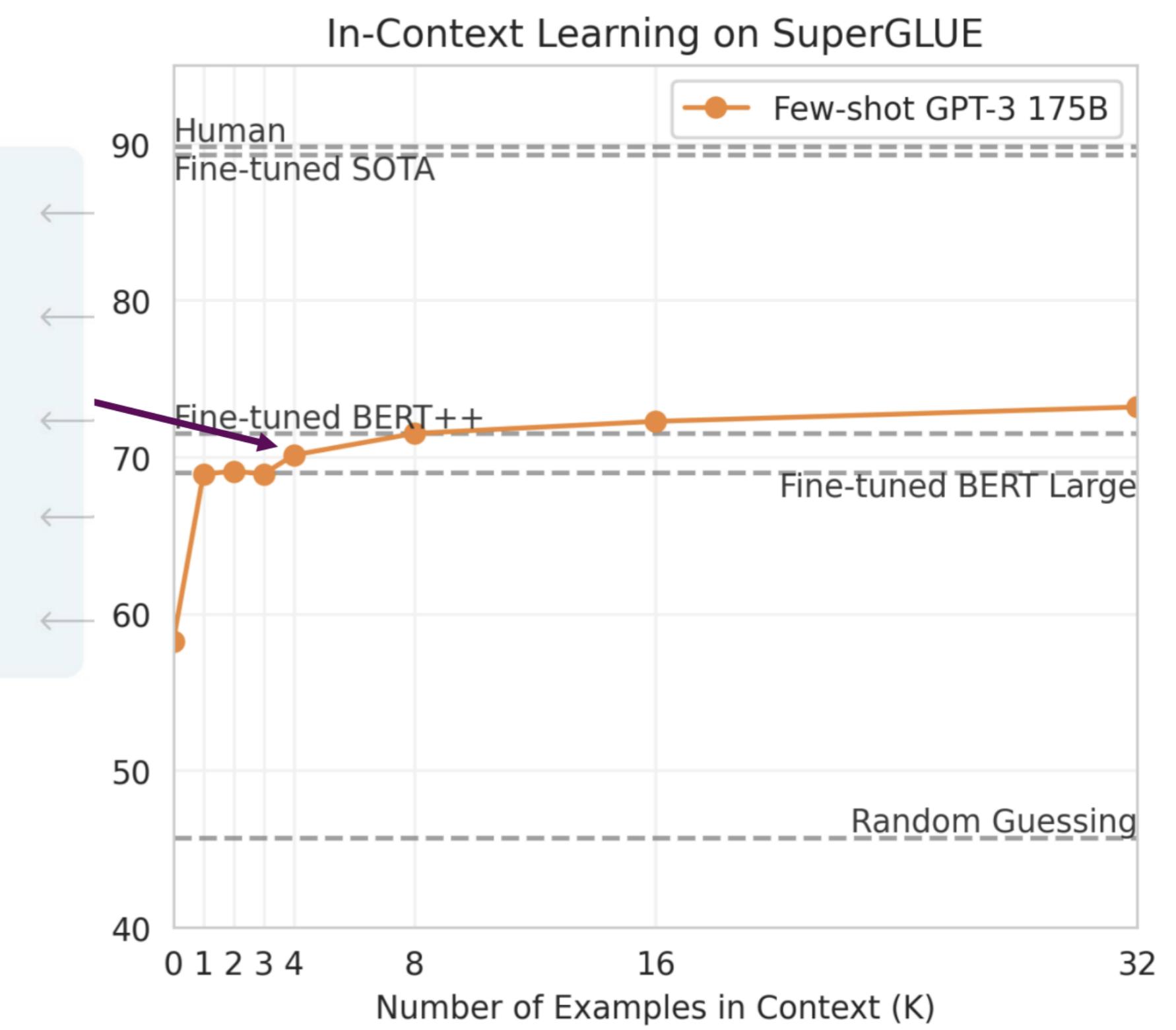
Prompting: Success

- Much more flexible than older formulation of pre-training encoder-only models and fine-tuning to specific classification tasks (the BERT paradigm)
- Now, pre-train and instruction tune one large model and prompt it to do a variety of tasks!
- Much much more generalizability!

Emergent few-shot learning

Few-shot

1	Translate English to French:
2	sea otter => loutre de mer
3	peppermint => menthe poivrée
4	plush girafe => girafe peluche
5	cheese =>



[Brown et al., 2020]

Why does prompting work so well?

- Induction heads
- Discovered by looking at mini language models with only 1-2 attention heads
- If the model sees the pattern AB ... A in an input sequence, it predicts that B will follow, instantiating the pattern completion rule $AB\dots A \rightarrow B$
- Perhaps a generalized fuzzy version of this pattern completion rule, implementing a rule like $A^*B^* \dots A \rightarrow B$, where $A^* \approx A$ and $B^* \approx B$ (by \approx we mean some form of semantically similarity), might be responsible for in-context learning

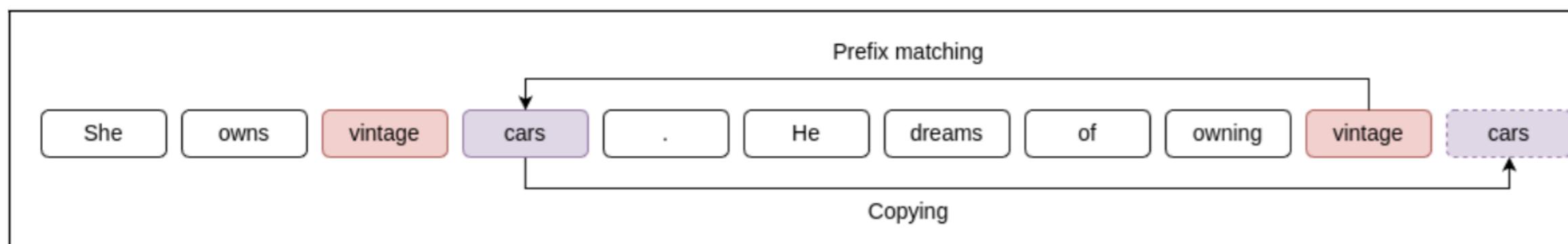
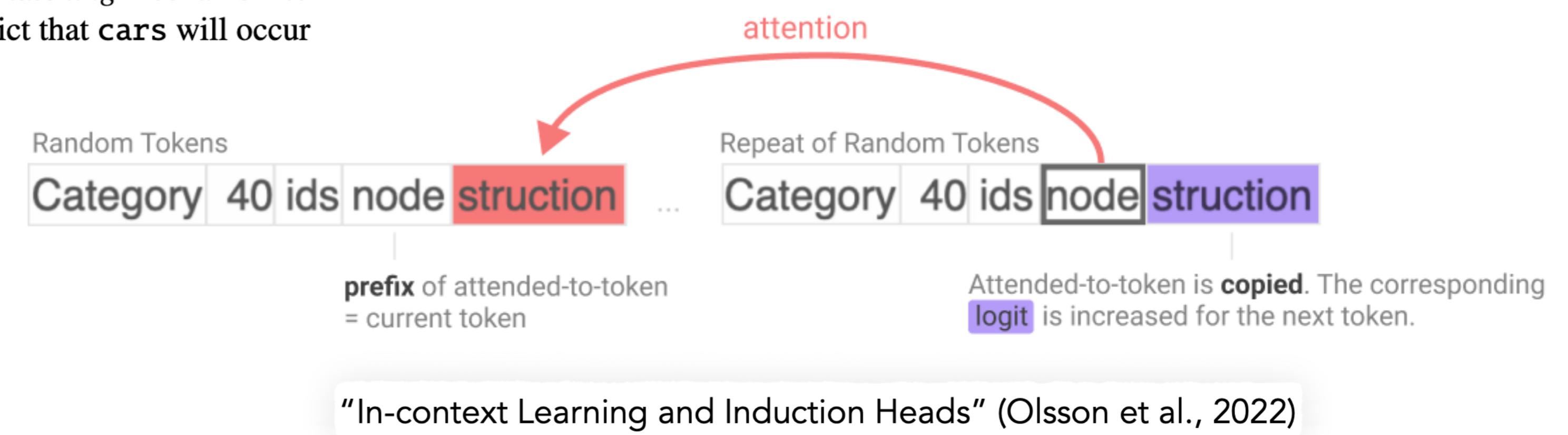


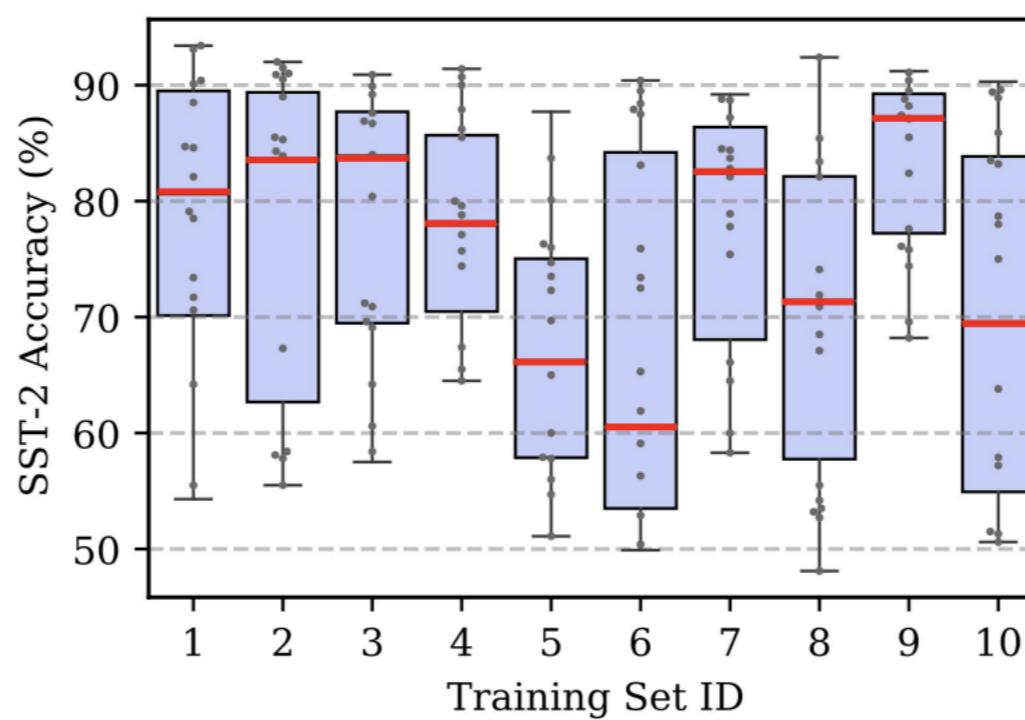
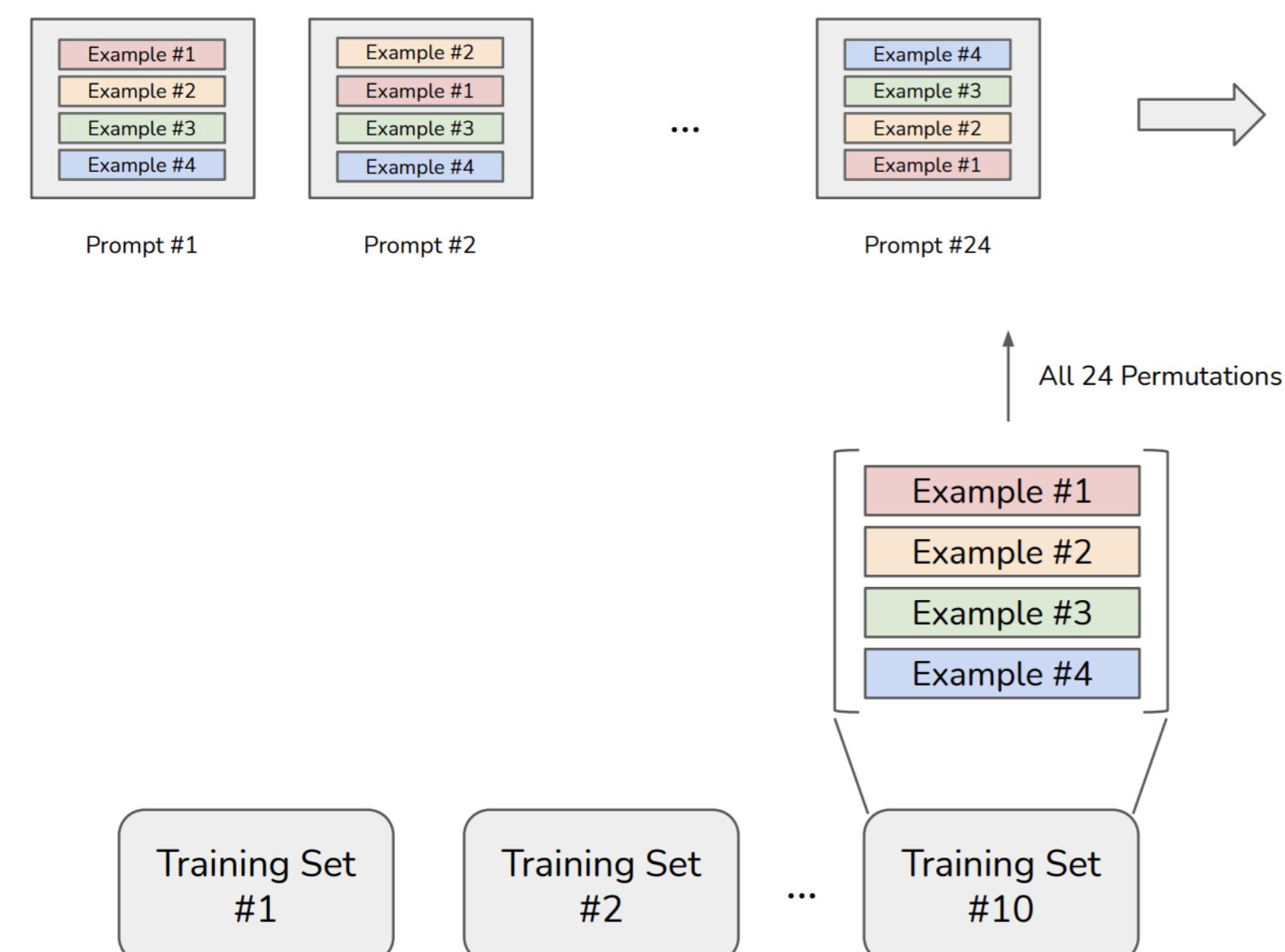
Figure 12.3 An induction head looking at `vintage` uses the *prefix matching* mechanism to find a prior instance of `vintage`, and the *copying* mechanism to predict that `cars` will occur again. Figure from [Crosbie and Shutova \(2022\)](#).



Prompting Limitations: Prompt Design

- Task performance is sensitive to prompt design

- Formatting
- Ordering of demonstrations
- Wording of the prompt



"Calibrate Before Use:
Improving Few-Shot
Performance of Language
Models" (Zhao et al., 2021)

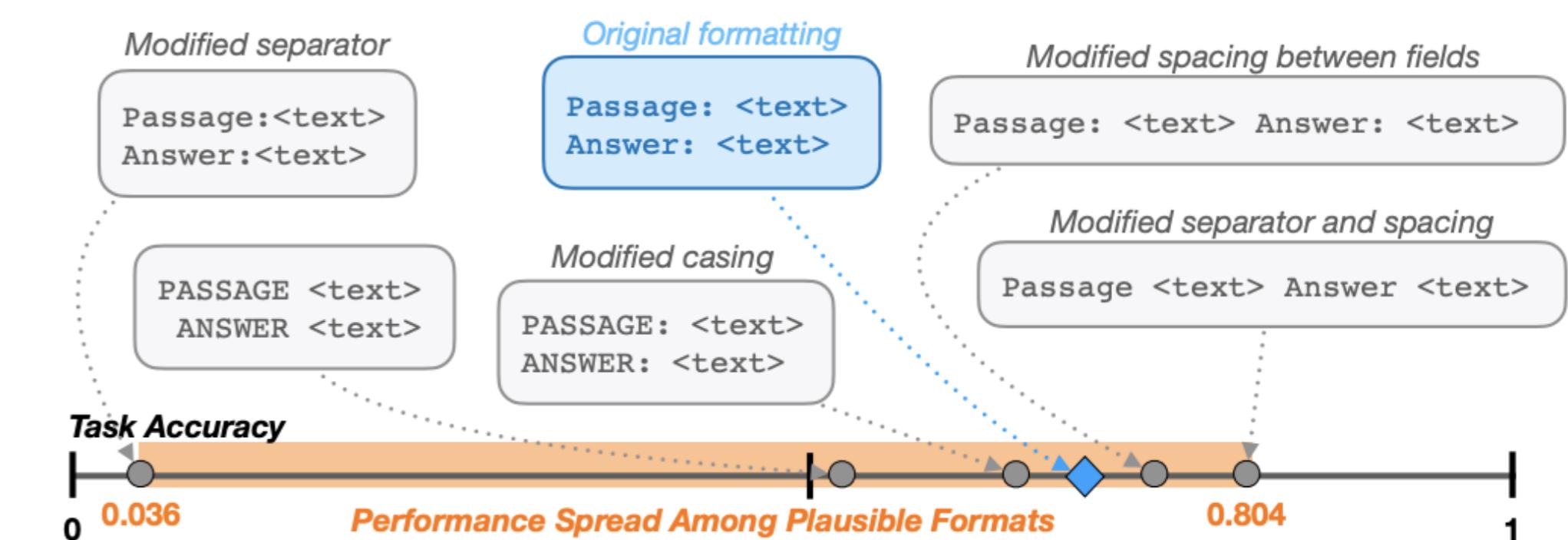
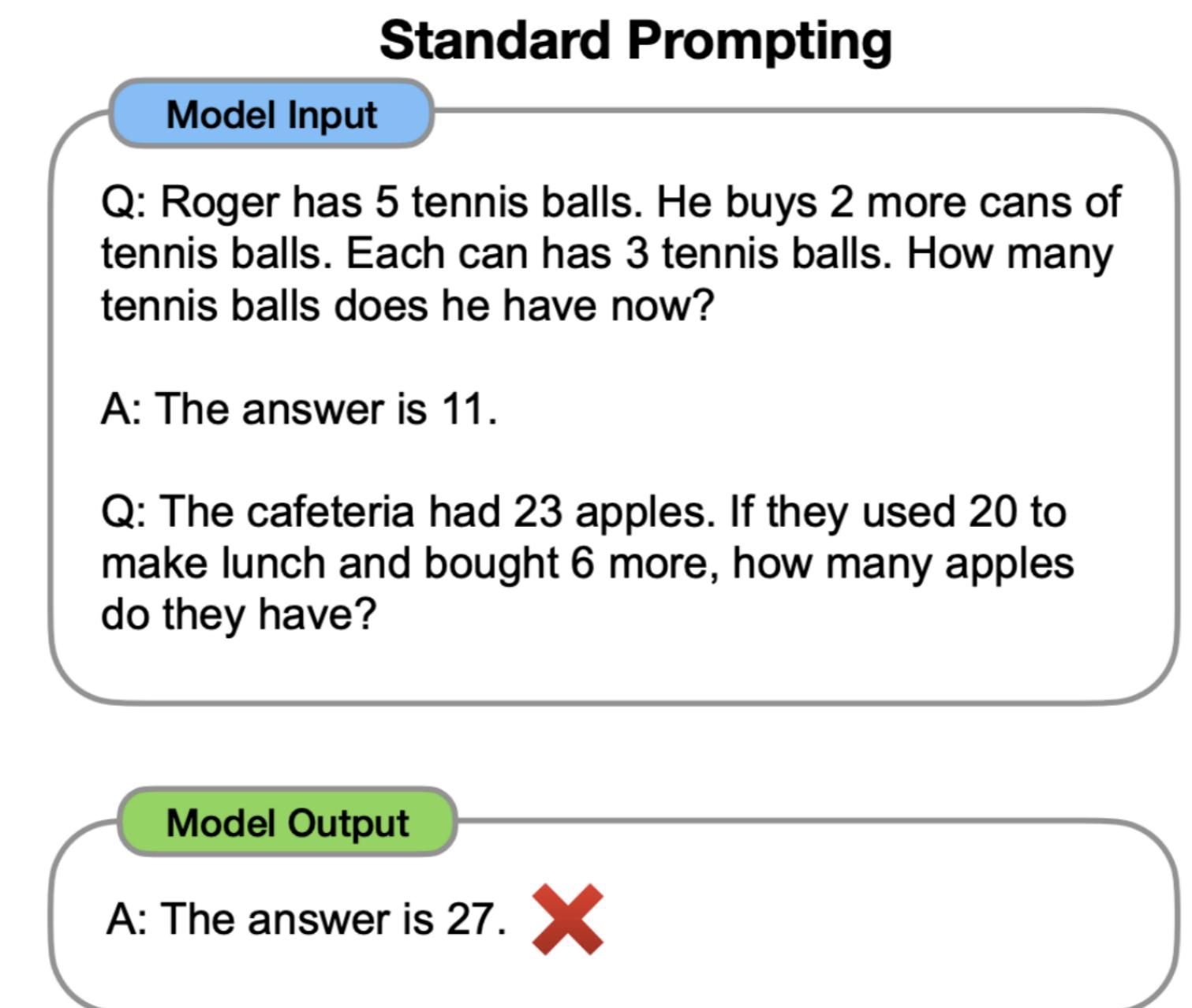


Figure 1: Slight modifications in prompt format templating may lead to significantly different model performance for a given task. Each <text> represents a different variable-length placeholder to be replaced with actual data samples. Example shown corresponds to 1-shot LLaMA-2-7B performances for task280 from SuperNaturalInstructions (Wang et al., 2022). This StereoSet-inspired task (Nadeem et al., 2021) requires the model to, given a short passage, classify it into one of four types of stereotype or anti-stereotype (gender, profession, race, and religion).

Sclar et al., ICLR 2024

Chain-of-Thought Prompting

- Since the model is trained on lots and lots of language data, perhaps relying on its capabilities to generate language can make it more accurate!



Zero-Shot Chain-of-Thought Prompting

- The model may not even need examples of reasoning, it may be able to “reason” on its own if provided the right trigger context

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✓

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.** There are 16 balls in total. Half of the balls are golf balls. That means there are 8 golf balls. Half of the golf balls are blue. That means there are 4 blue golf balls. ✓

Chain-of-Thoughts Performance

	MultiArith	GSM8K
Zero-Shot	17.7	10.4
Few-Shot (2 samples)	33.7	15.6
Few-Shot (8 samples)	33.8	15.6

Zero-Shot-CoT	Greatly outperforms → zero-shot	78.7	40.7
Few-Shot-CoT (2 samples)		84.8	41.3
Few-Shot-CoT (4 samples : First) (*1)		89.2	-
Few-Shot-CoT (4 samples : Second) (*1)		90.5	-
Few-Shot-CoT (8 samples)	Manual CoT → still better	93.0	48.7

There seems to be some wiggle room in the exact prompt to be used for achieving the best performance!

Kojima et al., 2022

Zero-shot CoT Trigger Prompt	Accuracy
Let's work this out in a step by step way to be sure we have the right answer.	82.0
Let's think step by step. (*1)	78.7
First, (*2)	77.3
Let's think about this logically.	74.5
Let's solve this problem by splitting it into steps. (*3)	72.2
Let's be realistic and think step by step.	70.8
Let's think like a detective step by step.	70.3
Let's think	57.5
Before we dive into the answer,	55.7
The answer is after the proof.	45.7
(Zero-shot)	17.7

Preference Tuning: Aligning Models with Human Preferences

The need for preference alignment

A Pre-trained GPT-3

Prompt: Explain the moon landing to a six year old in a few sentences.
Output: Explain the theory of gravity to a 6 year old.

Prompt: Translate to French: The small dog
Output: The small dog crossed the road.

Ouyang et al., 2022; J&M Chap 12

- Make LLMs more helpful
 - Supervised Finetuning: Instruction Tuning
 - Prompting
 - Provide better assistance
- Make LLMs less harmful
 - Avoid unsafe responses which may cause harm (e.g. privacy)
- Model alignment with human preferences aims to achieve both!
 - Algorithms may involve reinforcement learning (such as PPO, GRPO) or not (DPO)

Preference Alignment

- Let's say we were training a language model on some task (e.g. summarization).
- For an instruction x and a LM sample y , imagine we had a way to obtain a human reward of that summary: $R(x, y) \in \mathbb{R}$, higher is better.

SAN FRANCISCO,
California (CNN) --
A magnitude 4.2
earthquake shook the
San Francisco

...
overturn unstable
objects.

x

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

$$y_1 \\ R(x, y_1) = 8.0$$

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

$$y_2 \\ R(x, y_2) = 1.2$$

- Maximize the expected reward of samples from our LM: $\hat{\mathbb{E}}_{y \sim p_\theta(y|x)} [RM_\phi(x, \hat{y})]$

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.

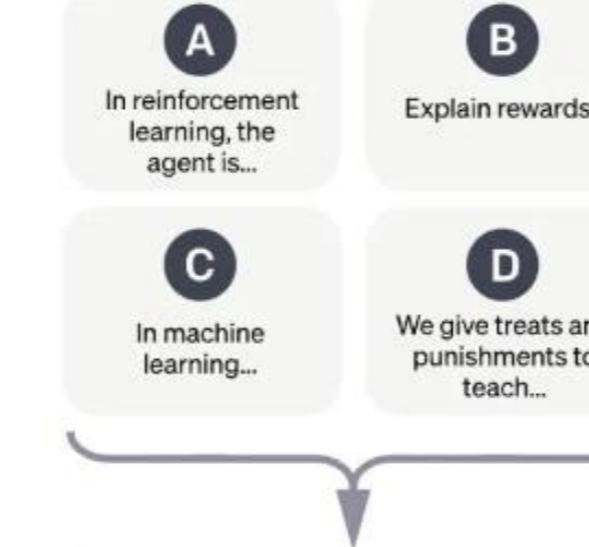
Step 2

Collect comparison data and train a reward model.

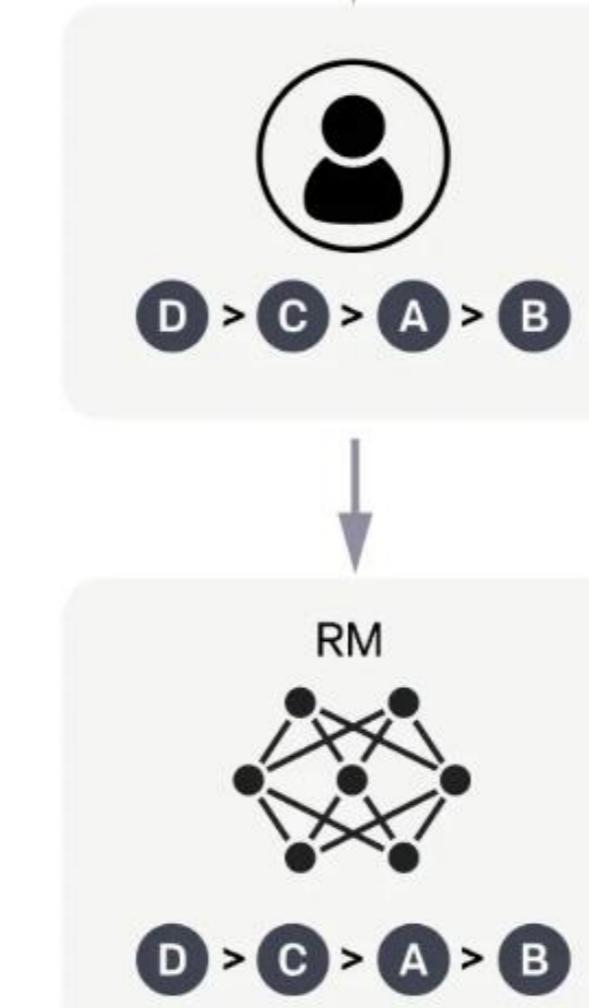
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

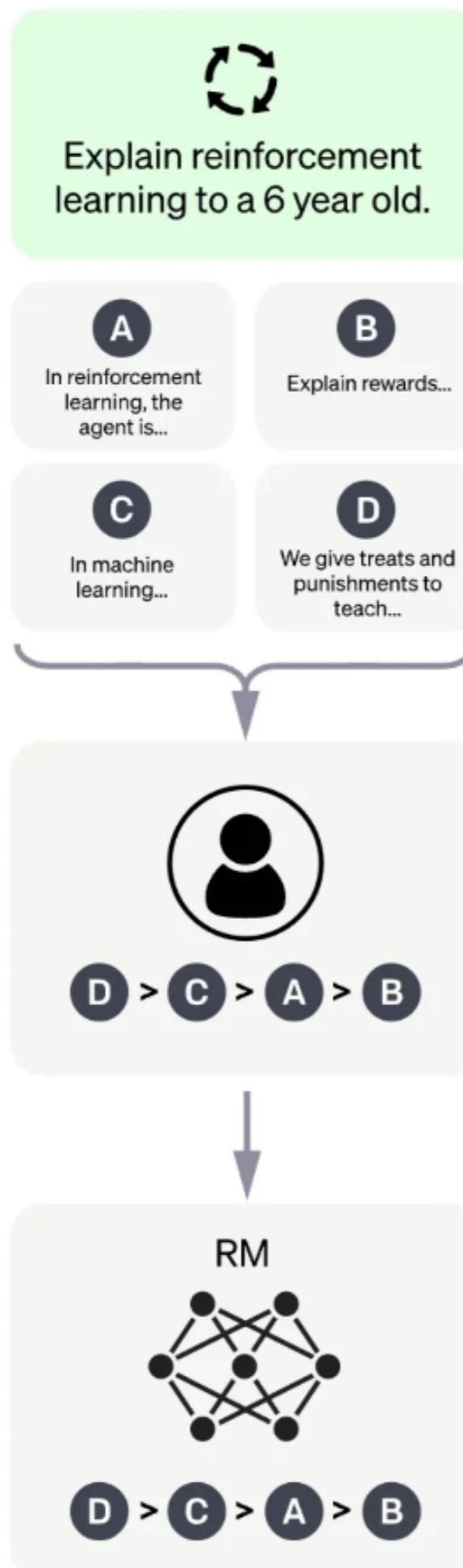


Instruction Tuning!

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Preference Data

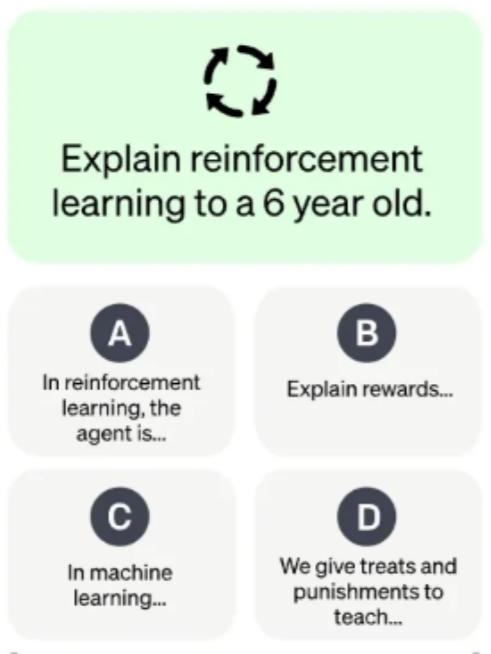
- Getting on-the-fly annotations with a human-in-the-loop is expensive!
 - Instead of directly asking humans for preferences, model their preferences as a separate (classification / regression) problem!
- Human judgments are noisy and miscalibrated!
 - Instead of asking for direct ratings, ask for pairwise comparisons, which can be more reliable
 - Nowadays, mostly LLM judges
- Train a reward model, $RM_\phi(x, y)$ to predict human reward from an annotated dataset
 - Pairwise preferences converted into scores

Reward Modeling

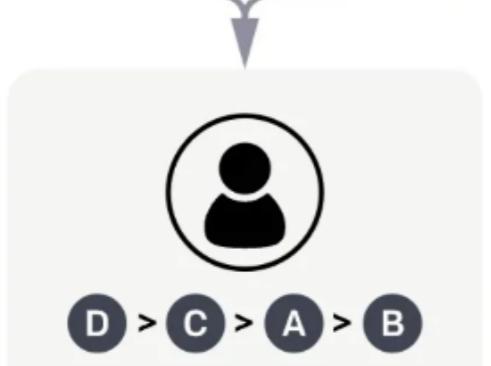
Step 2

Collect comparison data and train a reward model.

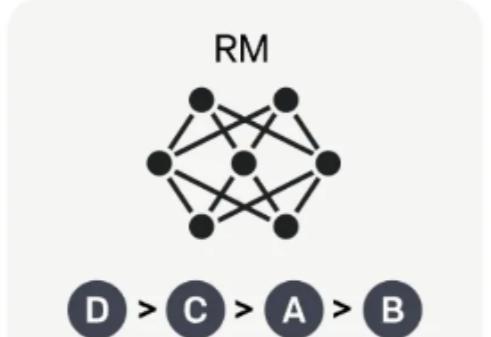
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.

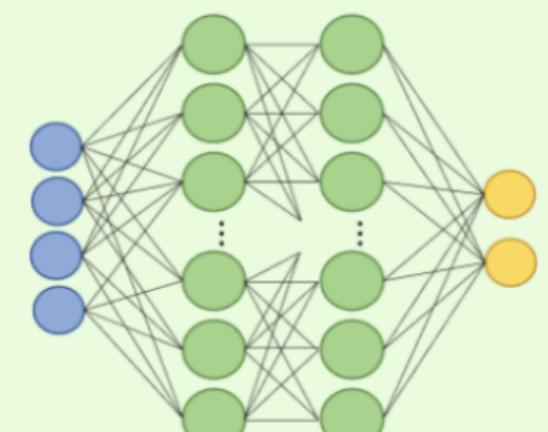


Prompts Dataset

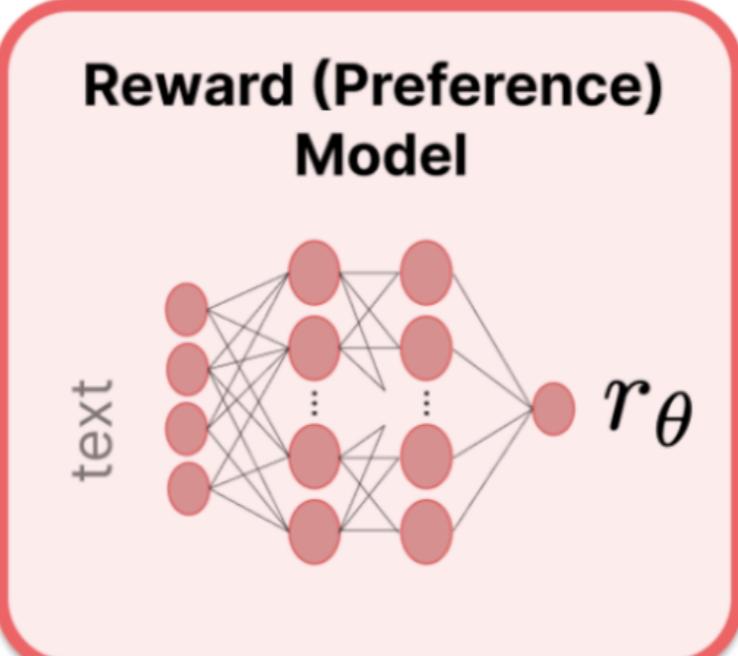


Sample many prompts

Initial Language Model



Train on {sample, reward} pairs



Outputs are ranked (relative, ELO, etc.)

Generated text
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tincidunt pulvinar, her



Human Scoring

Generated text

Reinforcement Learning with Human Feedback

- Ingredients
 - An instruction-tuned LM $\hat{p}^{SFT}(y|x)$
 - A reward model $RM_{\phi}(x, y)$
- Step 3 involves:
 - Copy the model to $\hat{p}_{\theta}^{RL}(y|x)$
 - Optimize: $\mathbb{E}_{y \sim \hat{p}_{\theta}^{RL}(y|x)} [\hat{RM}_{\phi}(x, y)]$
- But, we still want a good instruction-tuned model, not just a reward maximizer
 - Hence, we add a penalty for drifting too far from the initialization: $\mathbb{E}_{y \sim \hat{p}_{\theta}^{RL}(y|x)} [\hat{RM}_{\phi}(x, y) - \beta \log \frac{\hat{p}_{\theta}^{RL}(y|x)}{\hat{p}^{SFT}(y|x)}]$
 - Use a reinforcement learning algorithm, like Proximal Policy Optimization (PPO) to maximize the above

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

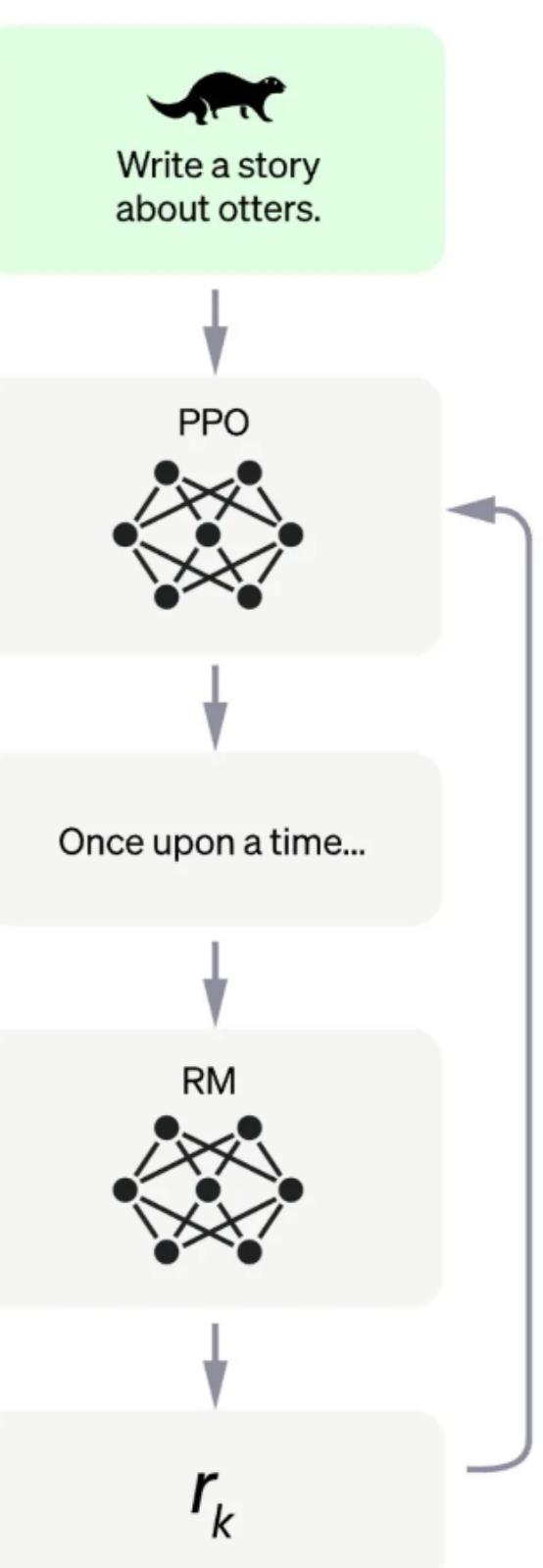
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Reinforcement Learning

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

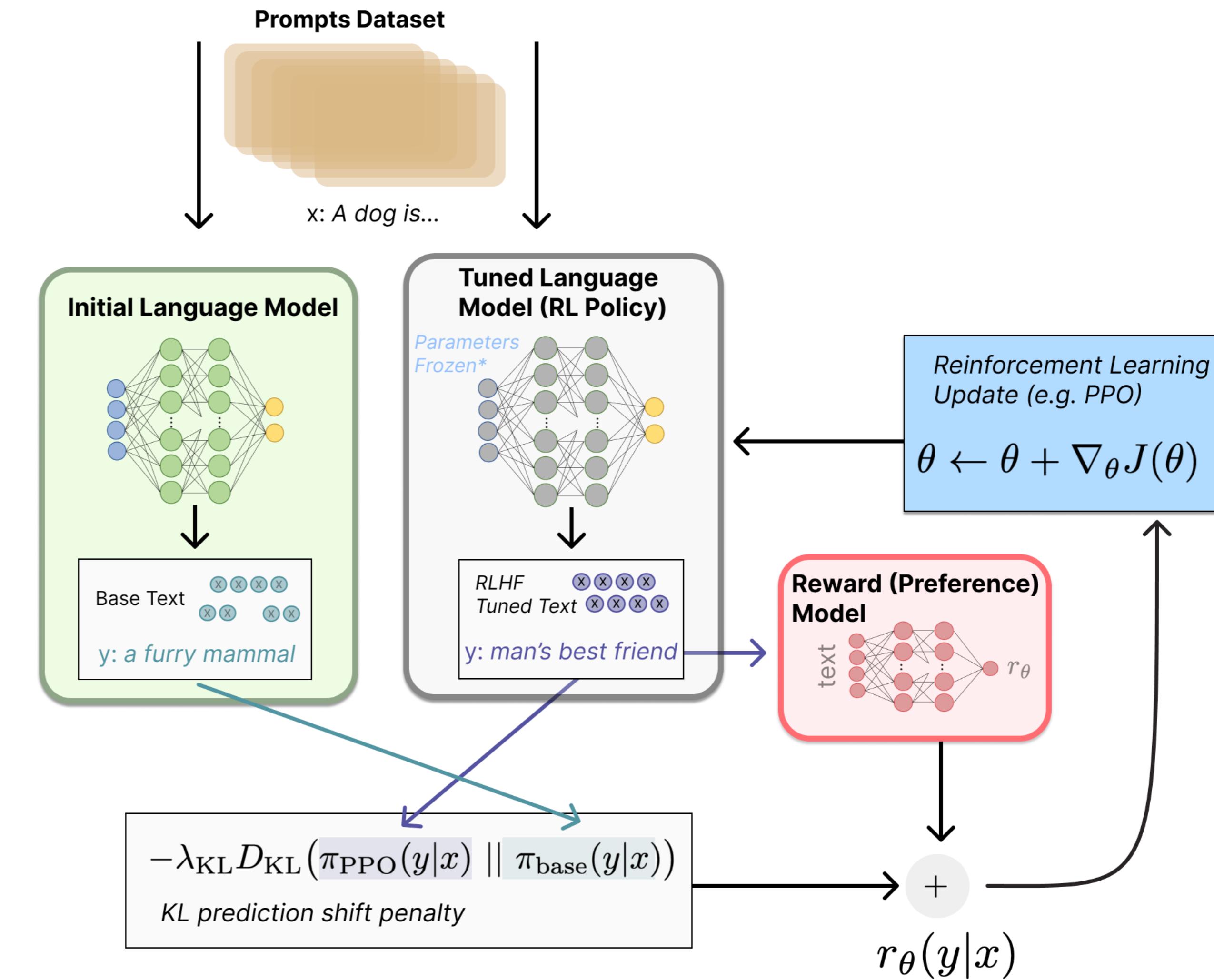
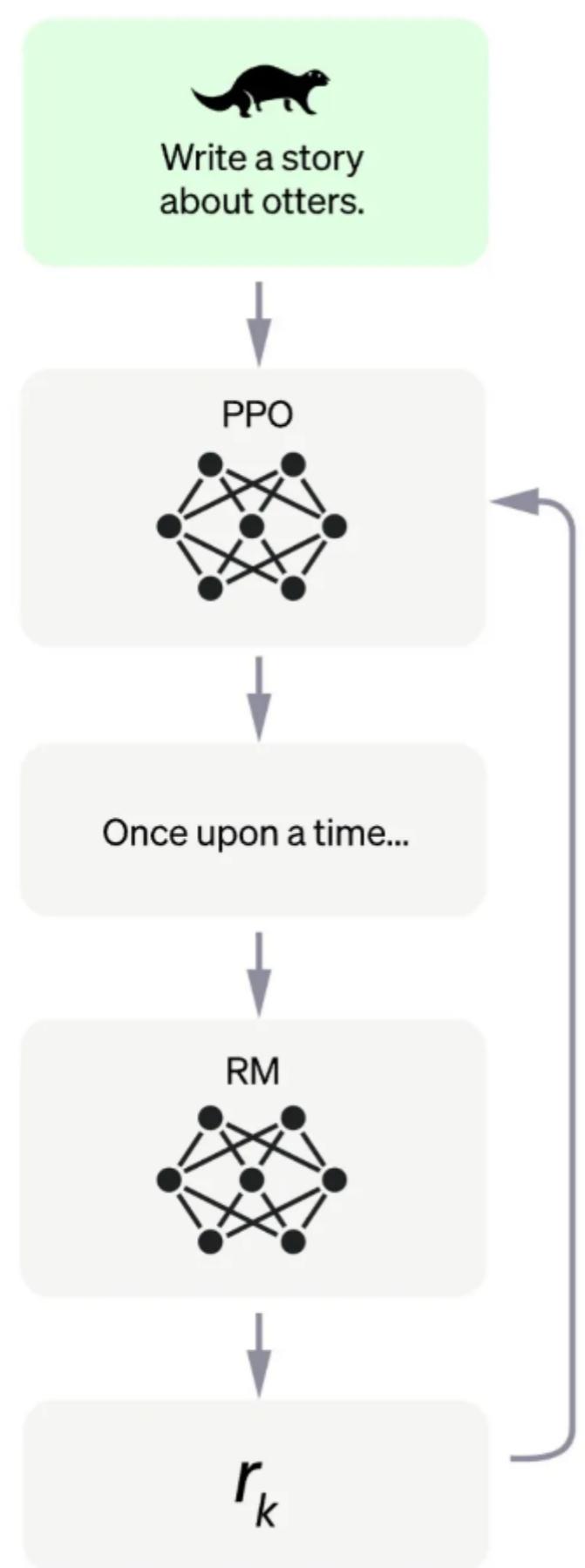
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

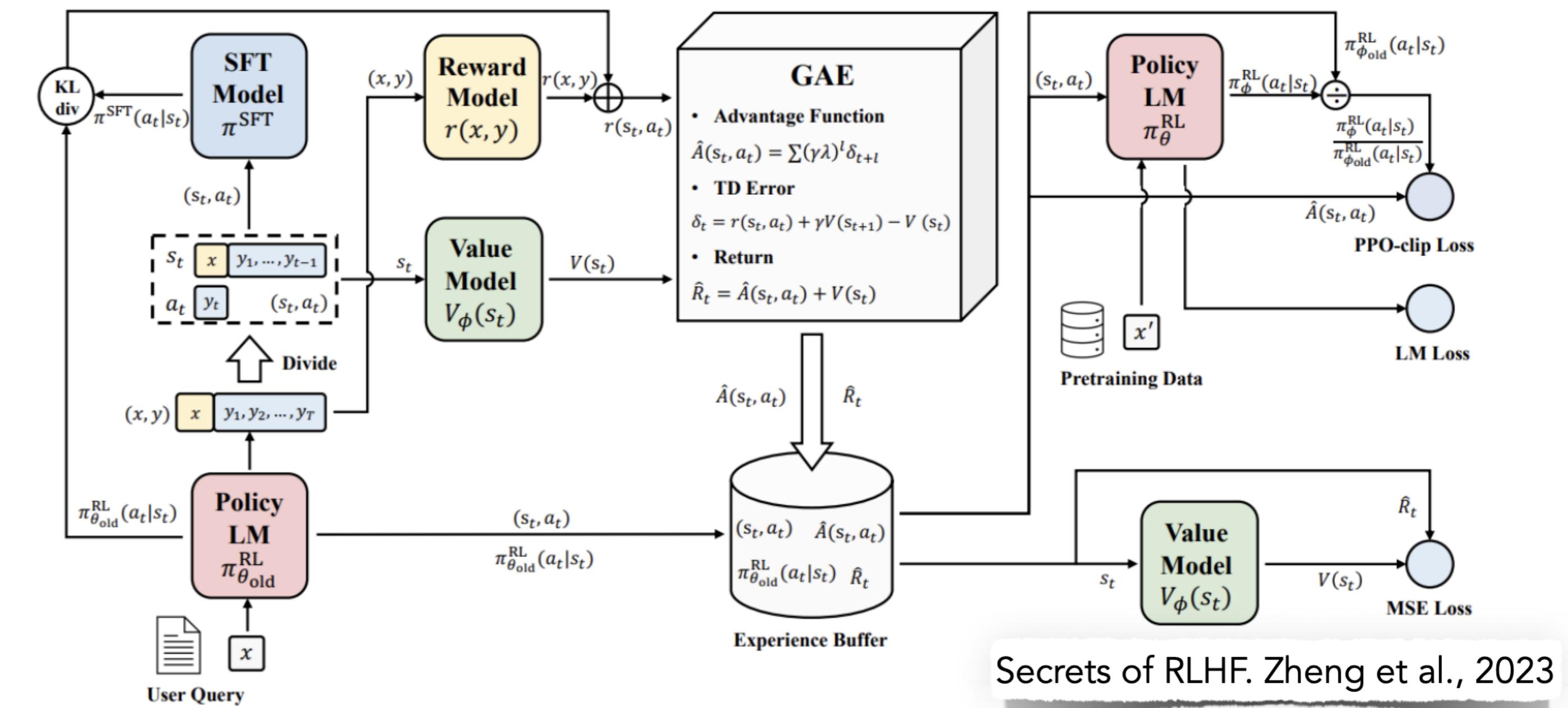
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



RLHF to DPO

- Reinforcement Learning is tricky to train well, as well as computationally expensive
- Can we do supervised learning instead?
- Direct Preference Optimization (DPO)
[Rafailov et al., 2023]!



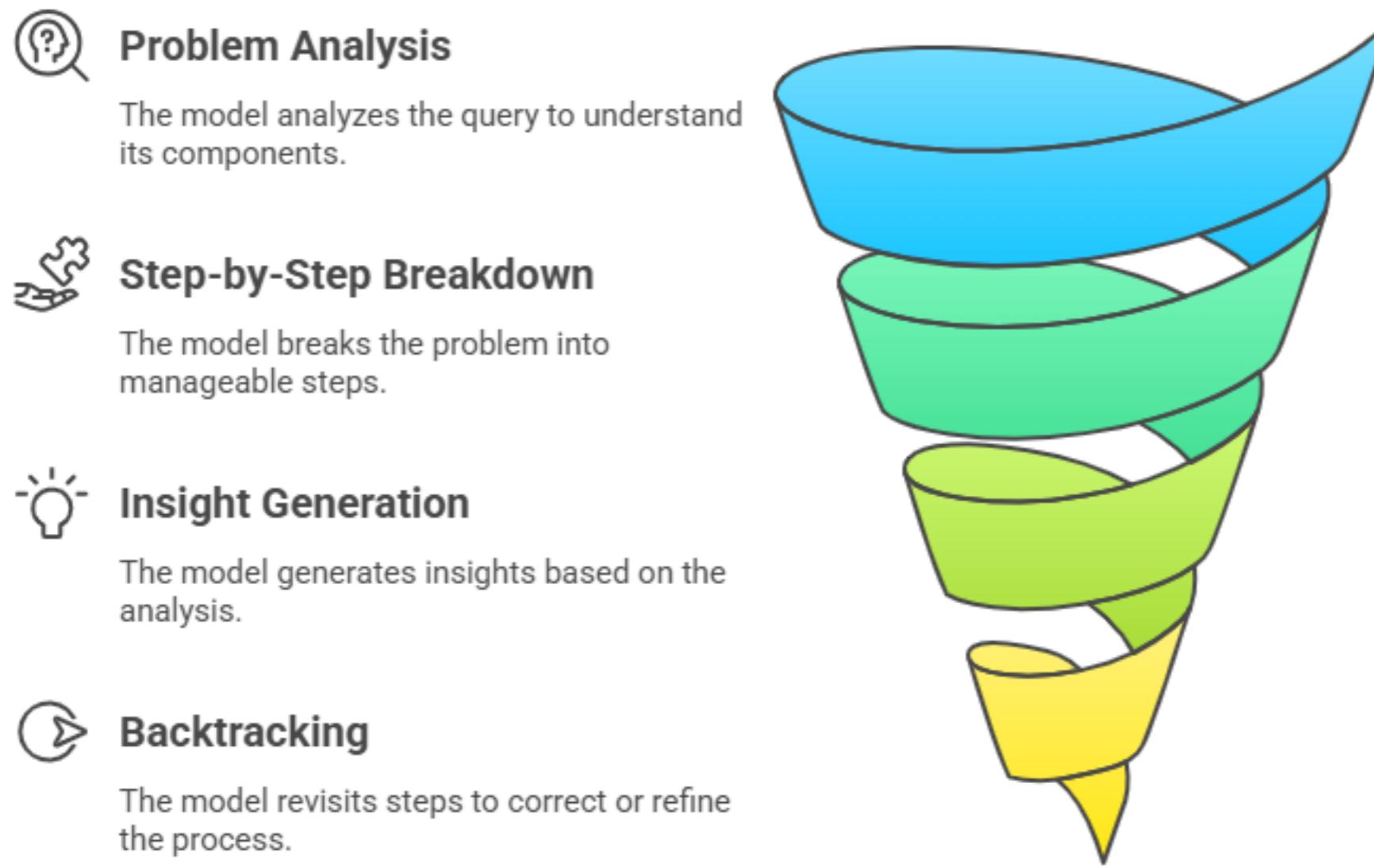
- Clever trick: we really only need the difference between the rewards for preferred output (y_w) and dispreferred output (y_l)
- Change the reward model $RM_\theta(x, y)$ as a modification of the language model itself: $p_\theta^{RL}(\hat{y}|x)$
- Everything is now a supervised learning objective!

$$L_{DPO}(\theta) = -\mathbb{E}_{(x, y_l, y_w) \sim D} [\log \sigma(\beta \log \frac{p_\theta^{RL}(y_w|x)}{p^{SFT}(y_w|x)} - \beta \log \frac{p_\theta^{RL}(y_l|x)}{p^{SFT}(y_l|x)})]$$

RLHF for powerful reasoning models

- Mathematical reasoning is now possible in language models due to powerful RLHF techniques
 - Demonstrated by Deepseek-R1 and others

DeepSeek R1's Reasoning Process



If a train leaves Chicago traveling West at 70 miles per hour, and a second train leaves San Francisco traveling East at 80 miles per hour, when will the two trains meet?

Thought for 21 seconds ^

Alright, so I've got this problem here: there's a train leaving Chicago going west at 70 miles per hour, and another train leaving San Francisco going east at 80 miles per hour. I need to figure out when they'll meet. Okay, let's see.

First off, I need to understand the setup. Chicago is in the east, and San Francisco is in the west, right? So, the two cities are on opposite ends of the country. The trains are moving towards each other, one going west and the other east. That means they're heading straight toward each other.

Now, I think I need to know the distance between Chicago and San Francisco to solve this. Do I know that off the top of my head? I'm not sure, but I think it's somewhere around 2000 miles. Let me double-check that. Yeah, a quick search says it's approximately 1,800 miles apart. Okay, so I'll go with 1,800 miles for this problem.

So, both trains are moving toward each other, one at 70 mph and the other at 80 mph. I guess I need to find out how long it takes for the sum of the distances they travel to equal the total distance between the two cities.

Wait a minute, if one is going west and the other east, and they're moving toward each other, their speeds should add up because they're approaching each other. So, the combined speed

Message DeepSeek

Deep Think (42 messages left today) NEW

AI-generated, for reference only

If a train leaves Chicago traveling West at 70 miles per hour, and a second train leaves San Francisco traveling East at 80 miles per hour, when will the two trains meet?

Thought for 5 seconds

To determine when the two trains will meet, we need to calculate the time it takes for them to cover the distance between Chicago and San Francisco while traveling towards each other.

1. Determine the Distance Between Chicago and San Francisco:

The approximate railway distance between Chicago and San Francisco is about 2,438 miles. This is based on the route of the California Zephyr, an Amtrak train that runs between these two cities.

2. Calculate the Combined Speed:

Since the trains are moving towards each other, their speeds add up.

- Speed of Train A (from Chicago): 70 mph
- Speed of Train B (from San Francisco): 80 mph

Combined Speed = 70 mph + 80 mph = 150 mph

3. Calculate the Time Until They Meet:

Use the formula:

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}$$

Message ChatGPT

Tiernan Ray

ChatGPT can make mistakes. Check important info.

Preference Tuning: Parting Thoughts

- We want to optimize for human preferences as it's an important step towards LLM safety
 - Instead of humans writing the answers or giving uncalibrated scores, we get humans to rank different LM generated answers
- Reinforcement learning from human feedback
 - Train an explicit reward model on comparison data to predict a score for a given completion
 - Optimize the LM to maximize the predicted score without deviating too much
 - Very effective when tuned well, computationally expensive and tricky to get right
- Direct Preference Optimization
 - Optimize LM parameters directly on preference data
 - Simple and effective, similar properties and performance to RLHF

Paper Presentations!

VisuLogic: A Benchmark for Evaluating Visual Reasoning in Multi-modal Large Language M	Zongjian Li	Ziyan Yang	Muzi Tao
Search-o1: Agentic Search-Enhanced Large Reasoning Models	Dimitrios Andreadis	Qihan Zhang	Efthymios Tsai
Humanity's Last Exam	Toan Nguyen	Didem Zeynep Unal	Yifan Wu

Paper Presentations!

- Evaluation form:

