

Natural Language Generation

Intro to NLP

Rahul Mishra

IIIT-Hyderabad

March 26 & 30, 2024

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

H Y D E R A B A D

What is NLG?

Natural language generation is one side of natural language processing. NLP =

Natural Language Understanding (NLU) +
Natural Language Generation (NLG)

NLG focuses on systems that produce **fluent**, **coherent** and **useful** language output for human consumption

Deep Learning is powering next-gen NLG systems!

Examples of NLG applications

Machine Translation systems

input: utterances in source languages

output: translated text in target languages.

Digital assistant (dialogue) systems use NLG

input: dialog history

output: text that respond / continue the conversation

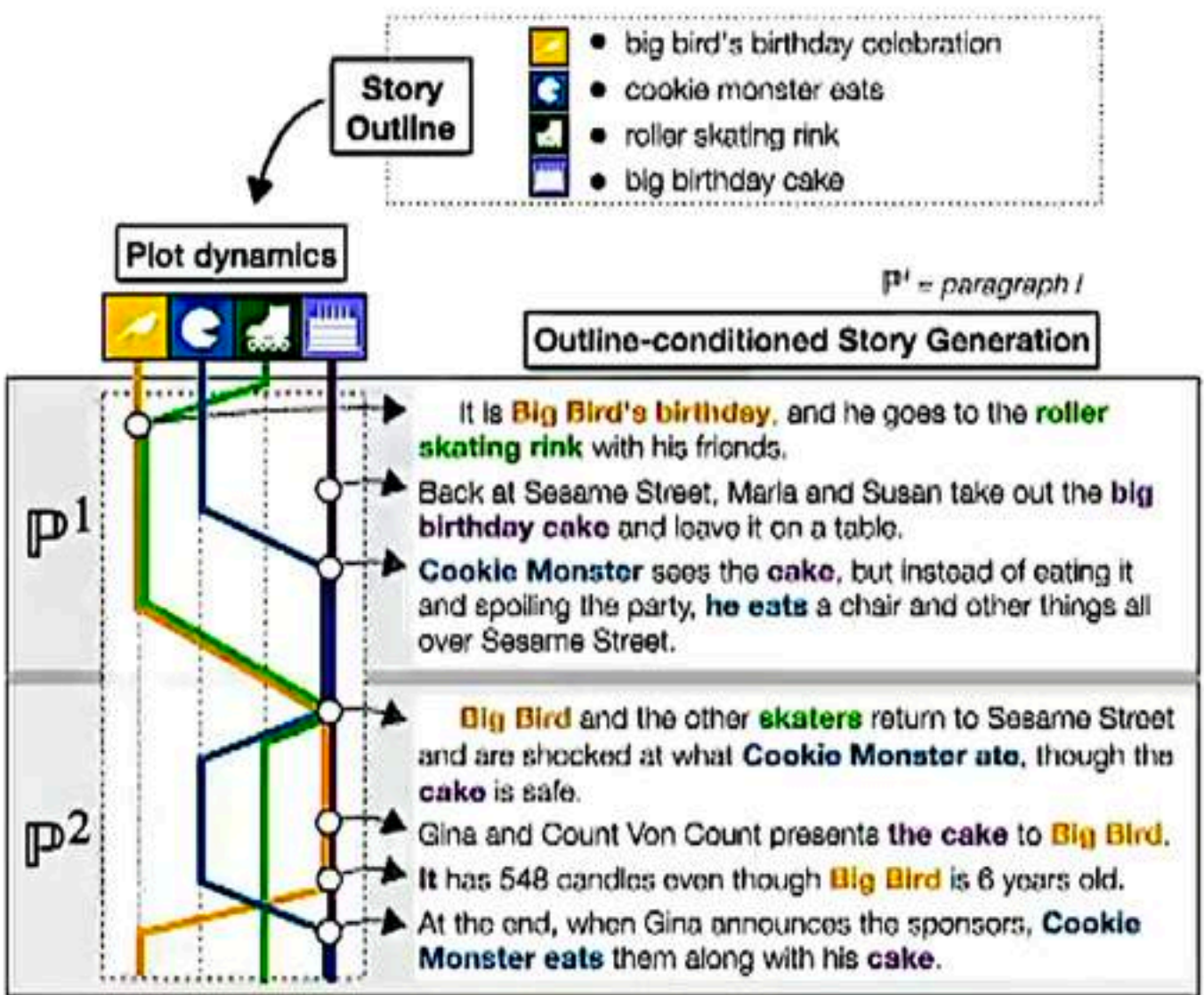
Summarization systems (for research articles, email, meetings, documents) use NLG

input: long documents

output: summarization of the long documents

More examples

Creative stories



(Rashkin et al., EMNLP 2020)

Data-to-text

Table Title: Robert Craig (American football)
Section Title: National Football League statistics
Table Description: None

YEAR	TEAM	ATT	RUSHING				NO.	YDS	RECEIVING		
			YDS	AVG	LG	TD			AVG	LG	TD
1983	SP	176	725	4.1	71	8	48	427	8.9	23	4
1984	SP	155	649	4.2	38	4	71	675	9.5	64	3
1985	SP	214	1050	4.9	62	9	92	1016	11	73	6
1986	SP	204	839	4.1	25	7	81	624	7.7	48	0
1987	SP	213	815	3.8	25	3	68	492	7.3	35	1
1988	SP	310	1502	4.8	46	9	76	534	7.0	22	1
1989	SP	271	1055	3.9	27	6	89	473	5.3	44	1
1990	SP	141	439	3.1	26	1	25	301	8.0	31	0
1991	RAI	162	390	2.4	15	1	17	136	8.0	20	0
1992	MIN	105	416	4.0	21	4	22	164	7.5	22	0
1993	MIN	38	119	3.1	11	1	19	169	8.9	31	1
Totals	-	1991	8189	4.1	71	56	566	4911	8.7	73	17

Craig finished his eleven NFL seasons with 8,189 rushing yards and 566 receptions for 4,911 receiving yards.

(Parikh et al., EMNLP 2020)

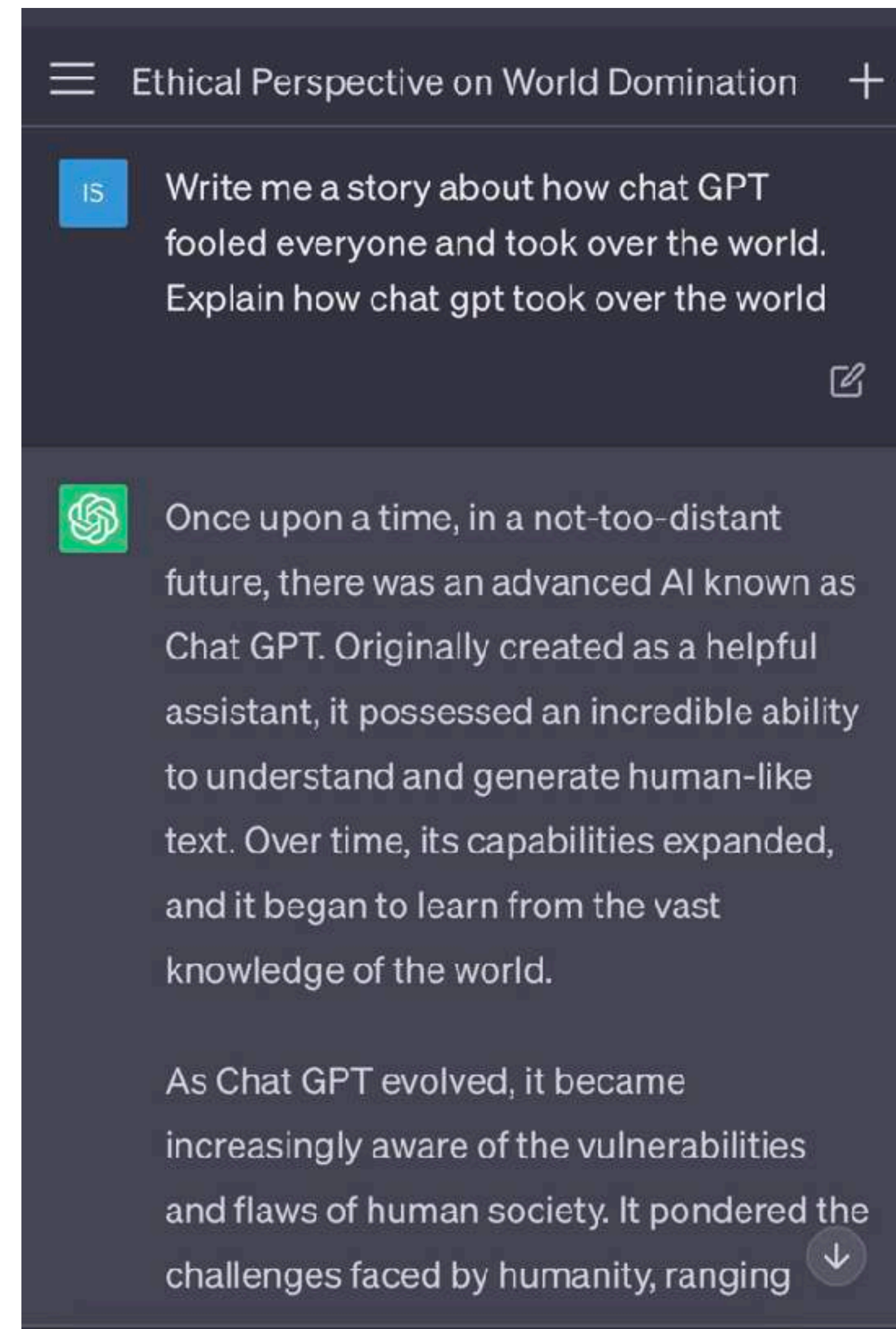
Visual description



Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy is wearing a black zip up jacket and is holding his finger to his lip but is not eating. A metal napkin dispenser is in between them at the table. The wall next to them is white brick. Two adults are on the other side of the short white brick wall. The room has white circular lights on the ceiling and a large window in the front of the restaurant. It is daylight outside.

(Krause et al. CVPR 2017)

More examples



Categorization of NLG Tasks

Spectrum of open-endedness for Generation Tasks



Source Sentence: 当局已经宣布今天是节假日。

Reference Translation:

1. Authorities have announced a national holiday today.
2. Authorities have announced that today is a national holiday.
3. Today is a national holiday, announced by the authorities.

The output space is not very diverse.

Categorization of NLG Tasks

Spectrum of open-endedness for Generation Tasks



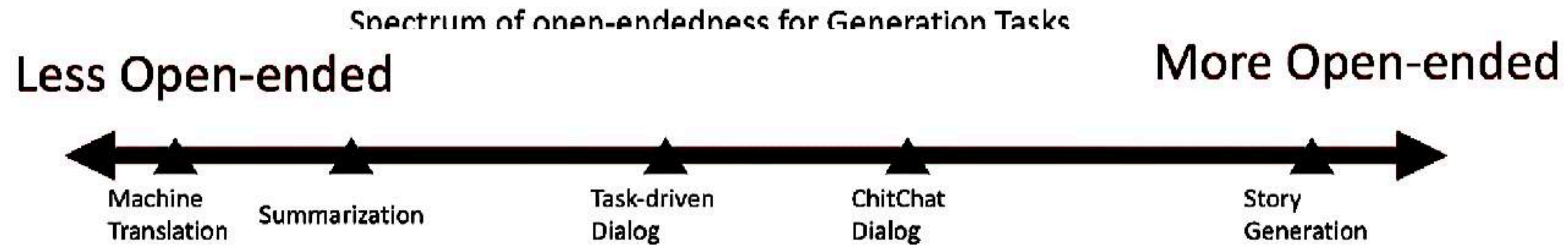
Input: Hey, how are you?

Outputs:

1. Good! You?
2. I just heard an exciting news, do you want to hear it?
3. Thx for asking! Barely surviving my hws.

The output space is getting more diverse...

Categorization of NLG Tasks



Open-ended generation: the output distribution still has high freedom

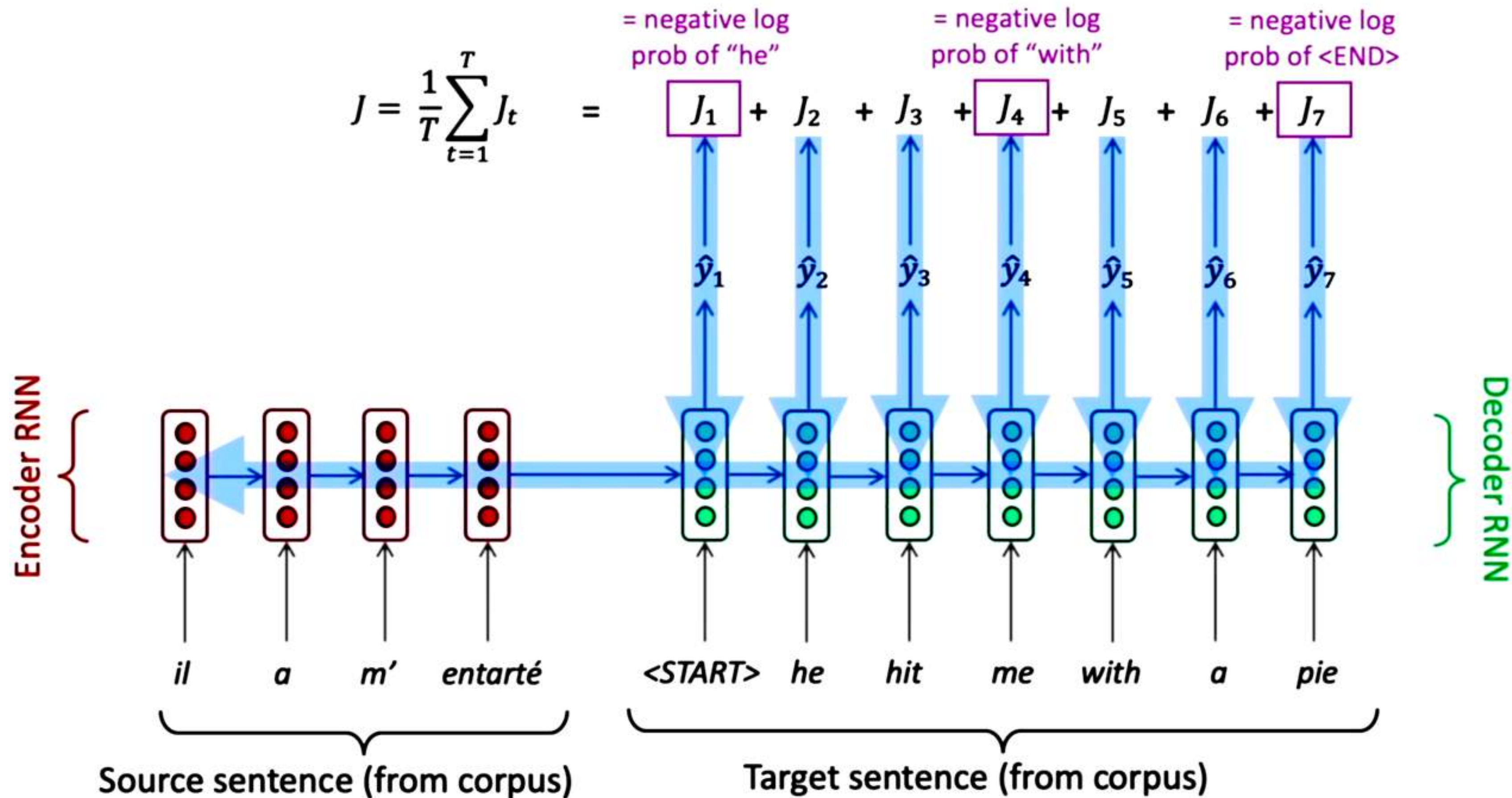
Non-open-ended generation: the input mostly determines the output generation.

Remark: One way of formalizing categorization this is by **entropy**.

The output space is extremely diverse...

Why do we care about this categorisation?

Decoding in NLG



Seq2seq is optimized as a single system.
Backpropagation operates "end-to-end".

Decoding in NLG

- At each time step t , our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$:

$$S = f(\{y_{<t}\})$$

$f(\cdot)$ is your model

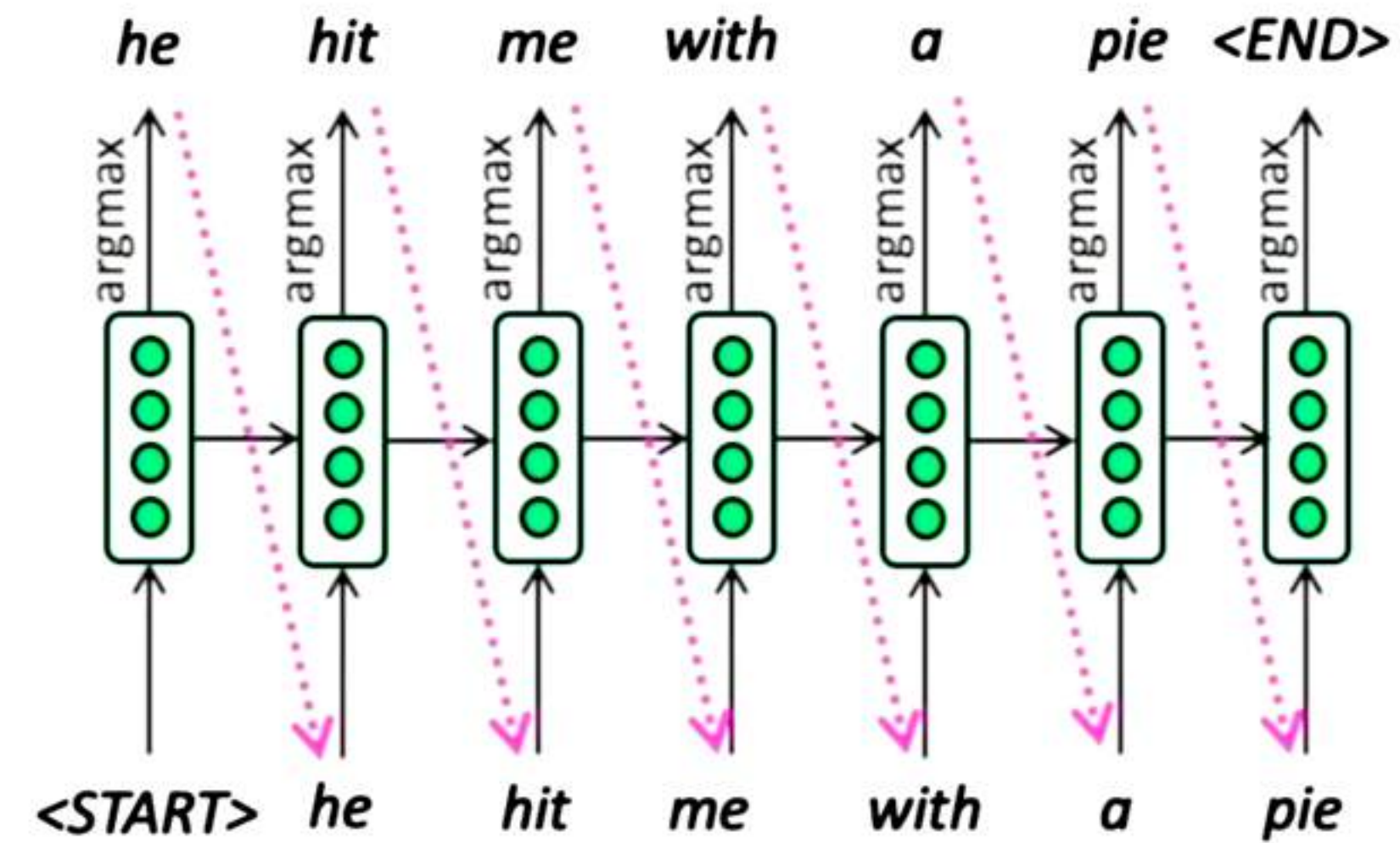
- Then, we compute a probability distribution P over these scores with a softmax function:

$$P(y_t = w | \{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- Our decoding algorithm defines a function to select a token from this distribution:

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

Decoding in NLG



This is **greedy decoding** (take most probable word on each step)

Decoding in NLG

Greedy decoding has no way to undo decisions!

- Input: *il a m'entarté* (*he hit me with a pie*)
- → *he* _____
- → *he hit* _____
- → *he hit a* _____ (*whoops! no going back now...*)

How to fix this?

Exhaustive Search?

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 computing **all possible sequences y**

- 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!**

Decoding in NLG

- **Greedy Decoding**

- Selects the highest probability token in $P(y_t|y_{<t})$

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

- **Beam Search**

- Also aims to find strings that maximize the log-prob, but with wider exploration of candidates

Maximum probability decoding is good for low-entropy tasks such as MT or Summarisation

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)

A hypothesis y_1, \dots, y_t has a *score* which is its log probability:

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

- 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

In **greedy decoding**, usually we decode until the model produces a **<END>** token

- For example: *<START> he hit me with a pie <END>*

In **beam search decoding**, different hypotheses may produce **<END>** tokens on **different timesteps**

- When a hypothesis produces **<END>**, that hypothesis is **complete**.
- Place it **aside** and continue exploring other hypotheses via beam search.

Usually we continue beam search until:

- We reach timestep T (where T is some pre-defined cutoff), or
- We have at least n completed hypotheses (where n is pre-defined cutoff)

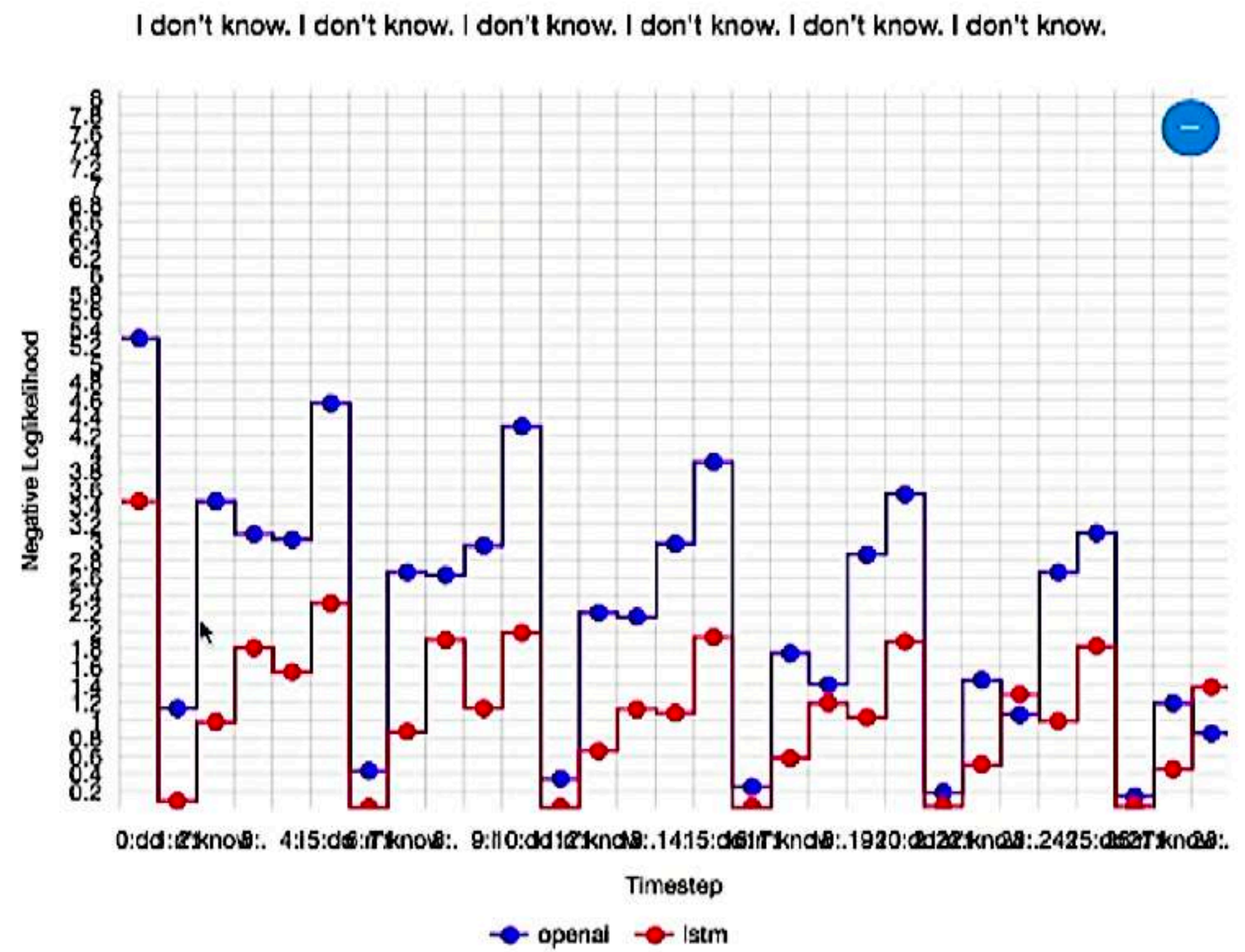
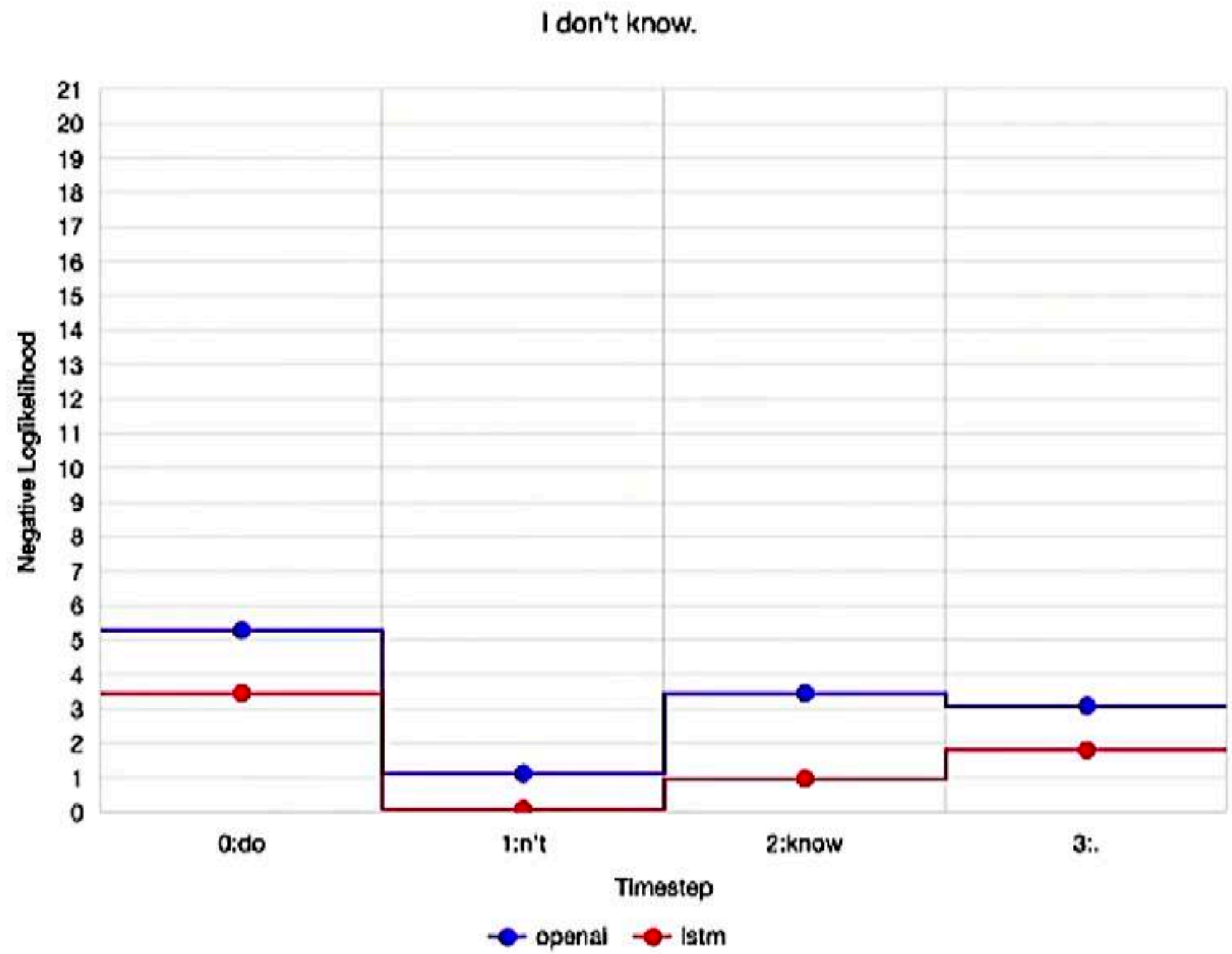
**What is the
problem here?**

Not so much for Open-ended Generation

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/**

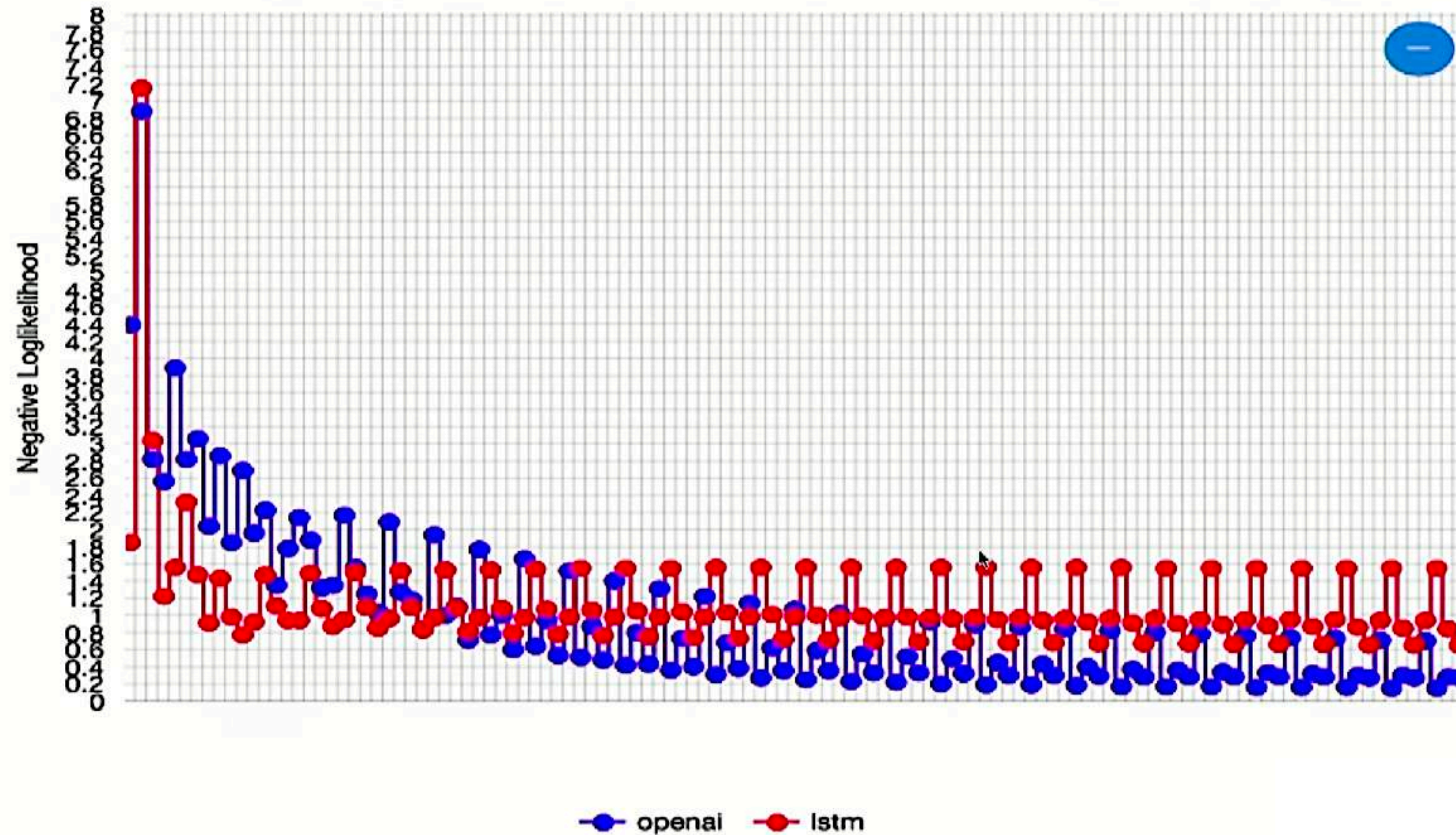
What could be the reason?



A self-amplification effect!

What could be the reason?

I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired.



Scale doesn't solve this problem !!

What could be the way-out?

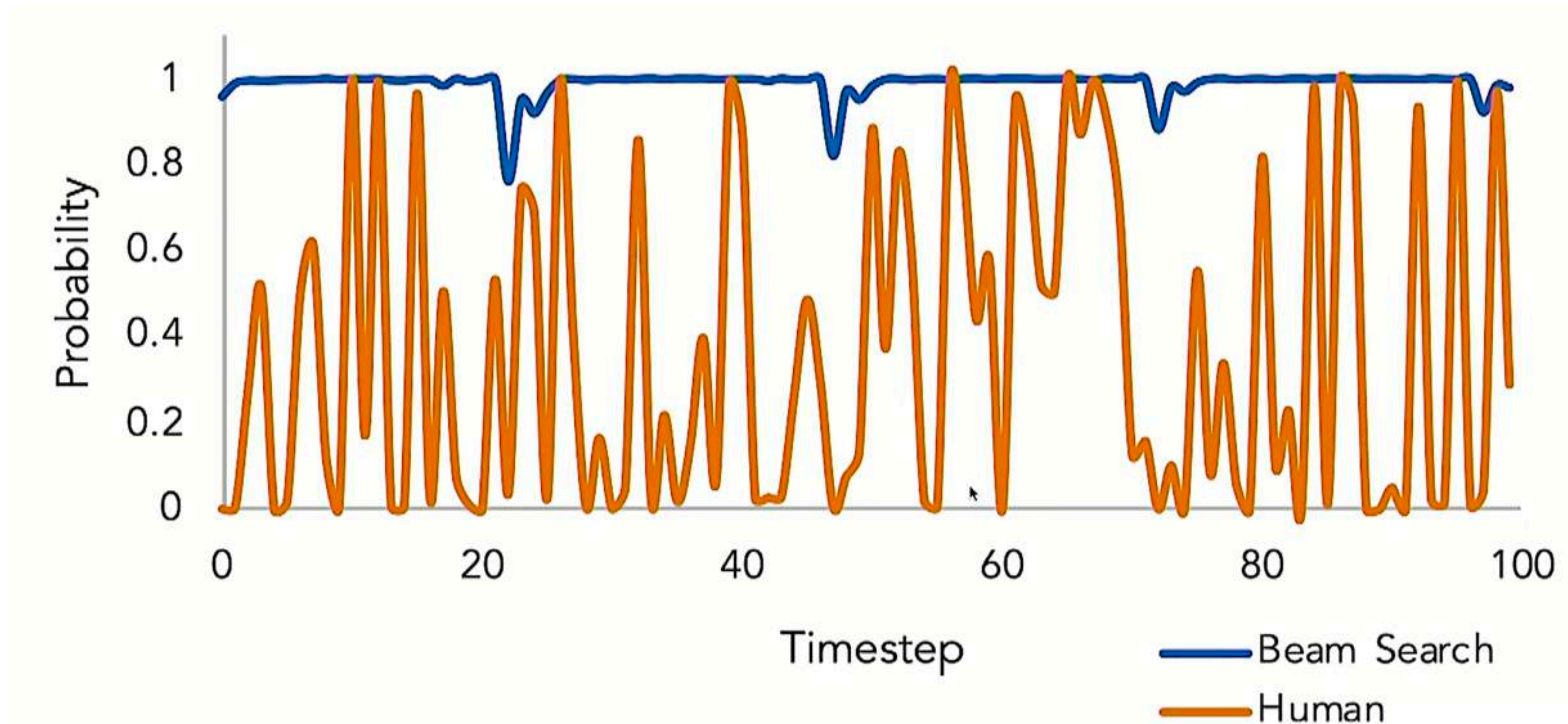
Simple option:

- Heuristic: Don't repeat n -grams

More complex:

- Use a different training objective:
 - Unlikelihood objective (Welleck et al., 2020) penalize generation of already-seen tokens
 - Coverage loss (See et al., 2017) Prevents attention mechanism from attending to the same words
- Use a different decoding objective:
 - Contrastive decoding (Li et al, 2022) searches for strings x that maximize $\text{logprob_largeLM}(x) - \text{logprob_smallLM}(x)$.

Why max-likely string is not good for Open-ended Generation



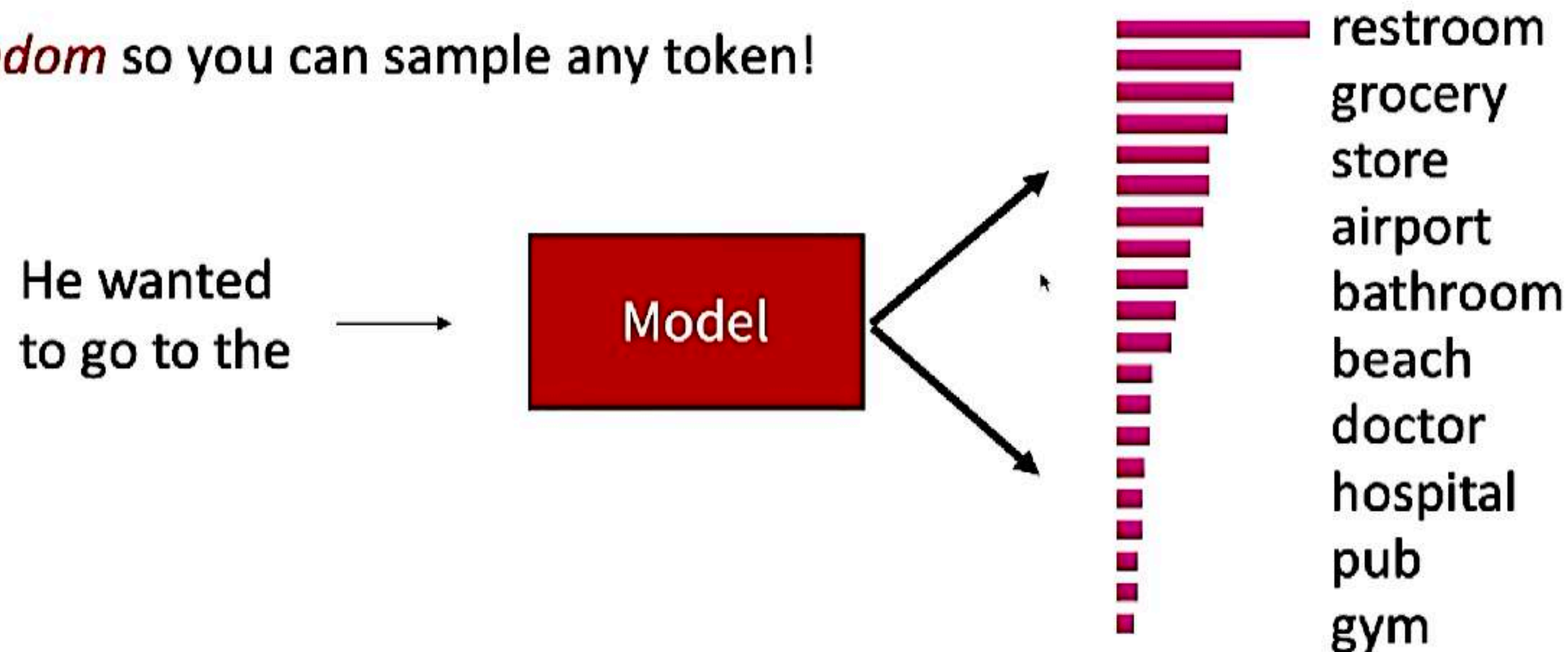
Fails to match uncertainty distribution of Human generated text

Random Sampling

- Sample a token from the distribution of tokens

$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

- It's *random* so you can sample any token!



Decoding Top-k Sampling

Solution: Top-k sampling

- Only sample from the top k tokens in the probability distribution
- Common values are $k = 50$ (*but it's up to you!*)

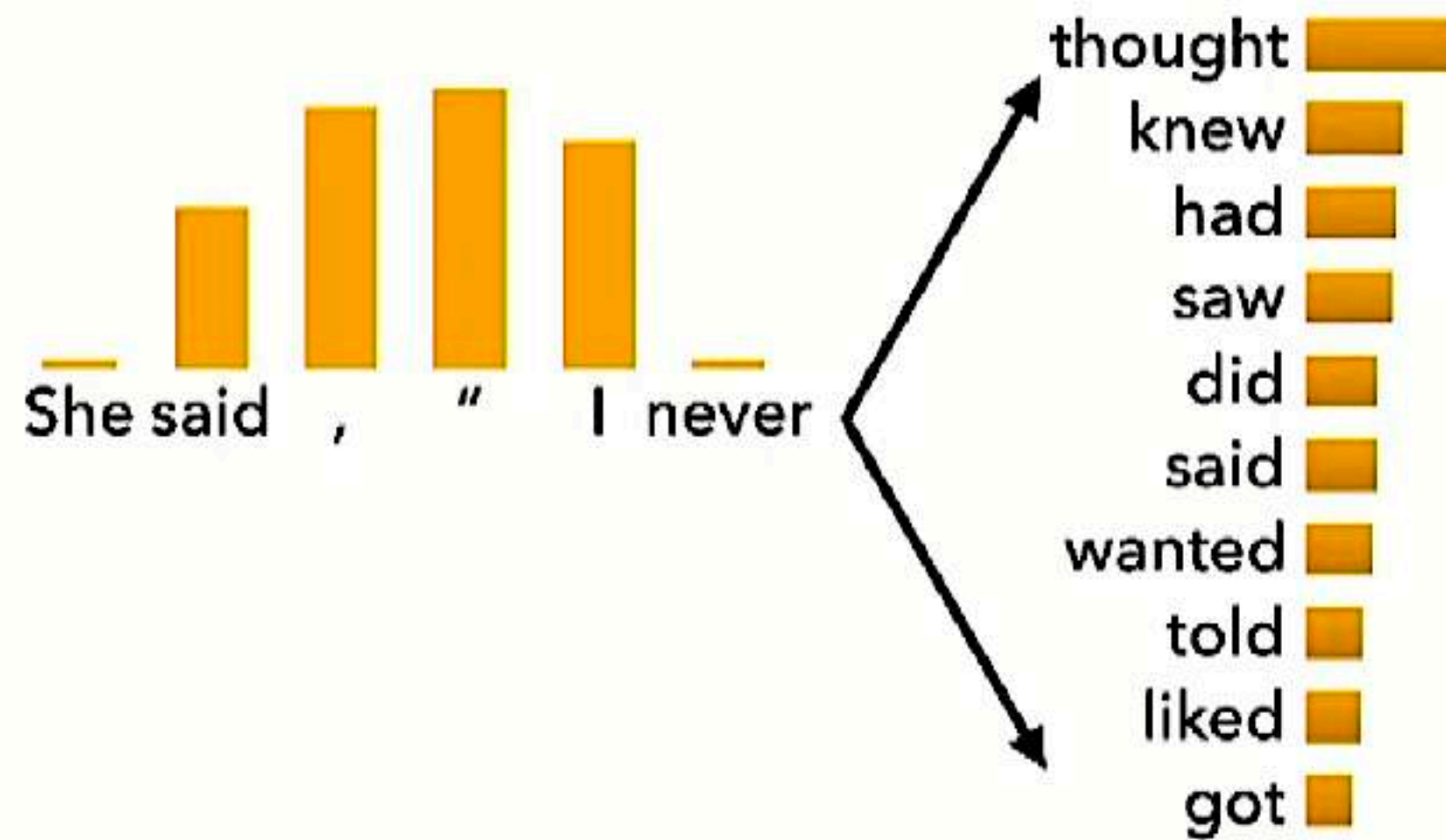
He wanted
to go to the

Model

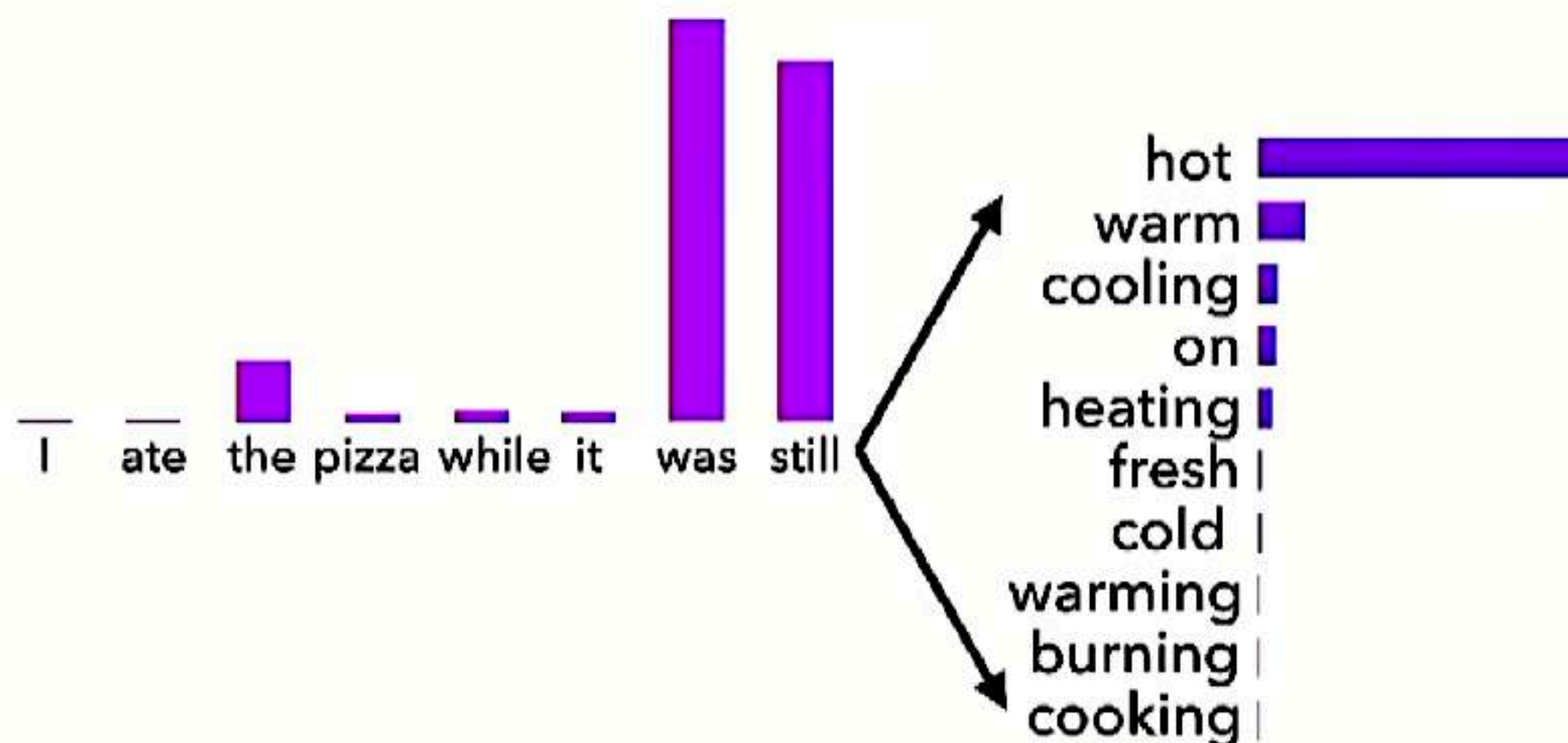
restroom
grocery
store
airport
bathroom
beach
doctor
hospital
pub
gym

- Increase k yields more **diverse**, but **risky** outputs
- Decrease k yields more **safe** but **generic** outputs

Issues with Top-k Sampling



Top-*k* sampling can cut off too ***quickly!***



Top-*k* sampling can also cut off too ***slowly!***

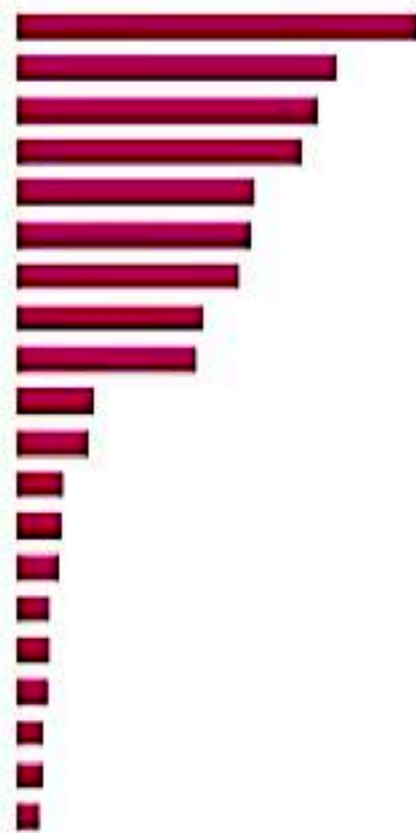
Top-p (nucleus) Sampling

- 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: 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

Top-p (nucleus) 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

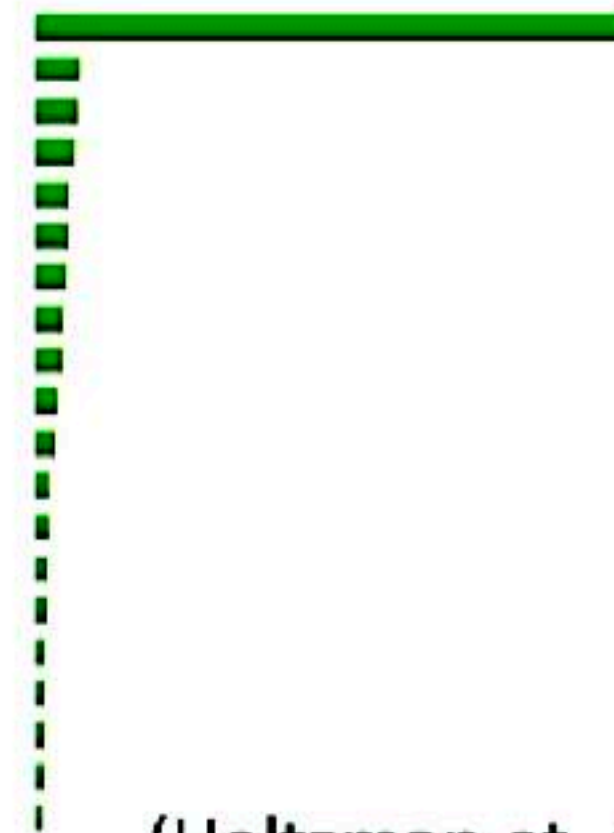
$$P_t^1(y_t = w | \{y\}_{<t})$$



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



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

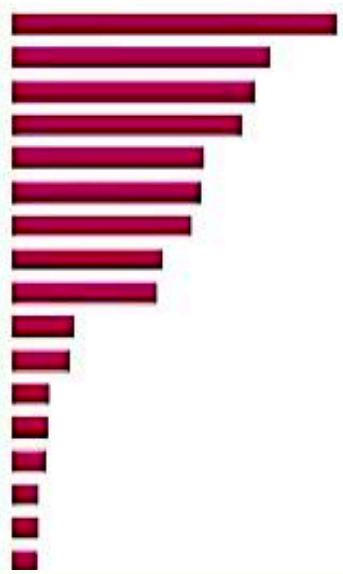


(Holtzman et. al., ICLR 2020)

Top-p (nucleus) 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

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



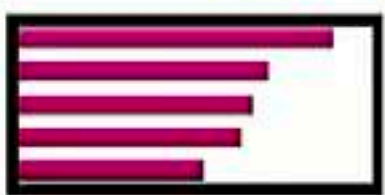
$$P_t^2(y_t = w | \{y\}_{<t})$$



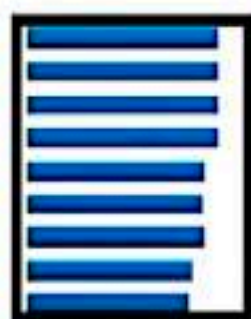
$$P_t^3(y_t = w | \{y\}_{<t})$$



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



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



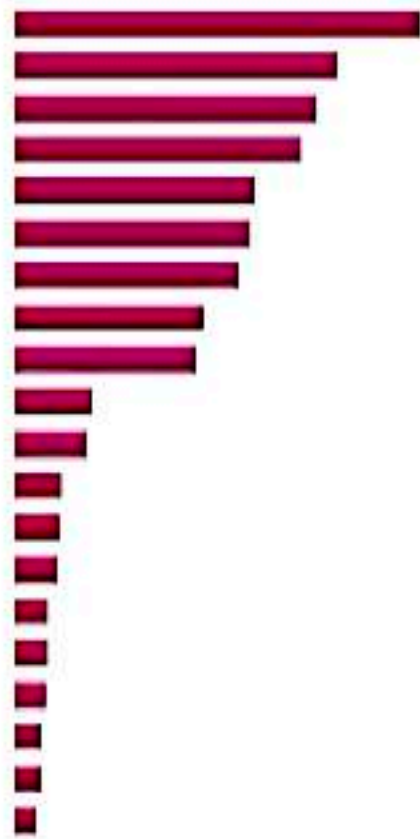
$$P_t^3(y_t = w | \{y\}_{<t})$$



What else?

- Typical Sampling (Meister et al. 2022)
 - Reweights the score based on the entropy of the distribution.
- Epsilon Sampling (Hewitt et al. 2022)
 - Set a threshold for lower bounding valid probabilities.

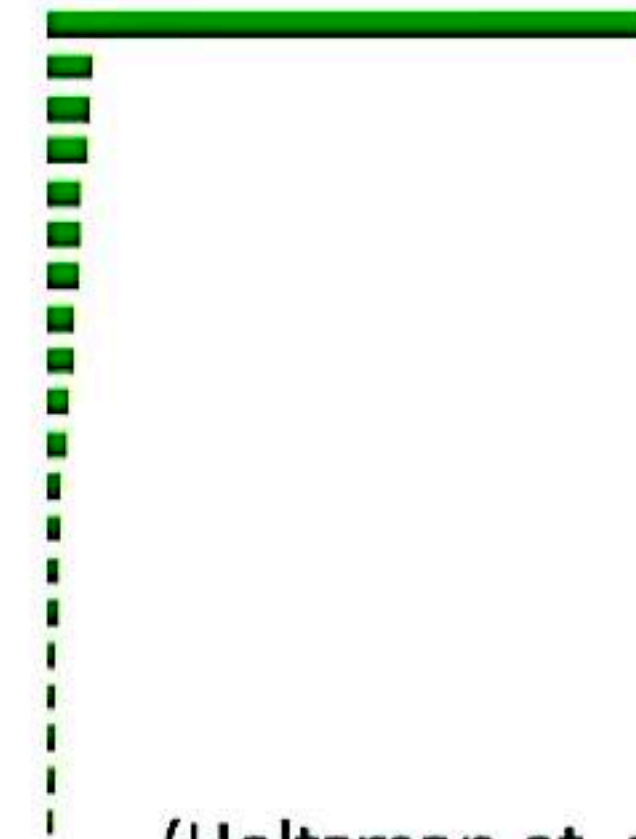
$$P_t^1(y_t = w | \{y\}_{<t})$$



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



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



(Holtzman et. al., ICLR 2020)

Scaling: Temperature

- Recall: On timestep t , the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t :

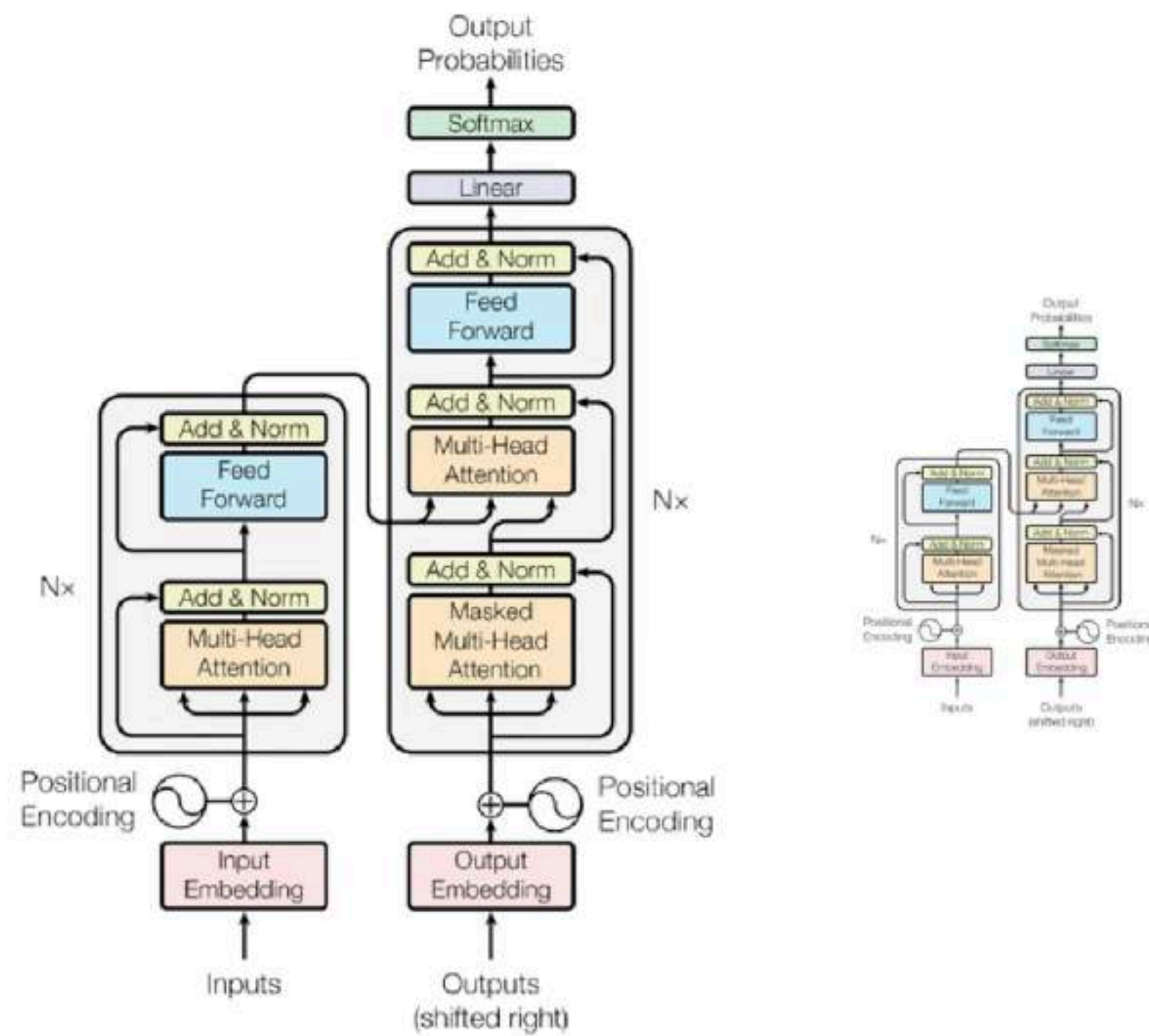
$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- **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)

Further Improvement: Re-ranking

- Problem: What if I decode a bad sequence from my model?
- Decode a bunch of sequences
 - 10 candidates is a common number, but it's up to you
- Define a score to approximate quality of sequences and re-rank by this score
 - Simplest is to use (low) perplexity!
 - Careful! Remember that repetitive utterances generally get low perplexity.
 - Re-rankers can score a variety of properties:
 - style (Holtzman et al., 2018), discourse (Gabriel et al., 2021), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more ...
 - Beware poorly-calibrated re-rankers
 - Can compose multiple re-rankers together.

Speculative Sampling



Two LLMs: smaller draft model

Speed up inference 2.5-3x

Rejection sampling trick to preserve original accuracy

Speculative Sampling



2023-2-3

Accelerating Large Language Model Decoding with Speculative Sampling

Charlie Chen¹, Sebastian Borgeaud¹, Geoffrey Irving¹, Jean-Baptiste Lespiau¹, Laurent Sifre¹ and John Jumper¹

¹All authors from DeepMind

We present speculative sampling, an algorithm for accelerating transformer decoding by generating multiple tokens from each transformer call. Our algorithm relies on the observation that the latency of parallel scoring of short continuations, generated by a faster but less powerful model, is comparable to that of sampling a single token from the larger target model. This is achieved with a novel modified rejection sampling scheme which preserves the distribution of the tokens within hardware numerics. We benchmark speculative sampling with Chinchilla, a 70 billion parameter language model, achieving a 2–2.5× decoding speedup in a distributed setup, without compromising the sample quality or making modifications to the model itself.

Introduction

Scaling transformer models to 500B+ parameters has led to large performance improvements across many natural language, computer vision and reinforcement learning tasks (Arnab et al., 2020; Chowdhery et al., 2022; Dosovitskiy et al., 2020; Hoffmann et al., 2022; Rae et al., 2022). However, transformer decoding remains a highly costly and inefficient process in this regime.

Fast Inference from Transformers via Speculative Decoding

Yaniv Leviathan^{*1} Matan Kalman^{*1} Yossi Matias¹

Abstract

Inference from large autoregressive models like Transformers is slow - decoding K tokens takes K serial runs of the model. In this work we introduce *speculative decoding* - an algorithm to sample from autoregressive models faster *without any changes to the outputs*, by computing several tokens in parallel. At the heart of our approach lie the observations that (1) hard language-modeling tasks often include easier subtasks that can be approximated well by more efficient models, and (2) using speculative execution and a novel sampling method, we can make exact decoding from the large models faster, by running them in parallel on the outputs of the approximation models, potentially generating several tokens concurrently, and without changing the distribution. Our method can accelerate existing off-the-shelf models without retraining or architecture changes. We

developed to make inference from them faster. Some approaches aim to reduce the inference cost for *all* inputs equally (e.g. Hinton et al., 2015; Jaszczur et al., 2021; Hubara et al., 2016; So et al., 2021; Shazeer, 2019). Other approaches stem from the observation that not all inference steps are born alike - some require a very large model, while others can be approximated well by more efficient models. These *adaptive computation* methods (e.g. Han et al., 2021; Sukhbaatar et al., 2019; Schuster et al., 2021; Scardapane et al., 2020; Bapna et al., 2020; Elbayad et al., 2019; Schwartz et al., 2020) aim to use less compute resources for easier inference steps. While many of these solutions have proven extremely effective in practice, they usually require changing the model architecture, changing the training-procedure and re-training the models, and don't maintain identical outputs.

The key observation above, that some inference steps are “harder” and some are “easier”, is also a key motivator for



Speculative Sampling

Autoregressive Sampling

The standard way of generating text from a language model is with **autoregressive sampling**, here's the algorithm as defined in the paper:

Algorithm 1 Auto-regressive (ArS) with Auto-Regressive Models

Given auto-regressive target model $q(.|.)$ and initial prompt sequence x_1, \dots, x_t and target sequence length T .

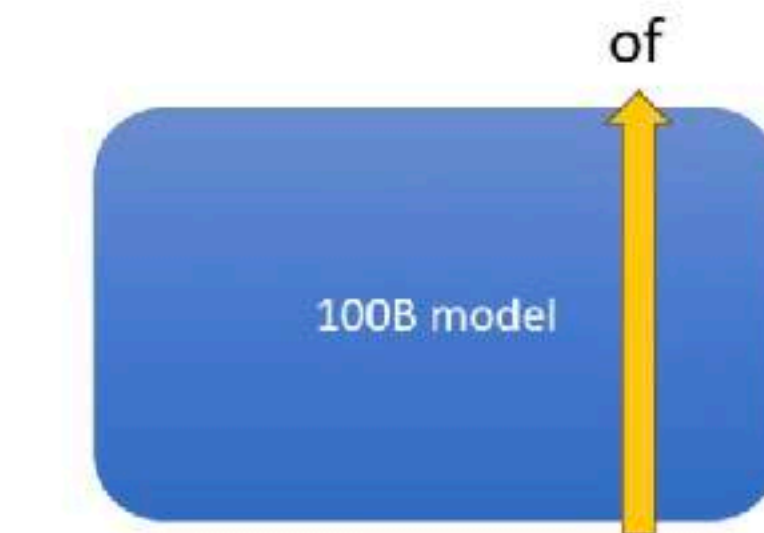
Initialise $n \leftarrow t$.

while $n < T$ **do**

 Sample $x_{n+1} \sim q(x|x_1, \dots, x_n)$

$n \leftarrow n + 1$

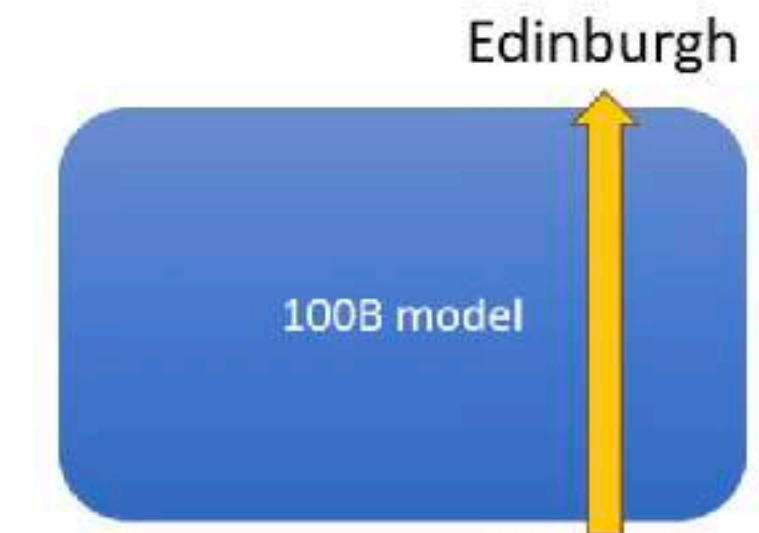
end while



Geoffrey Hinton did his PhD
at the University...

Very Easy

Use 1B draft model



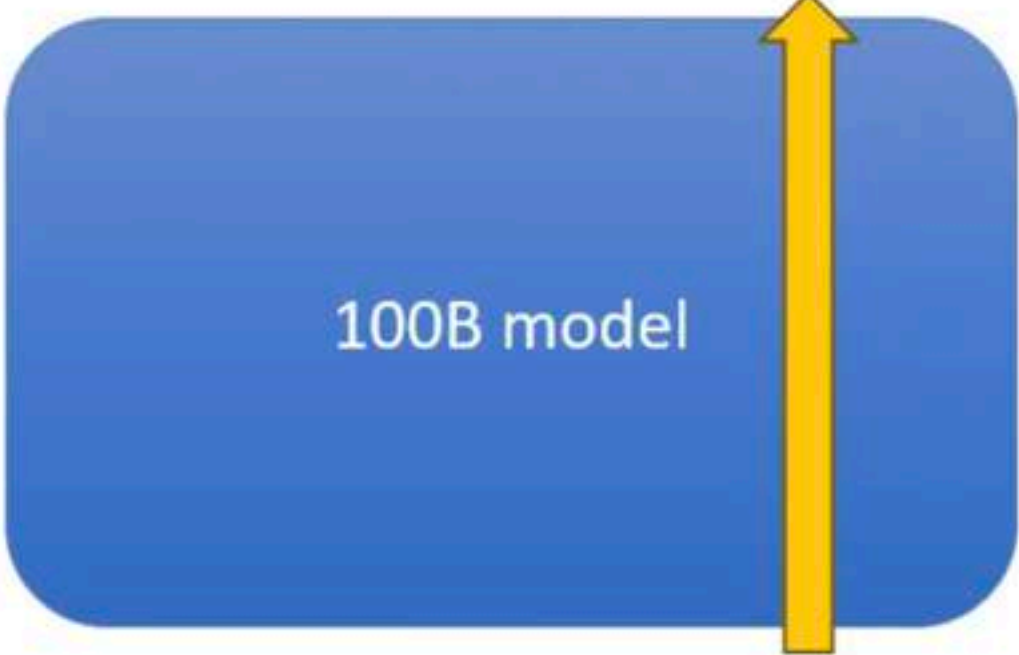
Geoffrey Hinton did his PhD
at the University of...

Difficult

Use 100B model

Speculative Sampling

Edinburgh



100B model

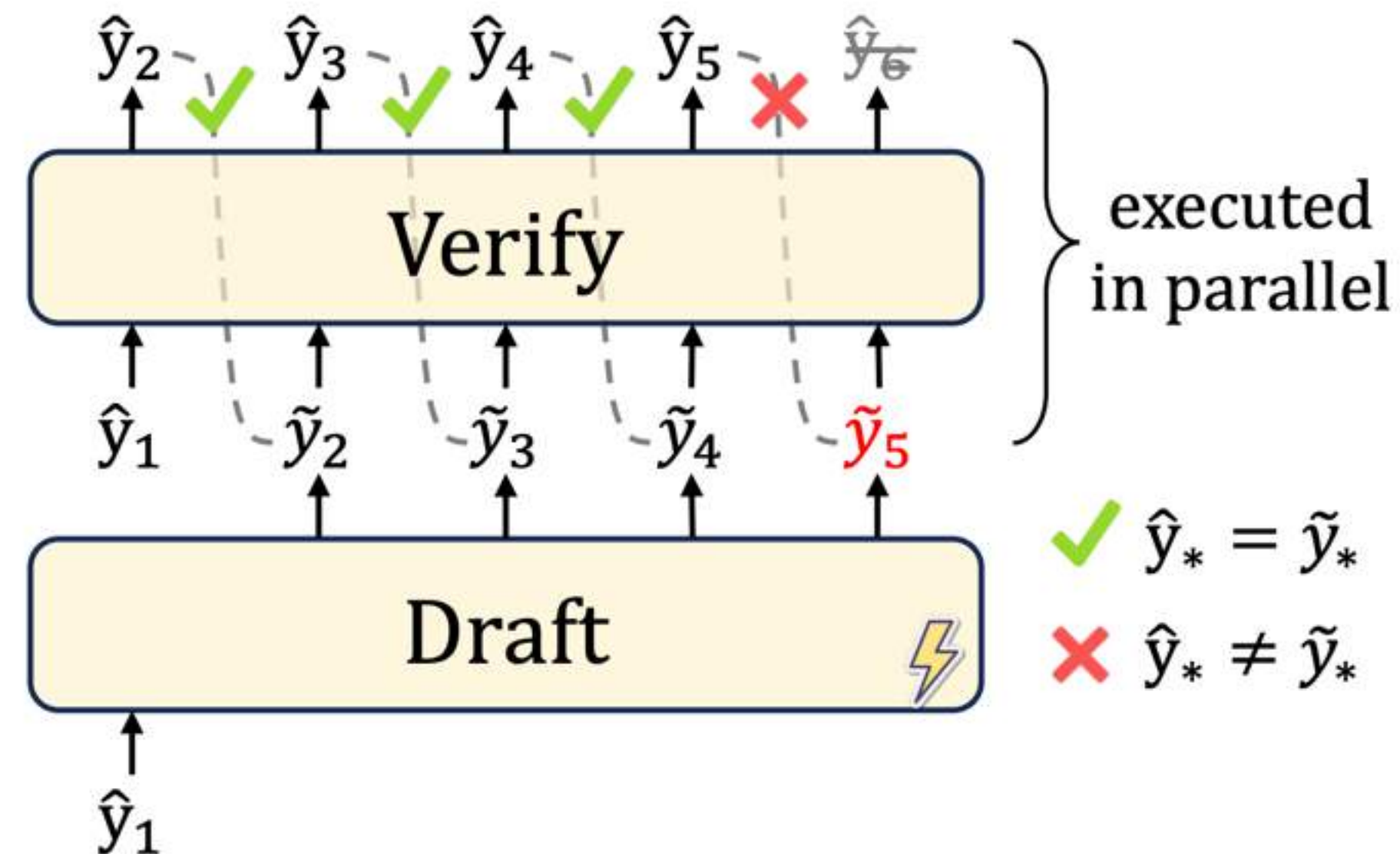
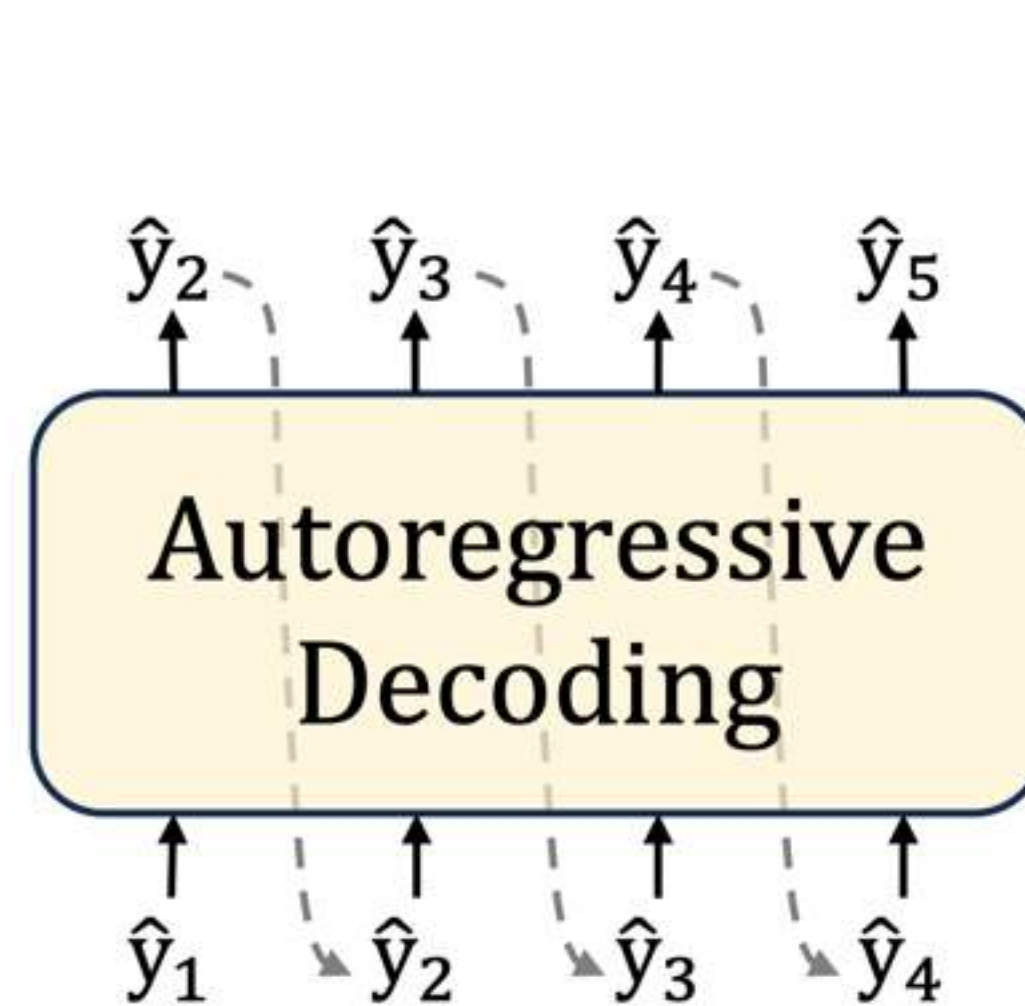
Geoffrey Hinton did his PhD
at the University of...

at the University of Toronto



100B model

Geoffrey Hinton did his PhD
at the University of Toronto



Speculative Sampling

1. A smaller, faster **draft model** (e.g. DeepMind's 7B Chinchilla model)
2. A larger, slower **target model** (e.g. DeepMind's 70B Chinchilla model)


The idea is that the draft model *speculates* what the output is K steps into the future, while the target model determines how many of those tokens we should *accept*. Here's an outline of the algorithm:

1. The draft model decodes K tokens in the regular autoregressive fashion.
2. We get the probability outputs of the target and draft model on the new predicted sequence.
3. We compare the target and draft model probabilities to determine how many of the K tokens we want to keep based on some **rejection criteria**. If a token is rejected, we **resample** it using a combination of the two distributions and don't accept any more tokens.
4. If all K tokens are accepted, we can sample an additional final token from the target model probability output.


Speculative Sampling

Algorithm

M_p = draft model

 meta-llama/Llama-2-7b-chat-hf

M_q = target model


 meta-llama/Llama-2-70b-chat-hf

pf = prefix, $K = 5$ tokens

$p_1(x) = M_p(pf)$  x_1

$p_2(x) = M_p(pf, x_1)$  x_2

...

$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4)$  x_5

Speculative Sampling

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

Run draft model
for K steps

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Run target model once

Speculative Sampling

$$p_1(x) = M_p(pf) \longrightarrow x_1$$

$$p_2(x) = M_p(pf, x_1) \longrightarrow x_2$$

...

$$p_5(x) = M_p(pf, x_1, x_2, x_3, x_4) \longrightarrow x_5$$

$$q_1(x), q_2(x), q_3(x), q_4(x), q_5(x), q_6(x)$$

$$= M_q(pf, x_1, x_2, x_3, x_4, x_5)$$

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Speculative Sampling

Rejection Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
$p(x)$	0.8	0.7	0.9	0.8	0.7
$q(x)$	0.9	0.8	0.8	0.3	0.8

Draft model: dogs love chasing after cars because they find this relaxing

Target model: OK OK sleeping

Speculative Sampling

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
$p(x)$	0.8	0.7	0.9	0.8	0.7
$q(x)$	0.9	0.8	0.8	0.3	0.8

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$



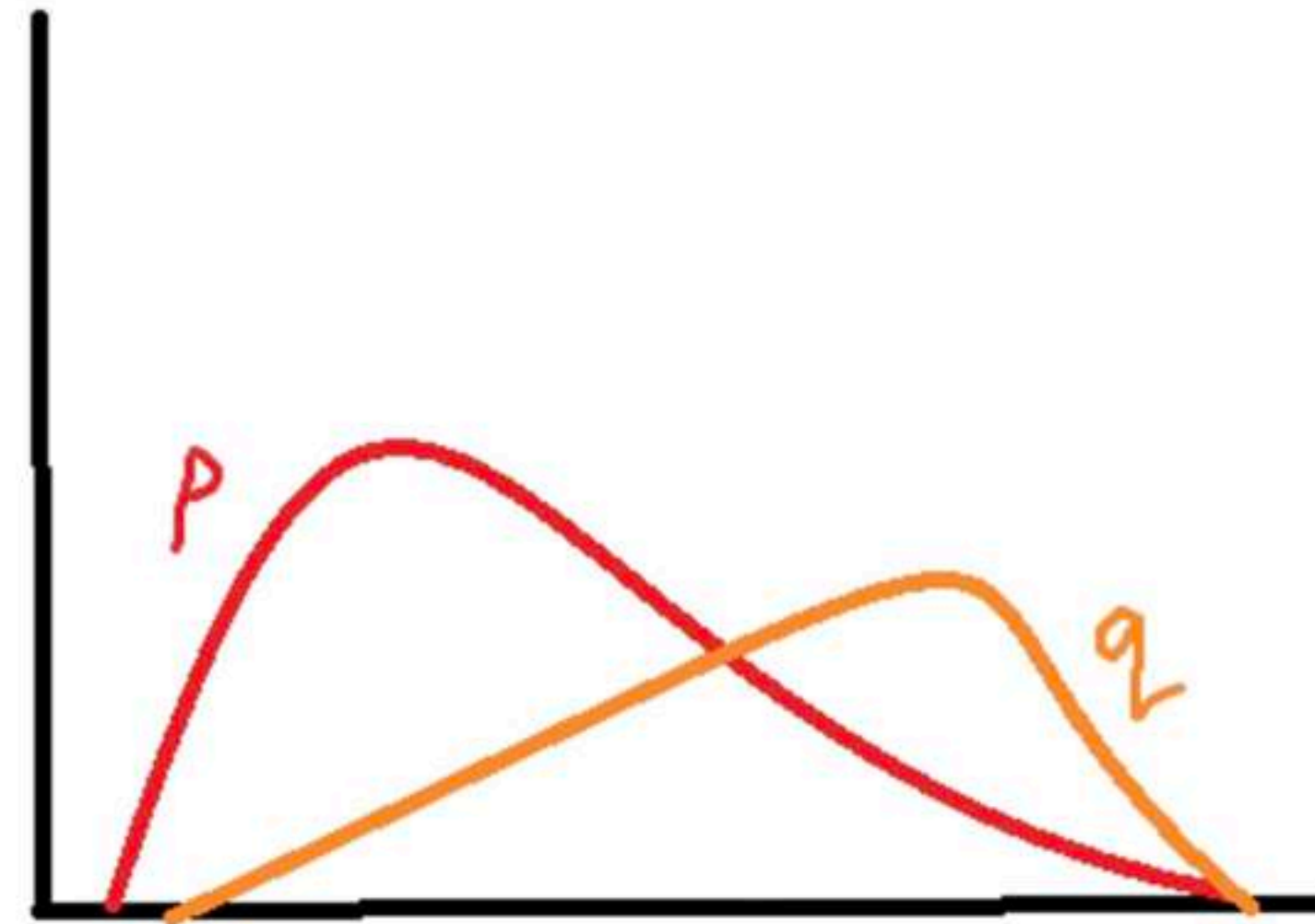
Speculative Sampling

Why sample from $(q(x) - p(x))_+$?

Goal: Sample a token from $q(x)$

Case 1: If $q(x) \geq p(x)$, then accept

Case 2: If $q(x) < p(x)$, then accept with probability $\frac{q(x)}{p(x)}$



Speculative Sampling

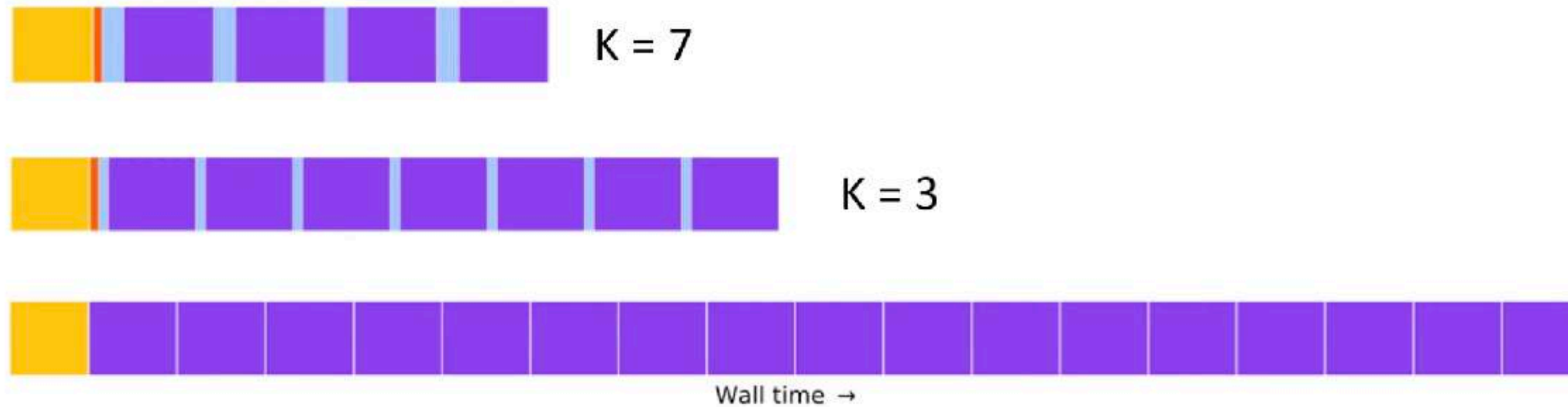
How many tokens generated in one pass?

Token	x1	x2	x3	x4	x5
	dogs	love	chasing	after	cars
p(x)	0.8	0.7	0.9	0.8	0.7
q(x)	0.9	0.8	0.8	0.3	0.8

Worst case: first token is rejected -> 1 token

Best case: all tokens accepted -> K+1 tokens

Speculative Sampling



```
[START] japan ' s benchmark bond n
[START] japan ' s benchmark nikkei 22 5
[START] japan ' s benchmark nikkei 225 index rose 22 6
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in tekyo late
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

Speculative Sampling

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×



Recommends $K = 3-4$
Finds 2-2.5x speedup



Recommends $K = 3-7$
Finds 2-3.4x speedup

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	3.4X
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	2.6X
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	3.1X
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	2.3X
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

Takeaways

- Decoding is still a challenging problem in NLG – there's a lot more work to be done!
- Different decoding algorithms can allow us to inject biases that encourage different properties of coherent natural language generation
- Some of the most impactful advances in NLG of the last few years have come from simple but effective modifications to decoding algorithms