COMP 3361 Natural Language Processing

Lecture 17: Natural language generation with LLMs (cont'd)

Spring 2025

Announcements

- Assignment 3 is out, due on May 9th.
  - Join #assignment-3 Slack channel for discussion
-

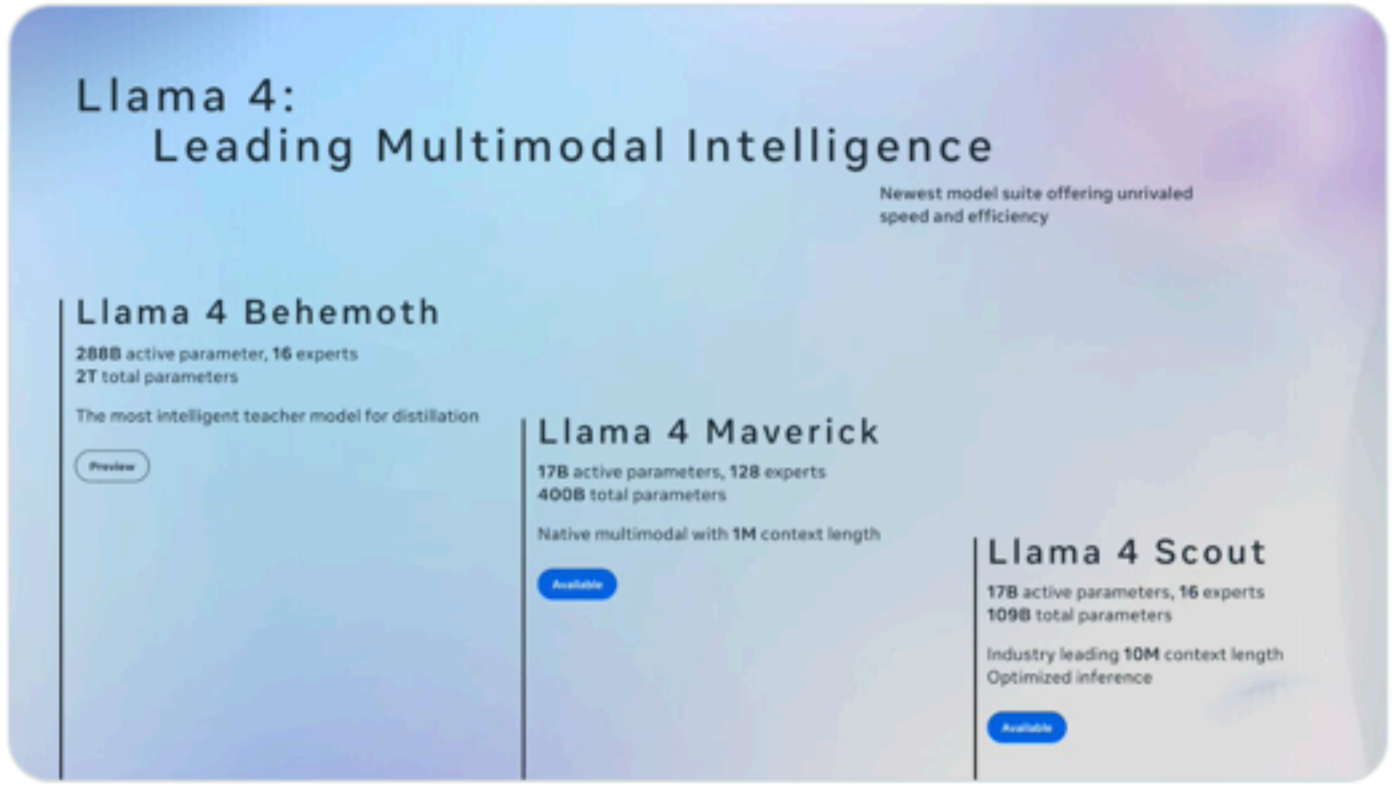
Latest AI news

AI at Meta @AIatMeta · Apr 6

Today is the start of a new era of natively multimodal AI innovation.

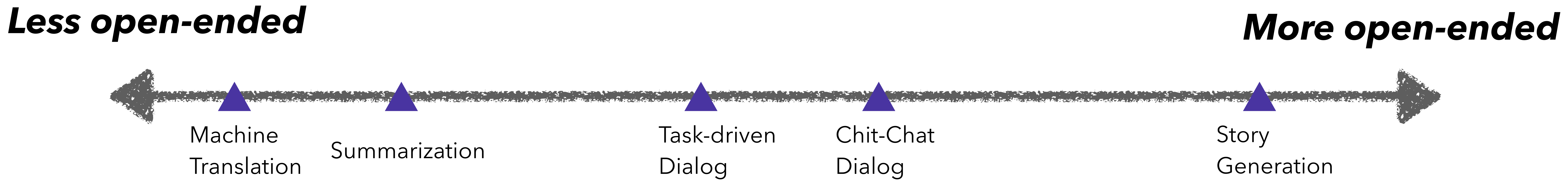
Today, we're introducing the first Llama 4 models: Llama 4 Scout and Llama 4 Maverick — our most advanced models yet and the best in their class for multimodality.

Llama 4 Scout
• 17B-active-parameter model
Show more



679 3.5K 12K 2.8M

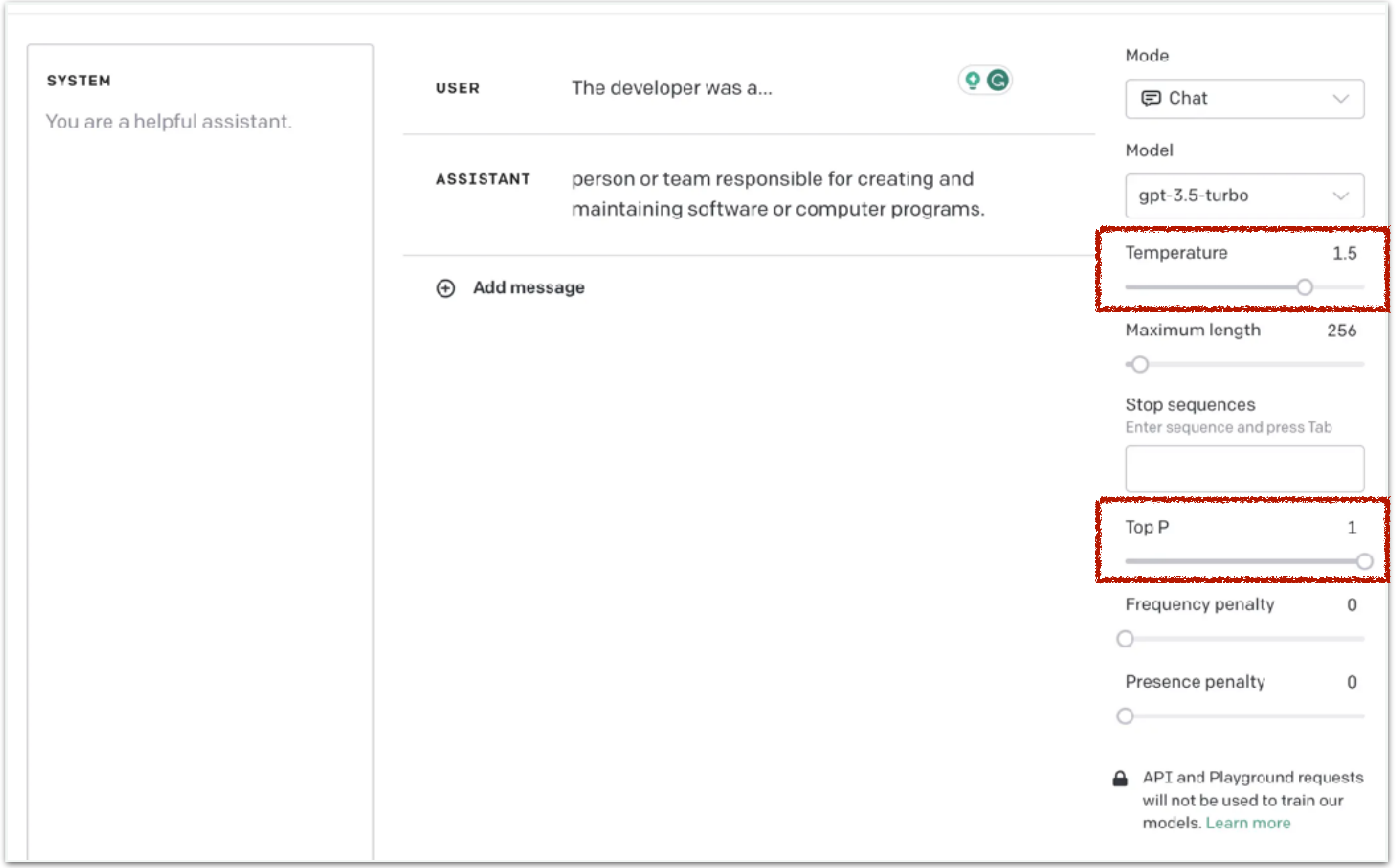
Categorization of NLG tasks



Less open-ended generation: the input mostly determines the correct output generation.

More open-ended generation: the output distribution still has high degree of freedom.

How to control open-endedness in ChatGPT?



ChatGPT API web interface

Decoding from LLMs

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

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

$f(\cdot\,;\theta)$ is your model

- Then, we compute a probability distribution P over $w \in V$ using these scores:

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

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

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

$g(\cdot)$ is your decoding algorithm

How to find the most likely text to generate?

- **Obvious method: Greedy Decoding**
  - Selects the highest probability token according to $P(y_t | y_{<t})$

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

- **Beam Search**
  - Also aims to find the string with the highest probability, but with a wider exploration of candidates.

How to find the most likely text to generate?

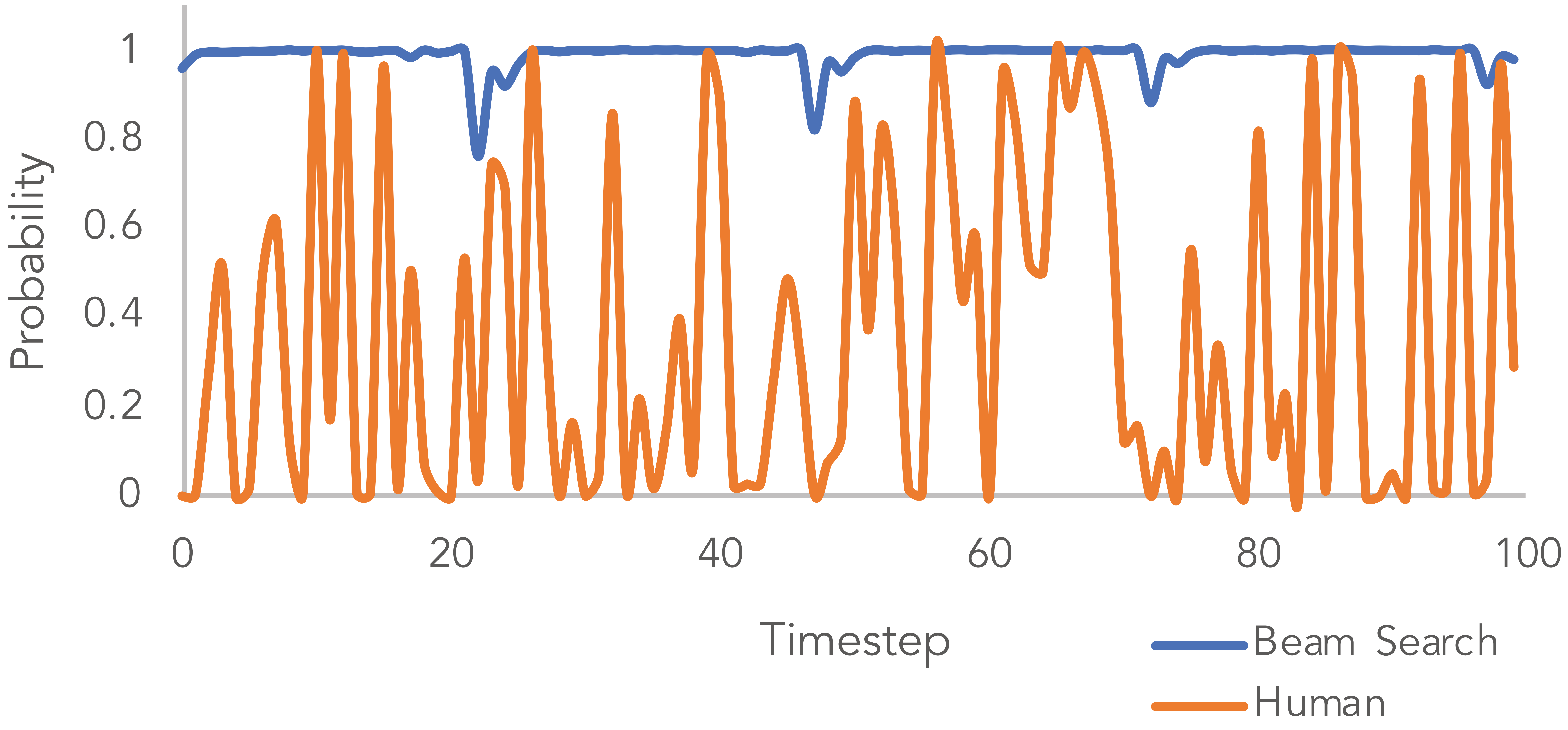
- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.
  - Compared to greedy decoding, beam search gives a better approximation of brute-force search over all sequences
  - A small overhead in computation due to beam width
    Time complexity: O(beam width * vocab size * generation length)

    ** Naive brute-force search: O(vocab size ^ generation length), hence intractable!*

Note: *Overall, greedy / beam search is widely used for low-entropy tasks like MT and summarization.*

But, are greedy sequences always the best solution? 🤔

Also, are greedy methods reasonable for open-ended generation?



Greedy methods fail to capture the <u>variance of human text distribution</u>.

(Holtzman et al. ICLR 2020)

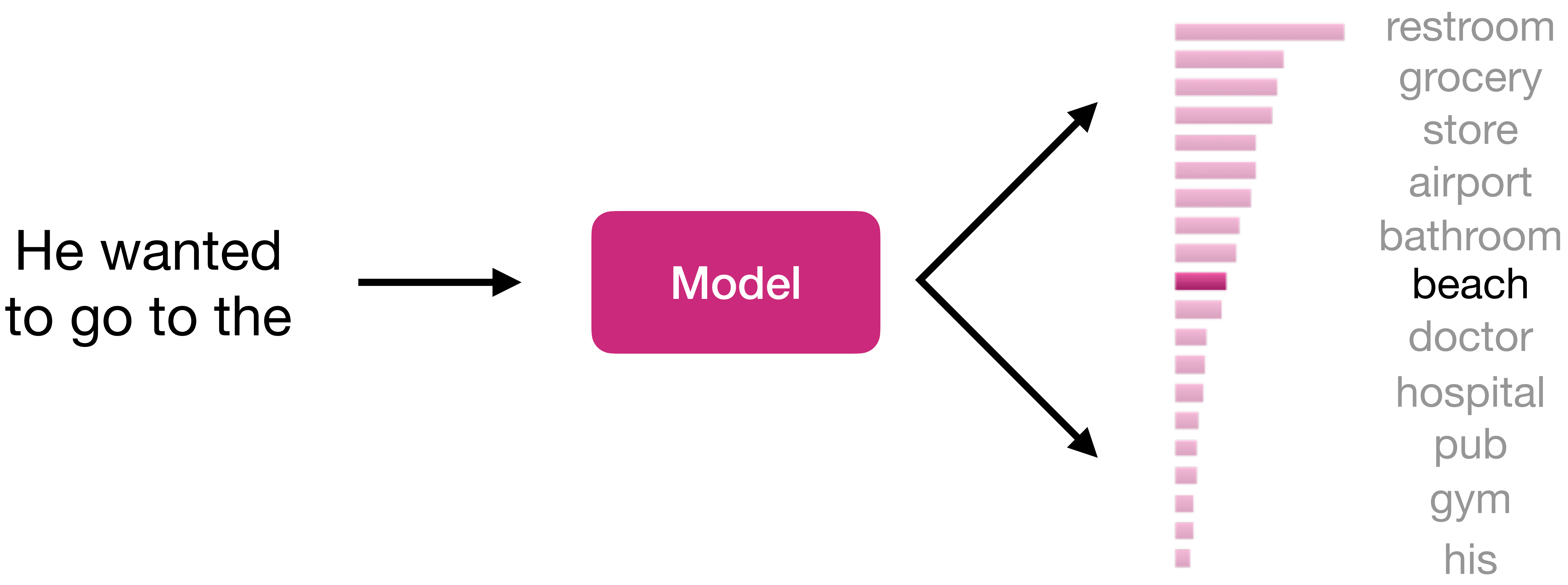
Sampling generation from LLMs

Time to get random: Sampling

- Sample a token from the token distribution at each step!

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

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



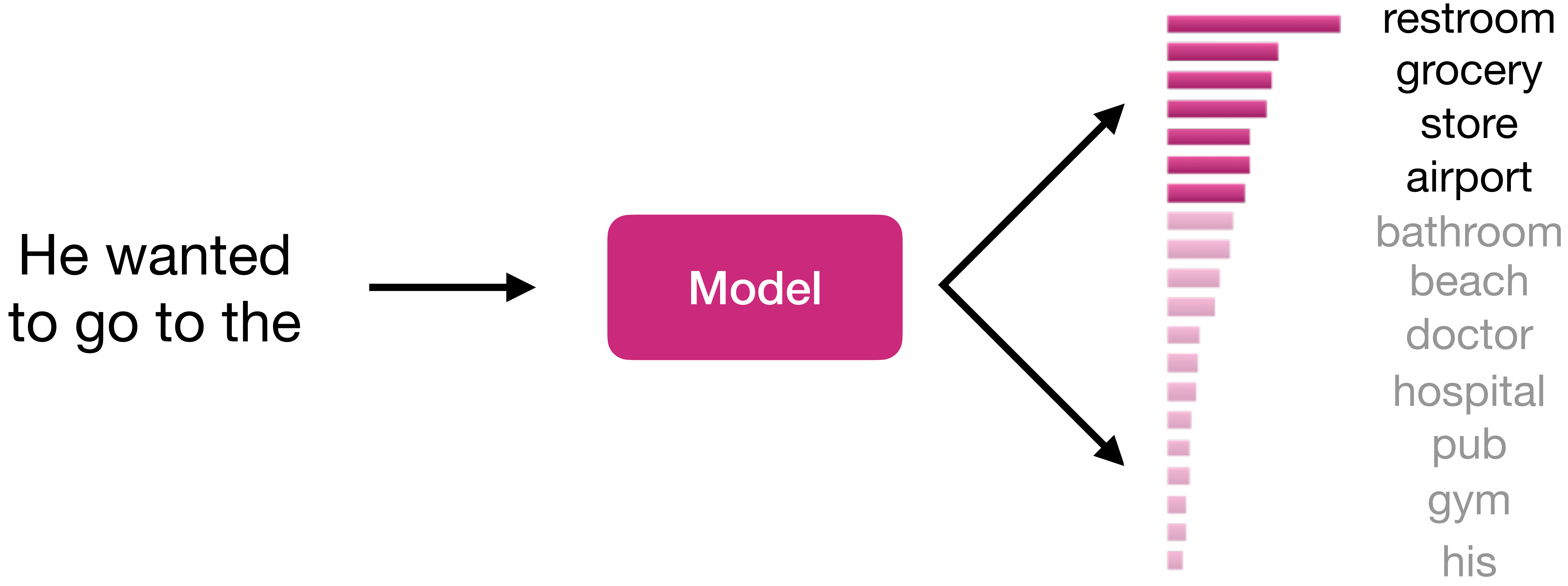
Decoding: Top-k Sampling

- Problem: Vanilla sampling makes *every token* in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have "heavy tailed" distributions)
  - Many tokens are probably really wrong in the current context.
  - Although *each of them* may be assigned a small probability, *in aggregate* they still get a high chance to be selected.

- Solution: Top-k sampling *(Fan et al., 2018)*
  - Only sample from the top k tokens in the probability distribution.

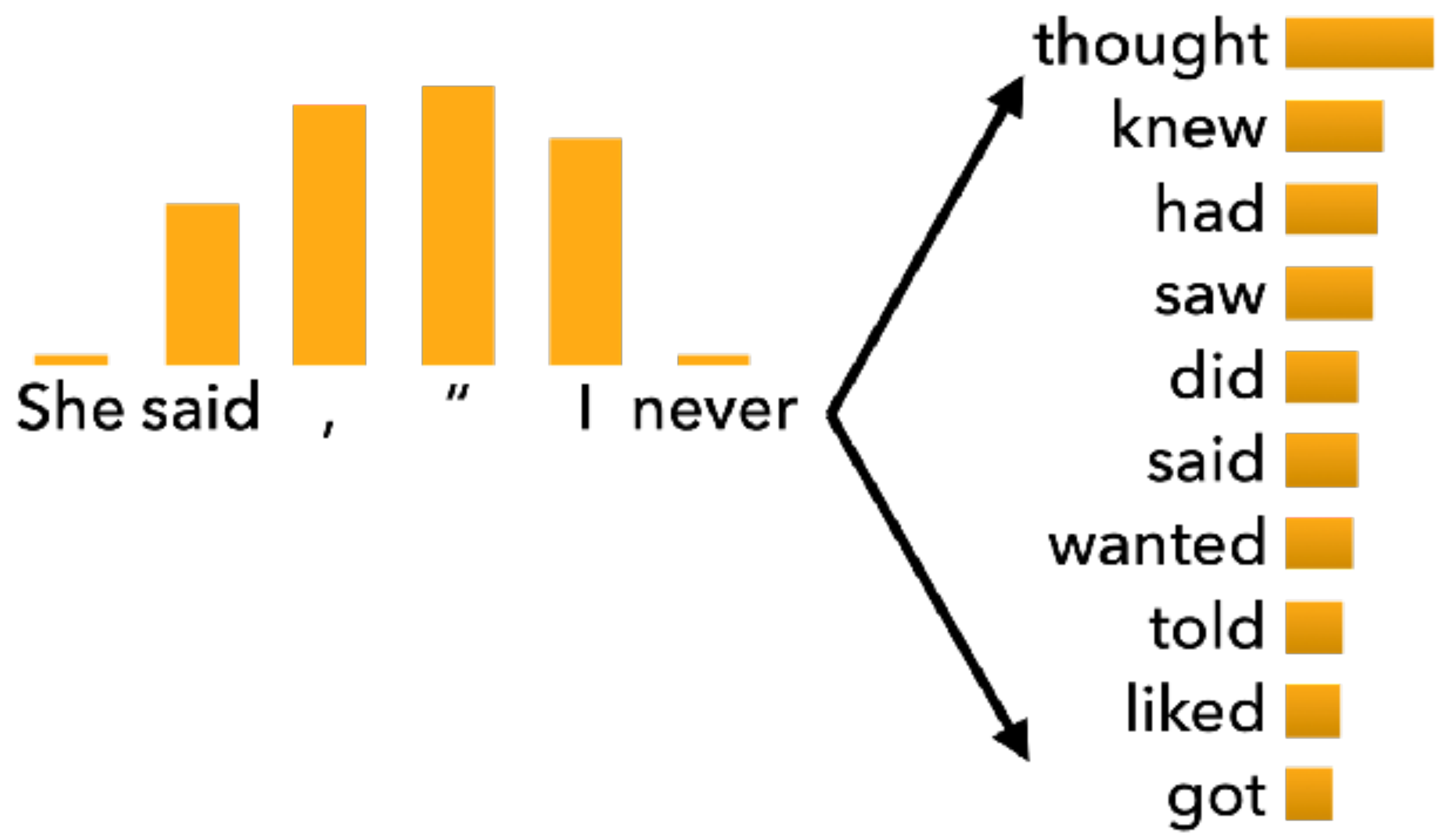
Decoding: Top-k Sampling

- Solution: Top-*k* sampling *(Fan et al., 2018)*
  - Only sample from the top *k* tokens in the probability distribution.
  - Common values for *k* = 10, 20, 50 (*but it's up to you!*)



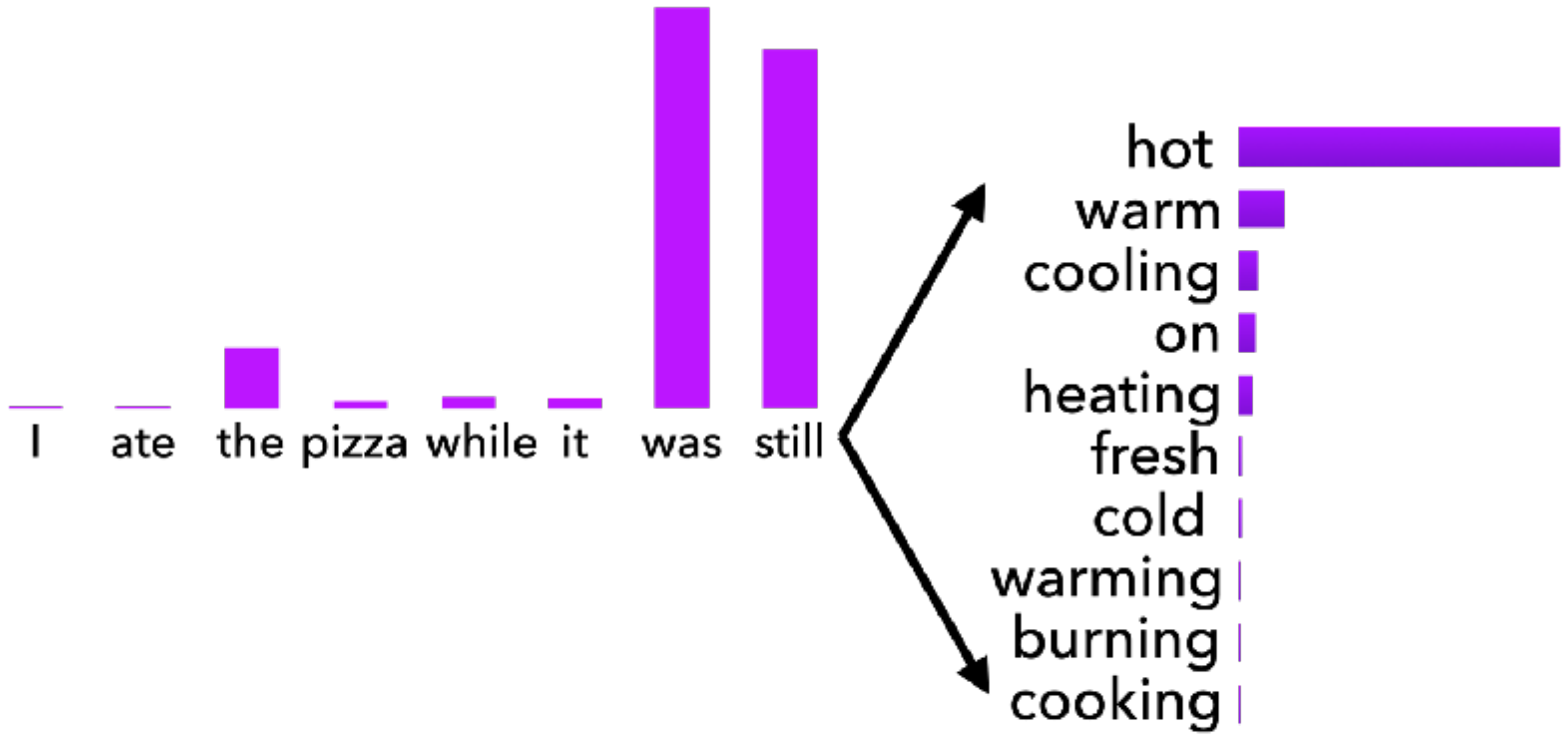
- Increasing *k* yields more **diverse**, but **risky** outputs
- Decreasing *k* yields more **safe** but **generic** outputs

Issues with Top-k Sampling



For *flat* distribution,

Top-*k* Sampling may cut off too **quickly**!



For *peaked* distribution,

Top-*k* Sampling may also cut off too **slowly**!

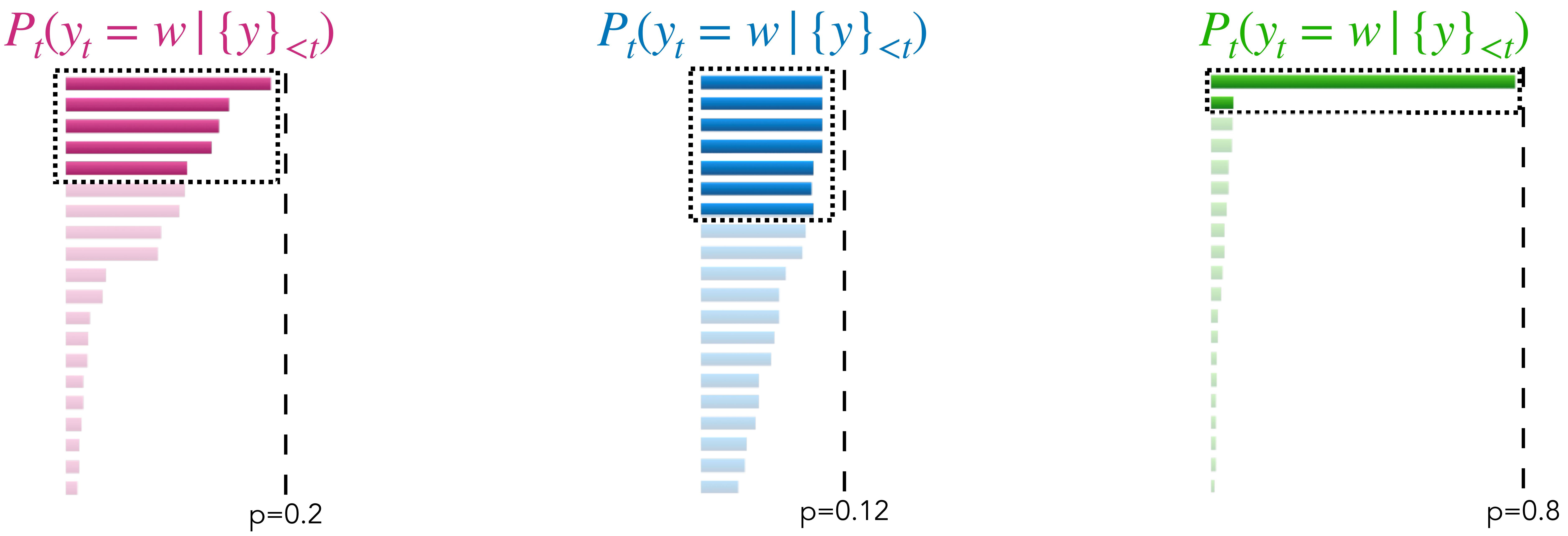
Decoding: Top-p (Nucleus) Sampling

- Problem: The token distributions we sample from are dynamic
  - When the distribution P_t is flat, small k removes many viable options.
  - When the distribution P_t is peaked, large k allows too many options a chance to be selected.

- Solution: Top-*p* sampling *(Holtzman et al., 2020)*
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k according to the uniformity of P_t

Decoding: Top-p (Nucleus) Sampling

- <u>Solution:</u> Top-*p* sampling *(Holtzman et al., 2020)*
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k according to the uniformity of P_t



Scaling randomness: Softmax temperature

- Recall: At time step t, model computes a distribution P_t by applying softmax to a vector of scores $S \in \mathbb{R}^{|V|}$

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

- Here, you can apply **temperature hyperparameter** τ to the softmax to rebalance P_t:

$$P_t(y_t = w \mid \{y_{<t}\}) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

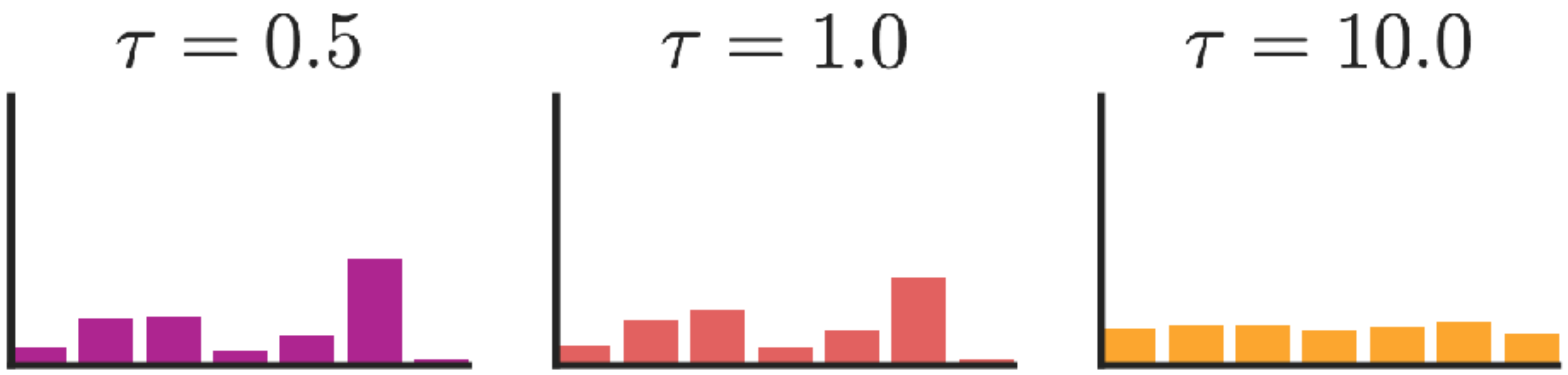
- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)

Scaling randomness: Softmax temperature

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

$$P_t(y_t = w \mid \{y_{<t}\}) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)



Scaling randomness: Softmax temperature

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

$$P_t(y_t = w \mid \{y_{<t}\}) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)

NOTE: Temperature is a hyperparameter for decoding algorithm, not an algorithm itself! It can be applied for both beam search and sampling methods.

Toward better generation: Re-ranking

- Problem: What if I already have decoded a bad sequence from my model?

- **Decode a bunch of sequences**
  - Sample $n = 10, 20, 50, \ldots$ sequences with the same input given
- Define a score to approximate quality of sequences and **re-rank by this score**
  - Simplest score: (low) perplexity
    - Careful! Remember that even the repetitive sequences get low perplexity in general…
  - Re-rankers can evaluate a variety of properties:
    - Style *(Holtzman et al., 2018)*, Discourse *(Gabriel et al., 2021)*, Factuality *(Goyal et al., 2020)*, Logical Consistency *(Jung et al. 2022)*, and many more
  - Can compose multiple re-rankers together.

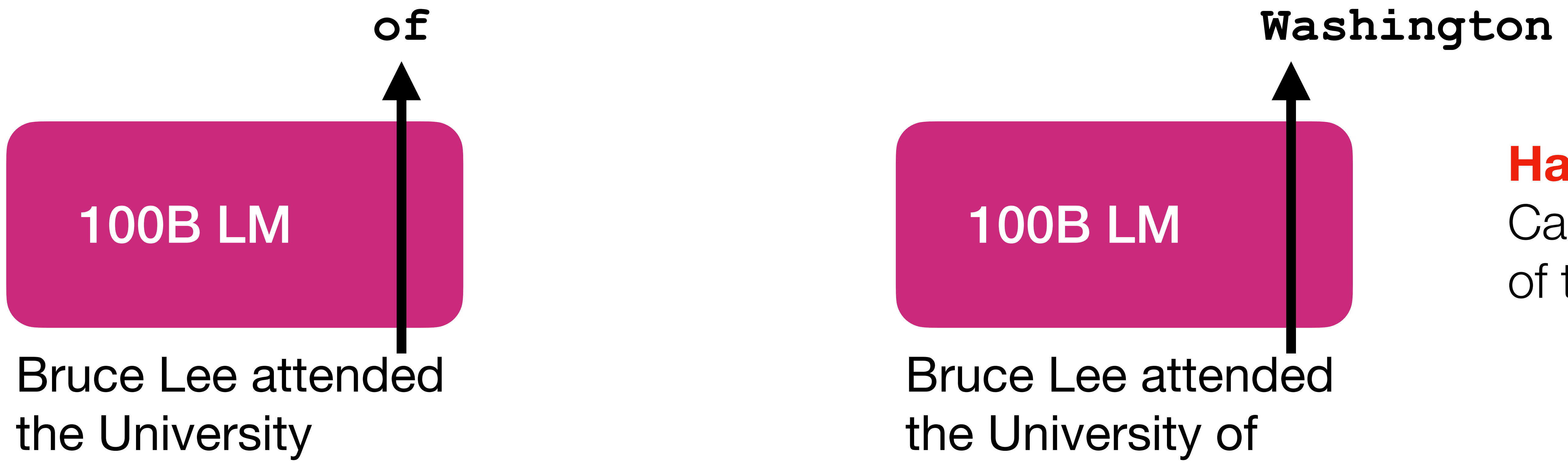
Speeding-up generation from LLMs

Speeding-up generation: Speculative Sampling

- <u>Problem:</u> Generating with a large LM takes a long time

- Intuition: Not all tokens are equally hard to generate!

Easy to predict:
May be a 1B LM
can predict this too



Hard to predict:
Can really make use
of the 100B LM here

- **<u>Idea</u>**: Use a generation from small LM to assist large LM generation

* *Same idea independently proposed from DeepMind and Google - see Chen et al., 2023; Leviathan et al., 2023*

Speeding-up generation: Speculative Sampling

- First, sample a draft of length K (= 5 in this example) from a small LM M_p

  $y_1 \sim p(\cdot \mid \underline{x}), y_2 \sim p(\cdot \mid x, y_1), \cdots, y_5 \sim p(\cdot \mid x, y_1, y_2, y_3, y_4)$

  Input prefix

- Then, compute the token distribution at each time step with a large target LM M_q

  $q(\cdot \mid x), q(\cdot \mid x, y_1), \underline{q(\cdot \mid x, y_1, y_2)}, \cdots, q(\cdot \mid x, y_1, \cdots, y_5)$

  Next token distribution of M_q, when given x, y_1, y_2

  - Note: This can be computed in a *single forward pass* of M_q (Why?)

- Let's denote $p_i = p(\cdot \mid x, y_1, \cdots, y_{i-1})$ and $q_i = q(\cdot \mid x, y_1, \cdots y_{i-1})$
  e.g., $q_2 = q(\cdot \mid x, y_1)$, *i.e. next token distribution predicted by the target model* M_q, *when given* x *and* y_1

Speeding-up generation: Speculative Sampling

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

	Token	y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether or not to accept the tokens generated by the draft model.

Speeding-up generation: Speculative Sampling

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

	Token	y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether or not to accept the tokens generated by the draft model.
- Case 1: $q_i \geq p_i$
  The target model (100B) likes this token, even more than the draft model (which generated it).
  => Accept this token!

Generation after step 1:
`dogs`

Speeding-up generation: Speculative Sampling

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

Token		y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8	0.3	0.8

- Starting from y_1, decide whether or not to accept the tokens generated by the draft model.
- Case 1: $q_i \geq p_i$
  The target model (100B) likes this token, even more than the draft model (which generated it).
  => Accept this token!

Generation after step 2:
`dogs love`

Speeding-up generation: Speculative Sampling

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

	Token	y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	after	cars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8	0.3	0.8

- Case 2: $q_i < p_i$ (accept)
  Target model doesn't like this token as much as the draft model...

  => Accept it with the probability $\dfrac{q_i}{p_i}$

Generation after step 3:
`dogs love chasing`

In this example, assume we accepted it with prob=0.8/0.9

Speeding-up generation: Speculative Sampling

- Now, we can compare the probability of each token assigned by draft model M_p and target model M_q

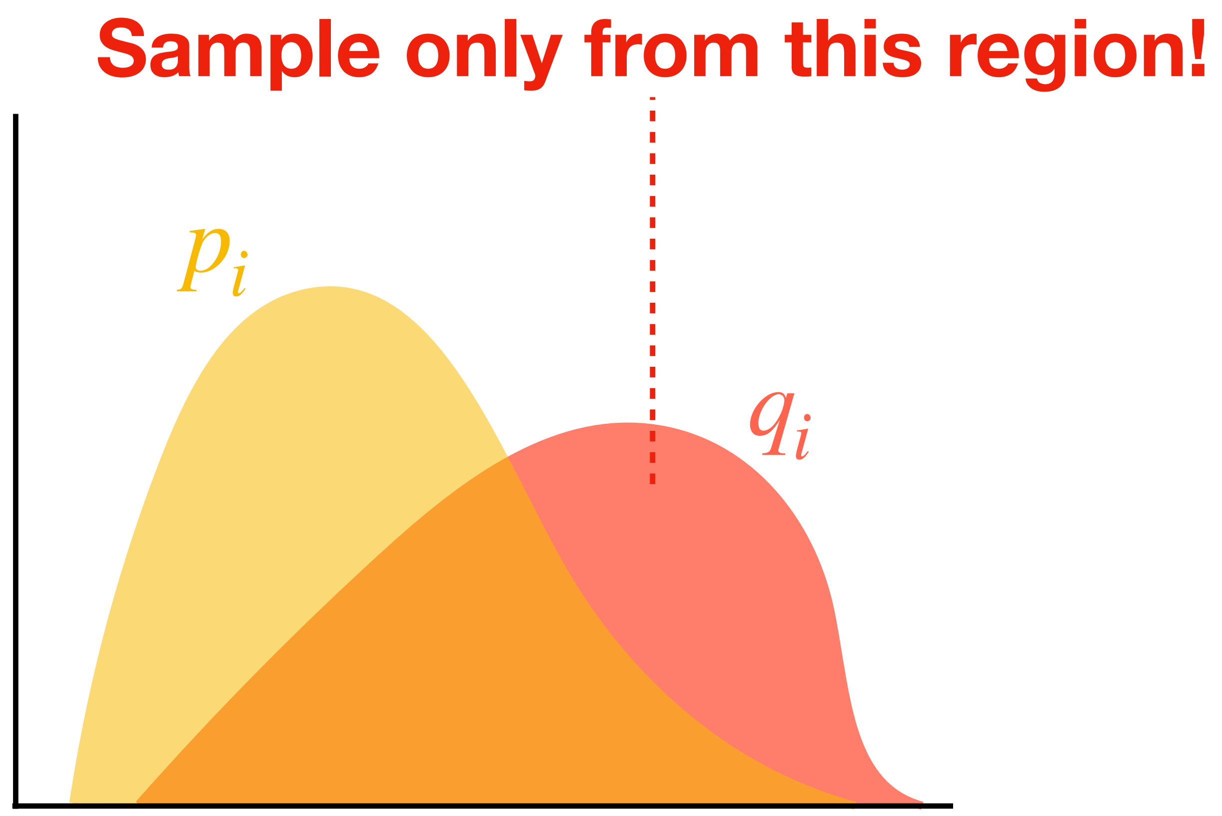
	Token	y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	af	ars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8		0.8

- Case 3: $q_i < p_i$ (reject)

  If $q_i <<< p_i$, we likely would have rejected it.

  In this case, we sample a new token from target model.

  - Specifically, we sample from $(q_i - p_i)_+$



Speeding-up generation: Speculative Sampling

- **Speculative sampling** uses idea of rejection sampling.
  - To sample from a easy-to-sample distribution p (small LM), in order to approximate sampling from a more complex distribution q (large LM).

- Using 4B LM as a draft model and 70B LM as a target model, we get **2~2.5x faster decoding speed** with negligible performance difference!

- Considerations before use
  - M_p and M_q should be pre-trained with the same tokenization scheme!
    (e.g., GPT-2 and GPT- 3 would work, but not GPT-3 and LLaMa-7B)
  - Hardware config matters: If you have 100 GPUs, running large model can actually be faster
    (rather than waiting for a small draft model that only takes up 10 GPU… => *GPU utilization bottleneck*, see page 5-6 in Chen et al.)

Decoding: 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