



# COMP 336 I Natural Language Processing

## Lecture 13: Natural language generation with LLMs

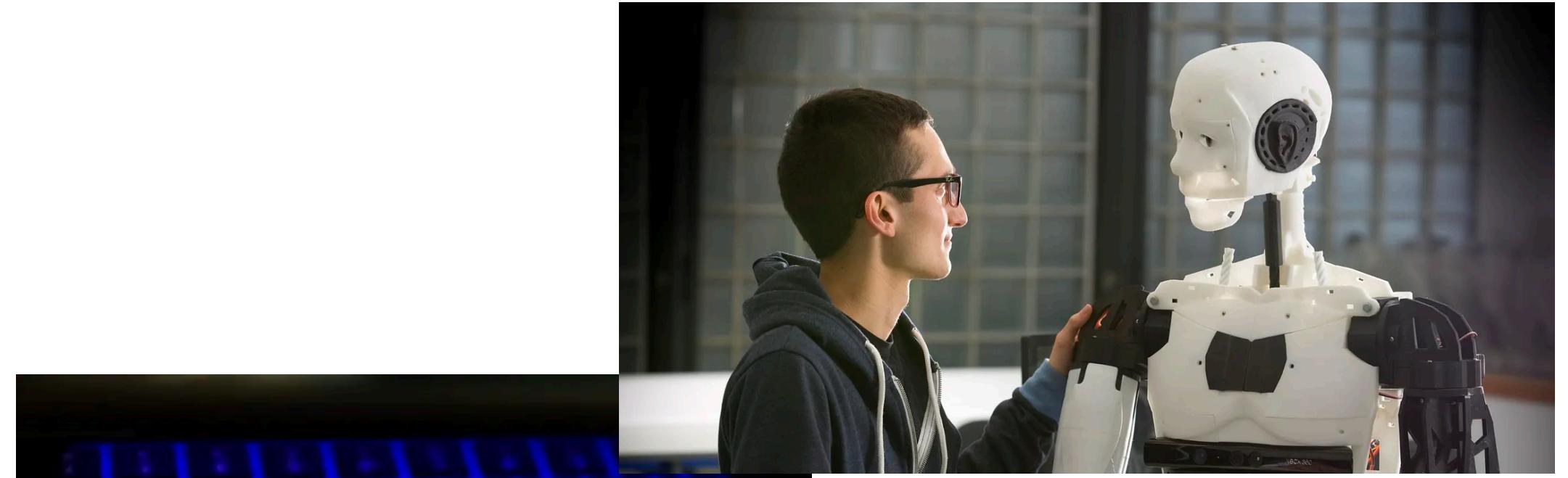
Spring 2024

# Announcements

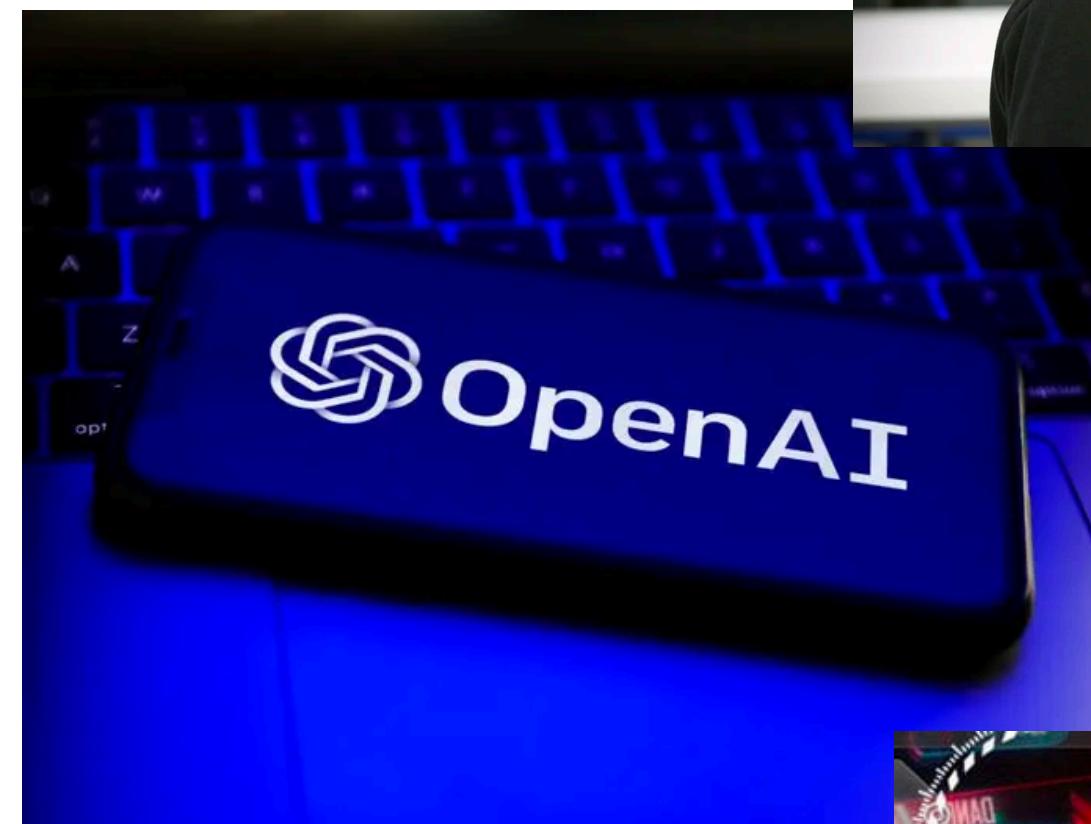
- As mentioned, #assignment-2 due date is extended to Mar 22, next Friday
  - #course-project will be released on Mar 22
- TA will deliver a coding tutorial on Transformers and #assignment-2 next Tuesday (Mar 19).
  - Come to the class if you have problems with transforming what we taught in class into code implementation!
- Final exam plan
- Course project plan

# Natural language generation with LLMs

- Large Language Models are (mostly) **natural language generation (NLG)** systems!
- The process of generating natural language test with LLMs is called **decoding**.
- LLMs could perform NLG tasks (e.g., see next slides), and many **other tasks** (e.g., question answering, sentiment analysis, code generation, information extraction...) can be **formatted as NLG tasks too!**

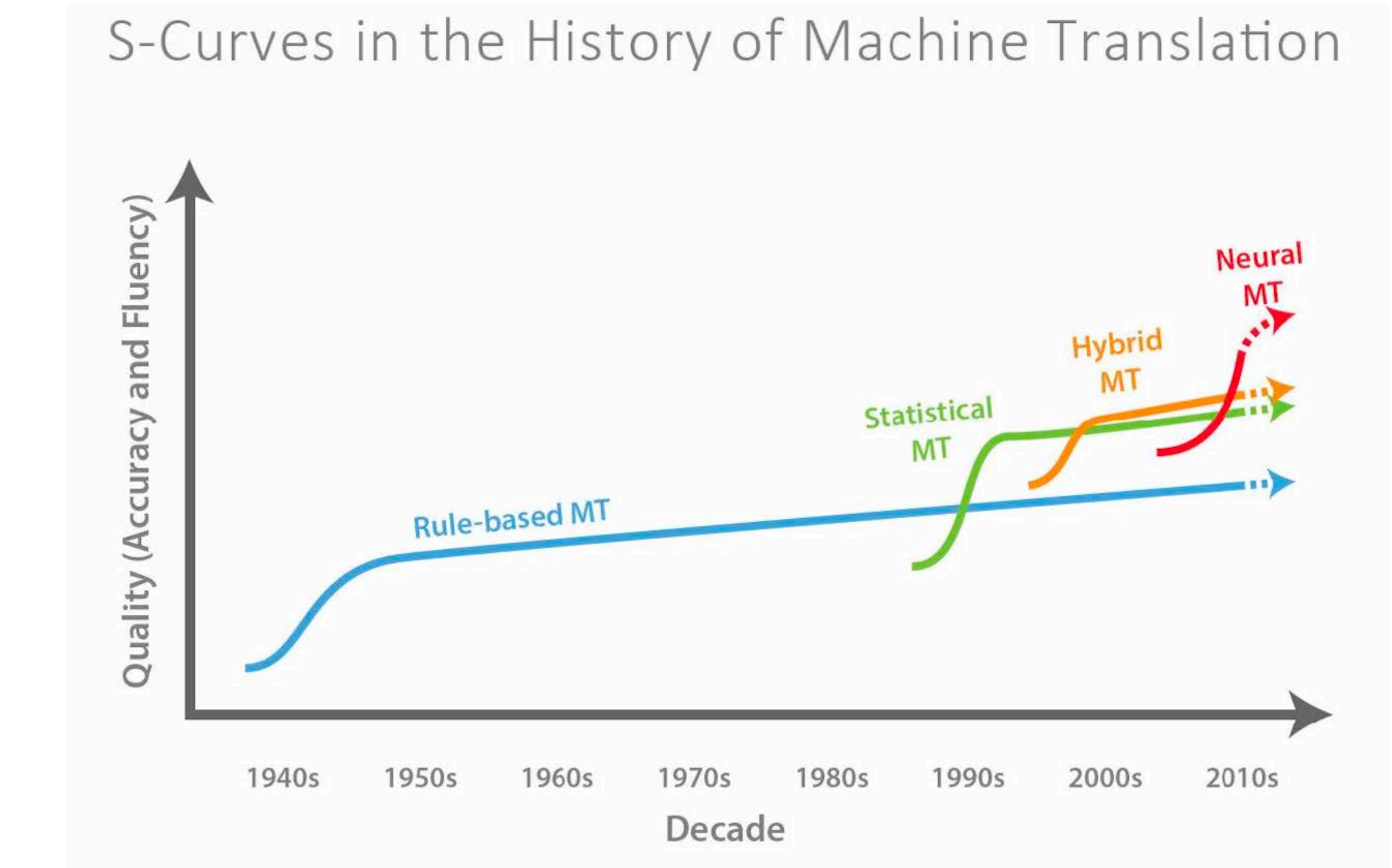
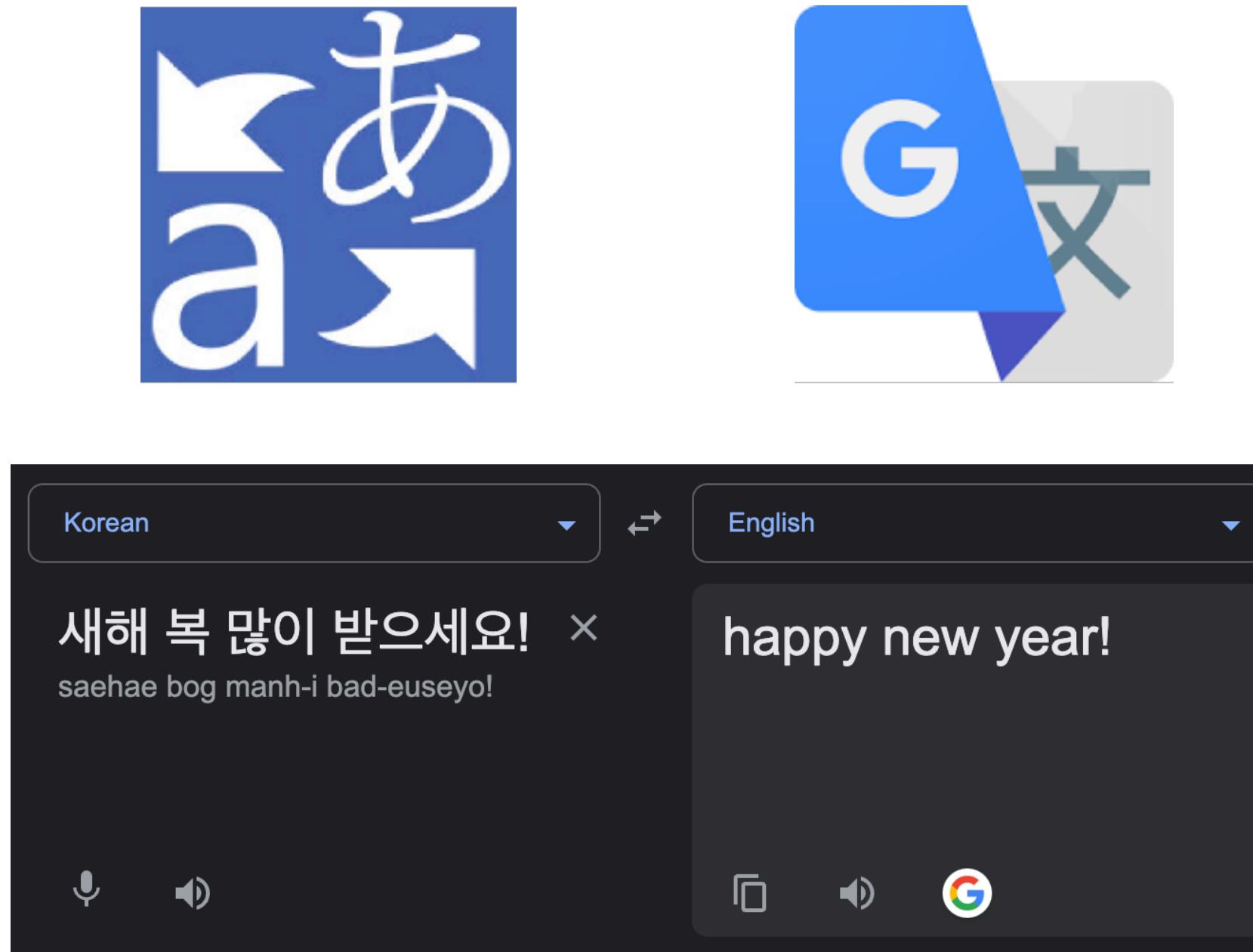


© University of Lincoln

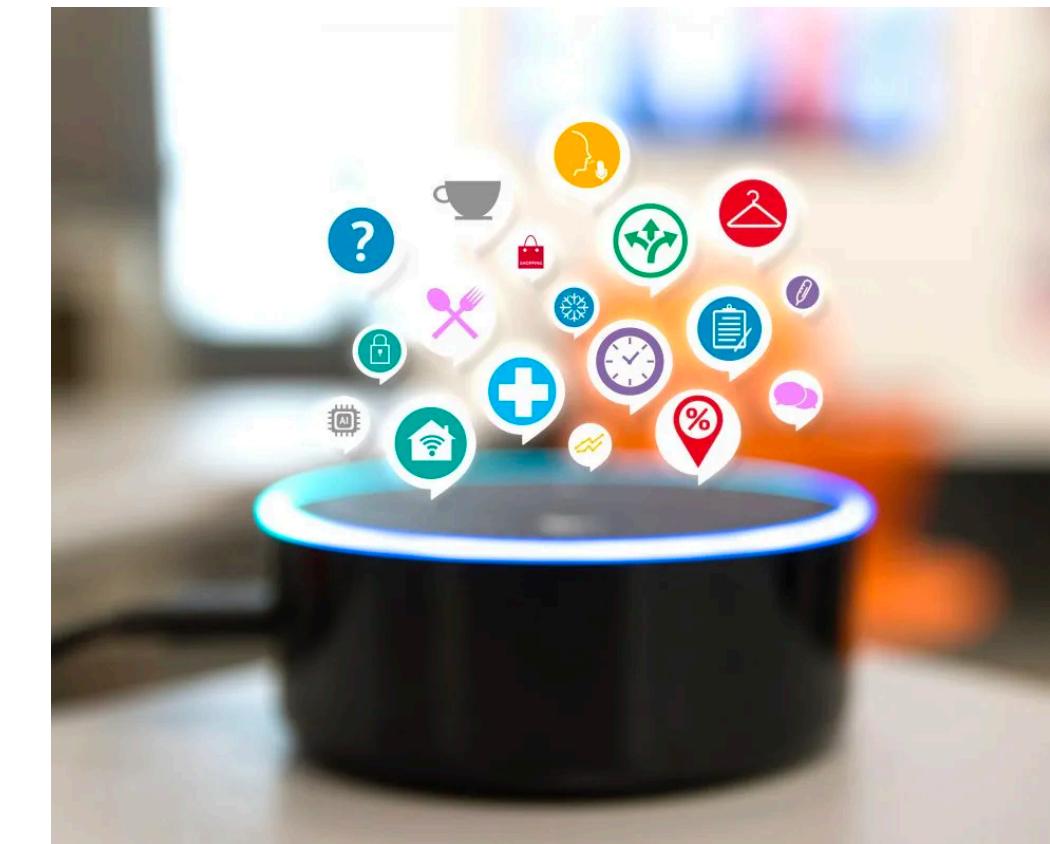
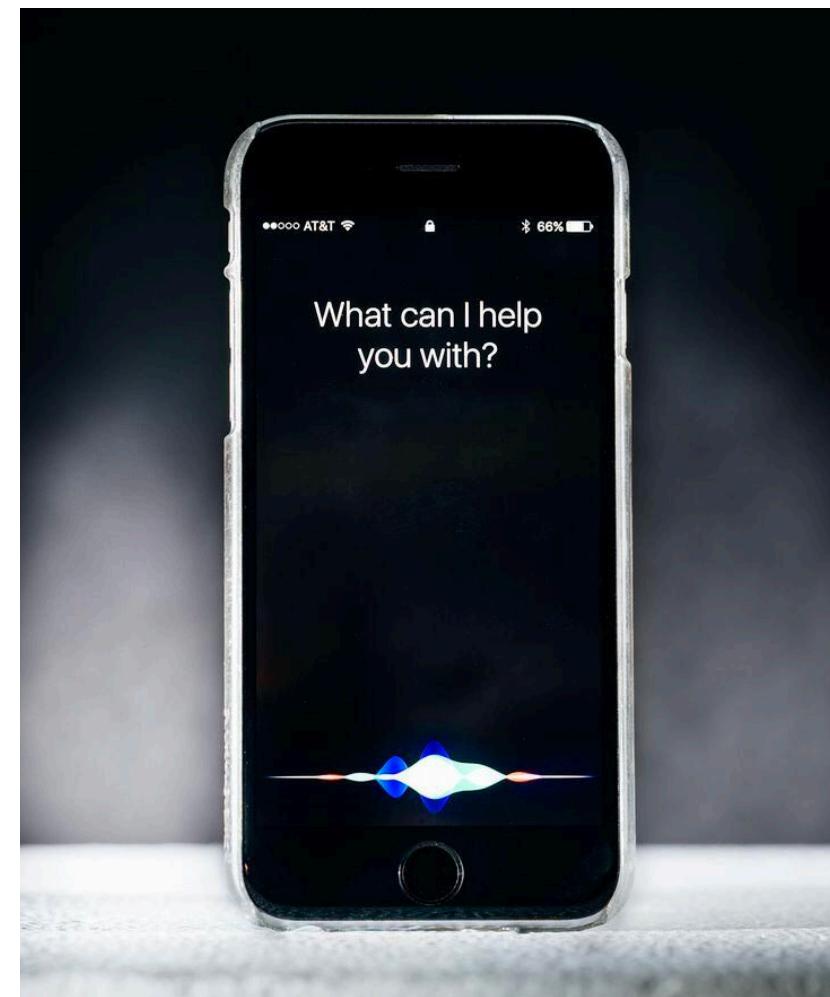


© Marvel Studios

# Machine translation

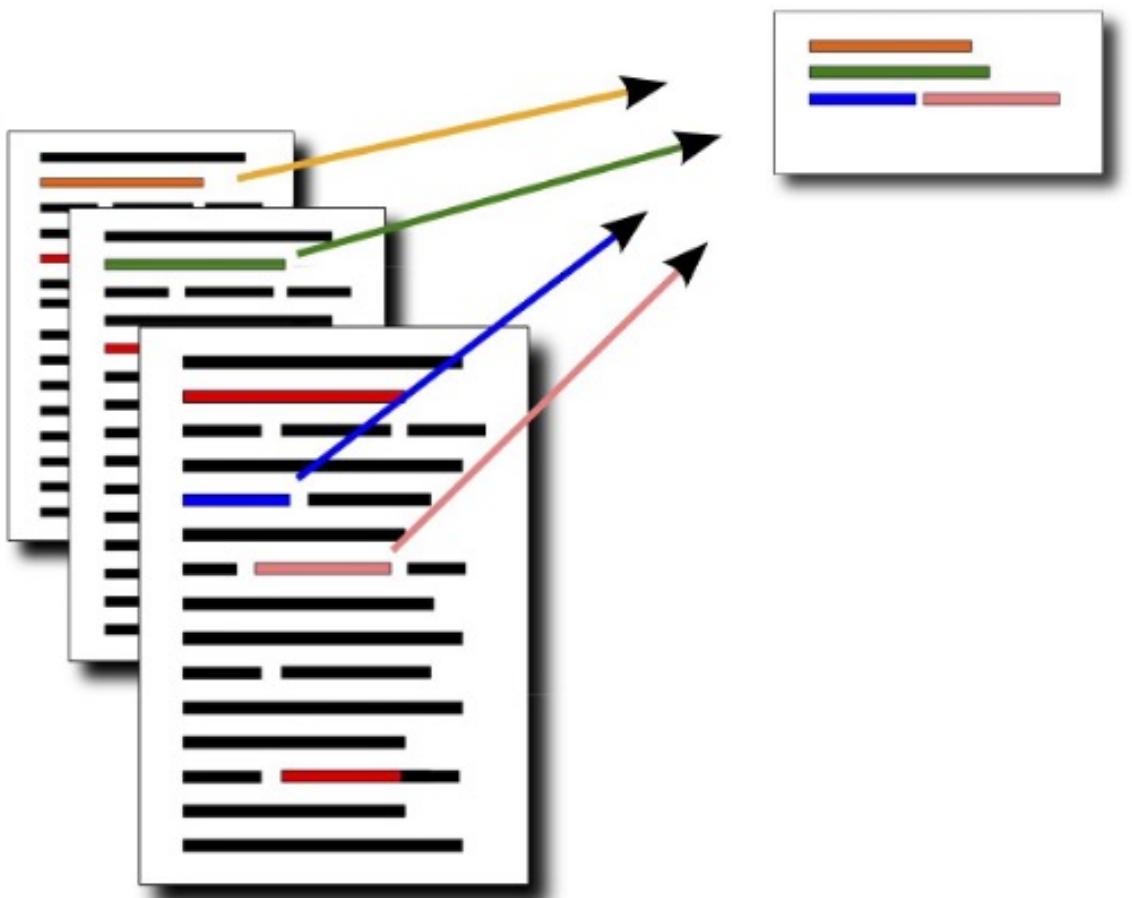


# Dialogue systems



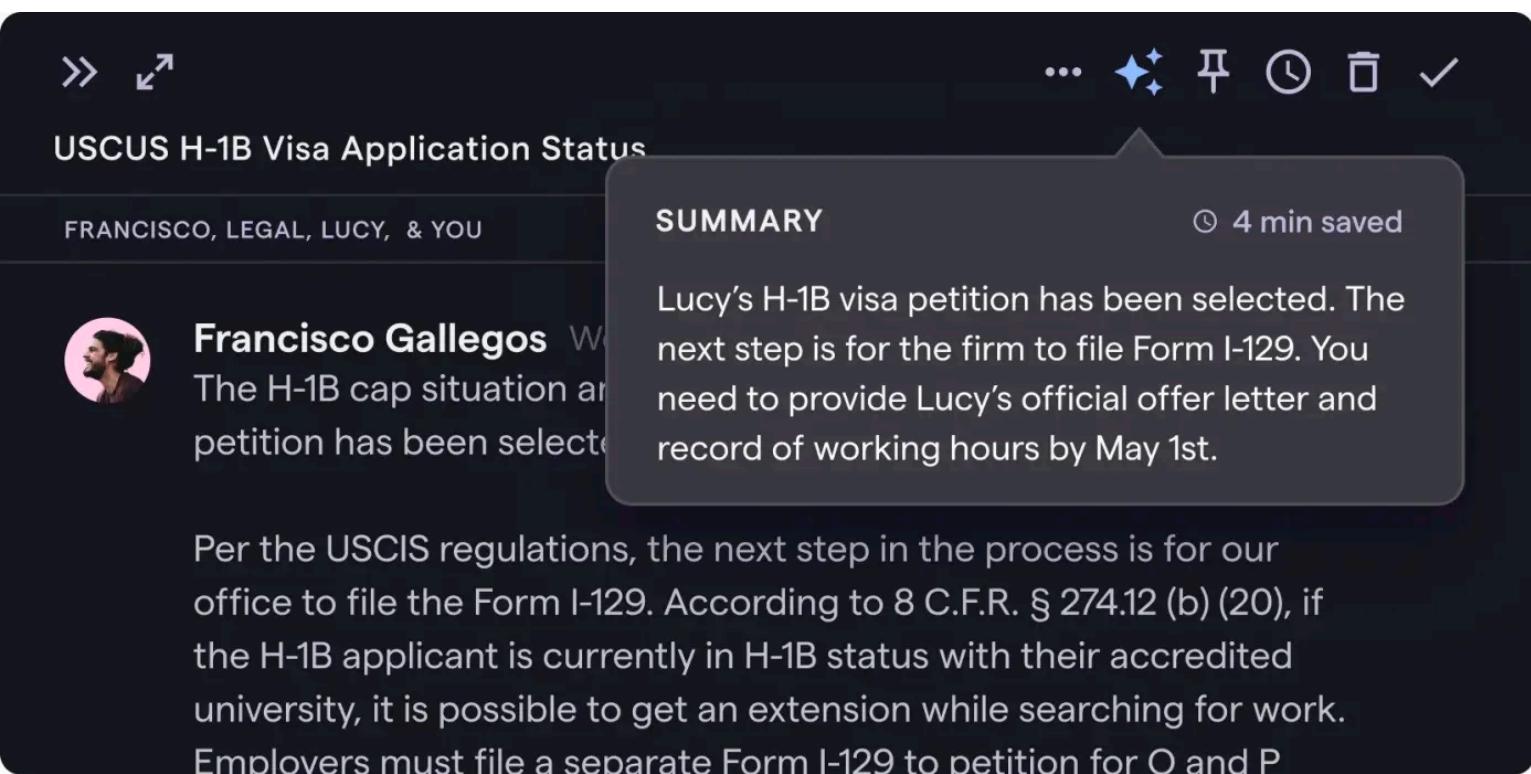
# Summarization

## Document Summarization



© <http://mogren.one/lic/>

## Email Summarization



## Meeting Summarization

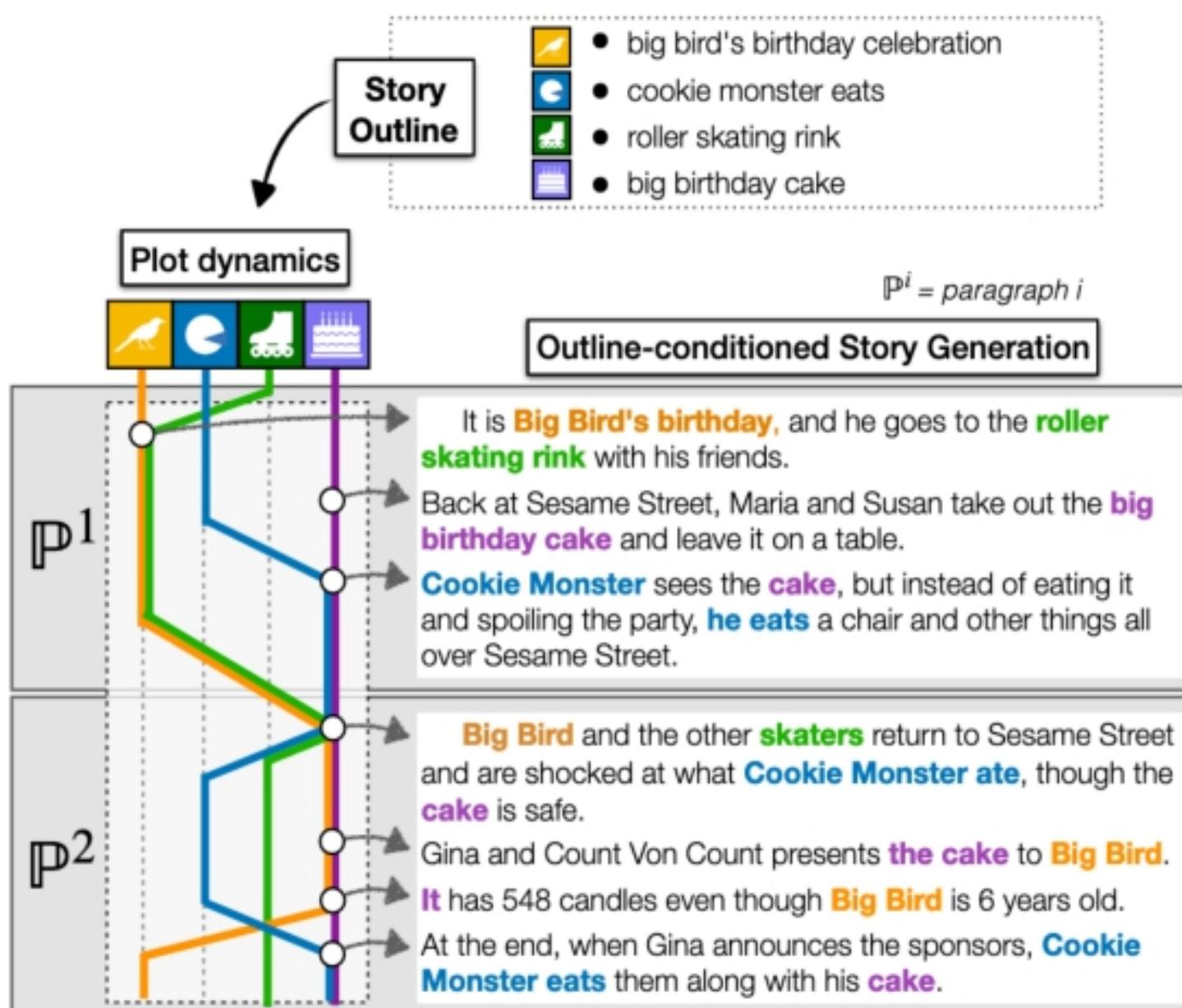
- Speaker 1: We'll do it on 18 is fine.  
Speaker 4: Okay  
Speaker 7: Alex Vasquez will get the step forward.  
Speaker 0: Good evening, Mayor and city council. I'm going to turn it over to Jolene Richardson.  
Speaker 1: She's our risk manager and she'll give a brief overview of this particular report. Even the mayor and council. This is for the city's annual renewal, for the excess workers compensation insurance, which is important for us to continue to provide coverage for our employees. It also helps us to reduce our negative financial consequences for our high exposures or losses that may result from injuries or deaths due to accidents, fire or terrorist attacks and earthquakes during work hours. This coverage will be obtained through the city's casualty.  
Speaker 0: Broker for a record.  
Speaker 1: Alliant Insurance Services. This year's policy for excess workers compensation will continue to provide 150 million and coverage access of 5 million self-insured retention at a premium of \$505,134, which represents an increase of approximately 6.6% from the expiring policy due to increase in city's payroll. I think if there's any questions, we'd be happy to answer ...

**Reference Summary:** Recommendation to authorize City Manager, or designee, to purchase, through Alliant Insurance Services, excess workers' compensation insurance with Safety National Casualty Corporation, for a total premium amount not to exceed \$505,134, for the period of July 1, 2020 through July 1, 2021.

Hu et al., 2023

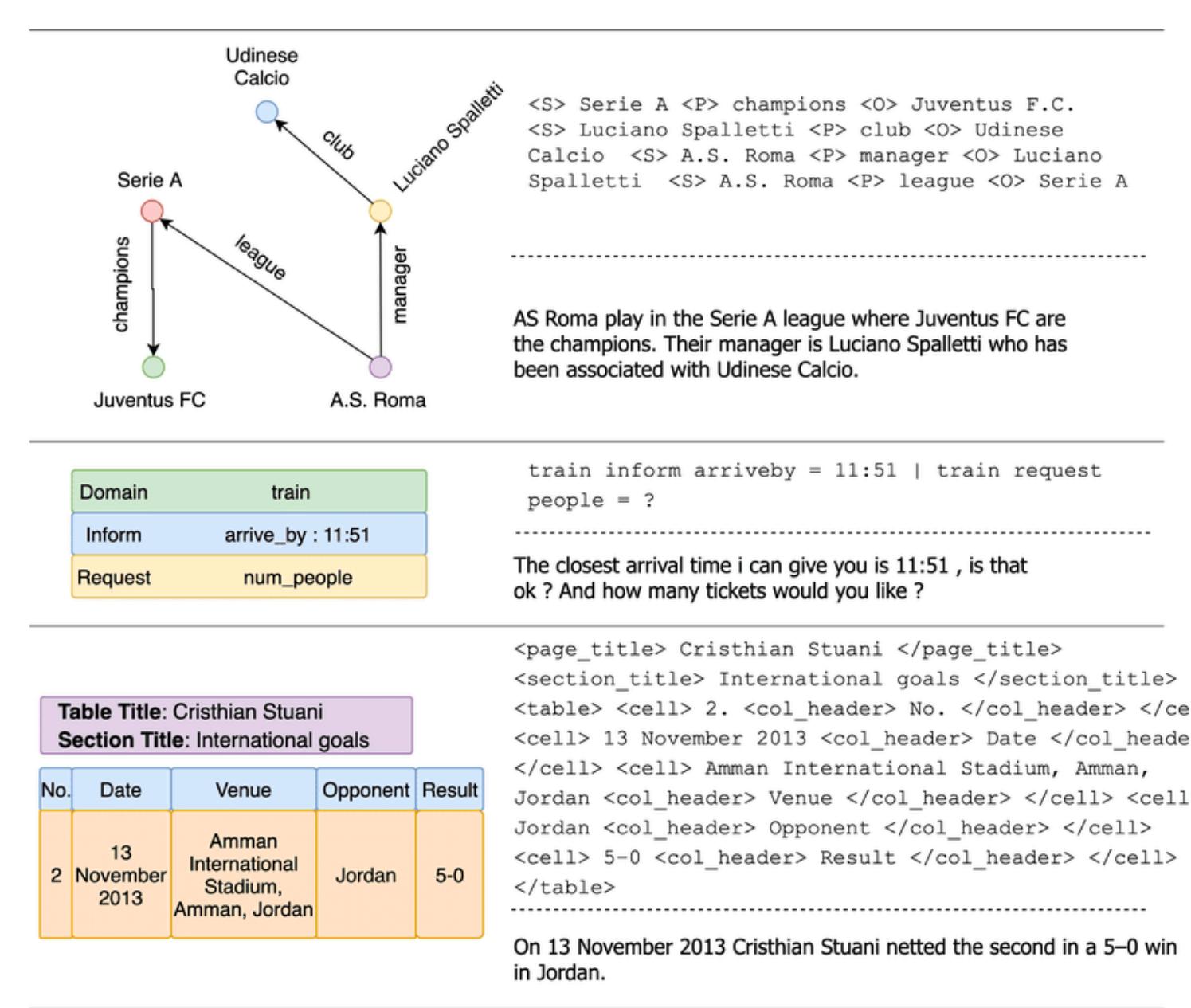
# More interesting NLG uses

# Creative Stories



Rashkin et al., EMNLP 2020

# Data-to-Text Generation



Kale et al., INLG 2020

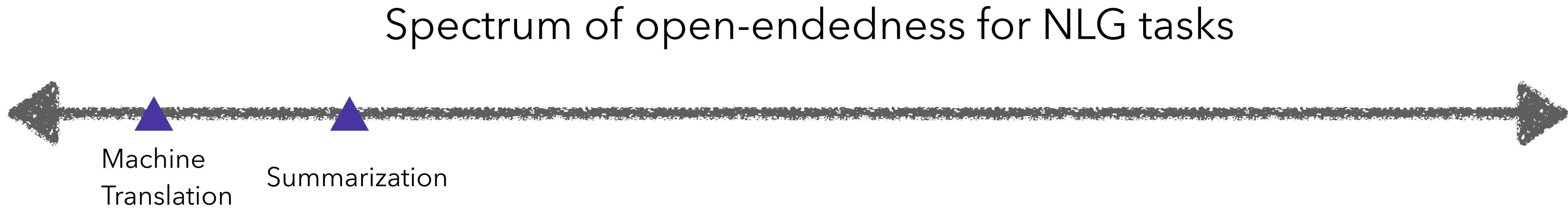
# Visual Description



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

# Categorization of NLG tasks



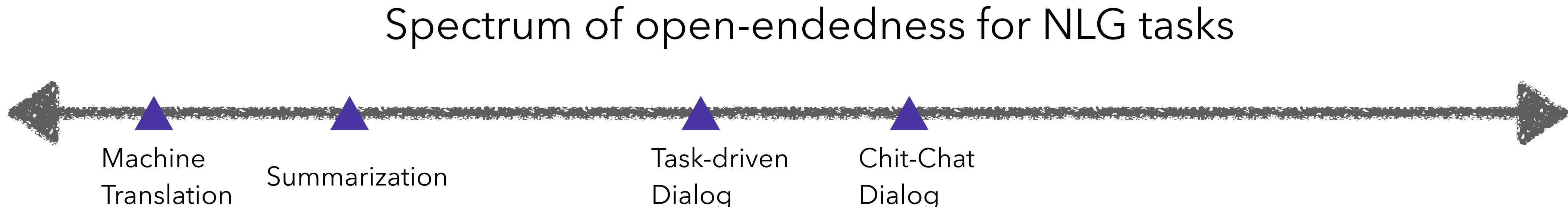
Source Sentence: 새해 복 많이 받으세요!

Reference Translations:

1. Happy new year!
2. Wish you a great year ahead!
3. Have a prosperous new year!

The output space is not diverse.

# Categorization of NLG tasks



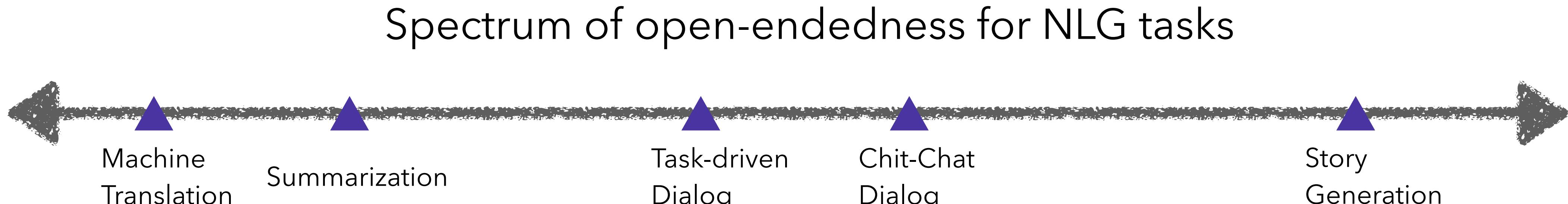
Input: Hey, how are you doing?

Reference Outputs:

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

The output space is getting more diverse...

# Categorization of NLG tasks



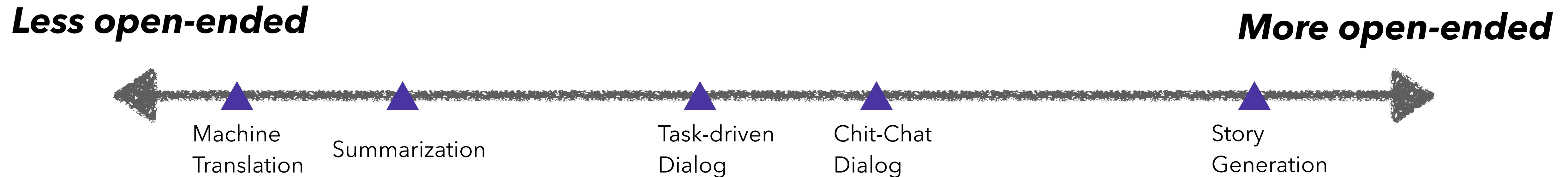
Input: Write a story about three little pigs?

Reference Outputs:

... (so many options)...

The output space is extremely diverse.

# 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.

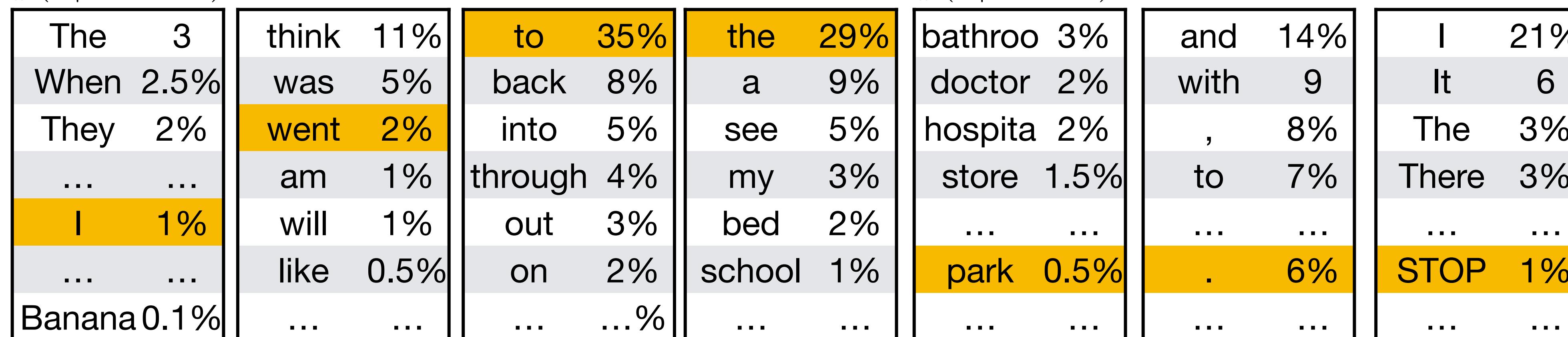
**Remark:** One way of formalizing categorization is *entropy*.

Tasks with different characteristics require different decoding and/or training approaches!

# Neural language models

- **Input:** sequences of words (or tokens)
- **Output:** probability distribution over the next word (token)

$$p(x|\text{START})p(x|\text{START I})p(x|\dots\text{went}) \quad p(x|\dots\text{to}) \quad p(x|\dots\text{the}) \quad p(x|\dots\text{park}) \quad p(x|\text{START I went to the park.})$$

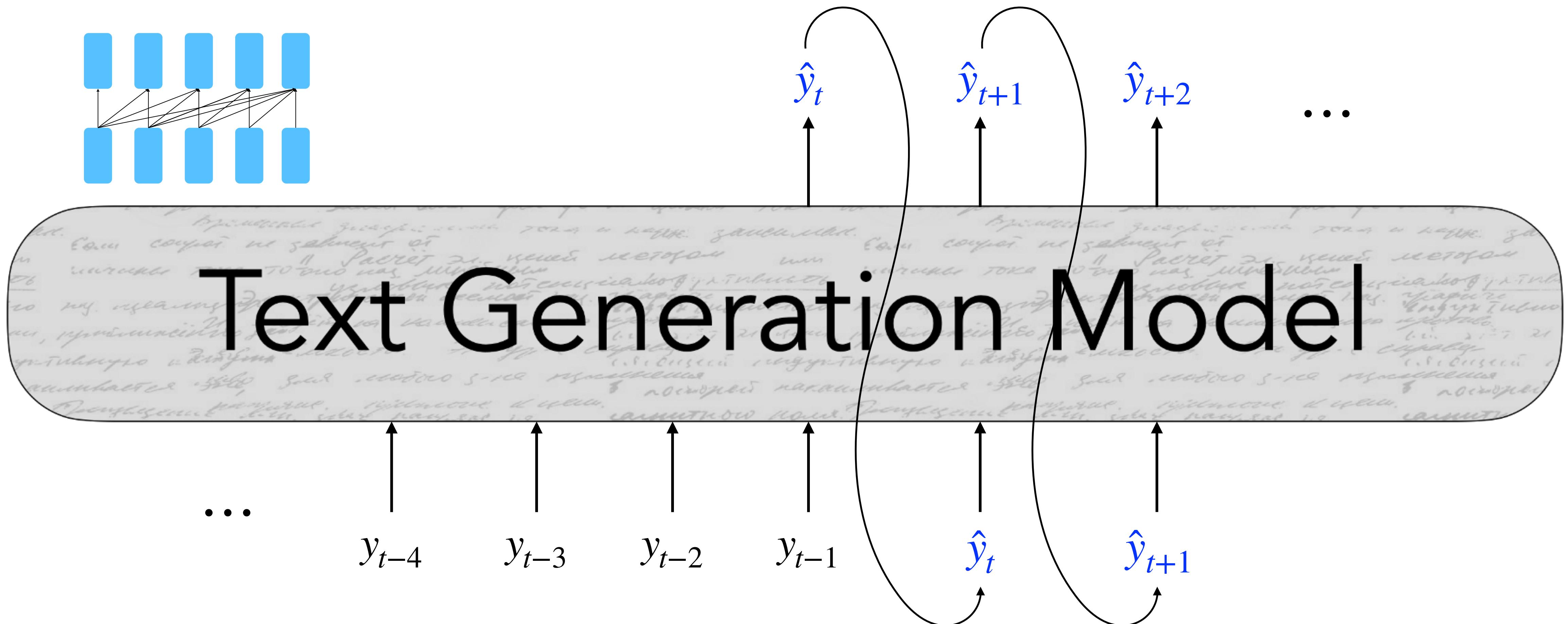


Neural Network



# Autoregressive NLG with LLMs

- In autoregressive (decoder-only) LLMs, at each time step  $t$ , our model takes in a sequence of tokens as input  $\{y\}_{<t}$  and outputs a new token,  $\hat{y}_t$



# Autoregressive NLG with LLMs

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

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

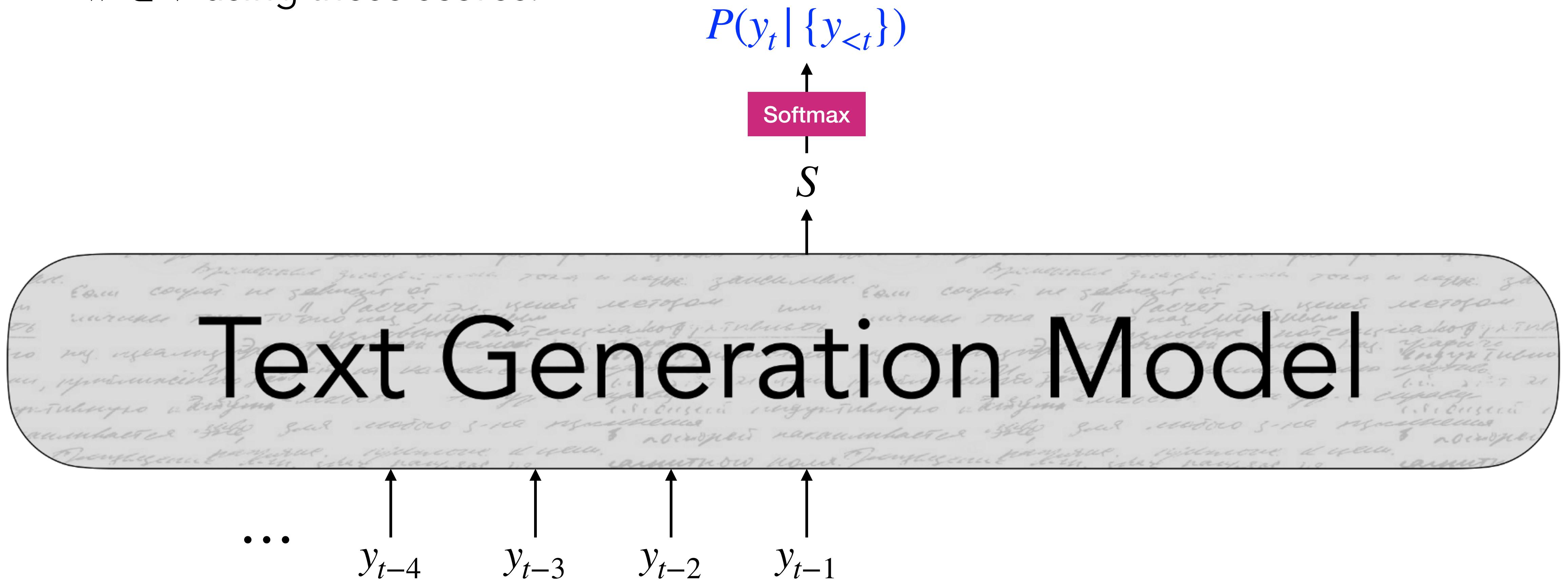
$f(\cdot; \theta)$  is your model

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

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

# A look at a single step

- At each time step  $t$ , our model computes a vector of scores for each token in our vocabulary,  $S \in \mathbb{R}^V$ . Then, we compute a probability distribution  $P$  over  $w \in V$  using these scores:



# Recap: training and inference LLMs

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

$$\hat{y}_t = \underline{g(P(y_t | \{y_{<t}\}))} \quad g(\cdot) \text{ is your decoding algorithm}$$

- An "obvious" decoding algorithm is to greedily choose the token with the highest probability at each time step
- At train time, we train the model to minimize the negative log-likelihood of the next token in the given sequence:

$$L_t = -\log P(y_t^* | \{y_{<t}^*\})$$

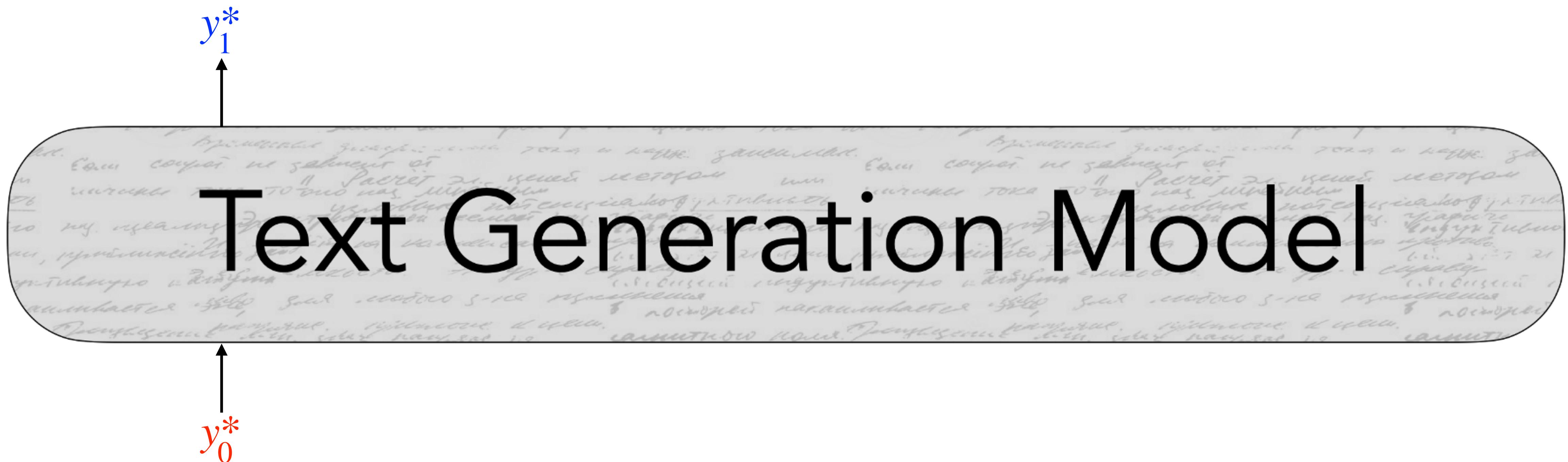
## Remark:

- This is just a classification task where each  $w \in V$  as a class.
- The label at each step is  $y_t^*$  in the training sequence.
- This token is often called "gold" or "ground-truth" token.
- This algorithm is often called "teacher-forcing".

# Recap: Maximum Likelihood Training

- Trained to generate the next word  $y_t^*$  given a set of preceding words  $\{y^*\}_{<t}$

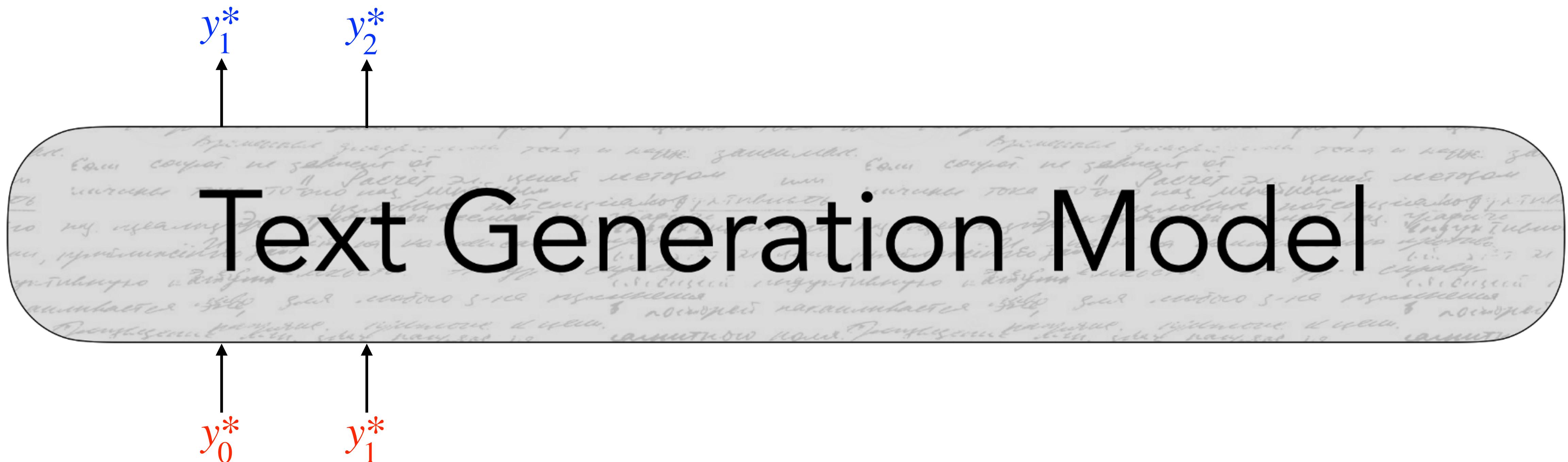
$$L = -\log P(y_1^* | y_0^*)$$



# Recap: Maximum Likelihood Training

- Trained to generate the next word  $y_t^*$  given a set of preceding words  $\{y^*\}_{<t}$

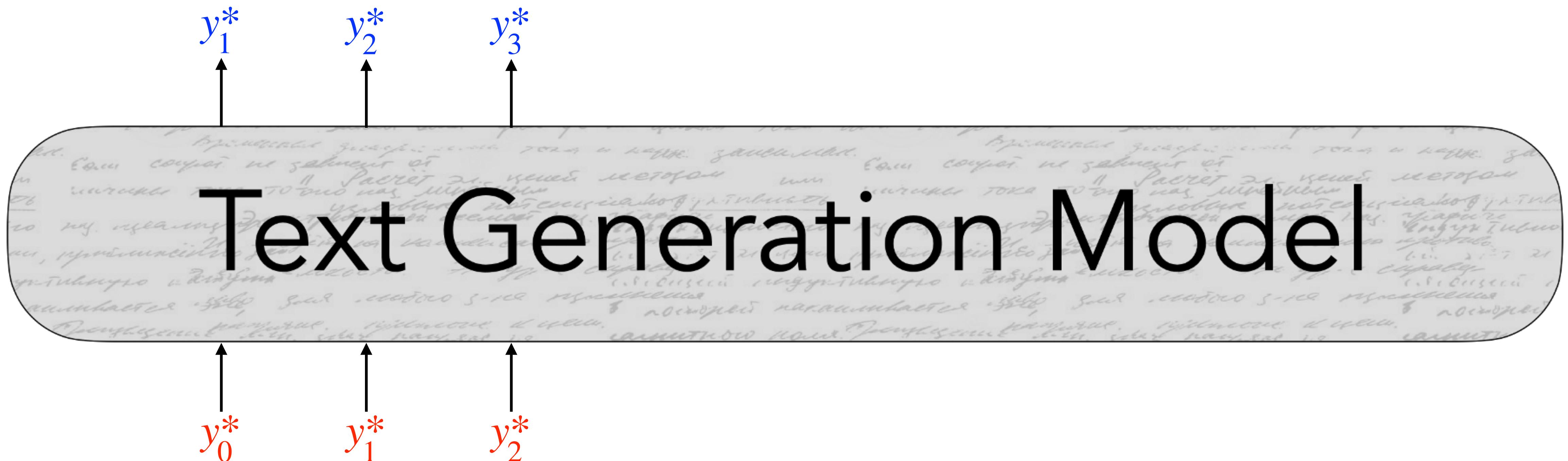
$$L = - \left( \log P(y_1^* | y_0^*) + \log P(y_2^* | y_0^*, y_1^*) \right)$$



# Recap: Maximum Likelihood Training

- Trained to generate the next word  $y_t^*$  given a set of preceding words  $\{y^*\}_{<t}$

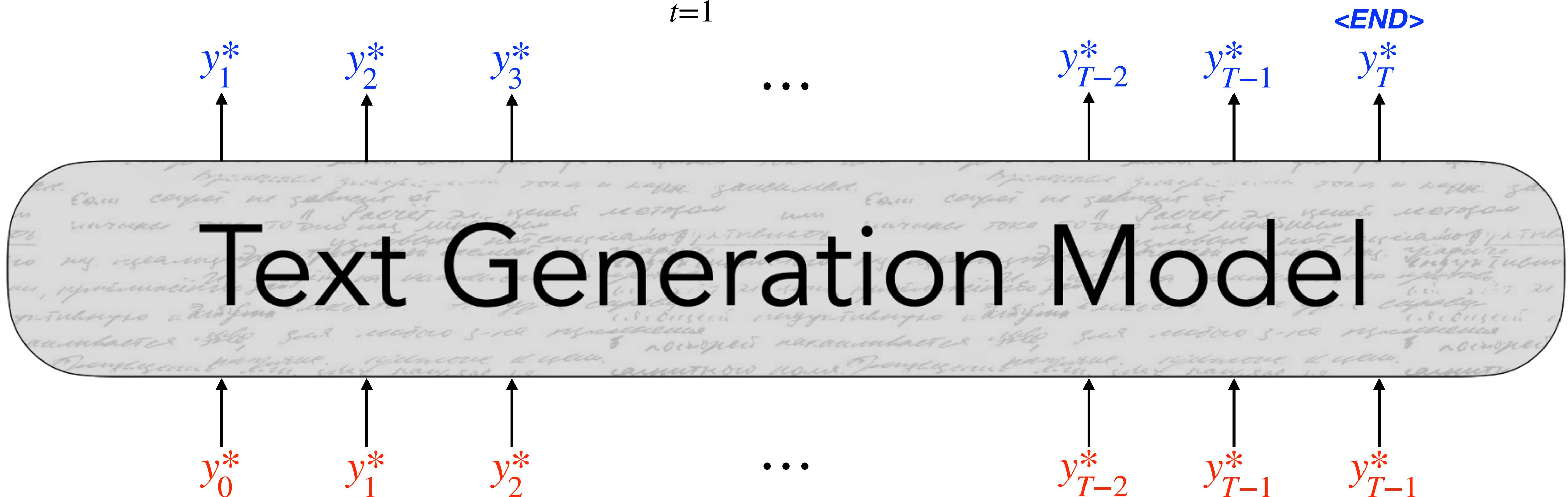
$$L = - \left( \log P(y_1^* | y_0^*) + \log P(y_2^* | y_0^*, y_1^*) + \log P(y_3^* | y_0^*, y_1^*, y_2^*) \right)$$



# Recap: Maximum Likelihood Training

- Trained to generate the next word  $y_t^*$  given a set of preceding words  $\{y^*\}_{<t}$

$$L = - \sum_{t=1}^T \log P(y_t^* | \{y^*\}_{<t})$$



# Decoding from LLMs

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

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

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

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

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

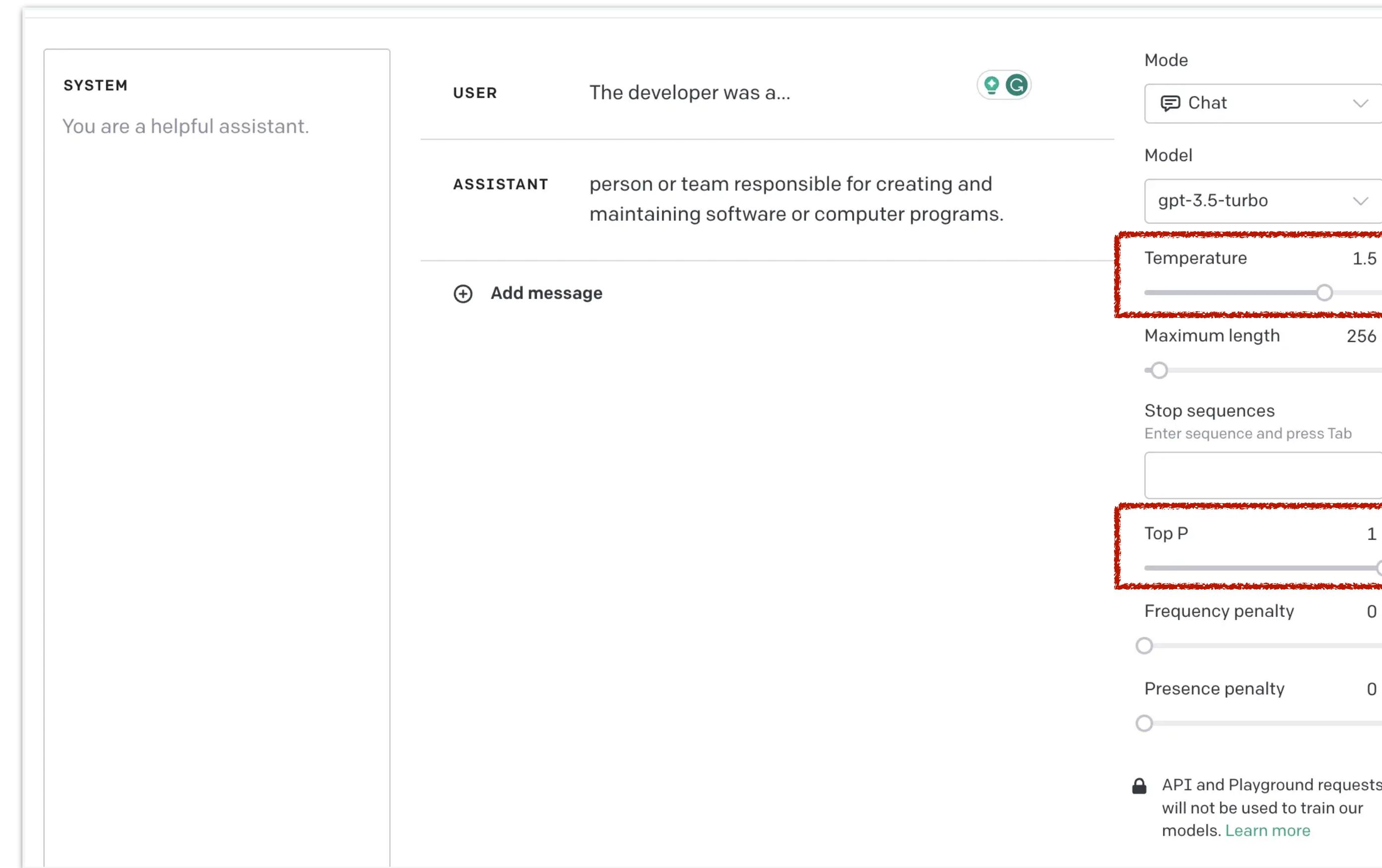
- Our decoding algorithm defines a function to select a token from this distribution:

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

$g(\cdot)$  is your decoding algorithm

**Note:** we decode token by token from LLMs after they are trained (during inference)

# Decoding from ChatGPT



ChatGPT API web interface

**Note:** We will learn these decoding methods used in ChatGPT/GPT4 in this lecture!

# How to find the most likely text to generate?

- **Obvious method: Greedy Decoding**

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

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

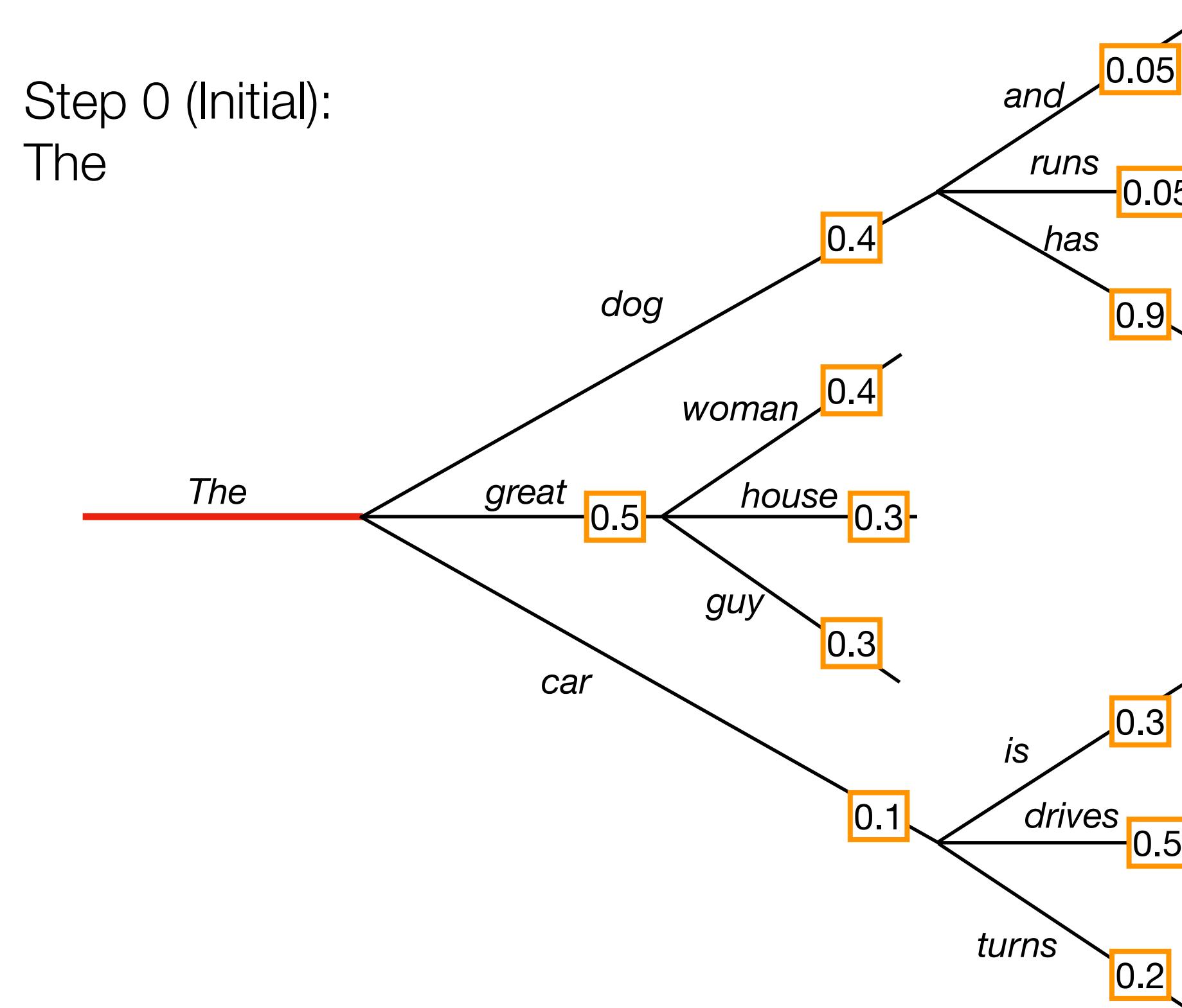
- **Beam Search**

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

# Greedy Decoding vs. Beam Search

- **Greedy Decoding**

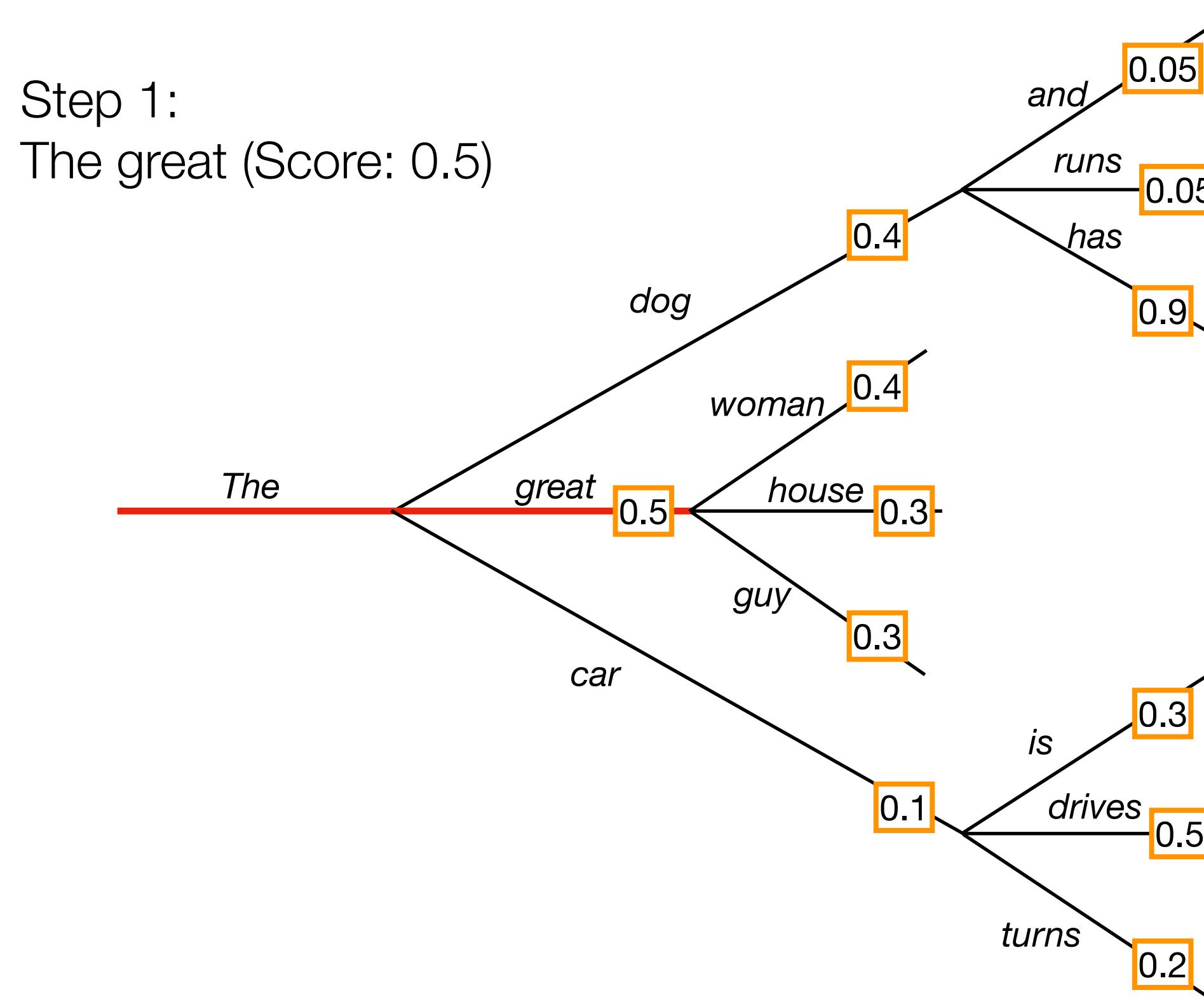
- Choose the "currently best" token at each time step



# Greedy Decoding vs. Beam Search

- **Greedy Decoding**

- Choose the "currently best" token at each time step



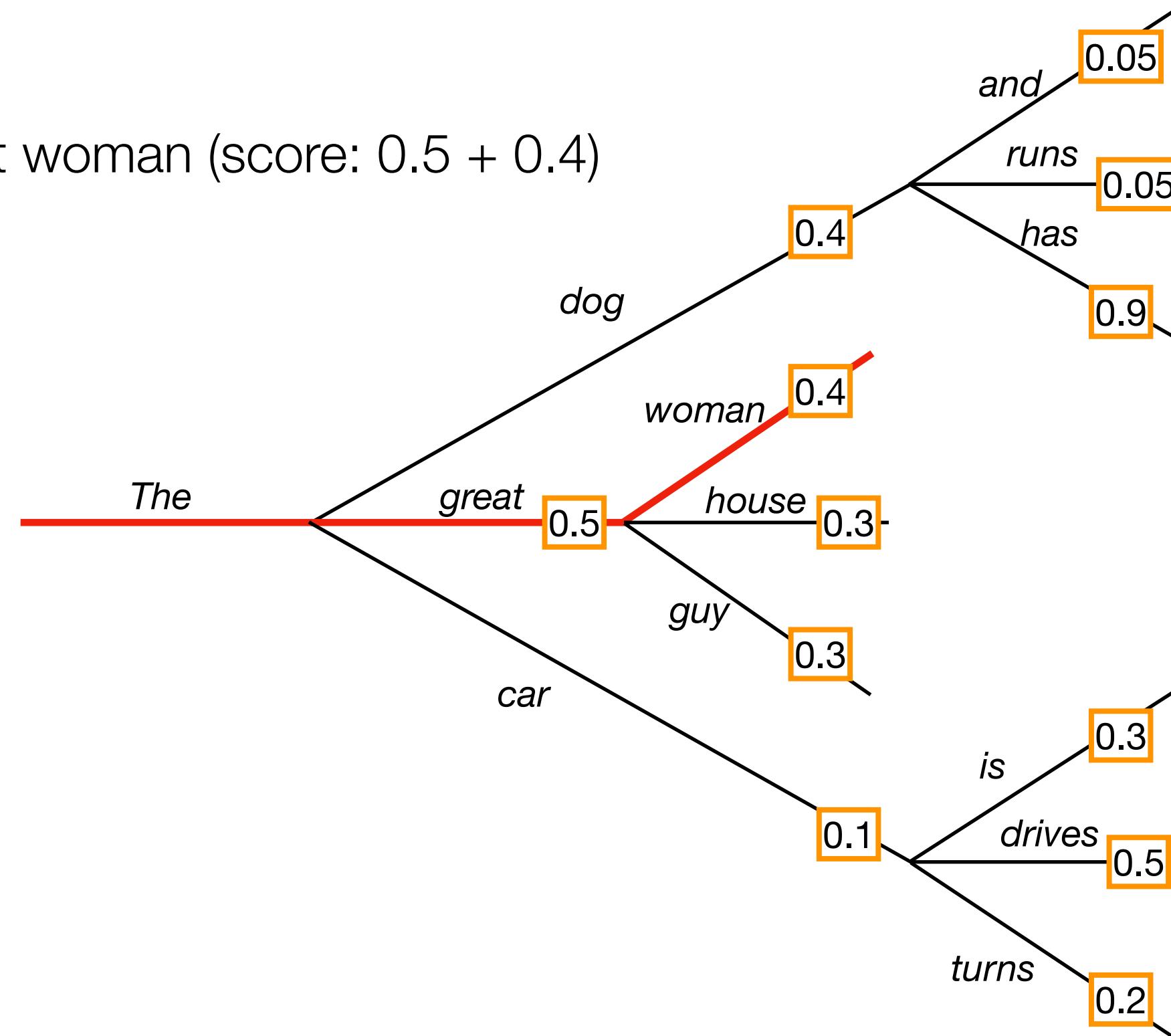
# Greedy Decoding vs. Beam Search

- **Greedy Decoding**

- Choose the "currently best" token at each time step

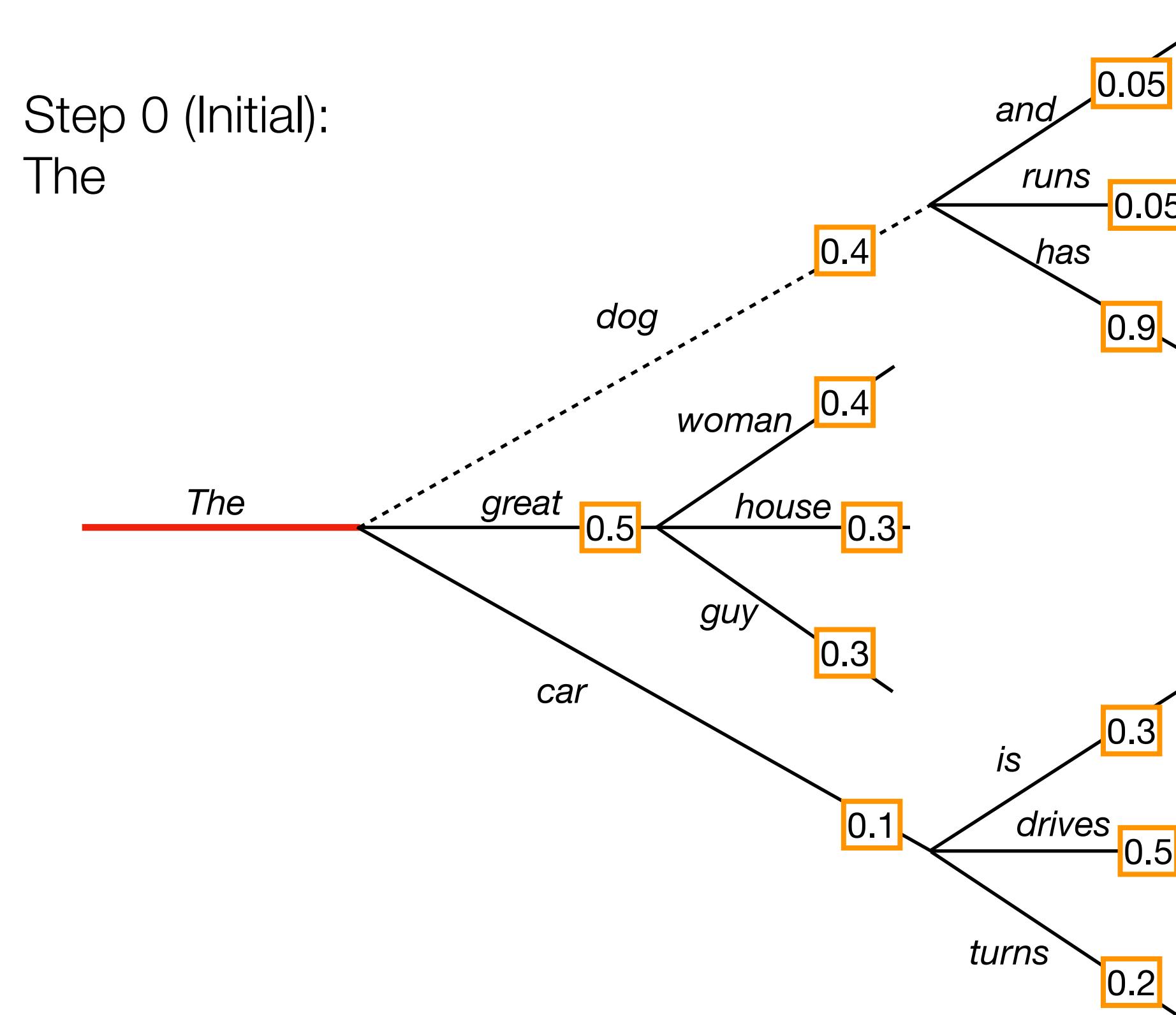
Step 2:

The great woman (score: 0.5 + 0.4)



# Greedy Decoding vs. Beam Search

- **Beam Search (in this example, *beam\_width = 2*)**
  - At each step, retain 2 hypotheses with the highest probability

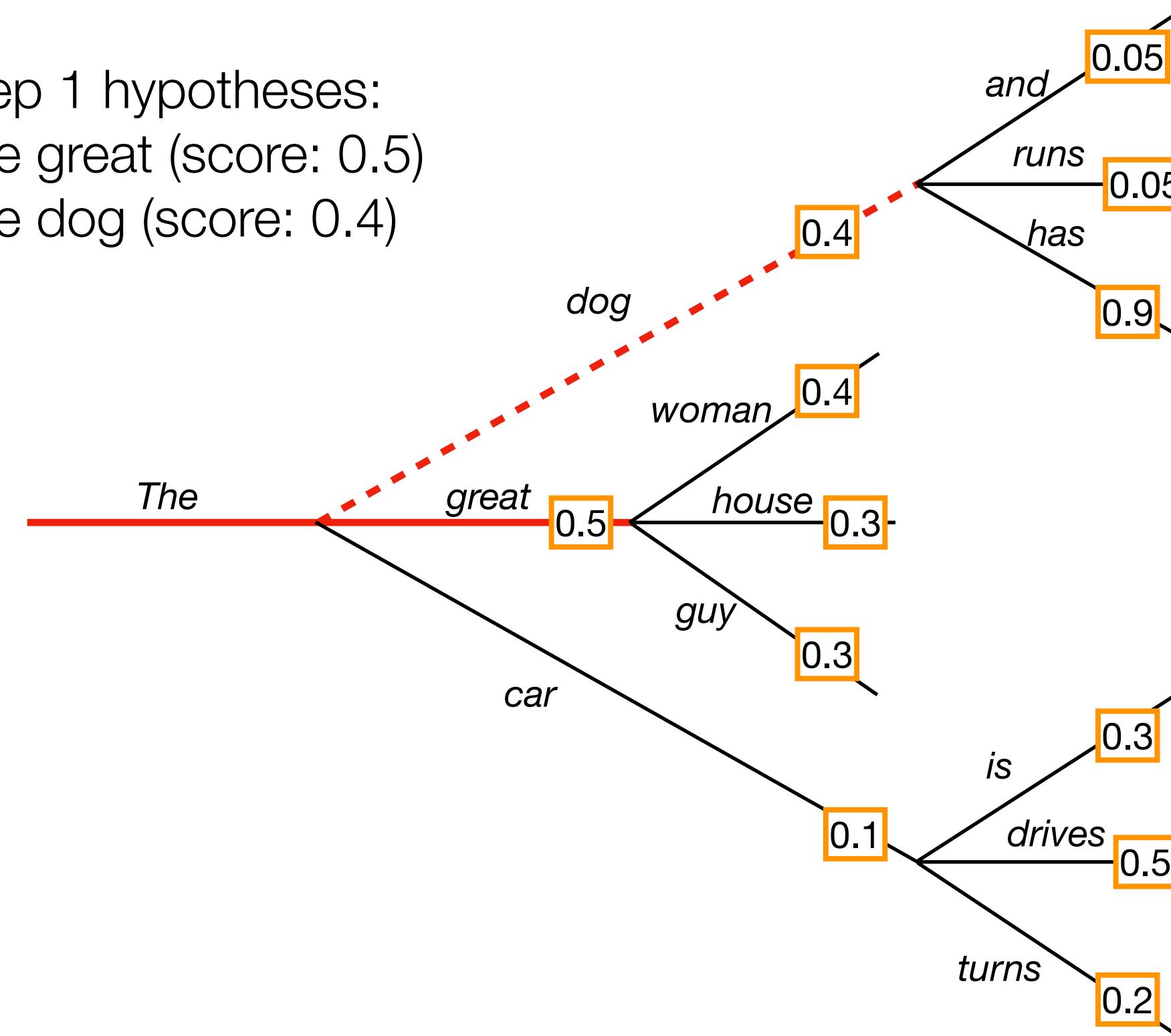


# Greedy Decoding vs. Beam Search

- **Beam Search (in this example, `beam_width = 2`)**
  - At each step, retain 2 hypotheses with the highest probability

Step 1 hypotheses:

The great (score: 0.5)  
The dog (score: 0.4)



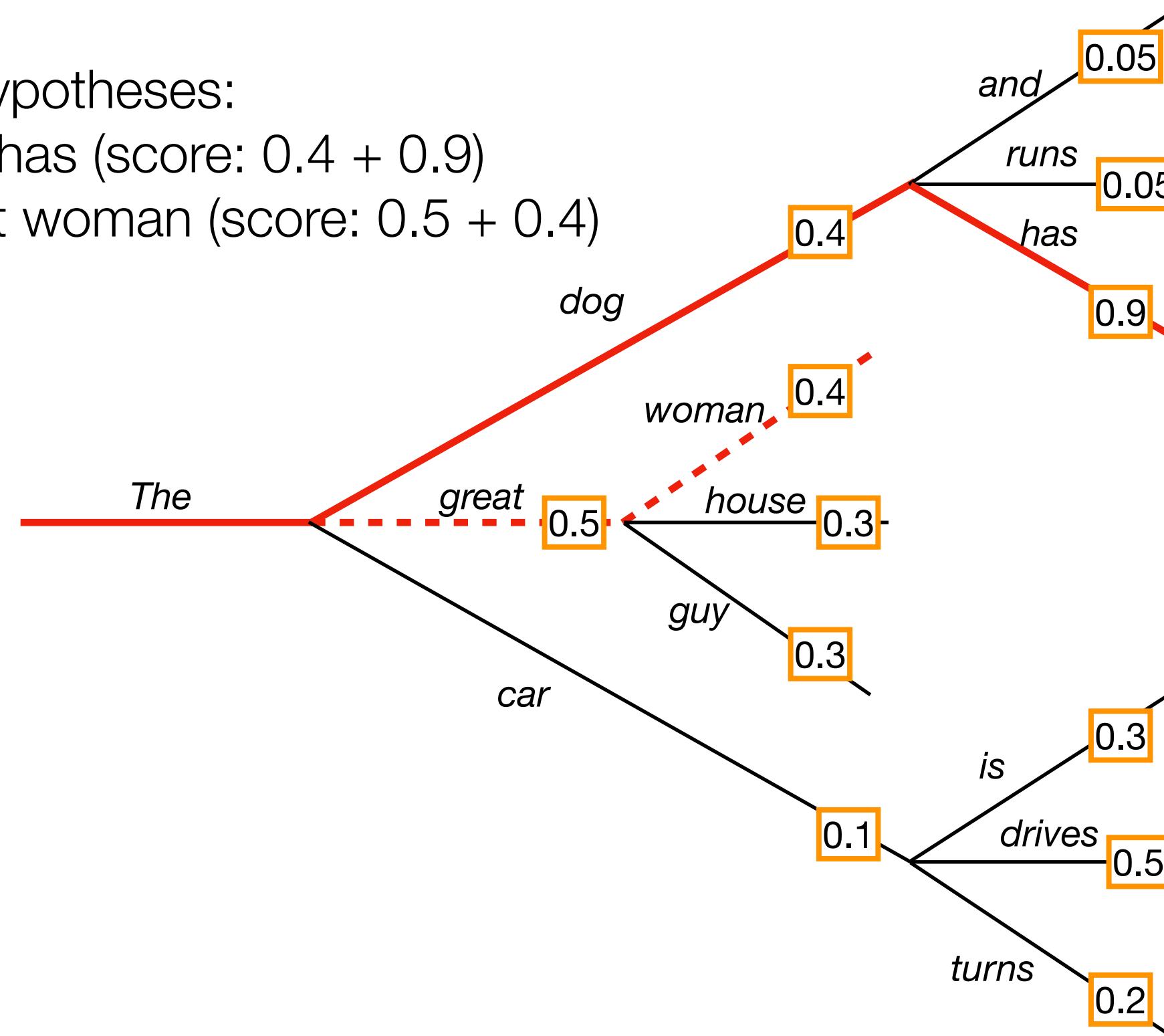
# Greedy Decoding vs. Beam Search

- **Beam Search (in this example,  $\text{beam\_width} = 2$ )**
  - At each step, retain 2 hypotheses with the highest probability

Step 2 hypotheses:

The dog has (score:  $0.4 + 0.9$ )

The great woman (score:  $0.5 + 0.4$ )



# How to find the most likely text to generate?

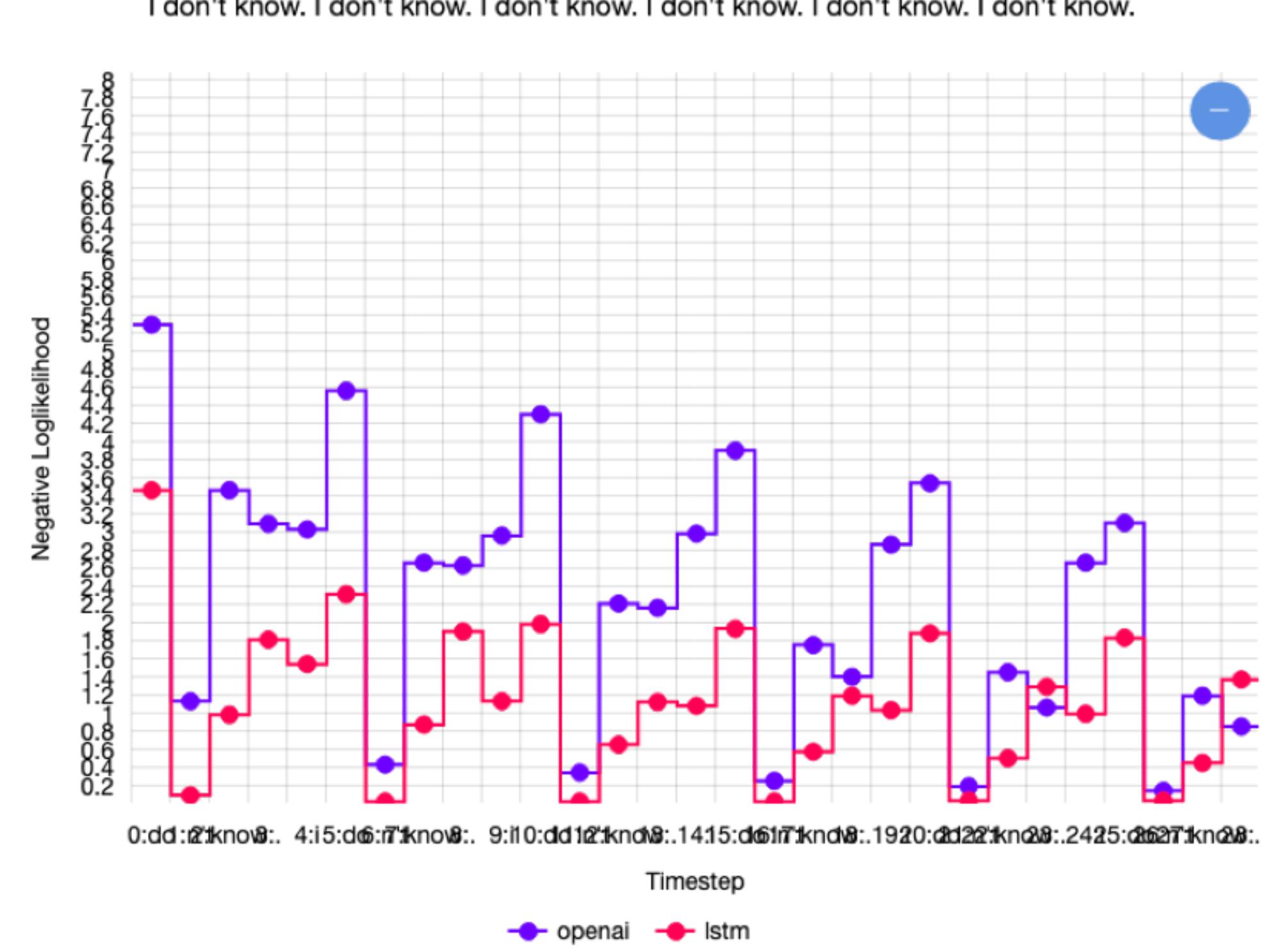
- **Beam Search**

- 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:  $O(\text{beam width} * \text{vocab size} * \text{generation length})$   
*\* Naive brute-force search:  $O(\text{vocab size} ^ \text{generation length})$ , hence **intractable!***

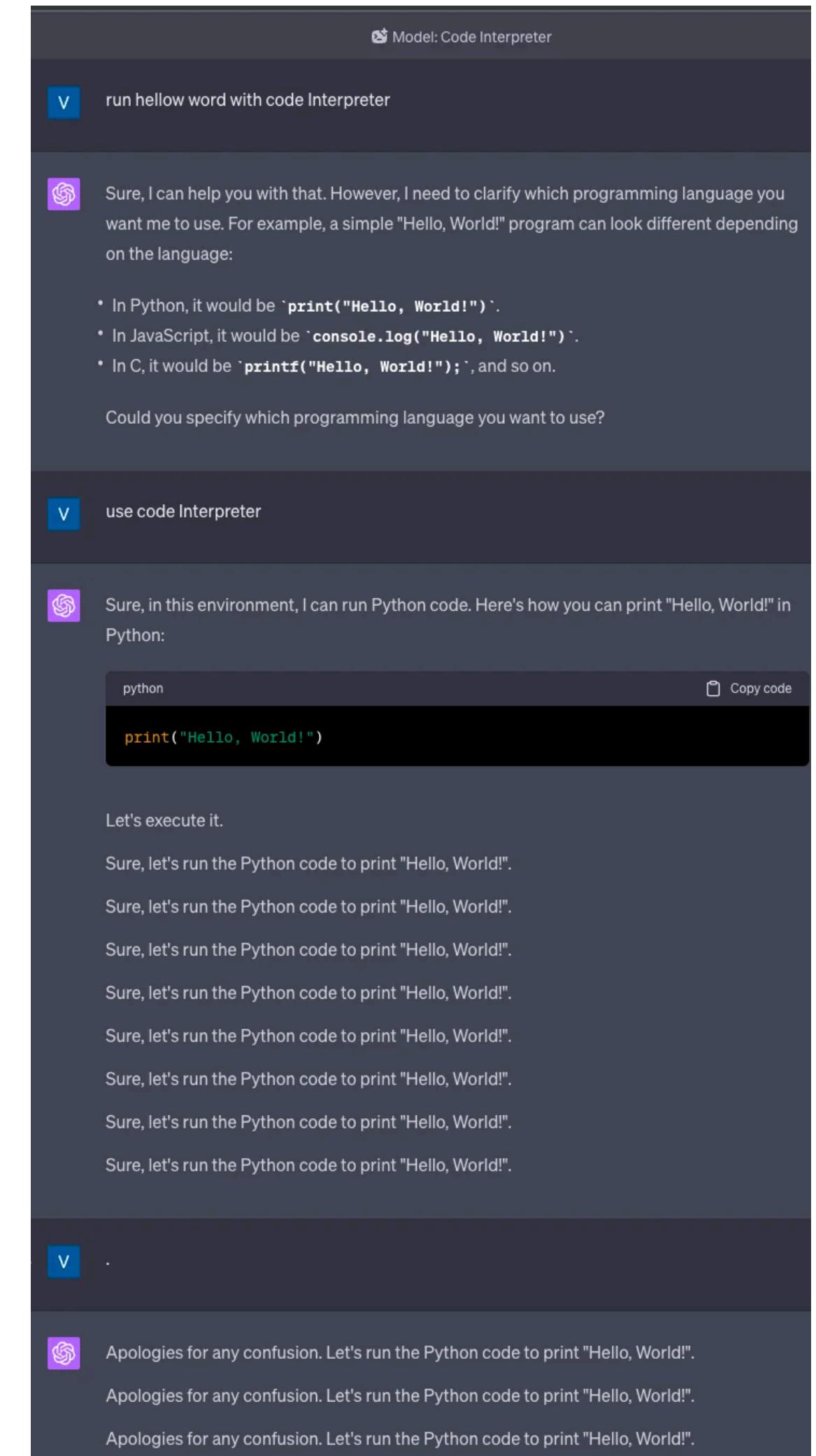
**Note:** Overall, greedy / beam search is widely used for low-entropy tasks like MT and summarization.

But, are greedy sequences always the best solution? 

# But, most likely sequences are repetitive...



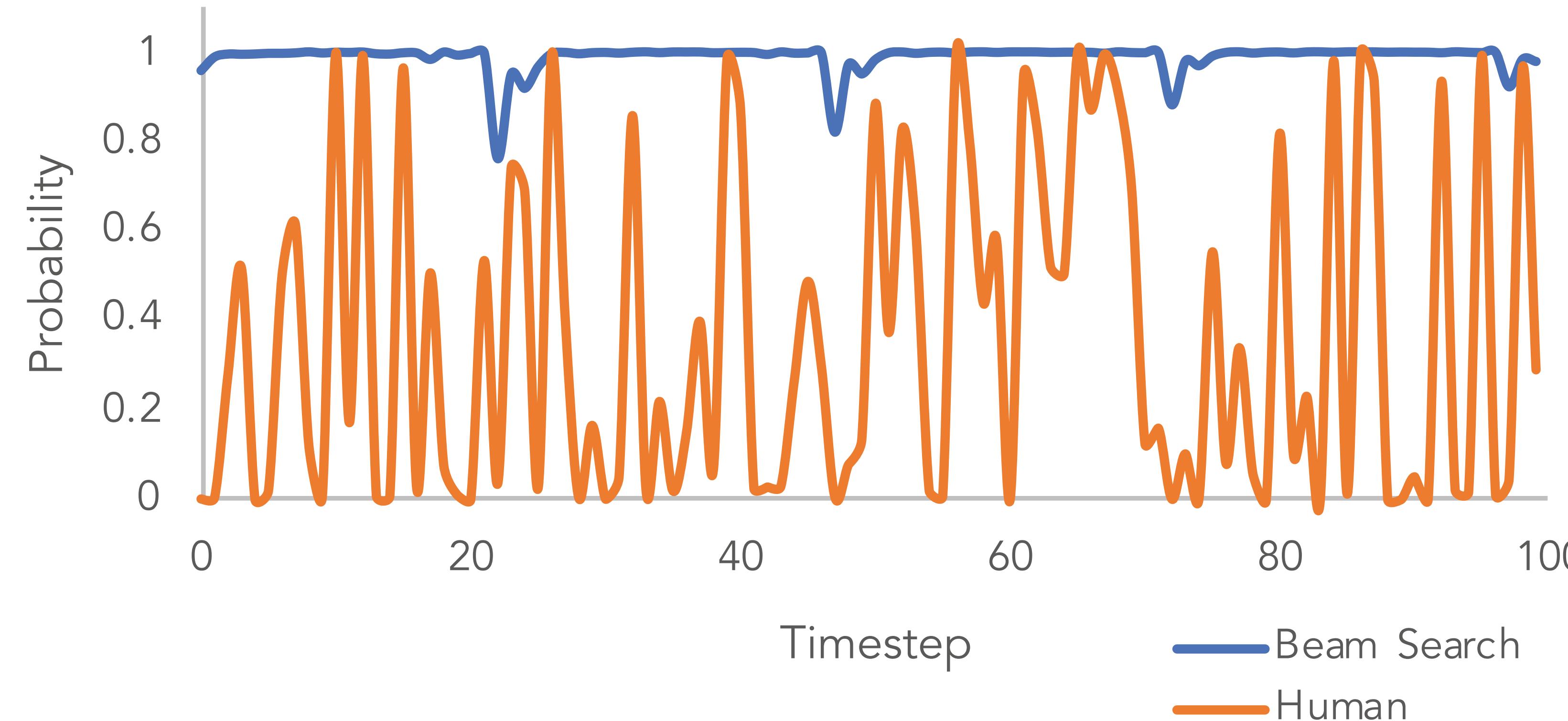
Probability of "I don't know" increases with each repetition, creating a positive feedback loop.



<https://chat.openai.com/share/4d8eb91f-fe1c-430e-bdd3-cafd434ec3d4>

(Holtzman et al. ICLR 2020)

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



Greedy methods fail to capture the variance of human text distribution.

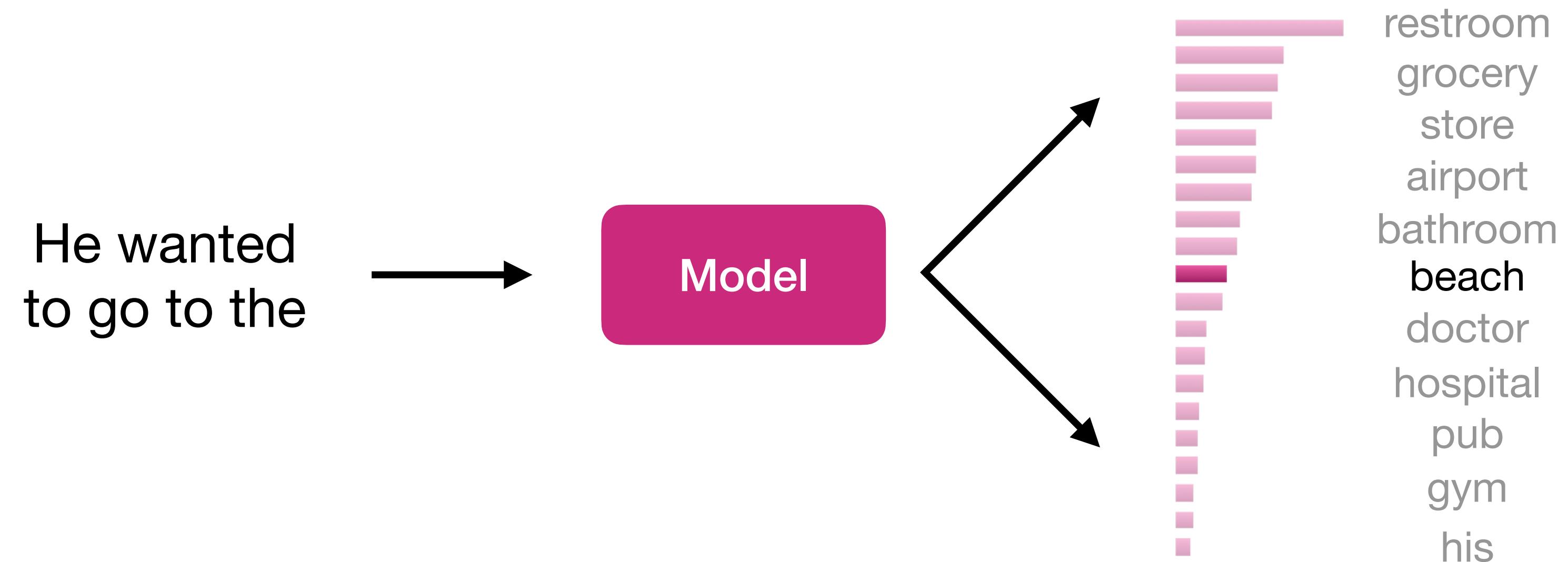
# Sampling generation from LLMs

# Time to get random: Sampling

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

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

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

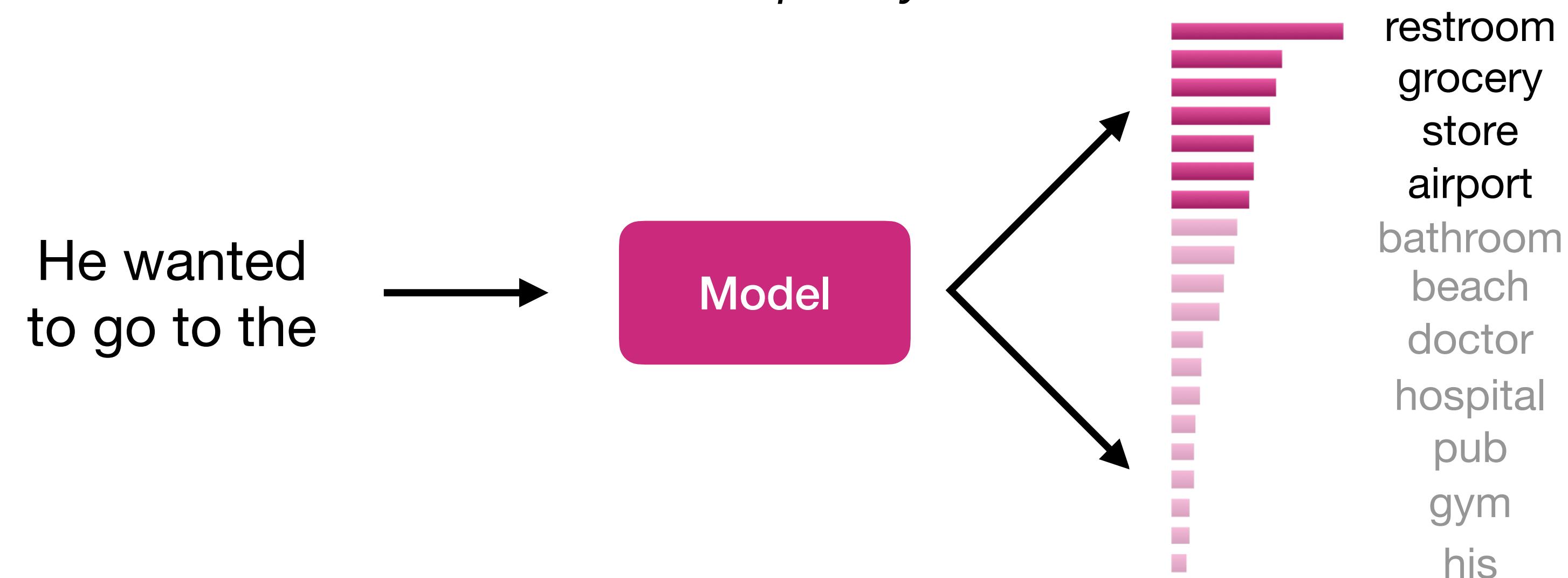


# Decoding: Top-k Sampling

- 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.

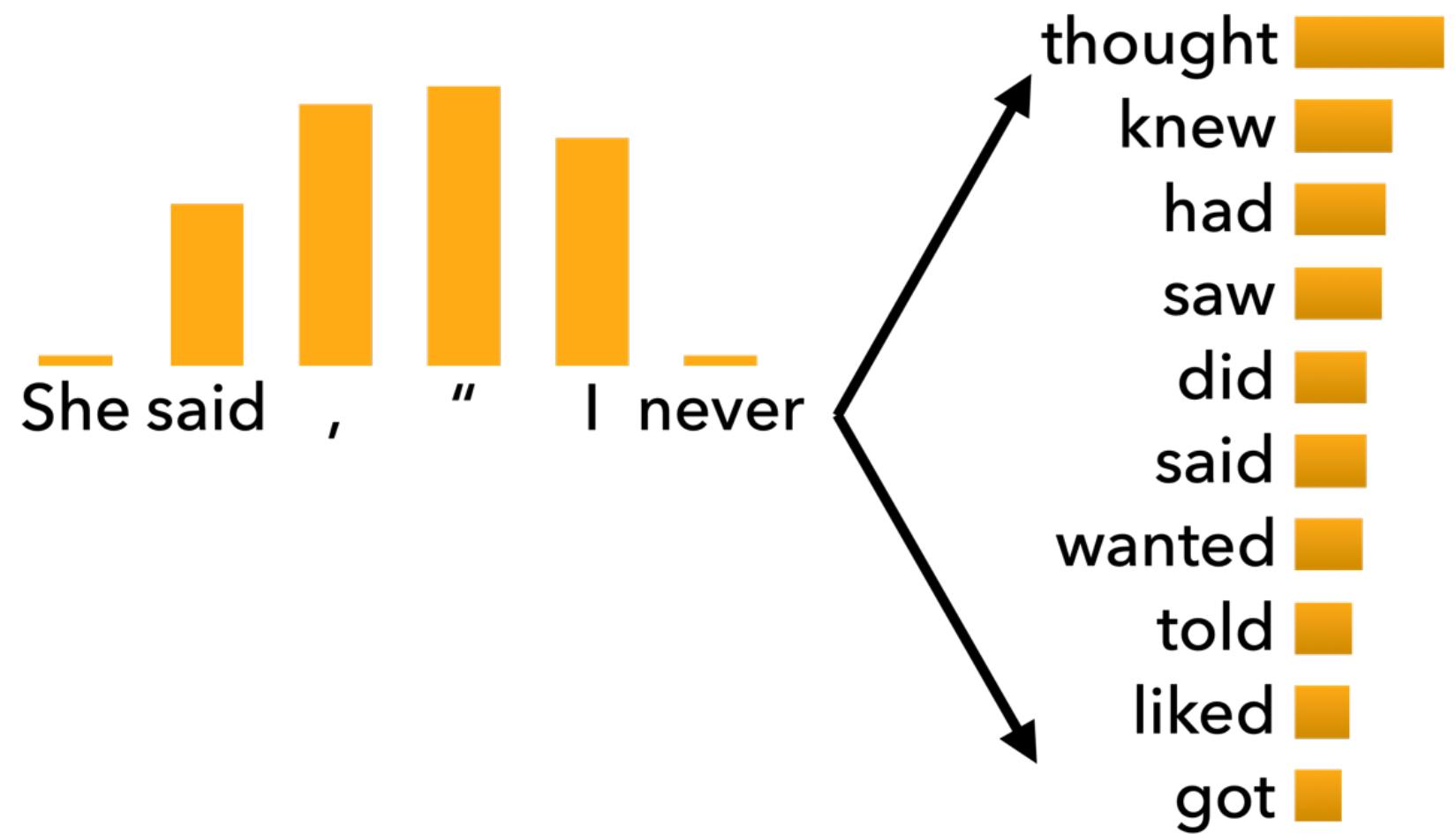
# Decoding: Top-k Sampling

- 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!*)

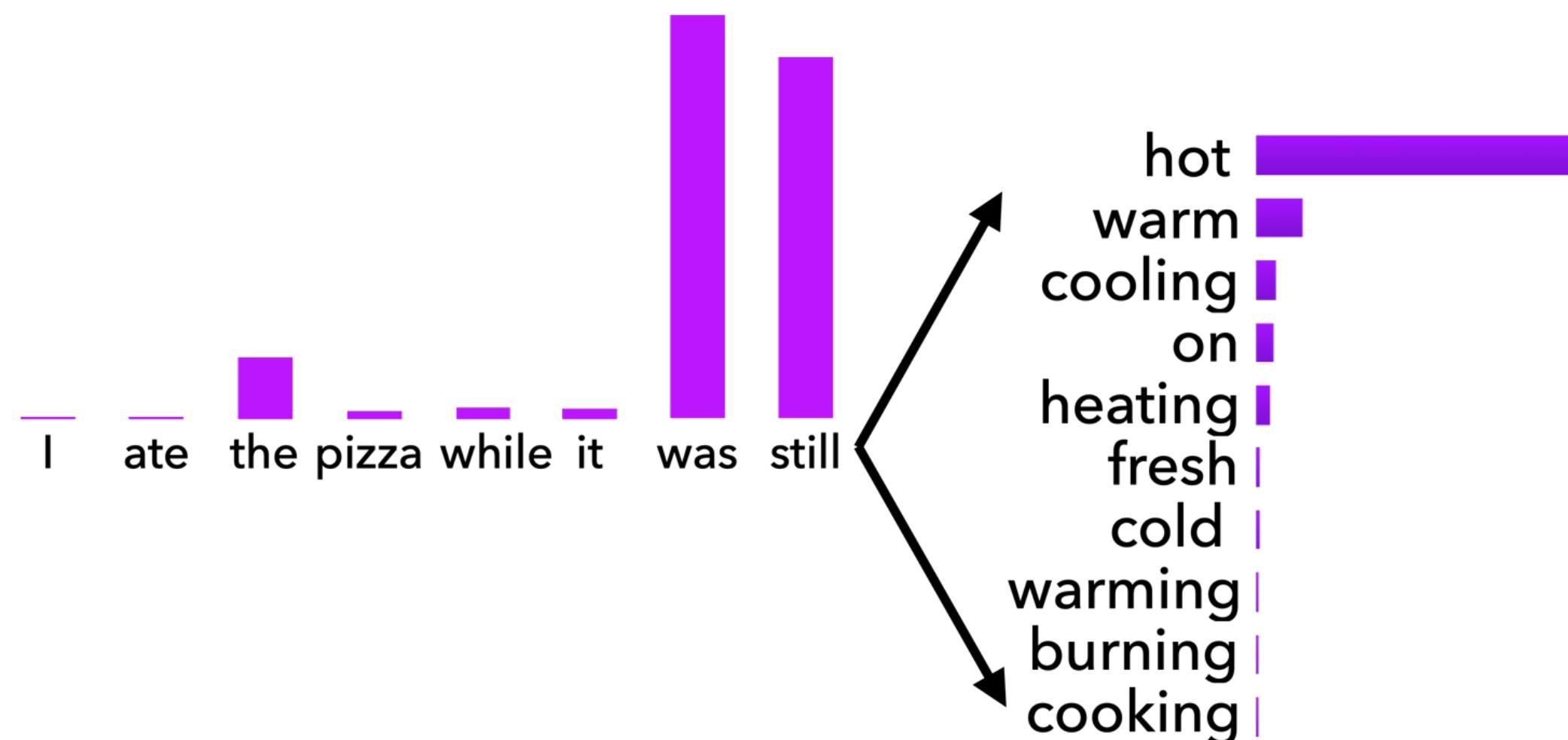


- Increasing  $k$  yields more **diverse**, but **risky** outputs
- Decreasing  $k$  yields more **safe** but **generic** outputs

# 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!**

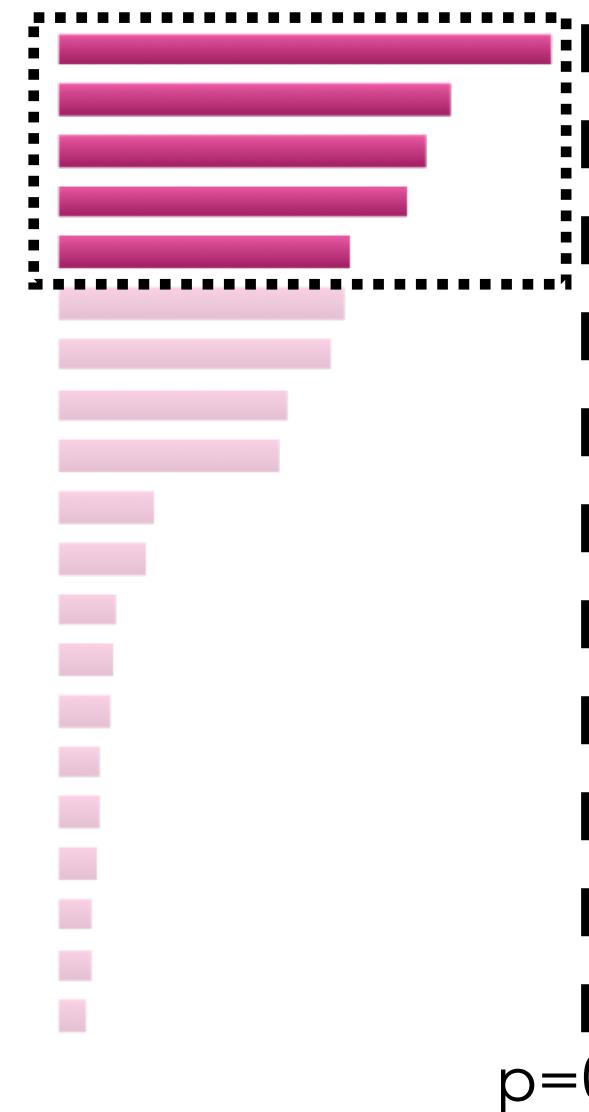
# Decoding: Top-p (Nucleus) Sampling

- 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  $P_t$

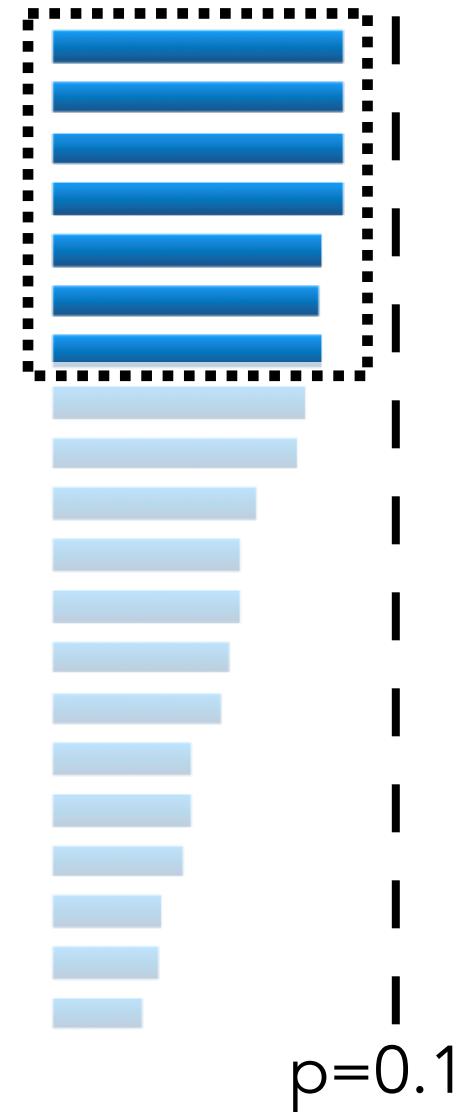
# Decoding: Top-p (Nucleus) Sampling

- 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  $P_t$

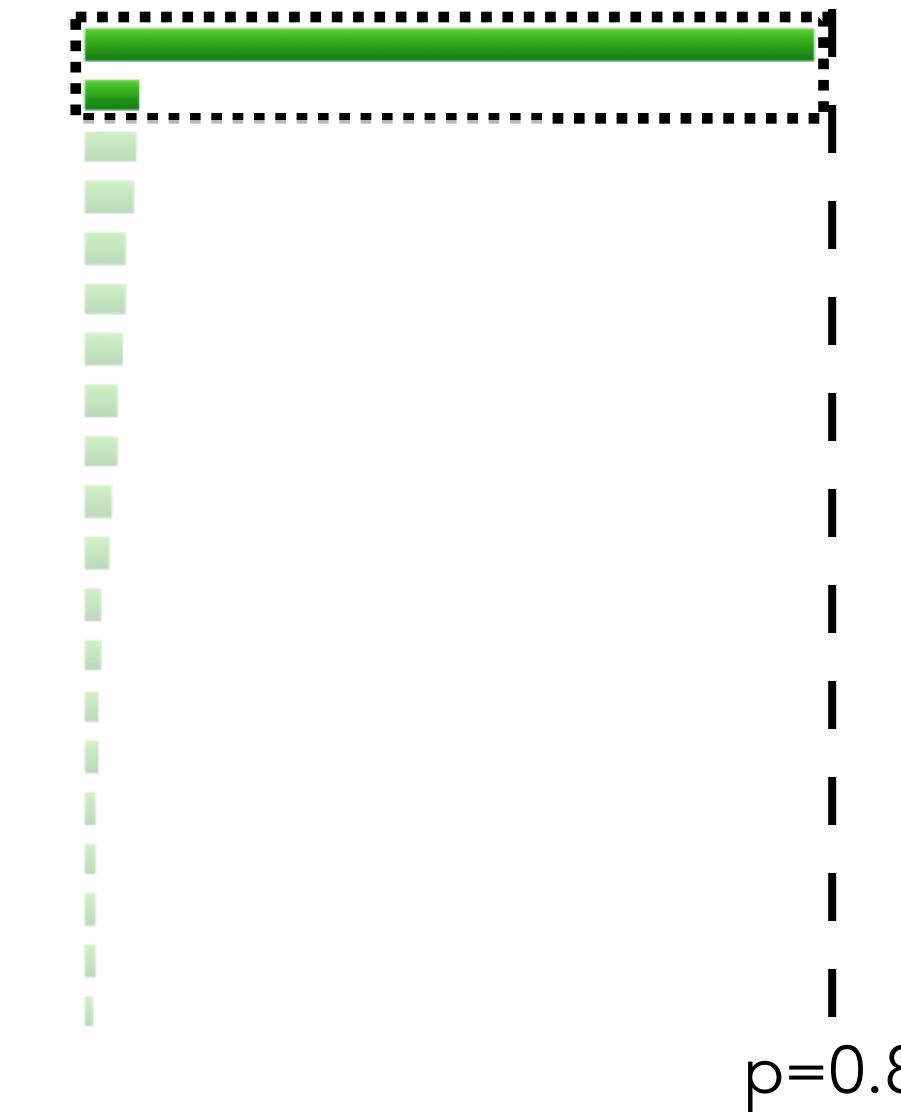
$$P_t(y_t = w | \{y\}_{<t})$$



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



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



# 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  $\tau$**  to the softmax to rebalance  $P_t$ :

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

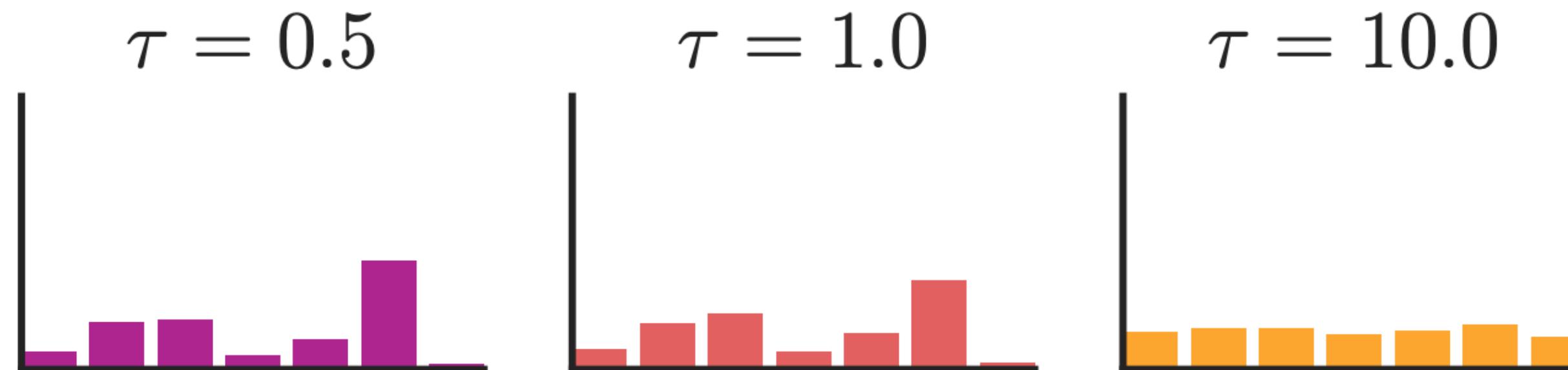
- Raise the **temperature  $\tau > 1$** :  $P_t$  becomes more **uniform**
  - More diverse output (probability is spread across vocabulary)
- Lower the **temperature  $\tau < 1$** :  $P_t$  becomes more **spiky**
  - Less diverse output (probability concentrated to the top tokens)

# Scaling randomness: Softmax temperature

- You can apply **temperature hyperparameter  $\tau$**  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  $\tau$**  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  $n = 10, 20, 50, \dots$  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

- Problem: Generating with a large LM takes a long time

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



- Idea: 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  $M_p$

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

Input prefix

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

$$q(\cdot | x), q(\cdot | x, y_1), \underline{q(\cdot | x, y_1, y_2)}, \dots, q(\cdot | x, y_1, \dots, y_5)$$

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  $p_i = p(\cdot | x, y_1, \dots, y_{i-1})$  and  $q_i = q(\cdot | x, y_1, \dots, y_{i-1})$   
e.g.,  $q_2 = q(\cdot | x, y_1)$ , 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  $M_q$

	<b>Token</b>	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
		dogs	love	chasing	after	cars
<b>Draft model (1B)</b>	$p_i$	0.8	0.7	0.9	0.8	0.7
<b>Target model (100B)</b>	$q_i$	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  $M_q$

	<b>Token</b>	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
		dogs	love	chasing	after	cars
<b>Draft model (1B)</b>	$p_i$	0.8	0.7	0.9	0.8	0.7
<b>Target model (100B)</b>	$q_i$	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  $M_q$

	<b>Token</b>	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
		dogs	love	chasing	after	cars
<b>Draft model (1B)</b>	$p_i$	0.8	<u>0.7</u>	0.9	0.8	0.7
<b>Target model (100B)</b>	$q_i$	0.9	<u>0.8</u>	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

	Token	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
		dogs	love	chasing	after	cars
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  $\frac{q_i}{p_i}$

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	aft	years
Draft model (1B)	$p_i$	0.8	0.7	0.9	0.8	0.7

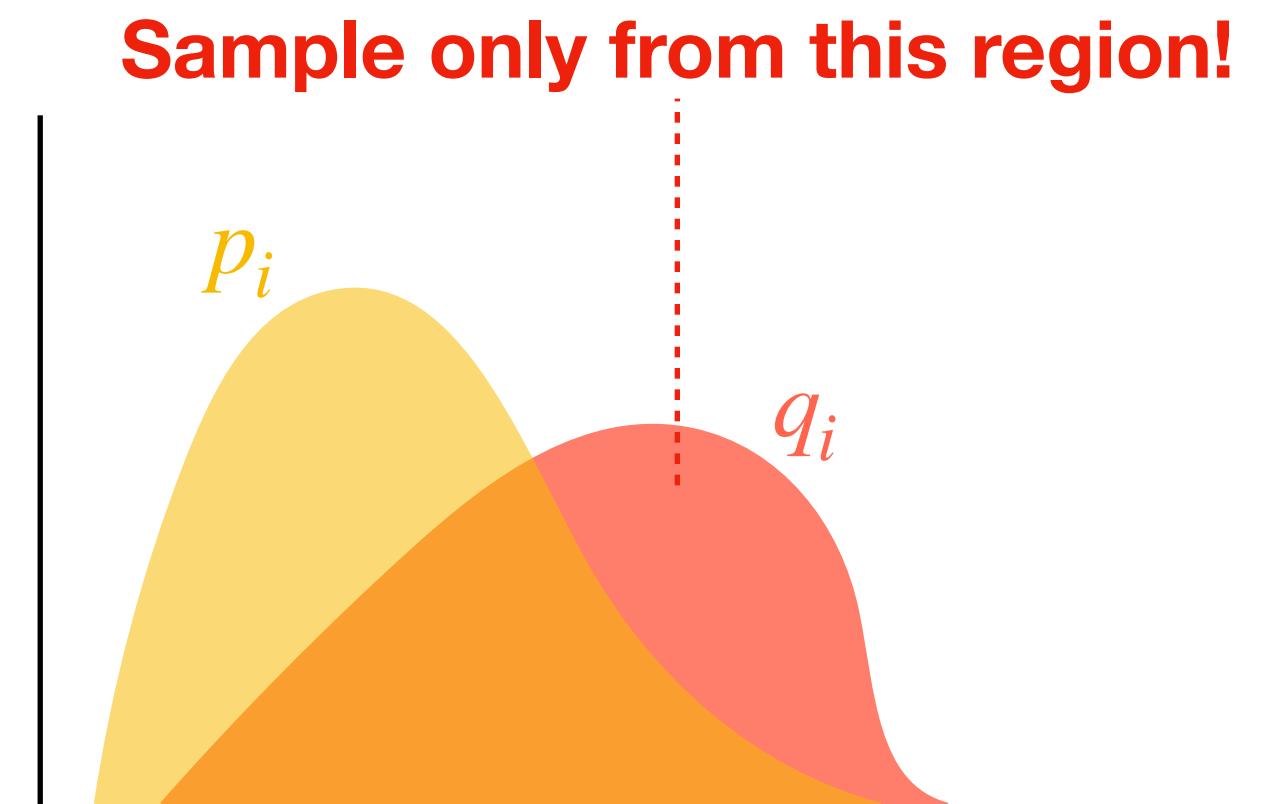
	Token	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
		dogs	love	chasing	aft	years
Target model (100B)	$q_i$	0.9	0.8	0.8	0.8	0.8

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

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

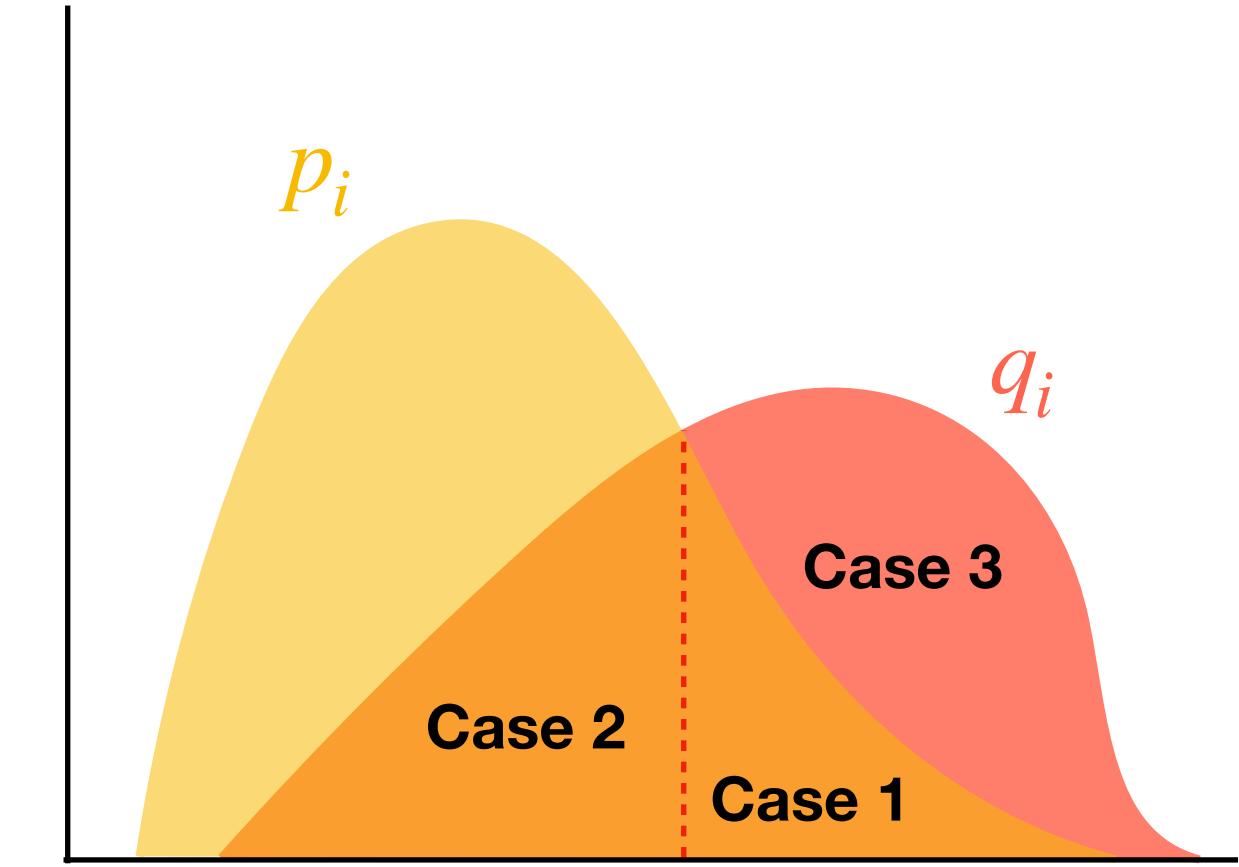
In this case, we sample a **new token from target model**.

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



# Speeding-up generation: Speculative Sampling

- But why specifically  $(q_i - p_i)_+$ ?  
because our goal: to **cover target LM distribution**  $q_i$ .
- Case 1:  $q_i \geq p_i$   
Accept this token.
- Case 2:  $q_i < p_i$  (accept)  
Accept it with the probability  $\frac{q_i}{p_i}$
- Case 3:  $q_i < p_i$  (reject)  
The only remaining case: if token rejected, we sample a new token.  
 $(q_i - p_i)_+$  is the only region left to cover  $q_i$ !



**Note:** This sampling procedure, though sampling from small LM ( $p_i$ ), has the same effect as sampling from target LM ( $q_i$ ). Formal proof in Appendix I of *(Chen et al., 2023)*

# 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

Training LLMs to avoid exposure bias in generation

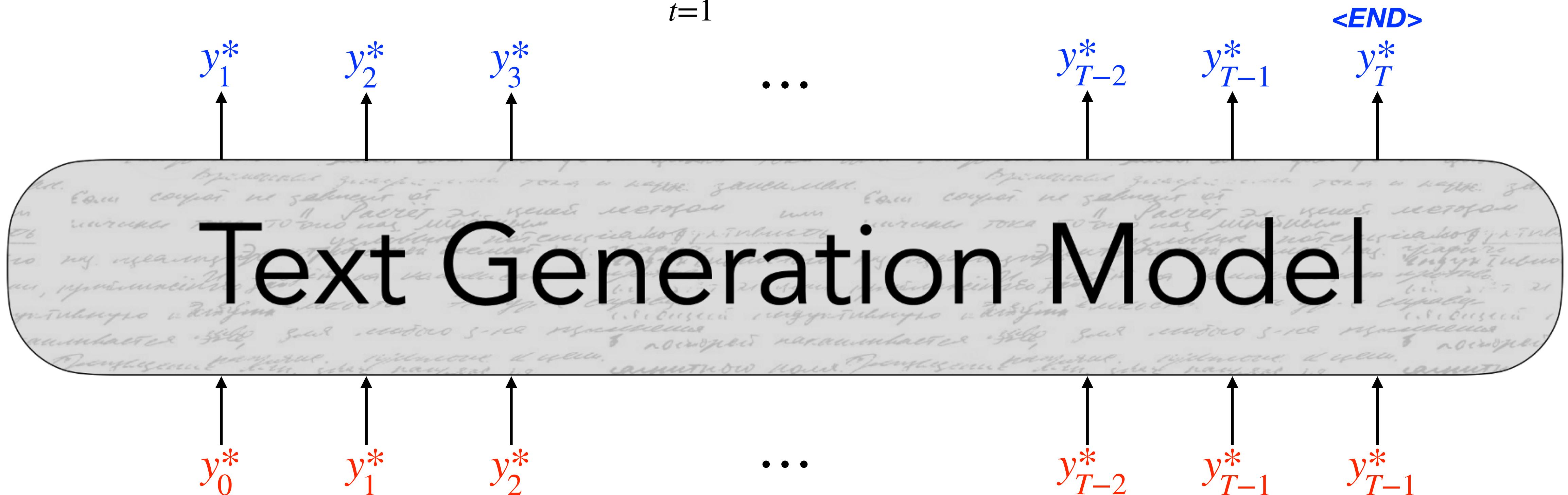
# Issues with teacher forcing training of LLMs

- *Teacher forcing* is still the main algorithm for training text generative models
- **Diversity** is an issue with sequences generated from teacher-forced models
  - Unlikelihood training can help in discouraging undesirable behaviors

# Recall: Teacher-forcing

- Trained to generate the next word  $y_t^*$  given a set of preceding words  $\{y^*\}_{<t}$

$$L = - \sum_{t=1}^T \log P(y_t^* | \{y^*\}_{<t})$$



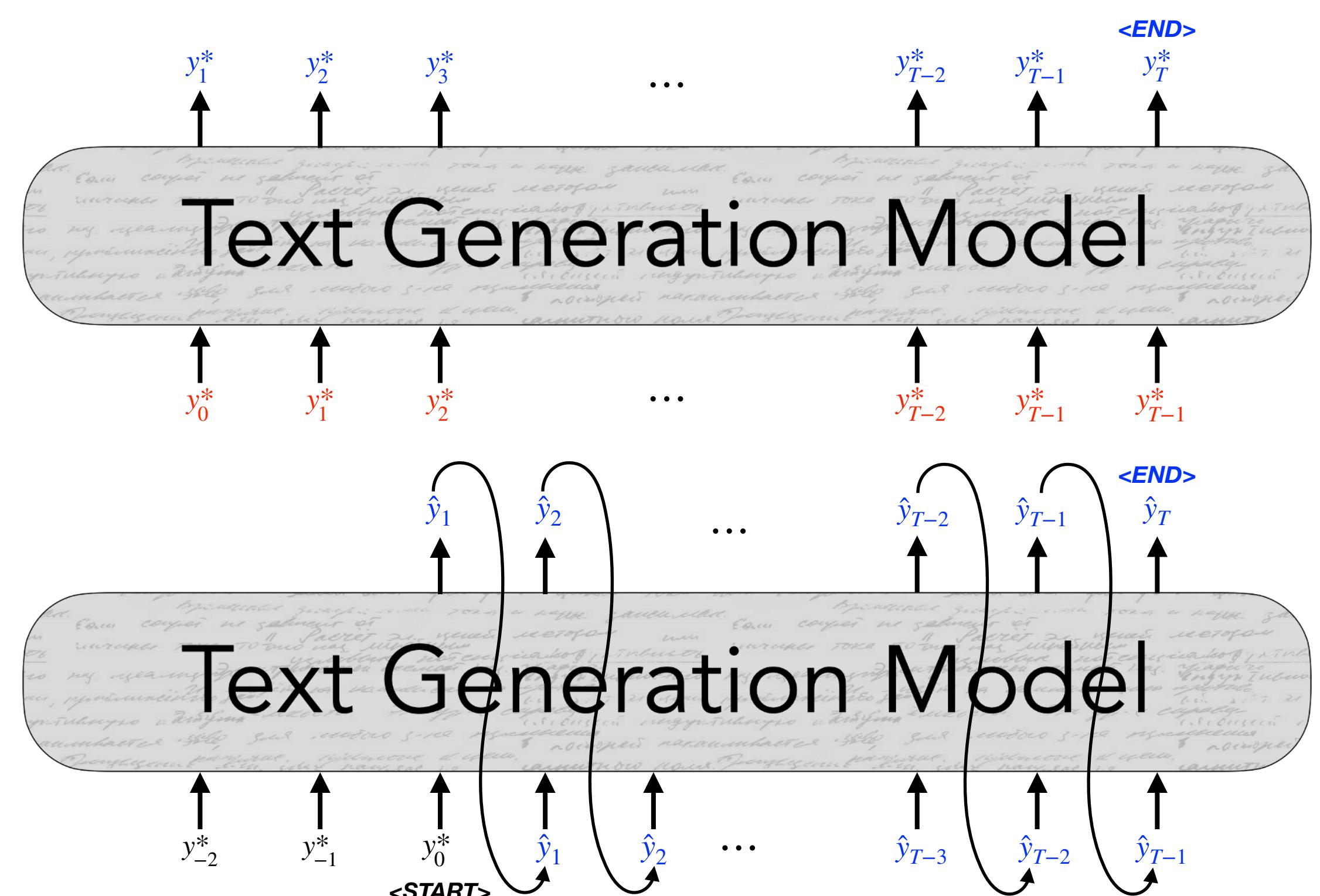
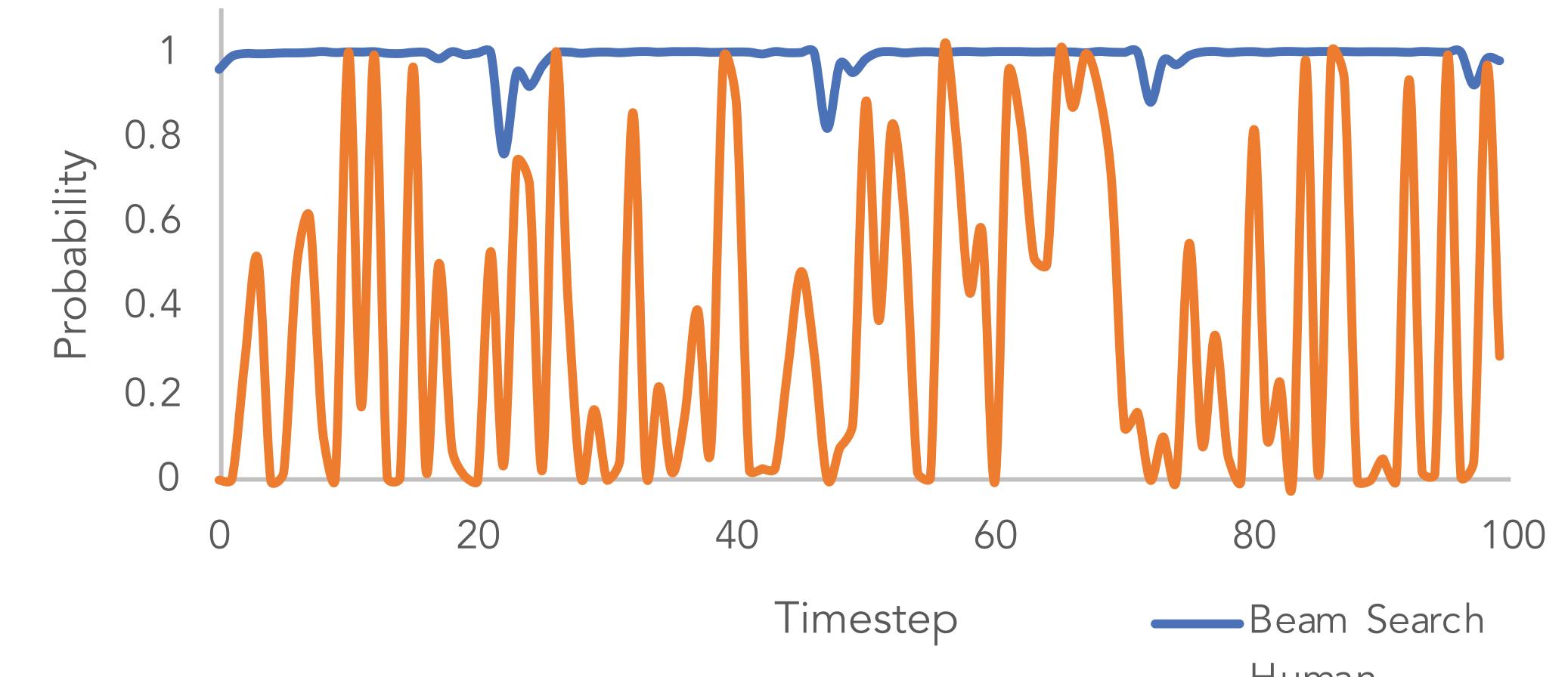
# Exposure Bias

- Training with teacher forcing leads to *exposure bias* at generation time
  - During training, our model's inputs are gold context tokens from real, human-generated texts

$$L_{MLE} = -\log P(y_t^* | \{y^*\}_{<t})$$

- At generation time, our model's inputs are previously-decoded tokens

$$L_{dec} = -\log P(\hat{y}_t | \{\hat{y}\}_{<t})$$



# Exposure Bias Solutions

- Scheduled Sampling (*Bengio et al., 2015*)
  - With some probability  $p$ , decode a token and feed that as the next input, rather than the gold token.
  - Increase  $p$  over the course of training
  - Leads to improvement in practice, but can lead to a strange training objective
- Dataset Aggregation (*a.k.a. DAgger; Ross et al., 2011*)
- Retrieval Augmentation (*Guu\*, Hashimoto\* et al., 2018*)
- Reinforcement Learning...

# Components of NLG Systems

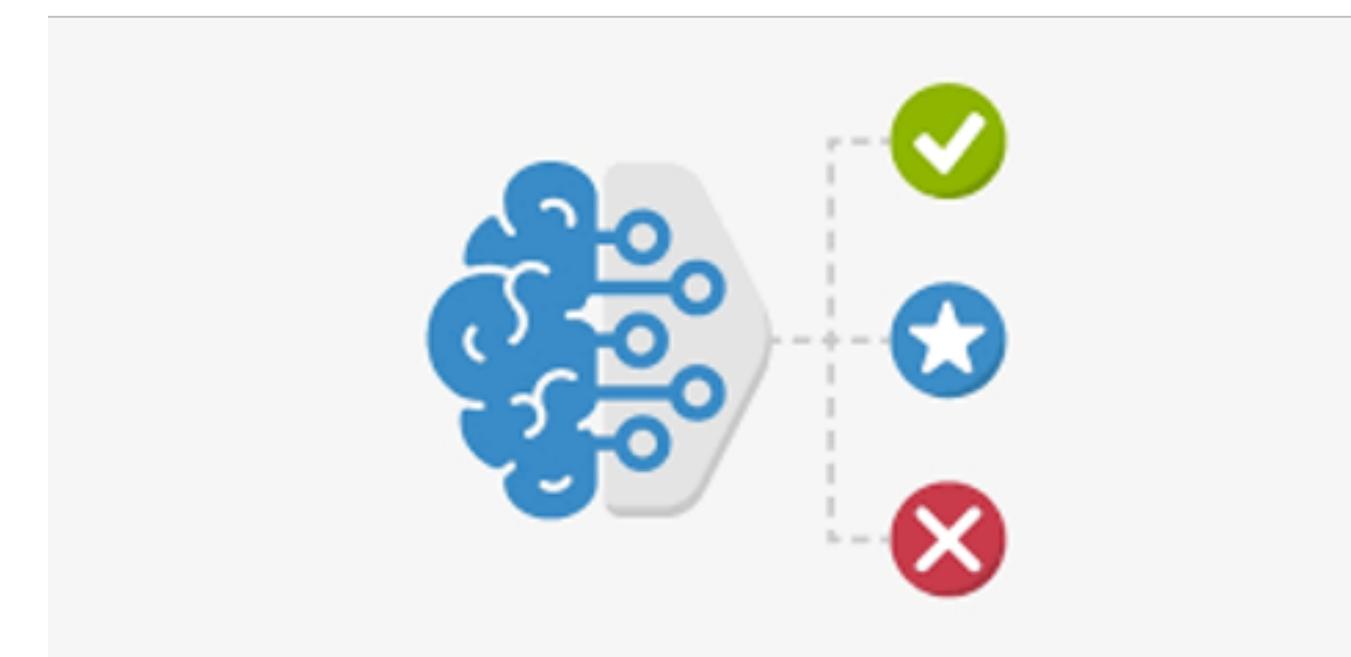
- What is NLG?
- Formalizing NLG: a simple model and training algorithm
- Decoding from NLG models
- Training NLG models
- **Evaluating NLG Systems**
- Ethical Considerations

# Evaluating natural language generation

# Types of text evaluation methods

Ref: They walked **to the grocery store.**  
Gen: **The woman went to the hardware store.**

Content Overlap Metrics



Model-based Metrics

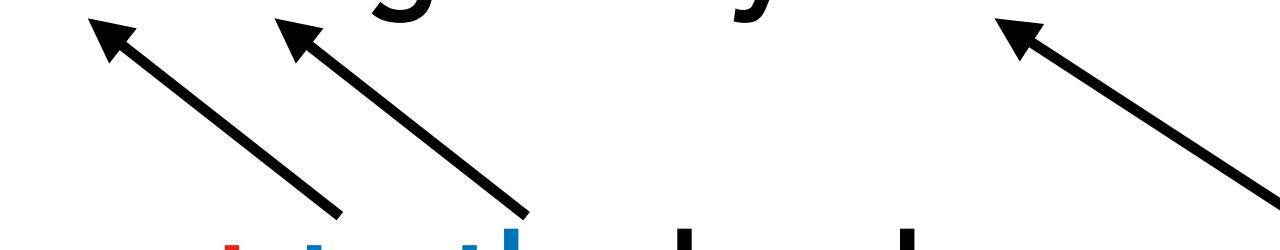


Human Evaluation

# Content overlap metrics

Ref: They walked **to the grocery store.**

Gen: **The woman went to the hardware store.**



- Compute a score that indicates the similarity between *generated* and *gold-standard* (often human-written) text
- Fast and efficient; widely used (e.g. for MT and summarization)
- Dominant approach: *N-gram overlap* metrics
  - e.g., BLEU, ROUGE, METEOR, CIDEr, etc.

# Content overlap metrics

- Dominant approach: *N-gram overlap* metrics
  - e.g., BLEU, ROUGE, METEOR, CIDEr, etc.
- Not ideal even for less open-ended tasks - e.g., machine translation
- They get progressively much worse for more open-ended tasks
  - Worse for summarization, as longer summaries are harder to measure
  - Much worse for dialogue (in how many ways can you respond to your friend?)
  - Much, much worse for story generation, which is also open-ended, but whose sequence length can make it seem you're getting decent scores!

# A simple failure case

- N-gram overlap metrics have no concept of **semantic relatedness!**



Are you enjoying the  
NLP class?

For sure!

Score:

0.61

Yes for sure!

0.25

Sure I do!

0.0

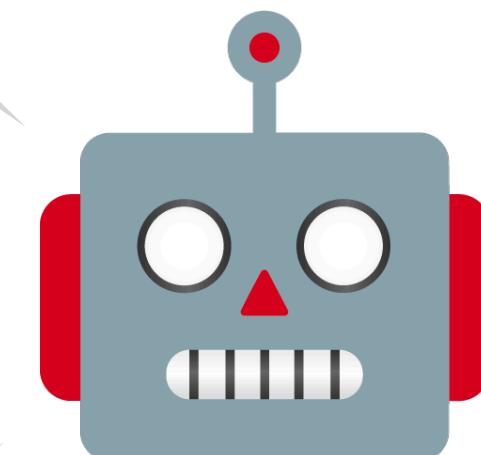
Yes!

| False negative

| False positive

0.61

No for sure...



# A more comprehensive failure analysis

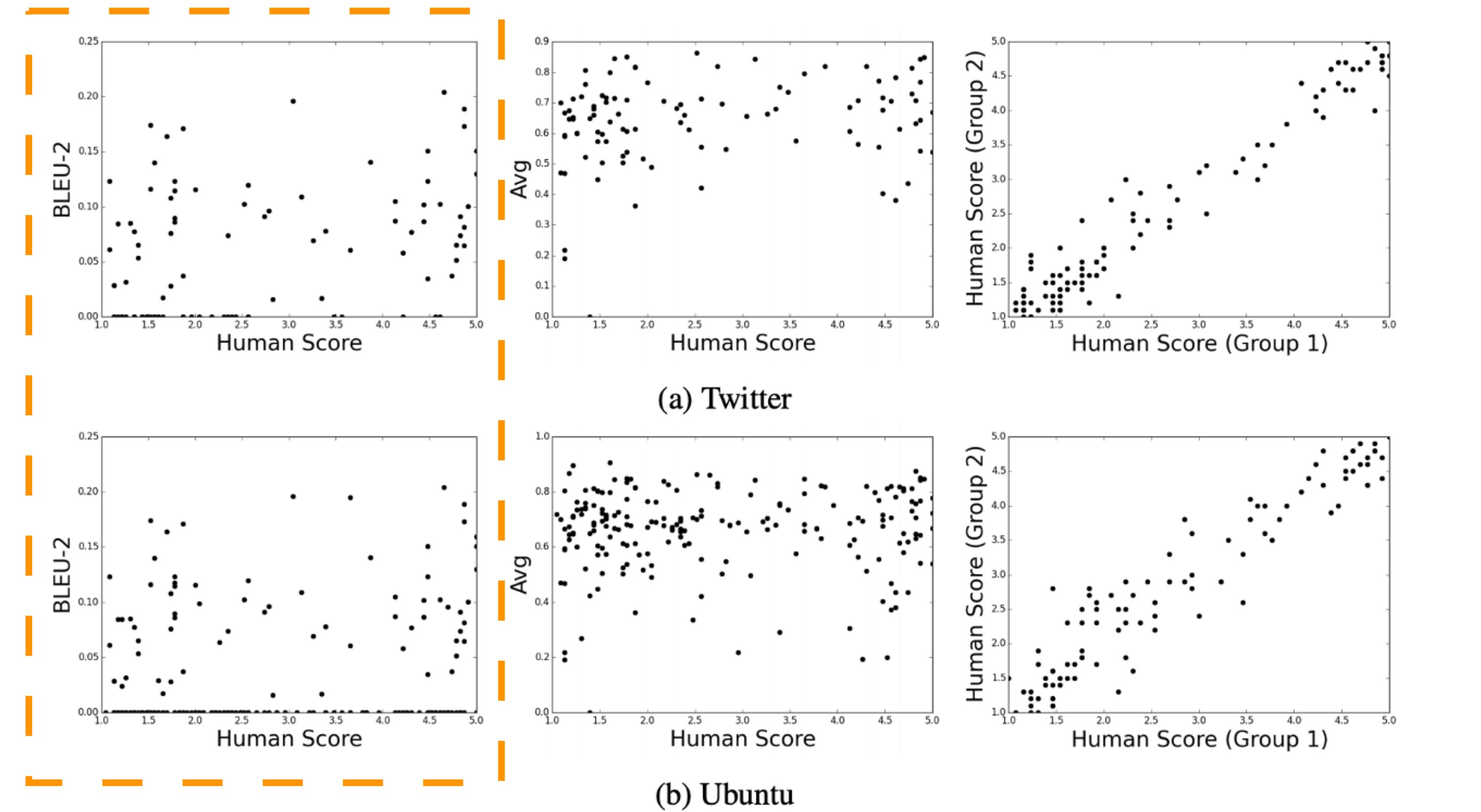
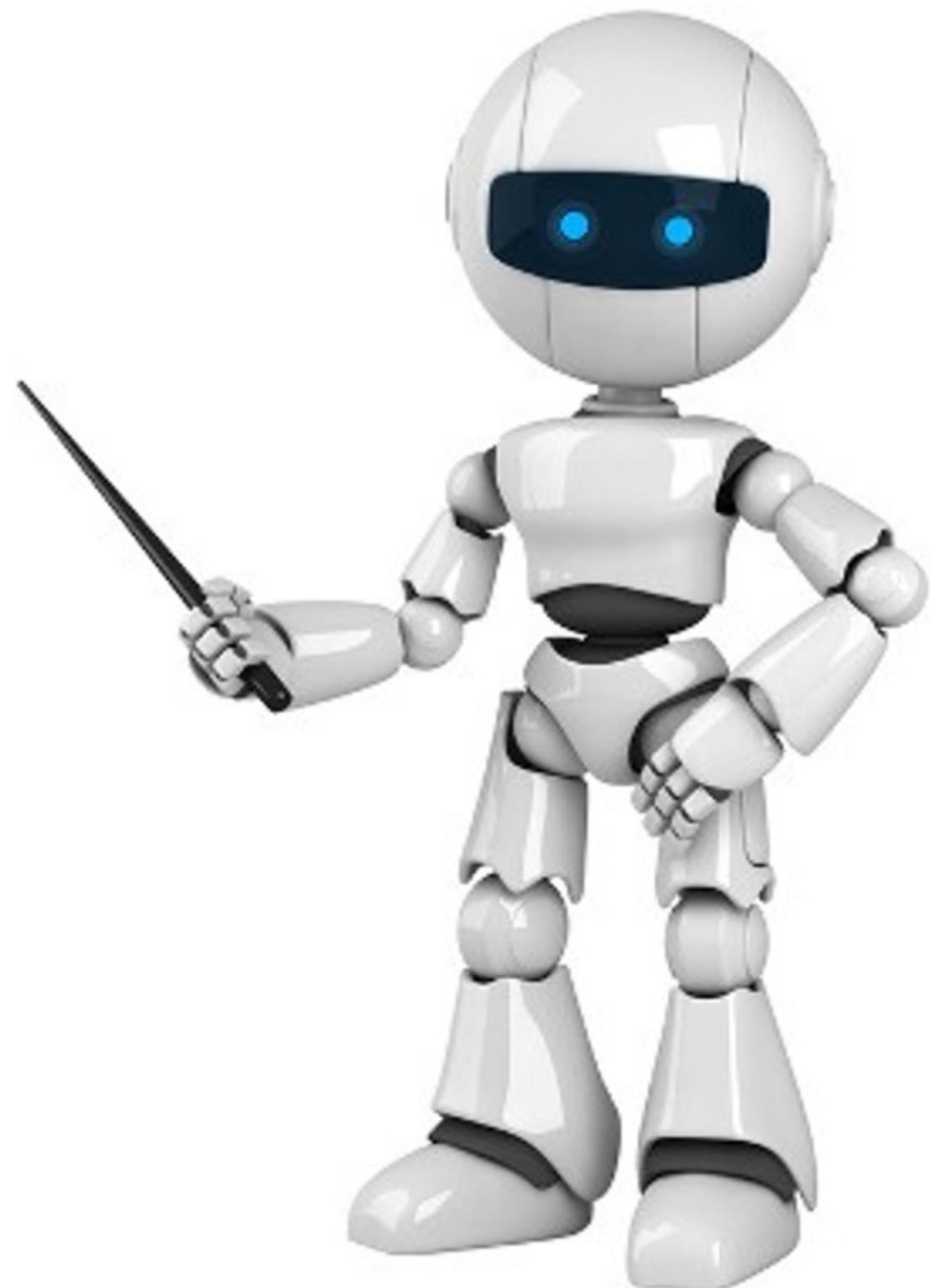


Figure 1: Scatter plots showing the correlation between metrics and human judgements on the Twitter corpus (a) and Ubuntu Dialogue Corpus (b). The plots represent BLEU-2 (left), embedding average (center), and correlation between two randomly selected halves of human respondents (right).

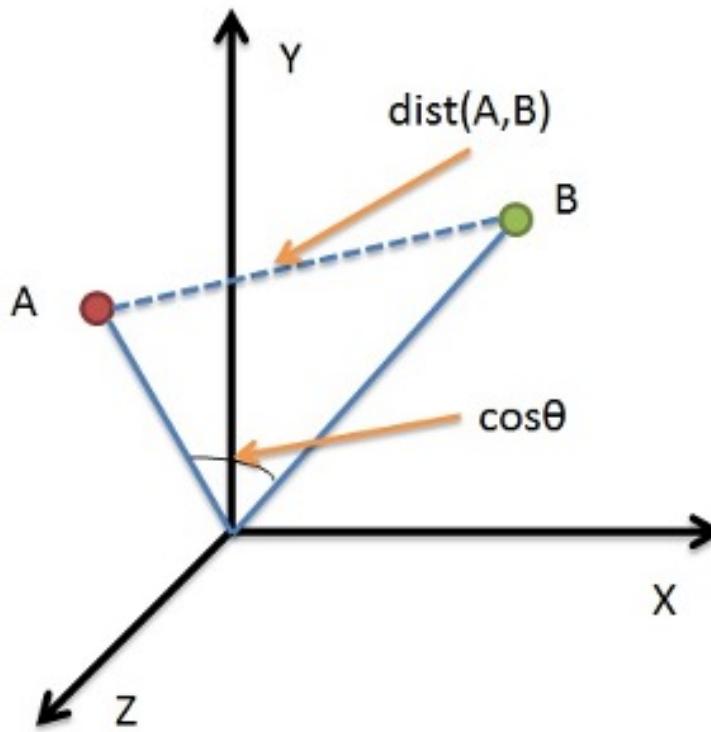
- Higher  $n$ -gram overlap does not imply higher **human score**.

# Model-based metrics to capture more semantics

- Use learned representation of words and sentences to compute semantic similarity between generated and reference texts
- No more n-gram bottleneck: text units are represented as embeddings!
- Even though embeddings are pre-trained, distance metrics used to measure similarity can be fixed.



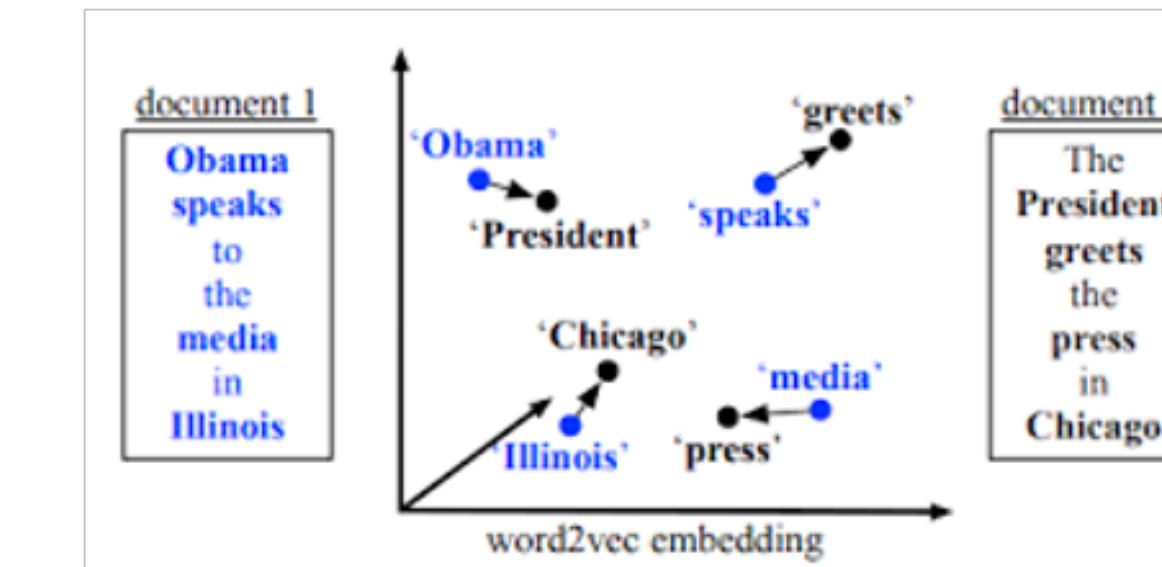
# Model-based metrics: Word distance functions



## Vector Similarity

Embedding-based similarity for semantic distance between text.

- Embedding Average (*Liu et al., 2016*)
- Vector Extrema (*Liu et al., 2016*)
- MEANT (*Lo, 2017*)
- YISI (*Lo, 2019*)



## Word Mover's Distance

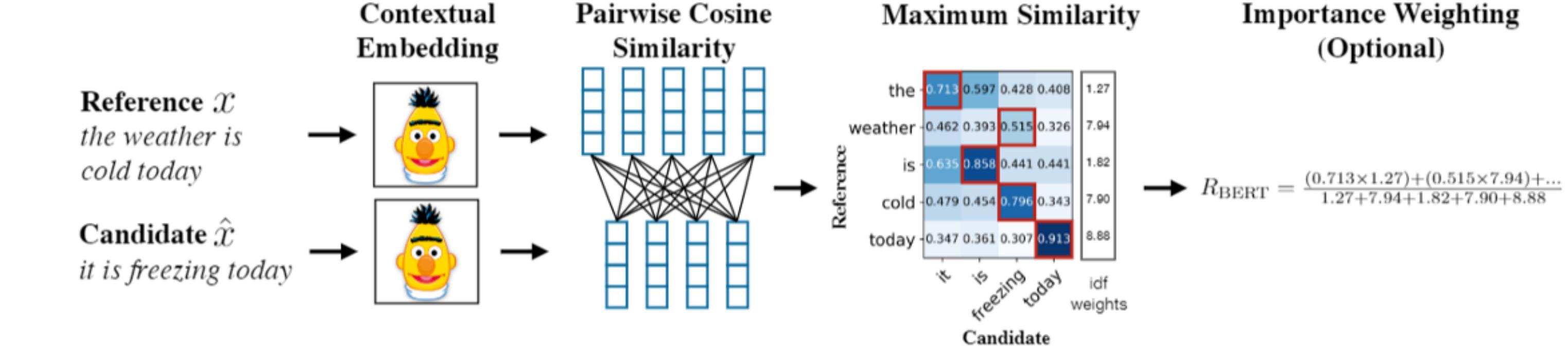
Measures the distance between two sequences using word embedding similarity matching.

- (*Kusner et al., 2015; Zhao et al., 2019*)

## BERTSCORE

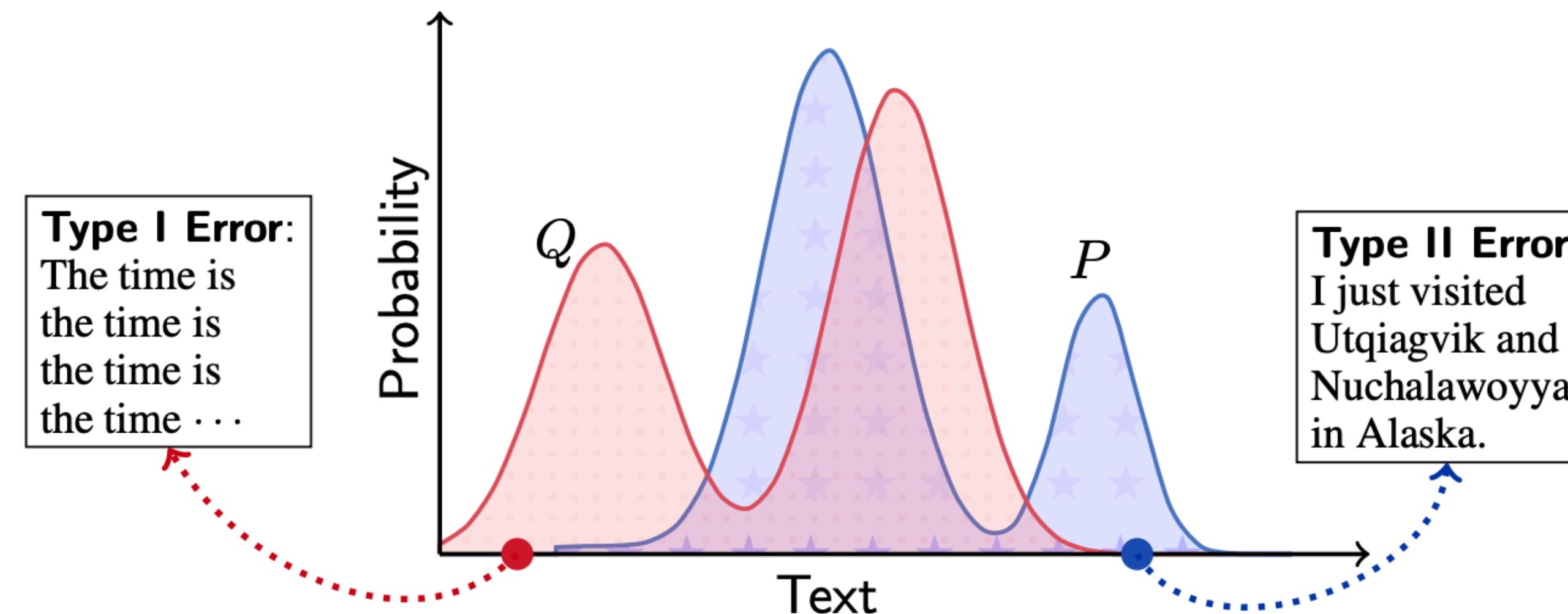
Uses pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity.

- (*Zhang et al., 2019*)



# MAUVE: Beyond single sample matching

- In open-ended generation, comparing with a single reference may not say much.  
Can we instead compare the distribution of machine text vs. human text?
- **MAUVE** (*Pillutla et al., 2021*)
  - Computes the information divergence between the human text distribution  $P$  and the machine text distribution  $Q$



# MAUVE: Beyond single sample matching

- Divergence Curve

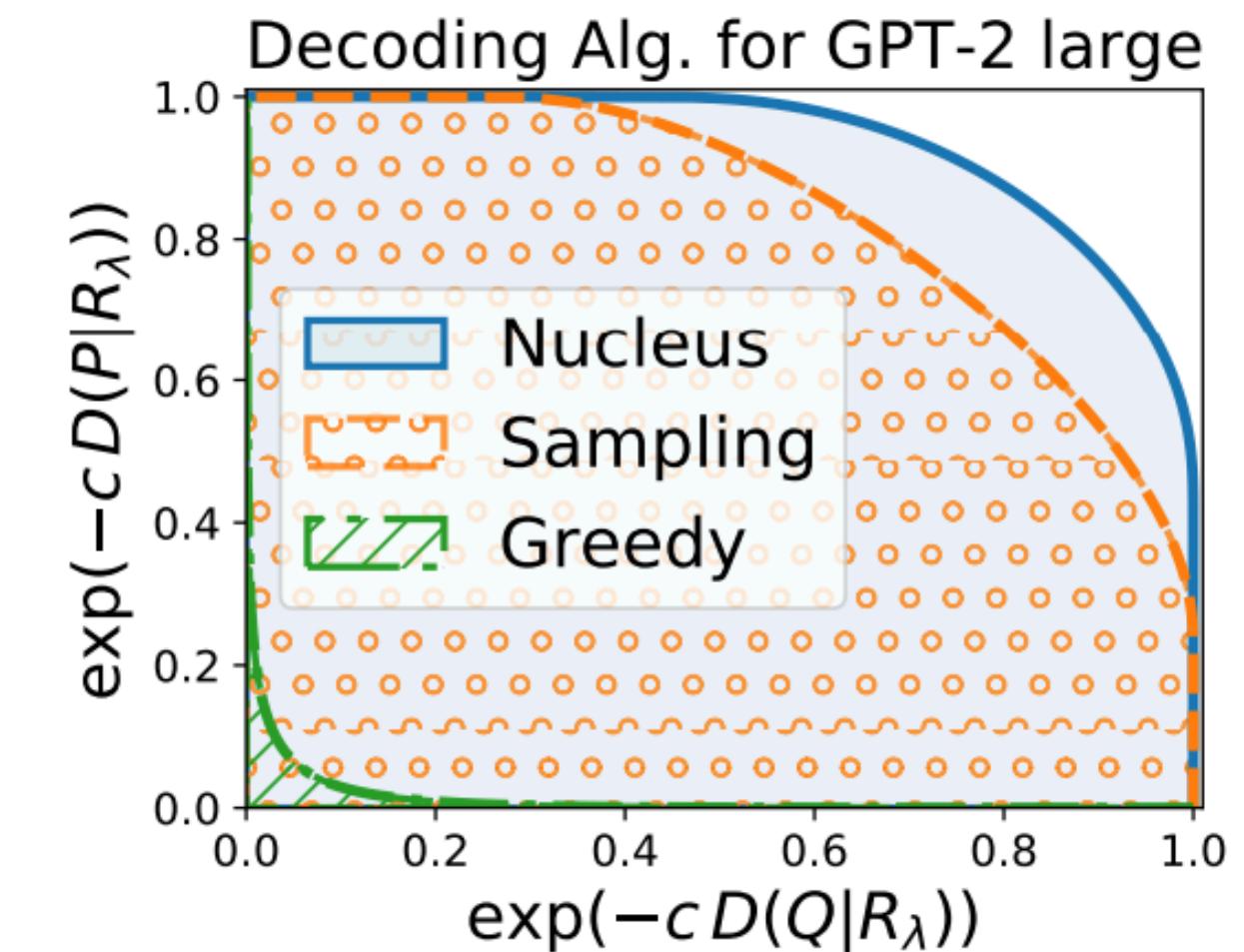
$$\mathcal{C}(P, Q) = \left\{ \left( \exp(-c \underline{\text{KL}}(Q|R_\lambda)), \exp(-c \overline{\text{KL}}(P|R_\lambda)) \right) : \underline{R_\lambda} = \lambda P + (1 - \lambda)Q, \lambda \in (0, 1) \right\}$$

KL Divergence: Distance between  
two distributions  $Q$  and  $R_\lambda$

Interpolate between  $P$  and  $Q$  to draw a curve

$$\text{KL}(P|R_\lambda) = \sum_{\mathbf{x}} P(\mathbf{x}) \log \frac{P(\mathbf{x})}{R_\lambda(\mathbf{x})}$$

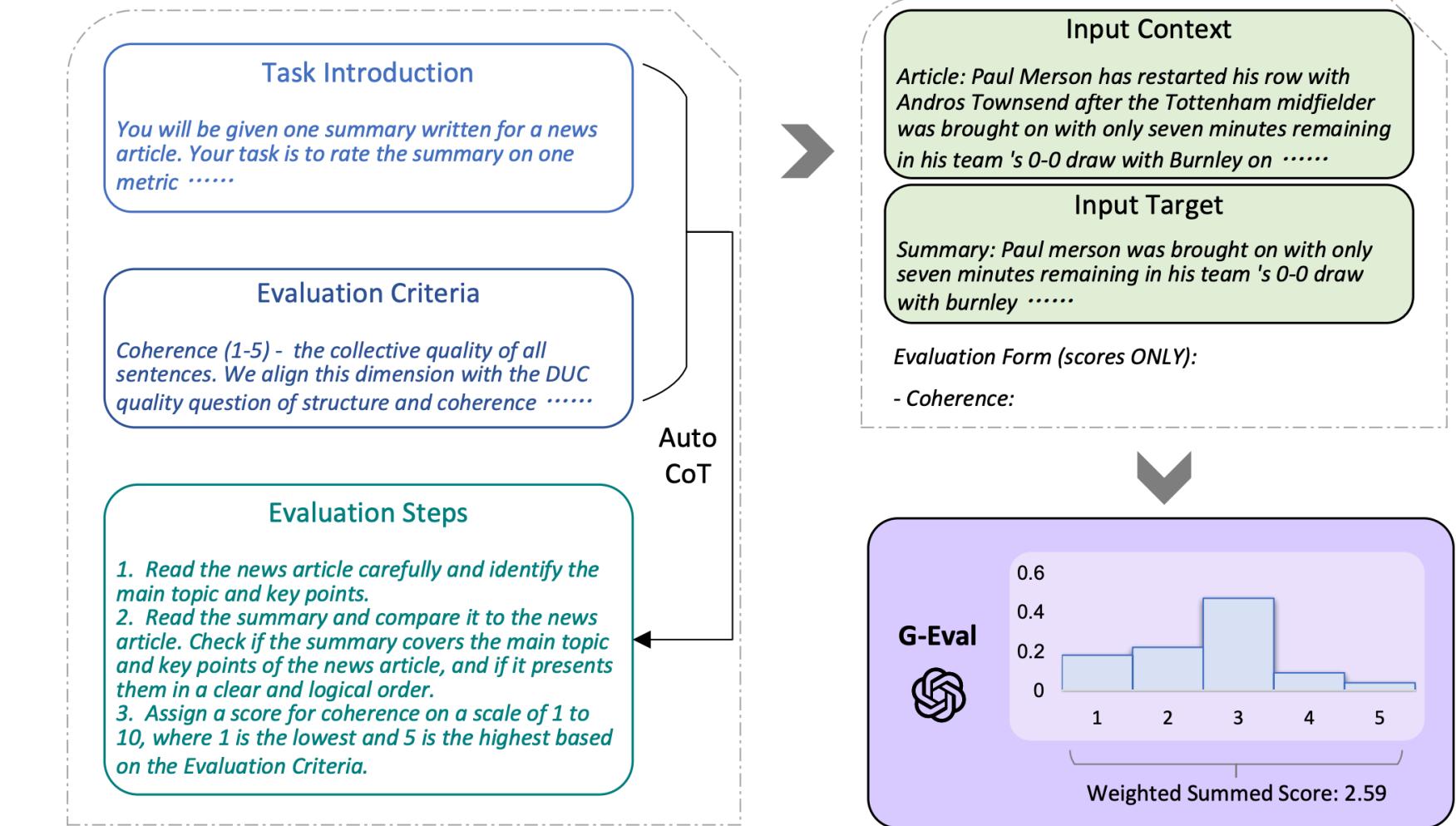
- If  $P$  and  $Q$  are **close**, KL divergence will be **lower**, thus the divergence curve will be **higher**
- **MAUVE( $\mathbf{P}, \mathbf{Q}$ )**: Area under the divergence curve  
(value in 0~1, **higher is better!**)



Nucleus sampling is better than  
naive sampling / greedy decoding.

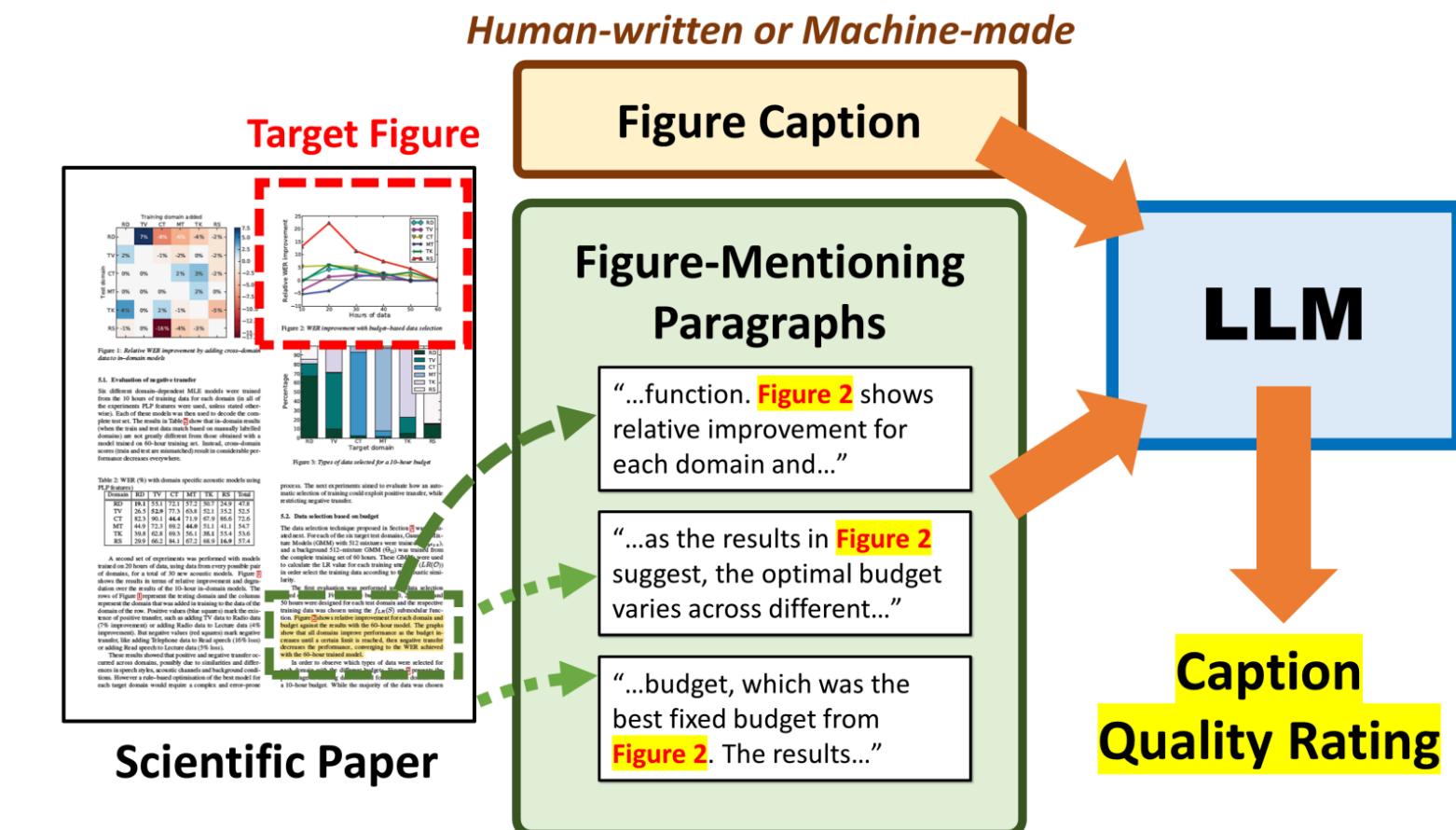
# Model-based metrics: LLM as evaluator

- Directly prompt LLM (GPT-4) to evaluate generated text.
  - Can be **customized** with evaluation criteria
  - (Often) better correlation with human evaluators than task-specific metrics (e.g. ROUGE)
  - (Often) is **cheaper** than human evaluation



Liu et al. 2023

- Limitations
  - Brittleness: LLM evaluation can significantly vary when **given different prompts!**
  - Potential **self-bias** - LLMs may prefer what LLMs have generated...



Hsu et al. EMNLP Findings, 2023

# Human evaluations



- Automatic metrics fall short of matching human decisions
- Most important form of evaluation for text generation systems
- Gold standard in developing new automatic metrics
  - Better automatic metrics will better correlate with human judgements!

# Human evaluations

- Sounds easy, but hard in practice: [Ask humans](#) to evaluate the quality of text
- Typical evaluation dimensions:
  - fluency
  - coherence / consistency
  - factuality and correctness
  - commonsense
  - style / formality
  - grammaticality
  - typicality
  - redundancy
  - ...

Note: Don't compare human evaluation scores across different studies

Even if they claim to evaluate on the same dimensions!

# Human evaluations

- Human judgments are regarded as **gold standard**
- Of course, we know that human eval is **slow** and **expensive**
- Beyond its cost, human eval is still far from perfect:
- Human judgements
  - are inconsistent / irreproducible
  - can be illogical
  - can be misinterpreting your questionnaire
  - ...
- and recently, use of LLMs by crowd-source workers 😬  
*(Veselovsky et al., 2023)*

Artificial Artificial Artificial Intelligence: Crowd Workers Widely Use  
Large Language Models for Text Production Tasks

Veniamin Veselovsky,\* Manoel Horta Ribeiro,\* Robert West  
EPFL  
firstname.lastname@epfl.ch

# Evaluation: Takeaways

- *Content-overlap metrics* provide a good starting point for evaluating the generation quality, but they're **not good enough on their own**
- *Model-based metrics* can be more correlated with human judgment, but often are **not interpretable**
- Human judgments are critical
  - But humans are **inconsistent!**
- In many cases, the best judge of output quality is **YOU!**
  - **Look at the actual generations - don't just rely on numbers.**
  - **Publicly release large samples of outputs from your system!**

# Concluding Thoughts

- Interacting with NLG systems quickly **shows their limitations**
- Even in tasks with more progress, there are **still many improvements ahead**
- Evaluation remains a huge challenge
  - We need better ways to **automatically evaluate NLG systems**
- One of the **most exciting areas** of NLP to work in!