DATA 8005 Advanced Natural Language Processing

Lecture 3: Introduction to LLMs

Fall 2024

Many materials from CSE447@UW (Taylor Sorensen and Jaehun Jung) and COS484@Princeton with special thanks!

Announcements

- Sign up for final projects
  - ~~In-class presentation: by Sep 22~~
  - Final projects: by Oct 4

Neural language models: generation

Categorization of NLG tasks

Spectrum of open-endedness for 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

Spectrum of open-endedness for 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

Spectrum of open-endedness for NLG tasks

Input: Write a story about three little pigs?

Reference Outputs:

... (so may options)...

The output space is extremely diverse.

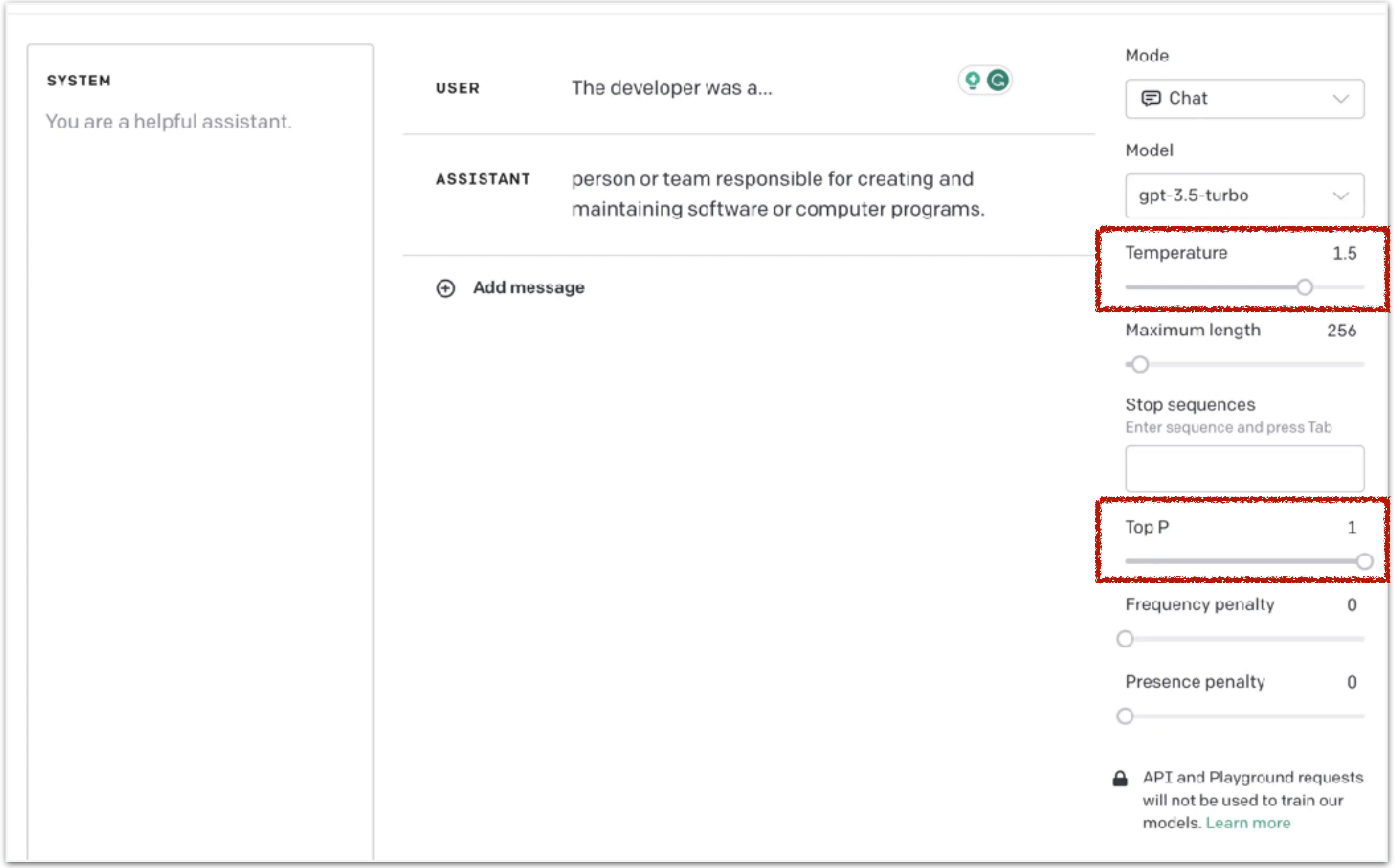
Categorization of NLG tasks

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!

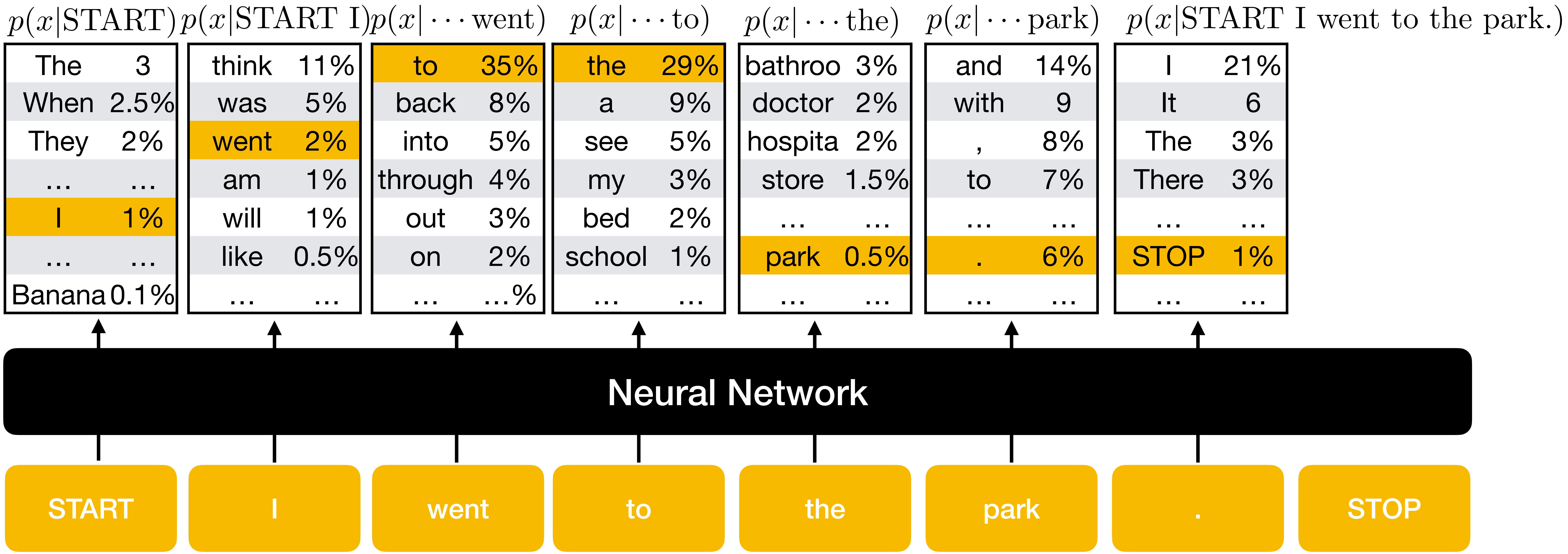
How to control open-endedness in ChatGPT?



ChatGPT API web interface

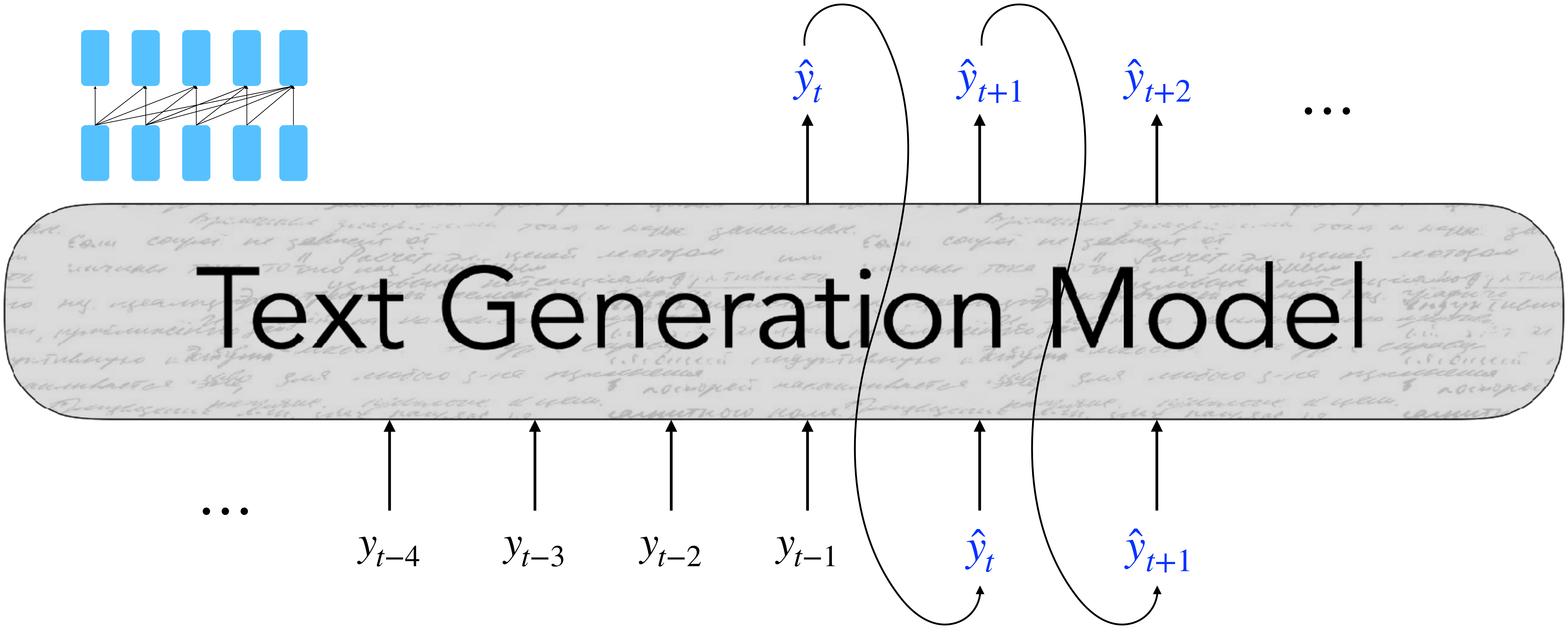
Neural language models

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



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, $S \in \mathbb{R}^V$:

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

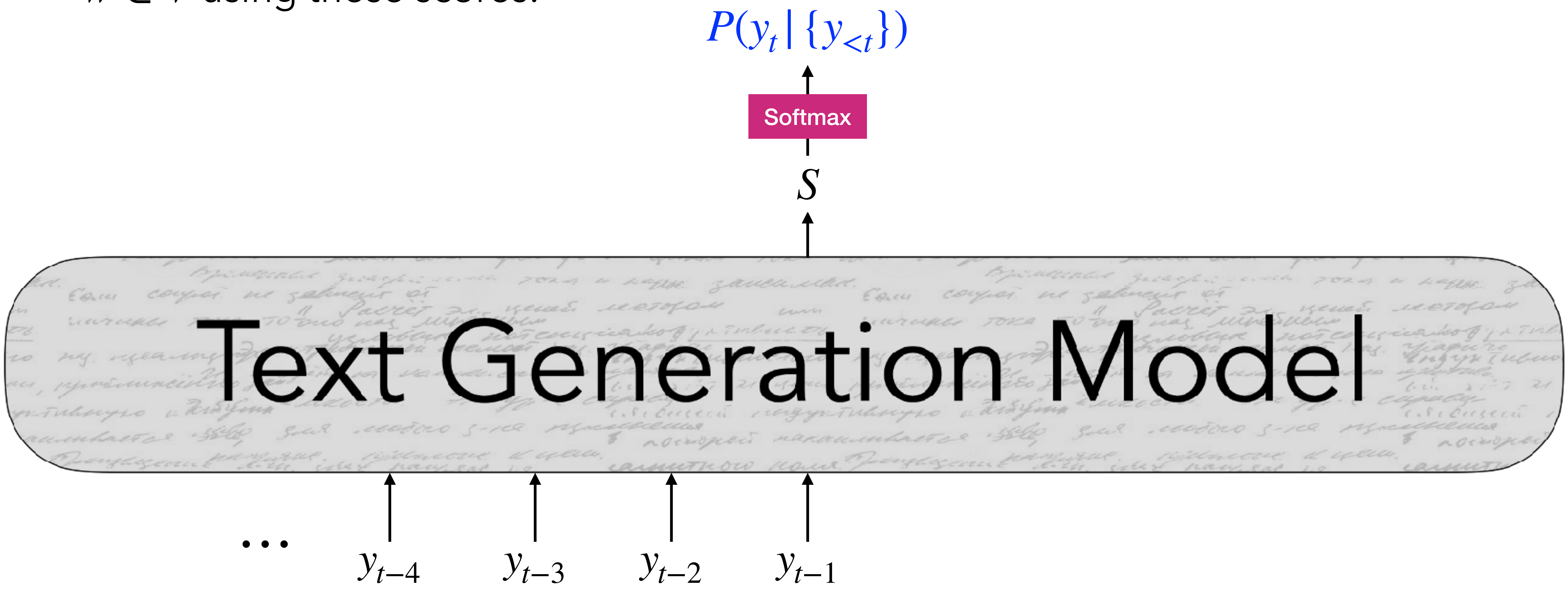
$f(\,\cdot\,;\theta)$ is your model

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

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

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:

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

$g(\cdot)$ 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:

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

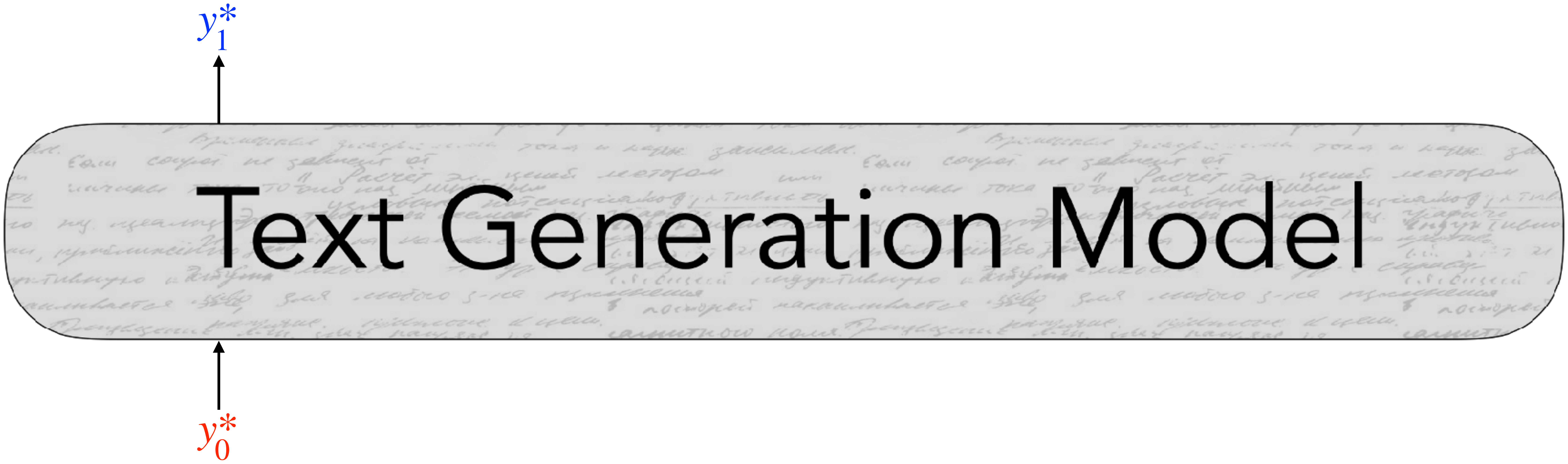
Remark:

- 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 $\{y^*\}_{<t}$

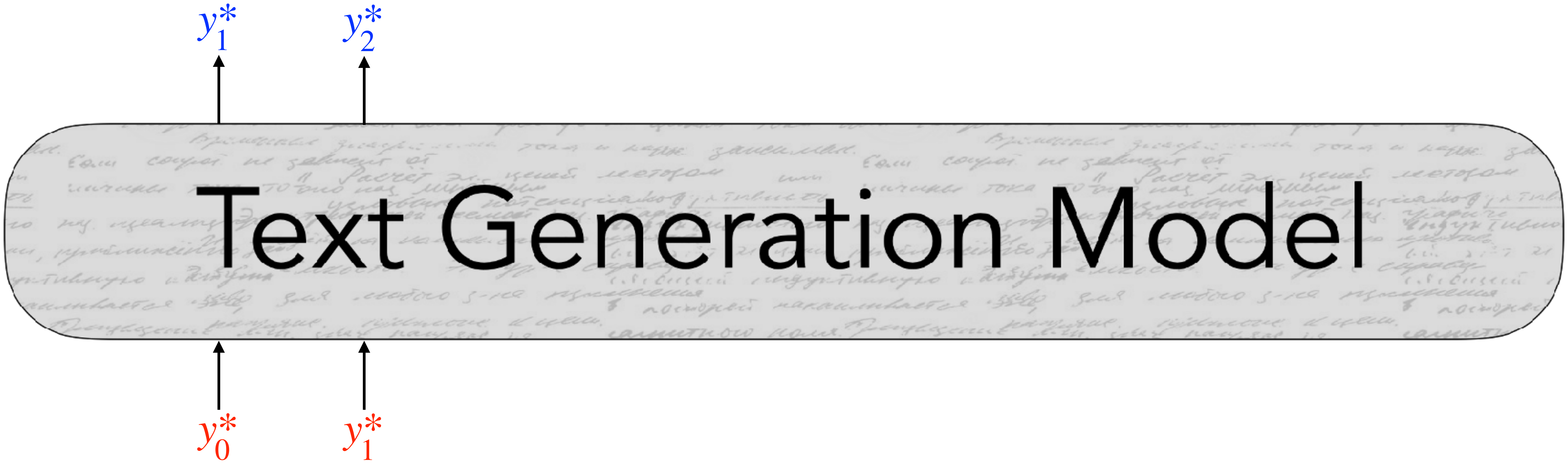
$$L = -\log P(y_1^* \mid y_0^*)$$



Recap: Maximum Likelihood Training

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

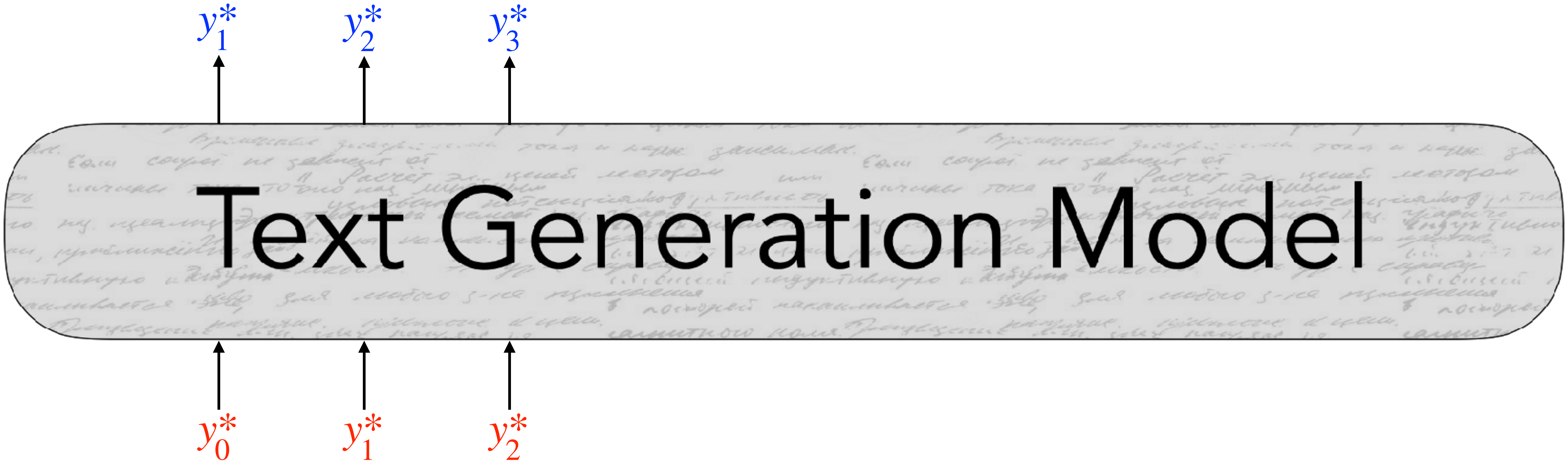
$$L = -\left(\log P(y_1^* \mid y_0^*) + \log P(y_2^* \mid y_0^*, y_1^*)\right)$$



Recap: Maximum Likelihood Training

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

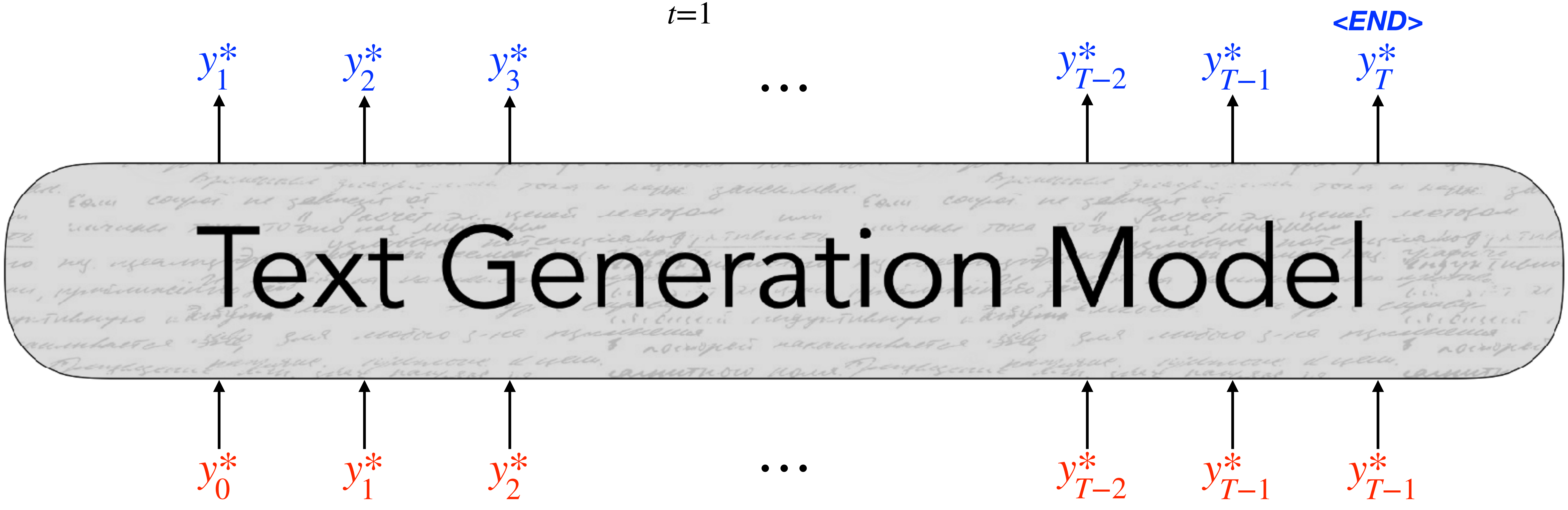
$$L = -\left(\log P(y_1^* \mid y_0^*) + \log P(y_2^* \mid y_0^*, y_1^*) + \log P(y_3^* \mid y_0^*, y_1^*, y_2^*)\right)$$



Recap: Maximum Likelihood Training

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

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



Decoding from LLMs

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

$$S = \underbrace{f(\{y_{<t}\}; \theta)}_{f(\cdot\,;\theta) \text{ is your model}}$$

- 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:

$$\hat{y}_t = \underbrace{g(P(y_t \mid \{y_{<t}\}))}_{g(\cdot) \text{ is your decoding algorithm}}$$

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

How to find the most likely text to generate?

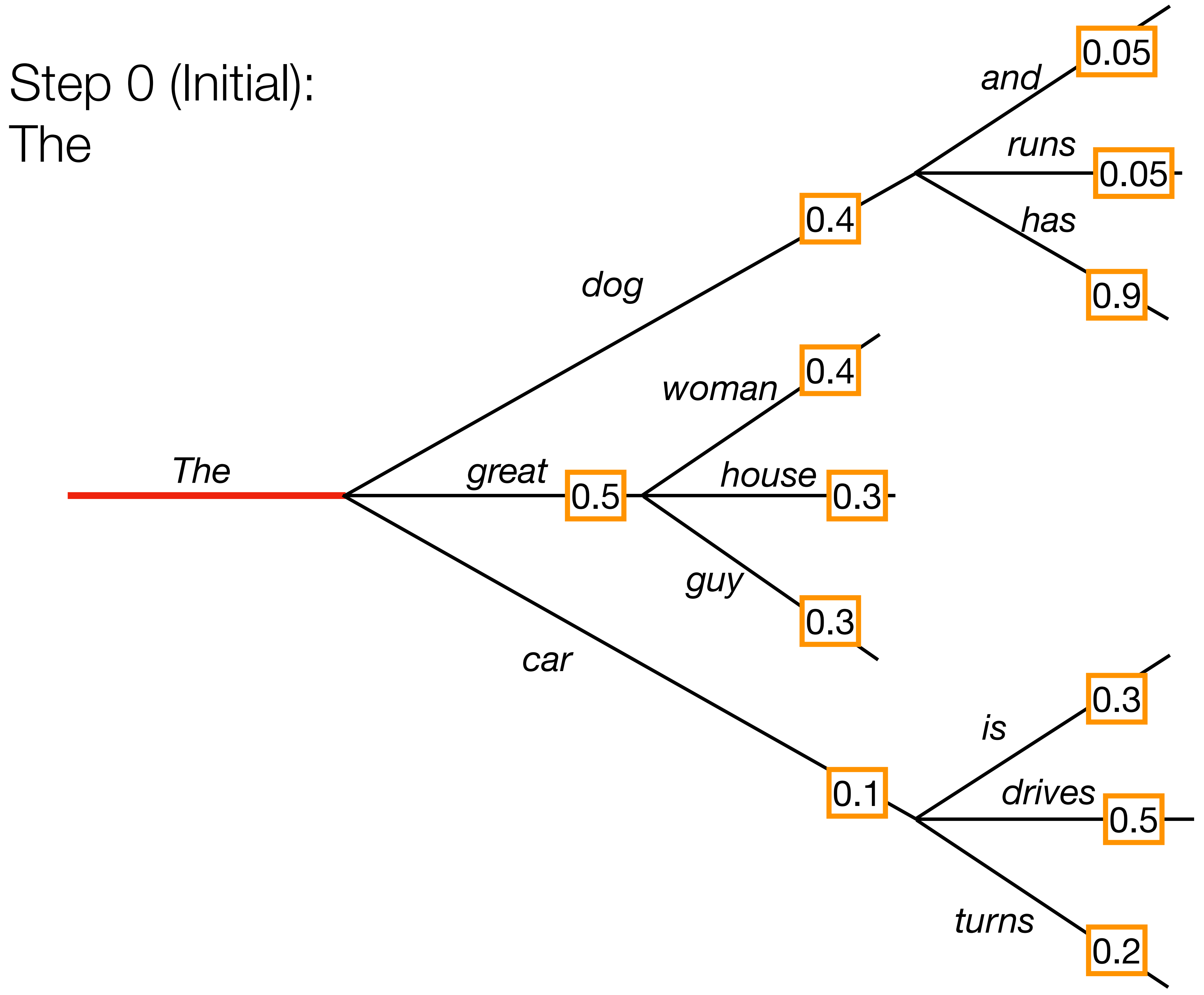
- **Obvious method: Greedy Decoding**
  - Selects the highest probability token according to $P(y_t | y_{<t})$

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

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

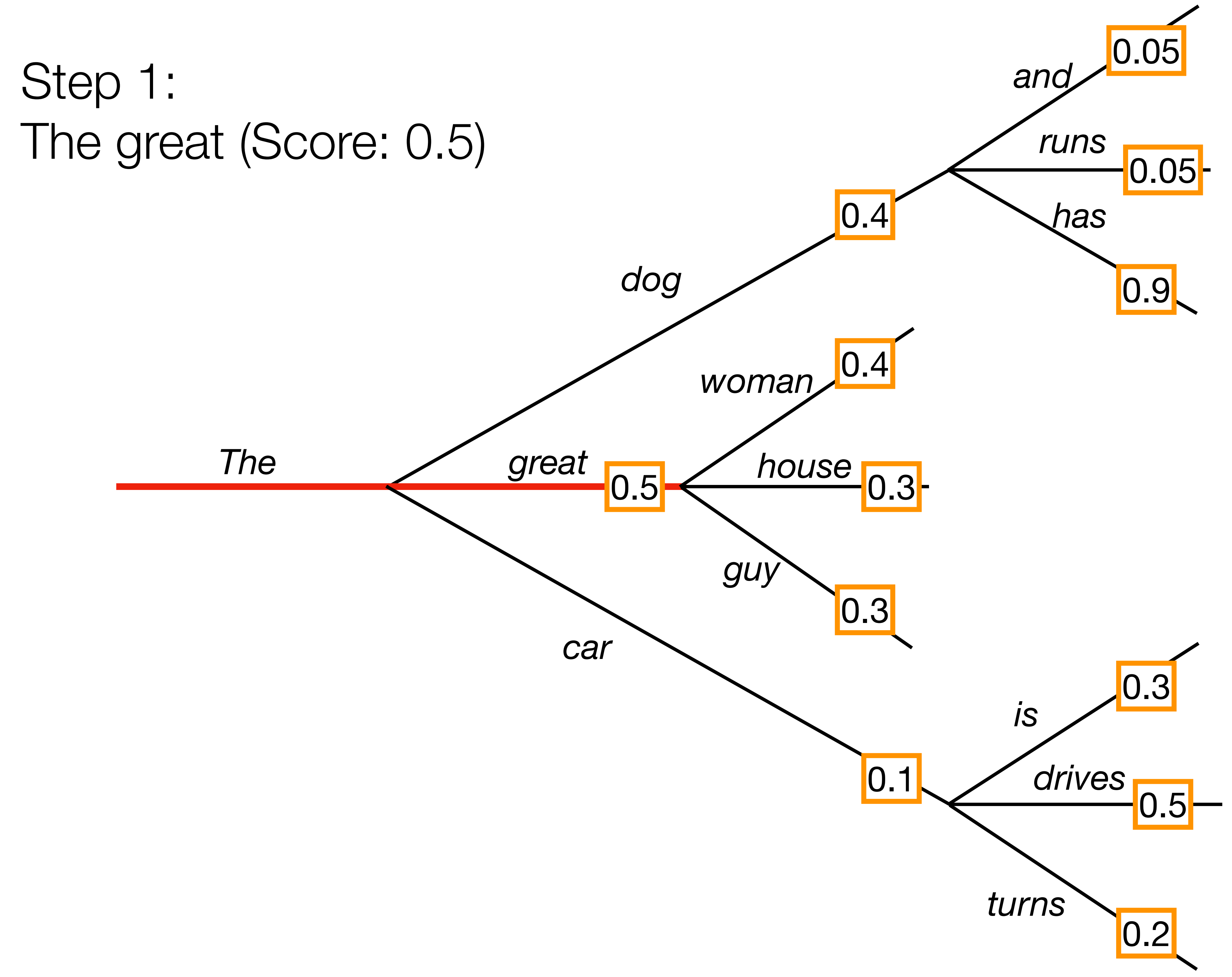
Greedy Decoding vs. Beam Search

- **Greedy Decoding**
  - Choose the "currently best" token at each time step



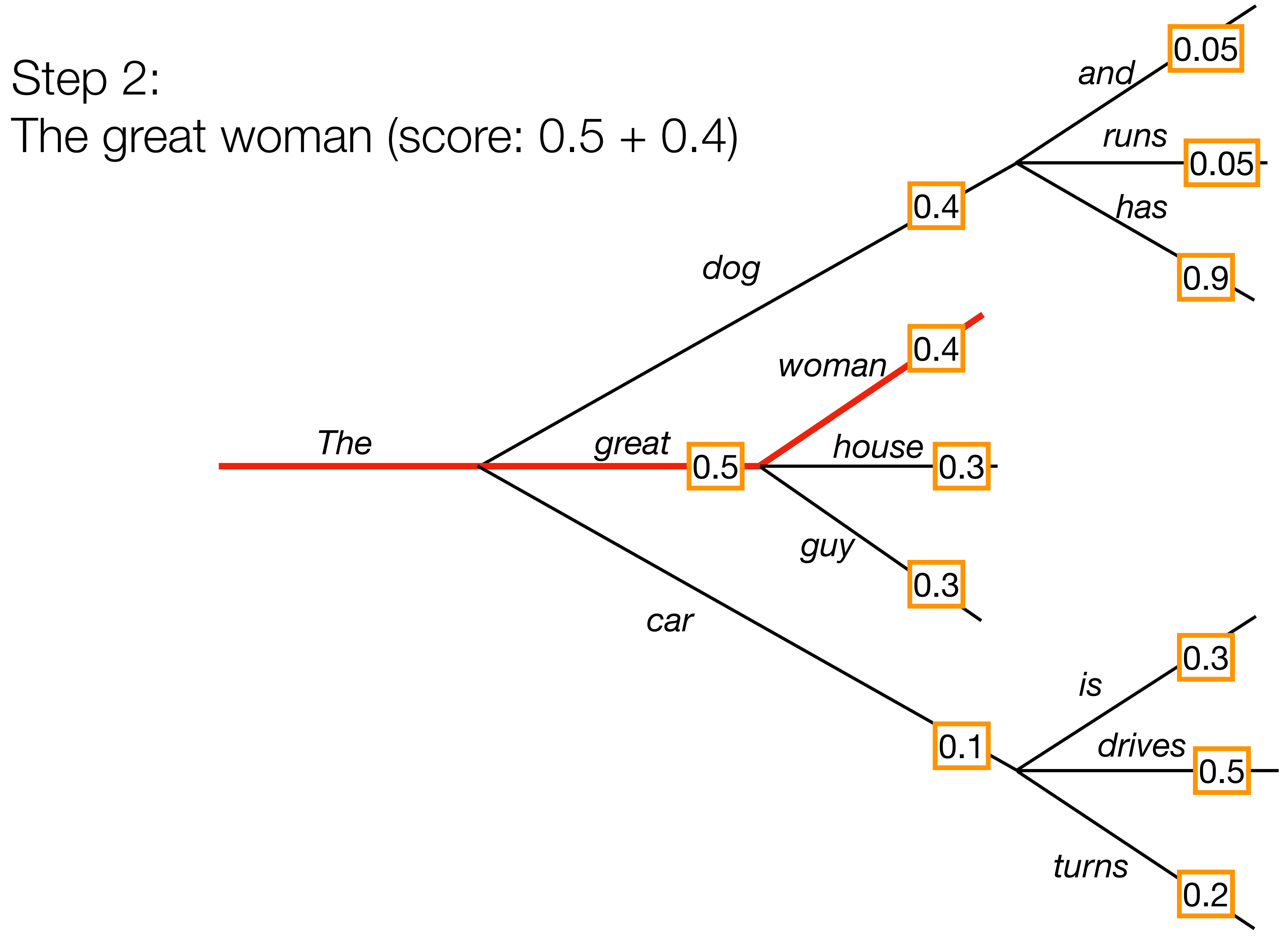
Greedy Decoding vs. Beam Search

- **Greedy Decoding**
  - Choose the "currently best" token at each time step



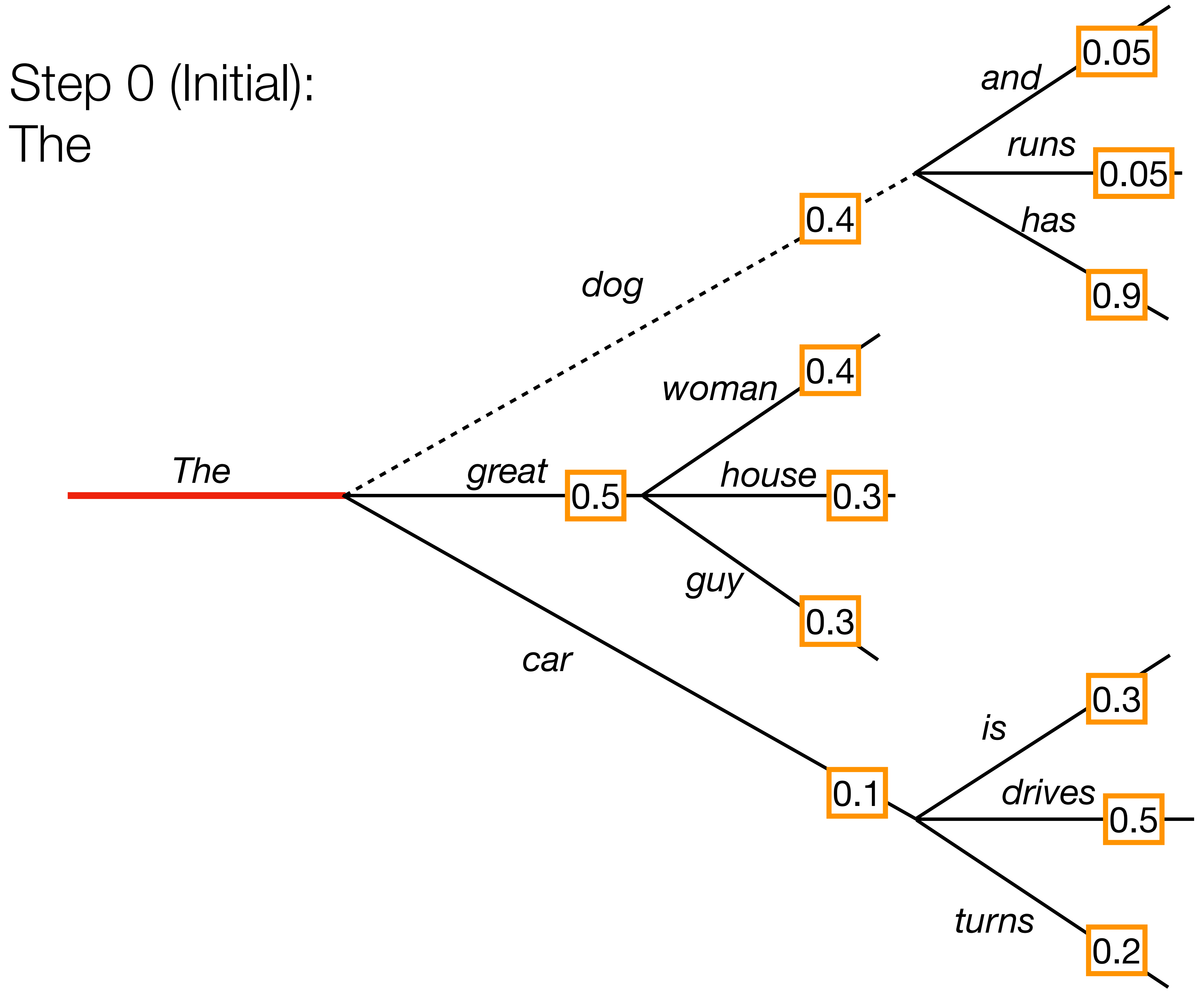
Greedy Decoding vs. Beam Search

- **Greedy Decoding**
  - Choose the "currently best" token at each time step



Greedy Decoding vs. Beam Search

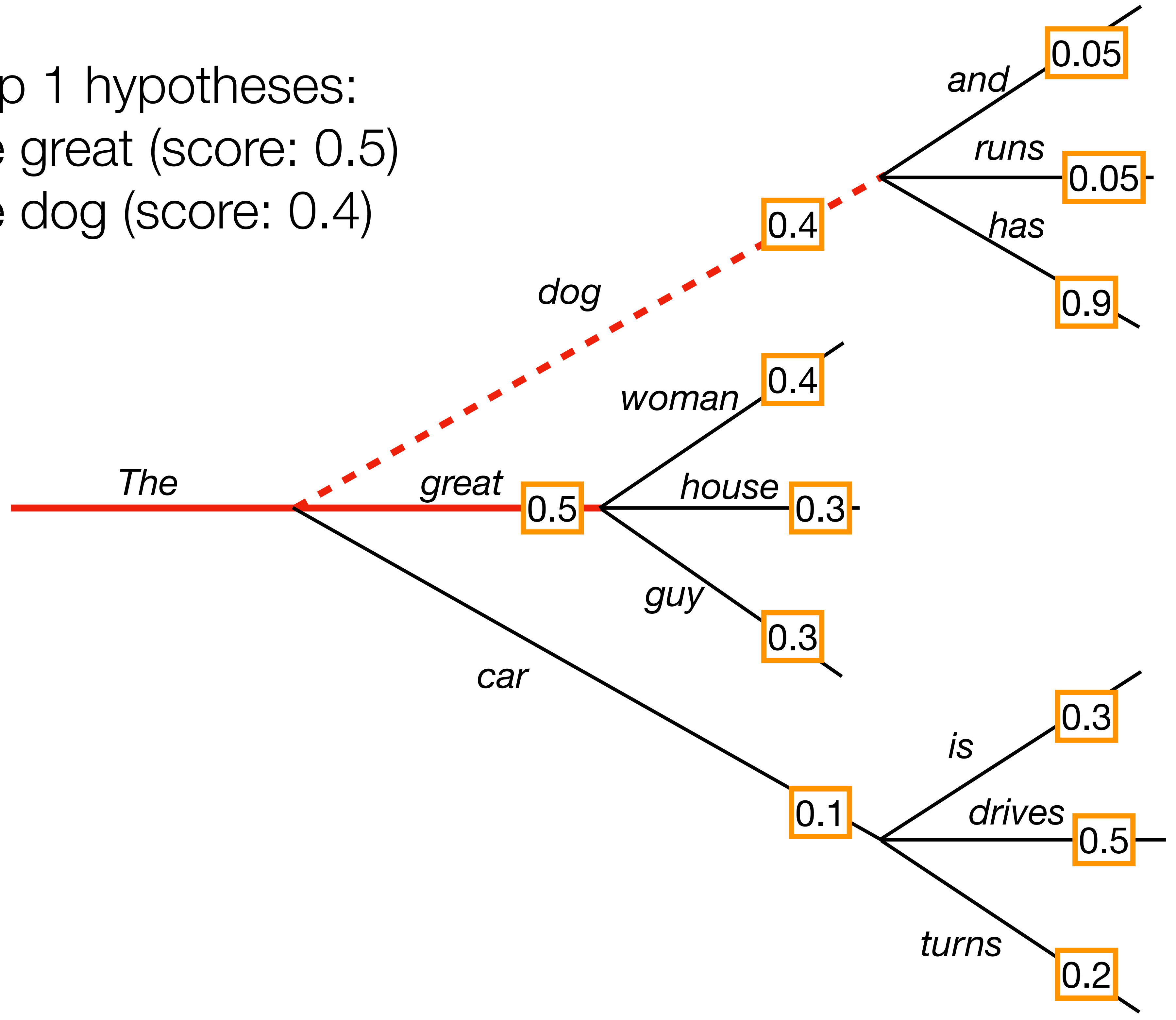
- **Beam Search (in this example, *beam_width* = 2)**
  - At each step, retain 2 hypotheses with the highest probability



Greedy Decoding vs. Beam Search

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

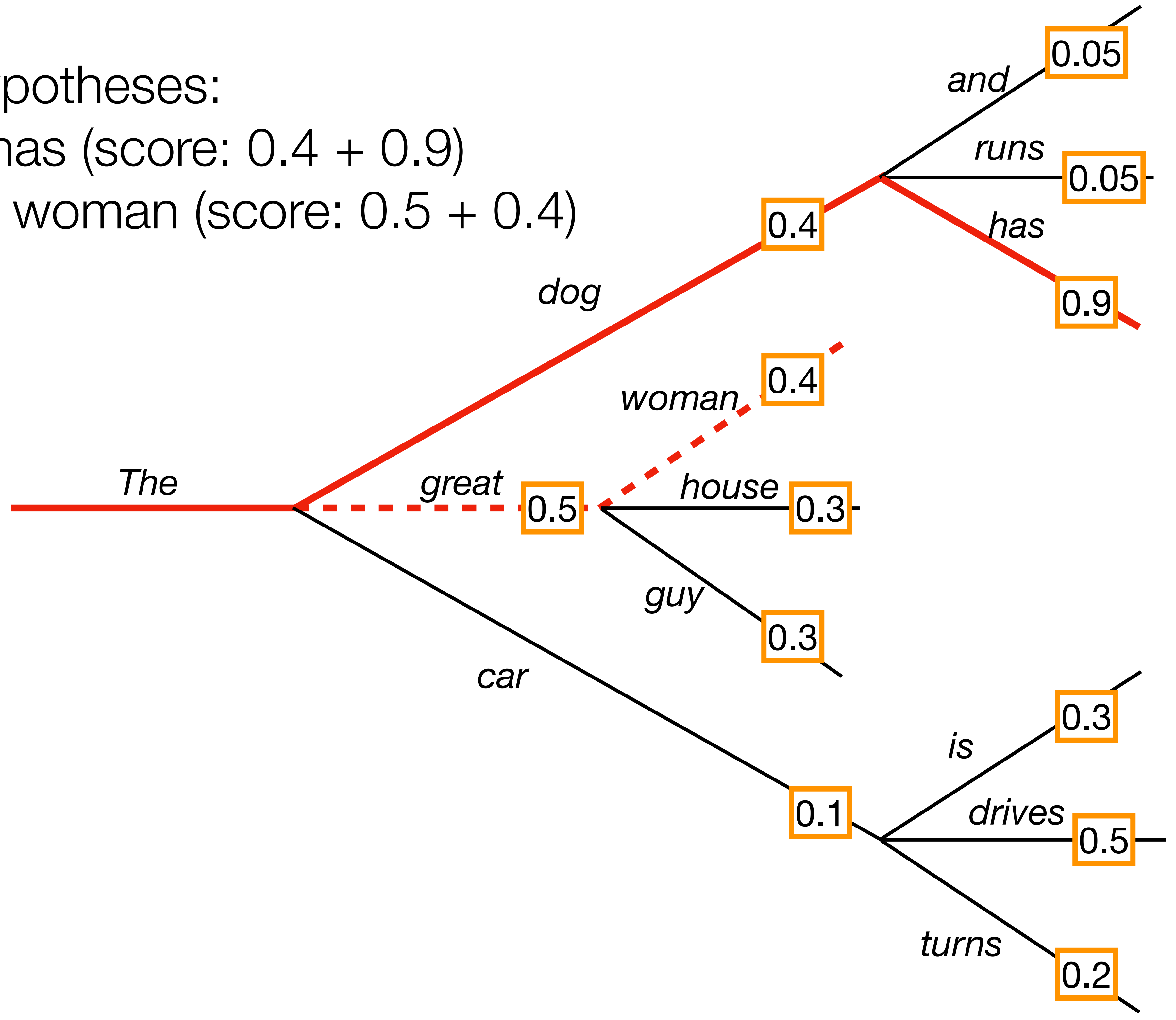
Step 1 hypotheses:
The great (score: 0.5)
The dog (score: 0.4)



Greedy Decoding vs. Beam Search

- **Beam Search (in this example, *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?

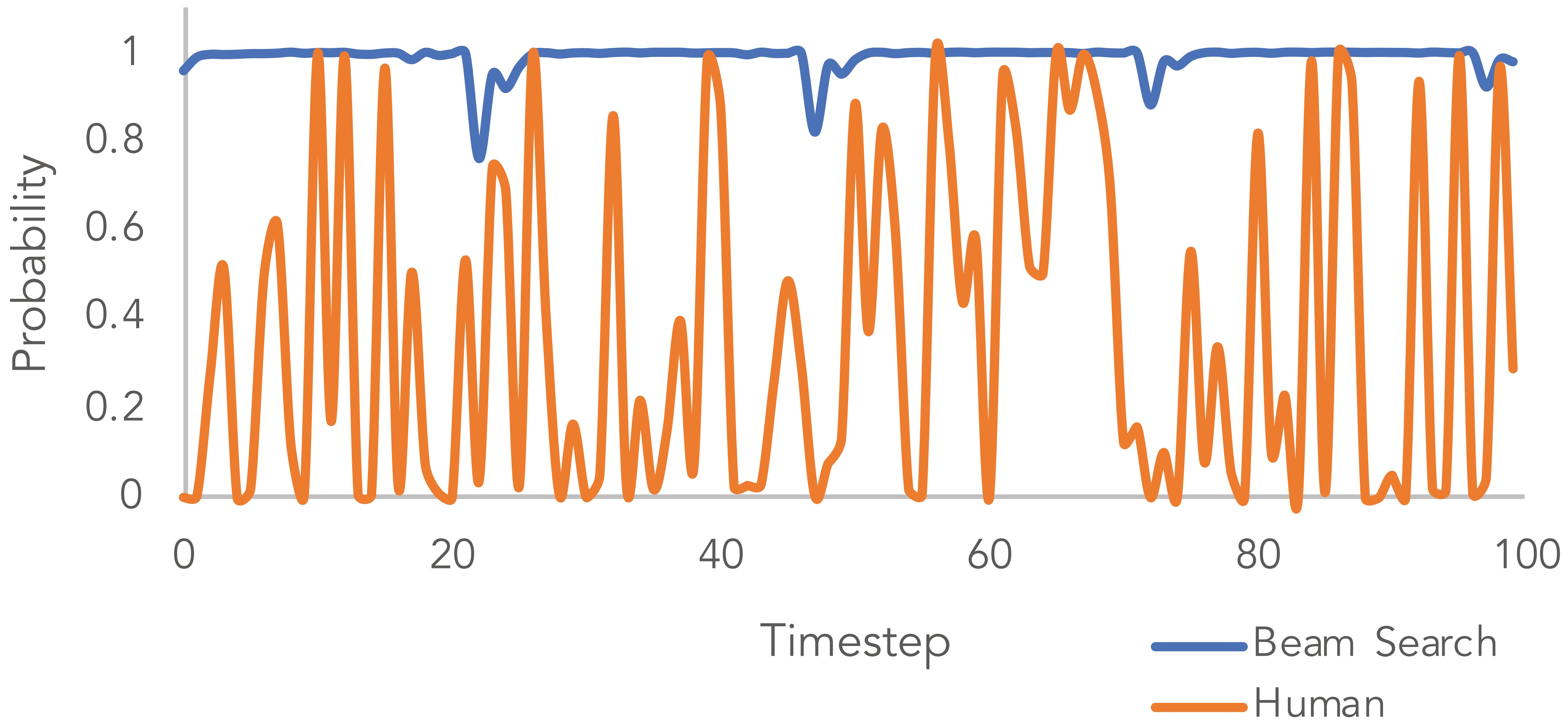
- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.
  - Compared to greedy decoding, beam search gives a better approximation of brute-force search over all sequences
  - A small overhead in computation due to beam width
    Time complexity: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

Greedy decoding for open-ended generation?



Beam Search

...to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and...

Human

...which grant increased life span and three years warranty. The Antec HCG series consists of five models with capacities spanning from 400W to 900W. Here we should note that we have already tested the HCG-620 in a previous review and were quite satisfied With its performance. In today's review we will rigorously test the Antec HCG-520, which as its model number implies, has 520W capacity and contrary to Antec's strong beliefs in multi-rail PSUs is equipped...

The probability assigned to tokens generated by Beam Search and humans, given the same context.

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

(Holtzman et al. ICLR 2020)

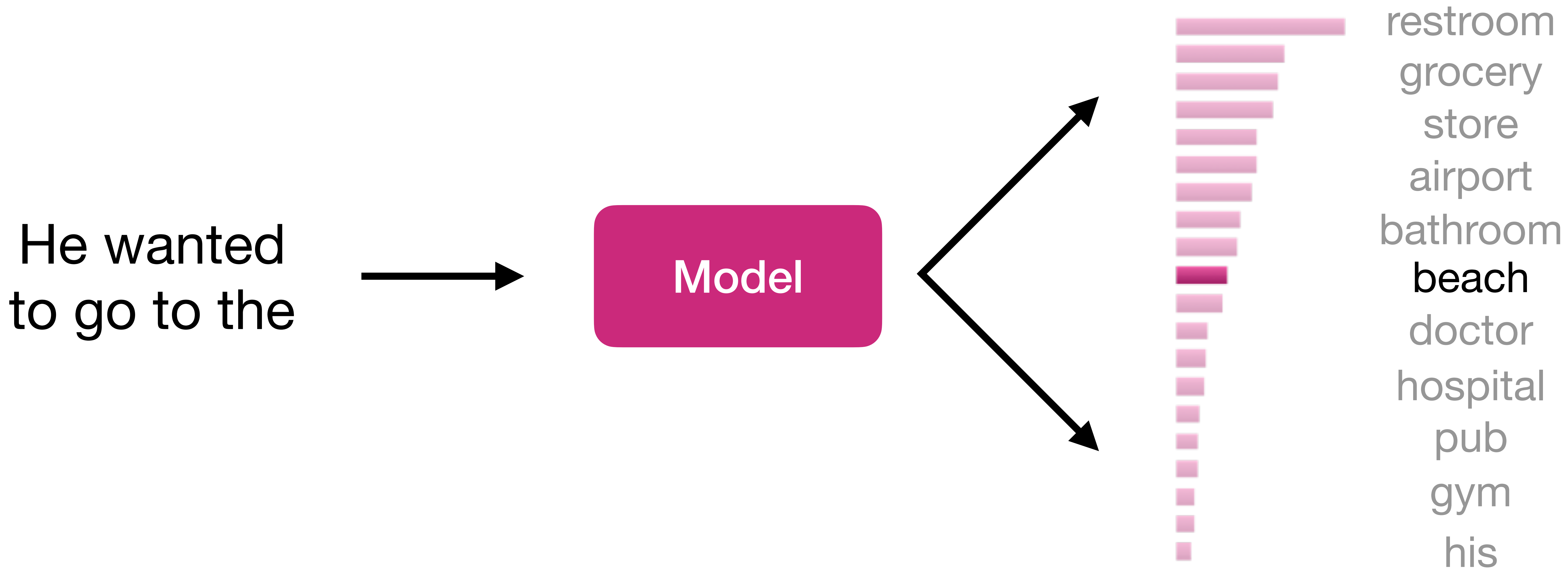
Sampling generation from LLMs

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.



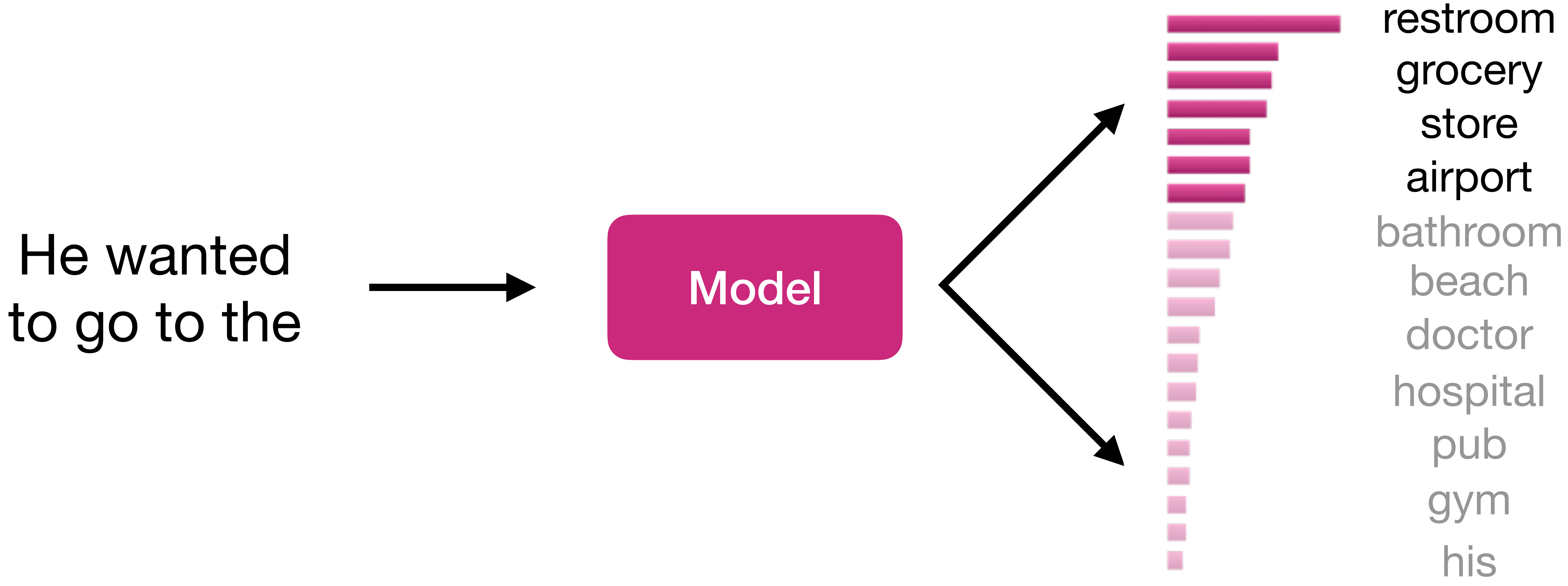
Decoding: Top-k Sampling

- 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.

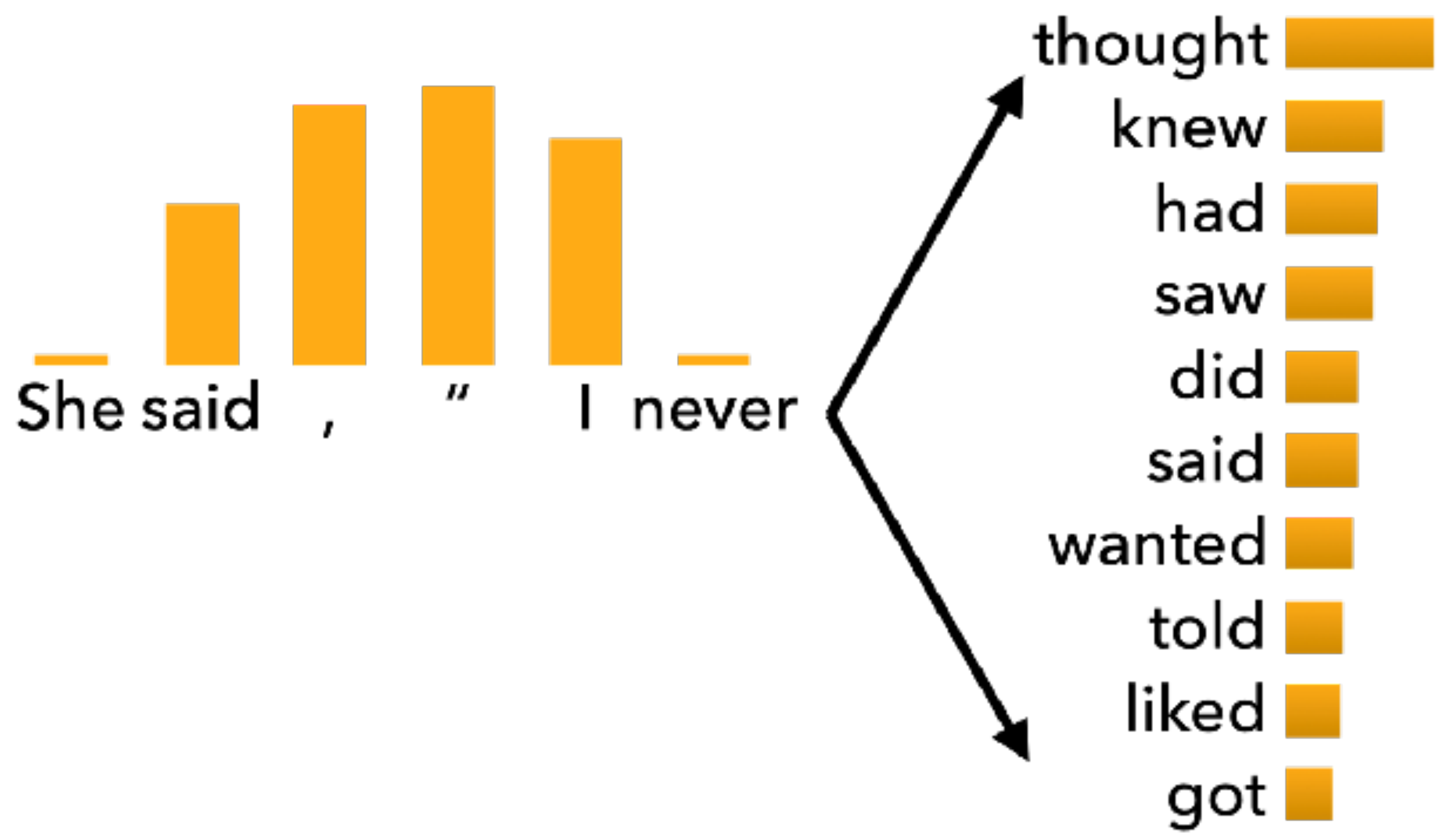
Decoding: Top-k Sampling

- <u>Solution:</u> 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!*)



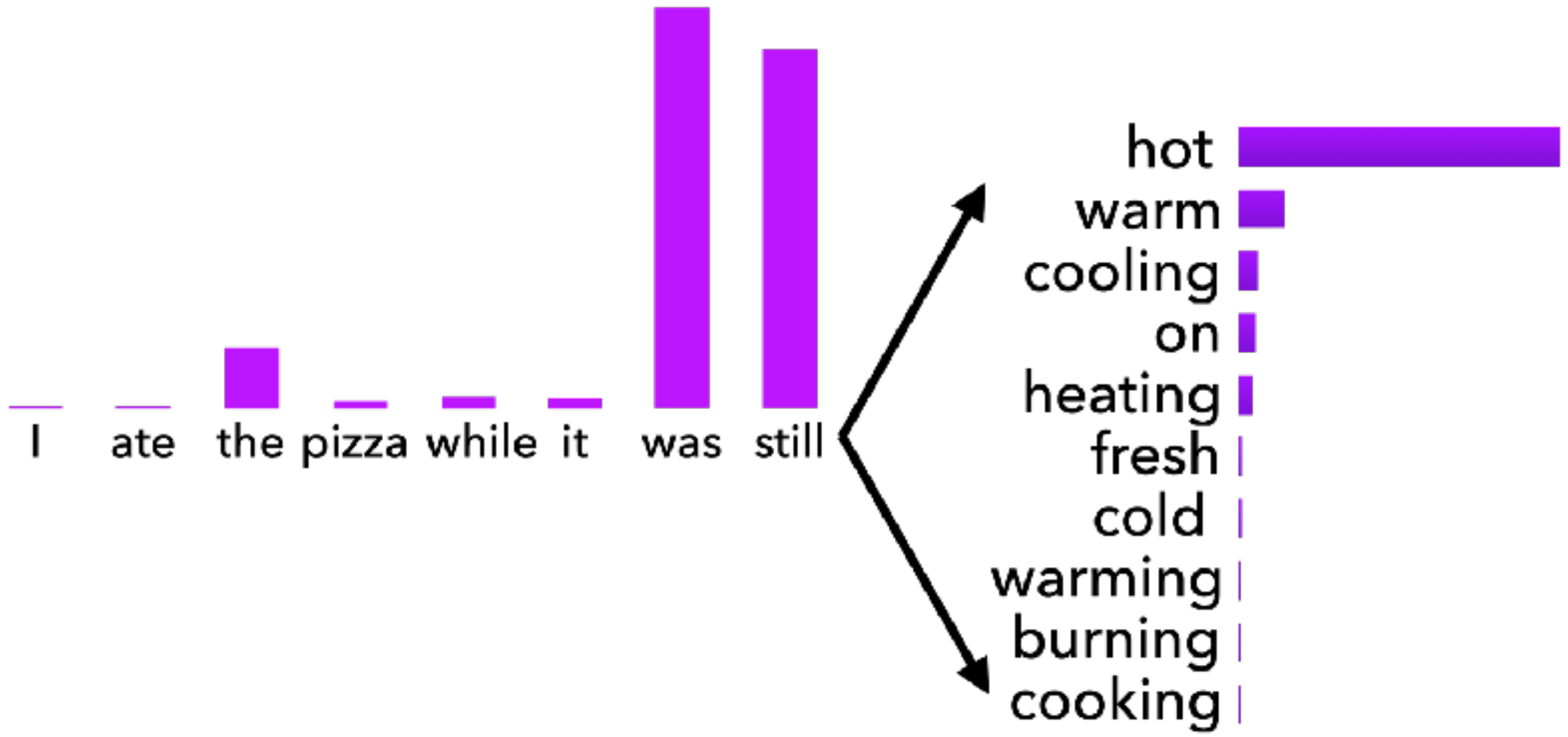
- Increasing *k* yields more **diverse**, but **risky** outputs
- Decreasing *k* yields more **safe** but **generic** outputs

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**!

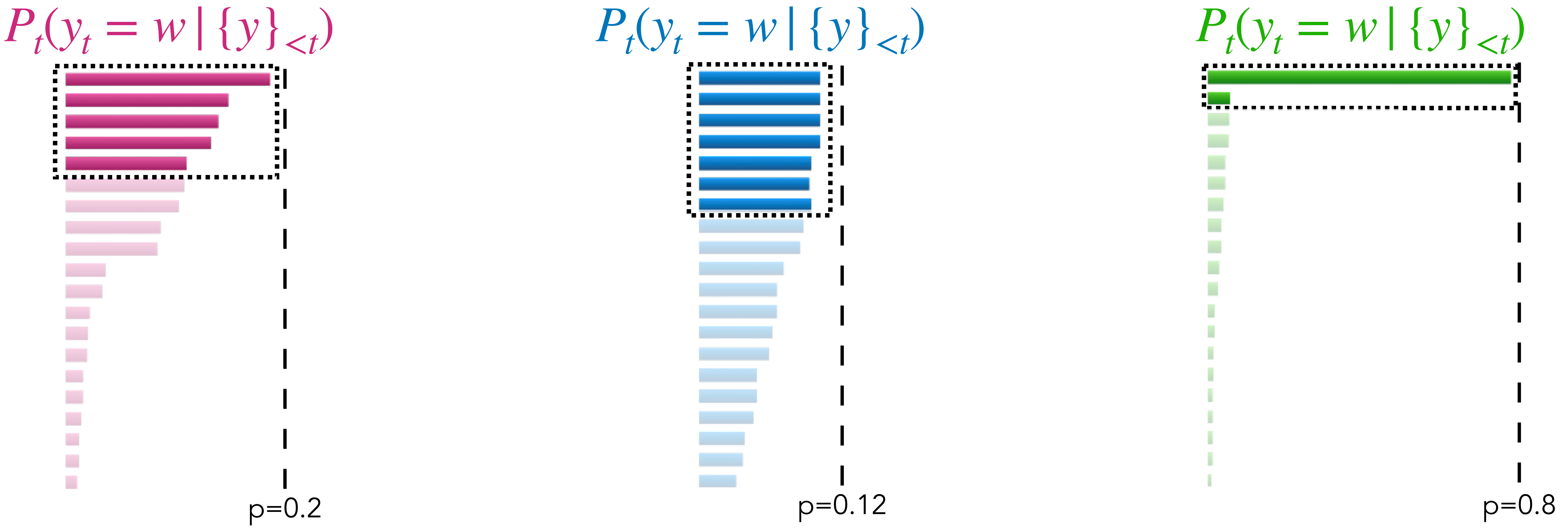
Decoding: Top-p (Nucleus) Sampling

- 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



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:

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

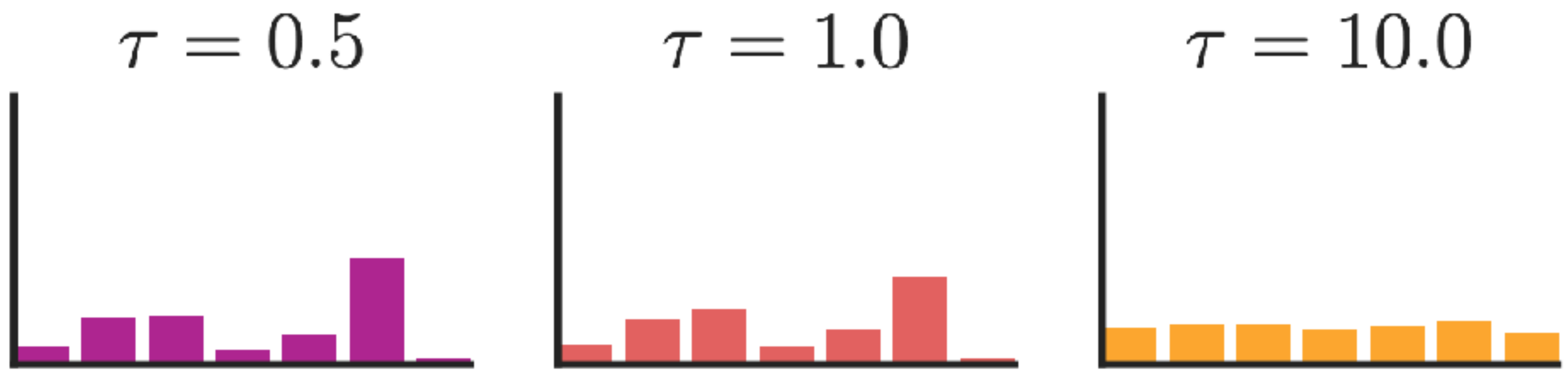
- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)

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.

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

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!

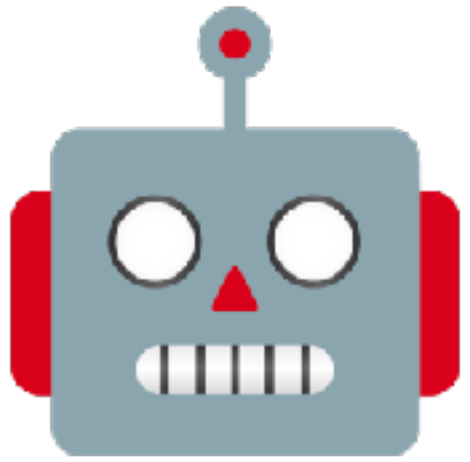
Score:

0.61 — Yes for sure!

0.25 — Sure I do!

False negative — 0.0 — Yes!

False positive — 0.61 — No for sure...

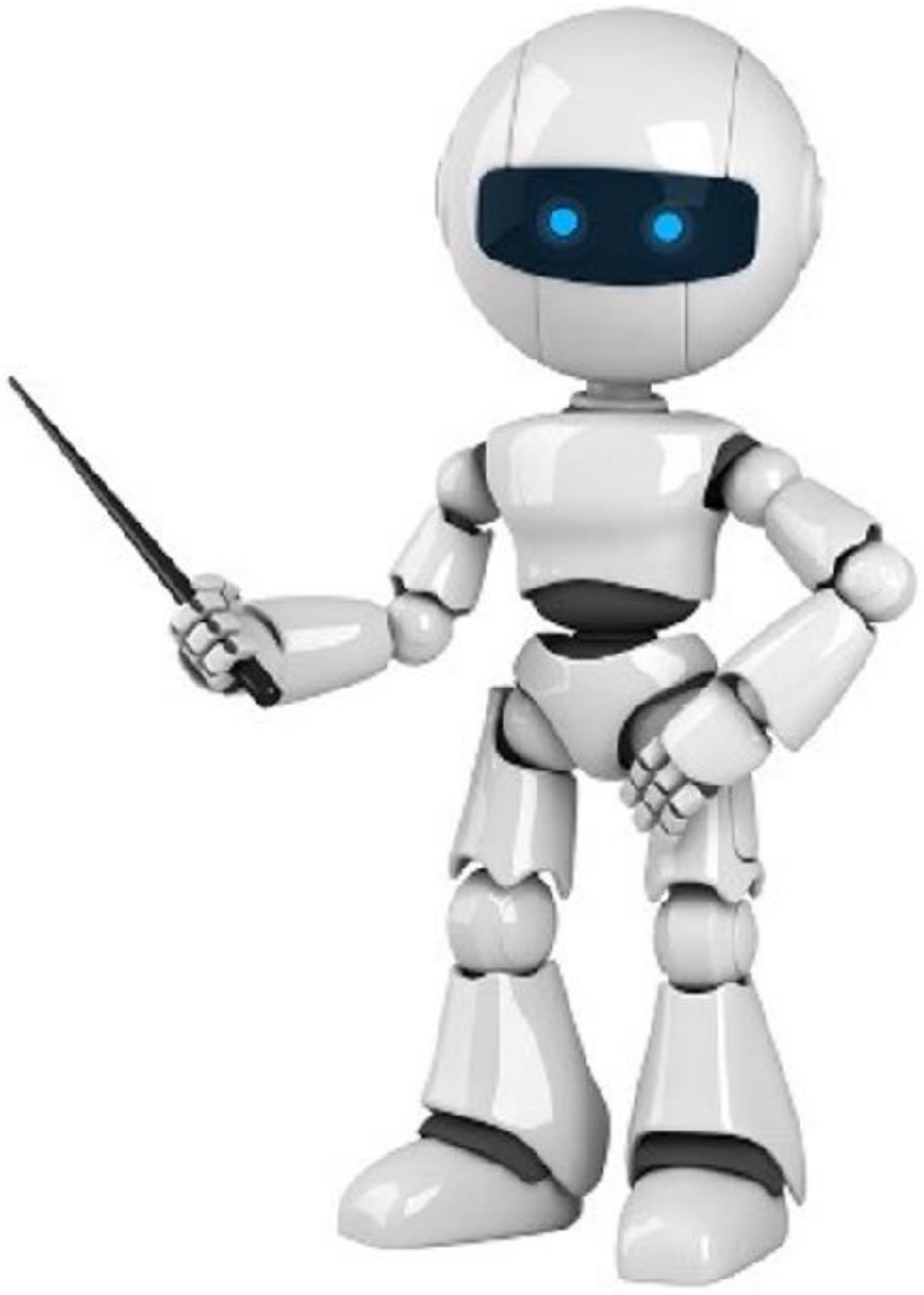


Model-based metrics to capture more semantics

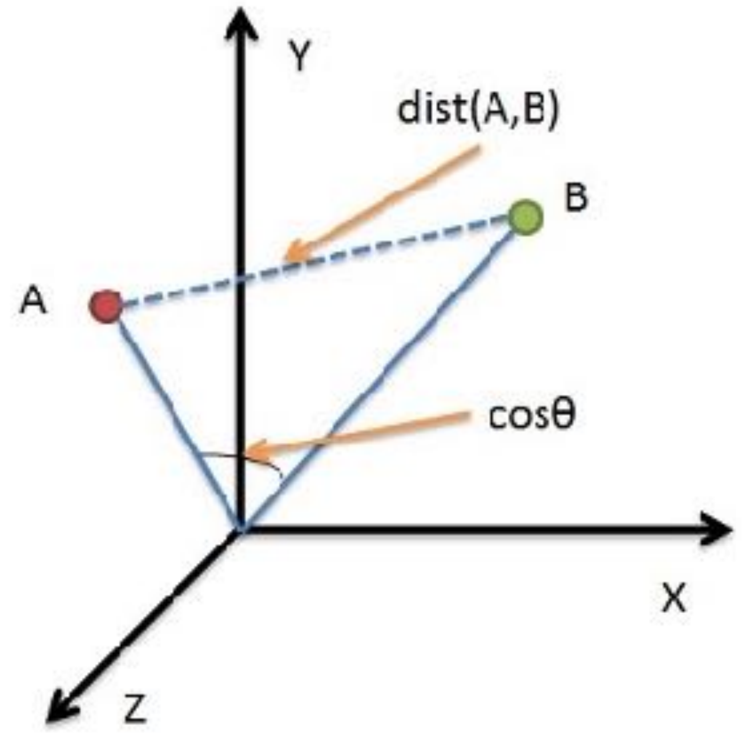
- Use learned representation of words and sentences to compute semantic similarity between generated and reference texts

- 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.



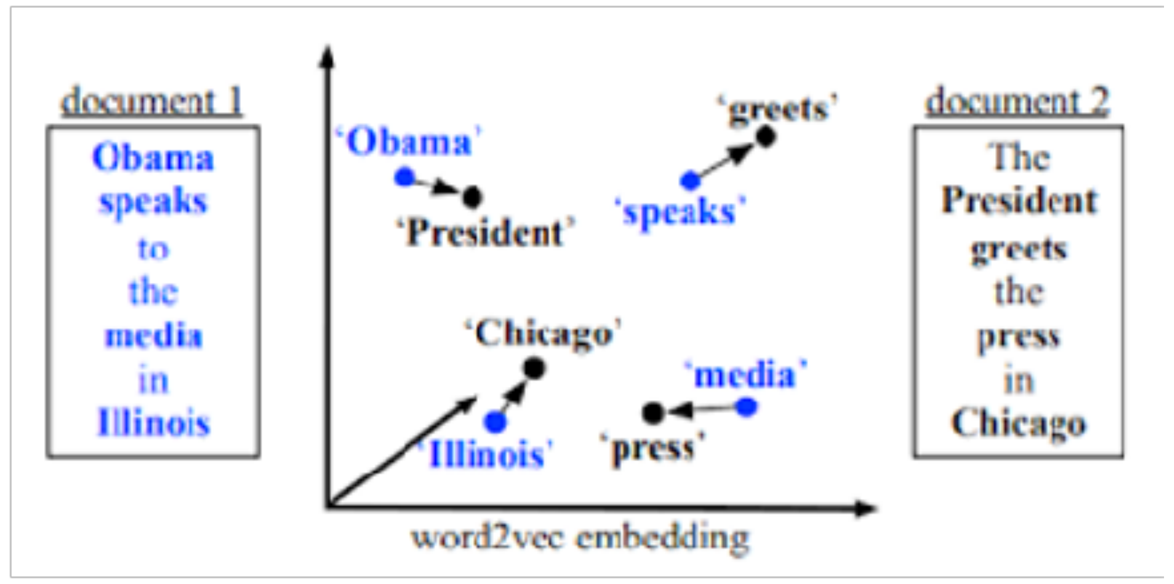
Model-based metrics: Word distance functions



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

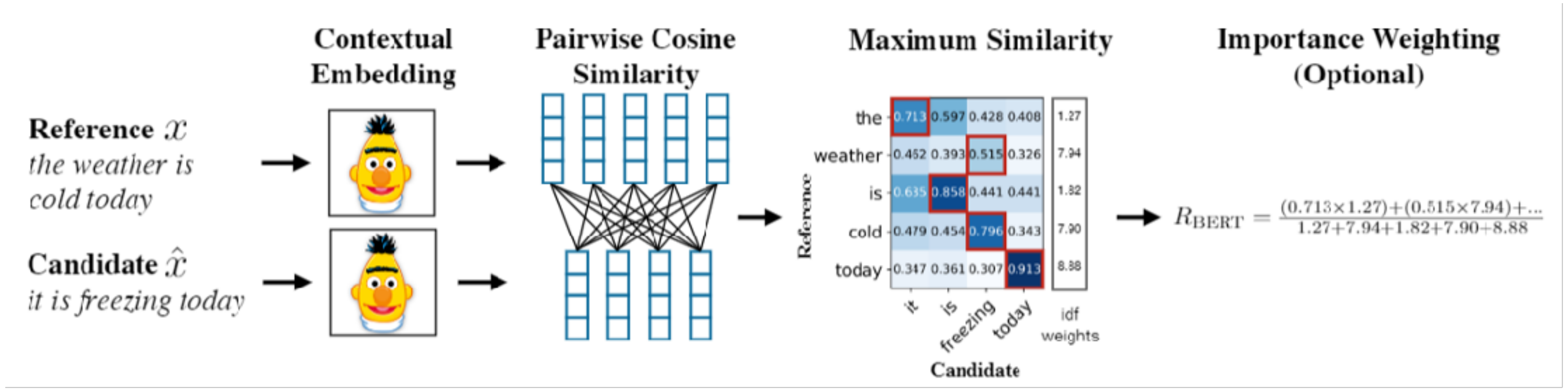
Measures the distance between two sequences using word embedding similarity matching.

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

BERTSCORE

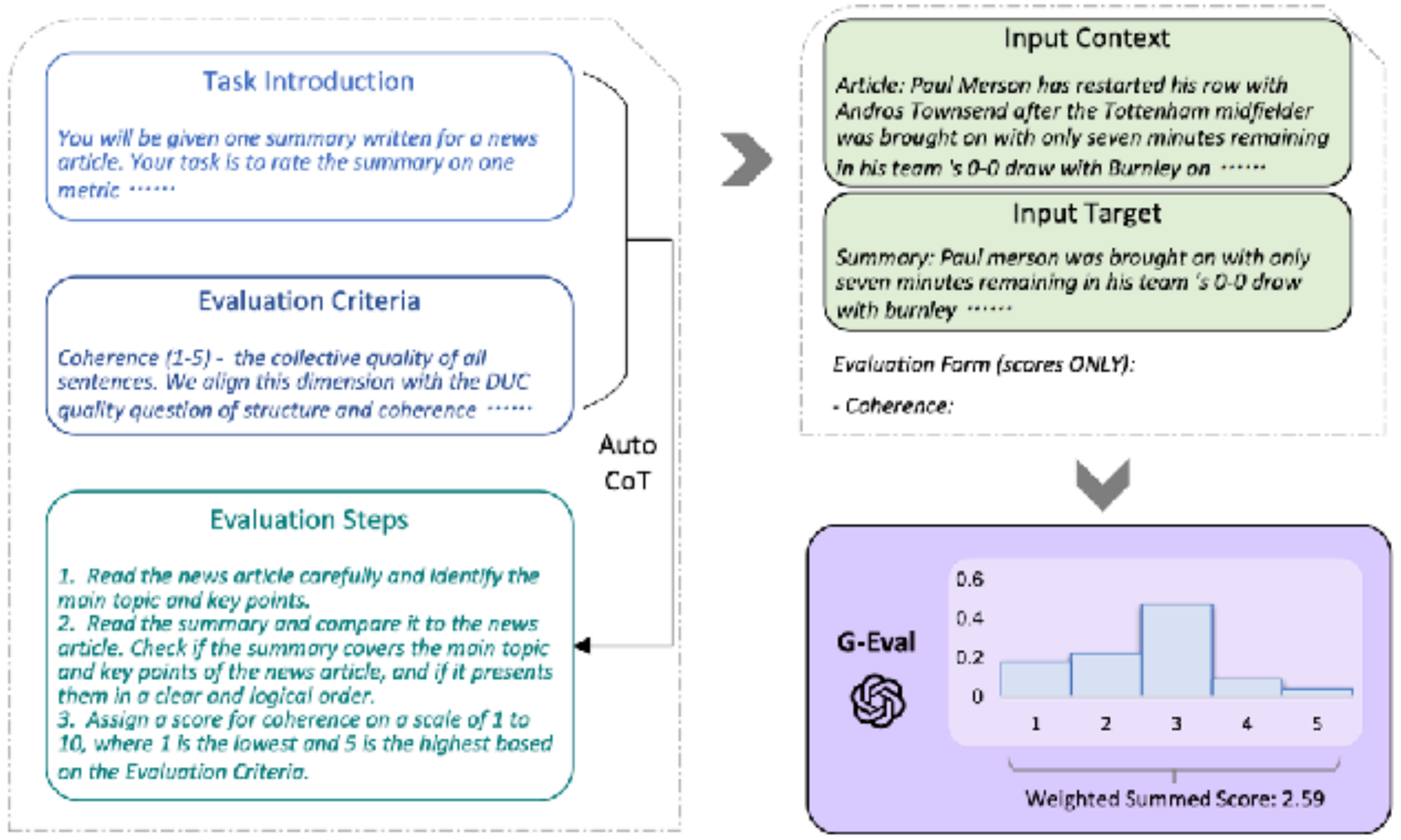
Uses pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity.

- *(Zhang et al., 2019)*



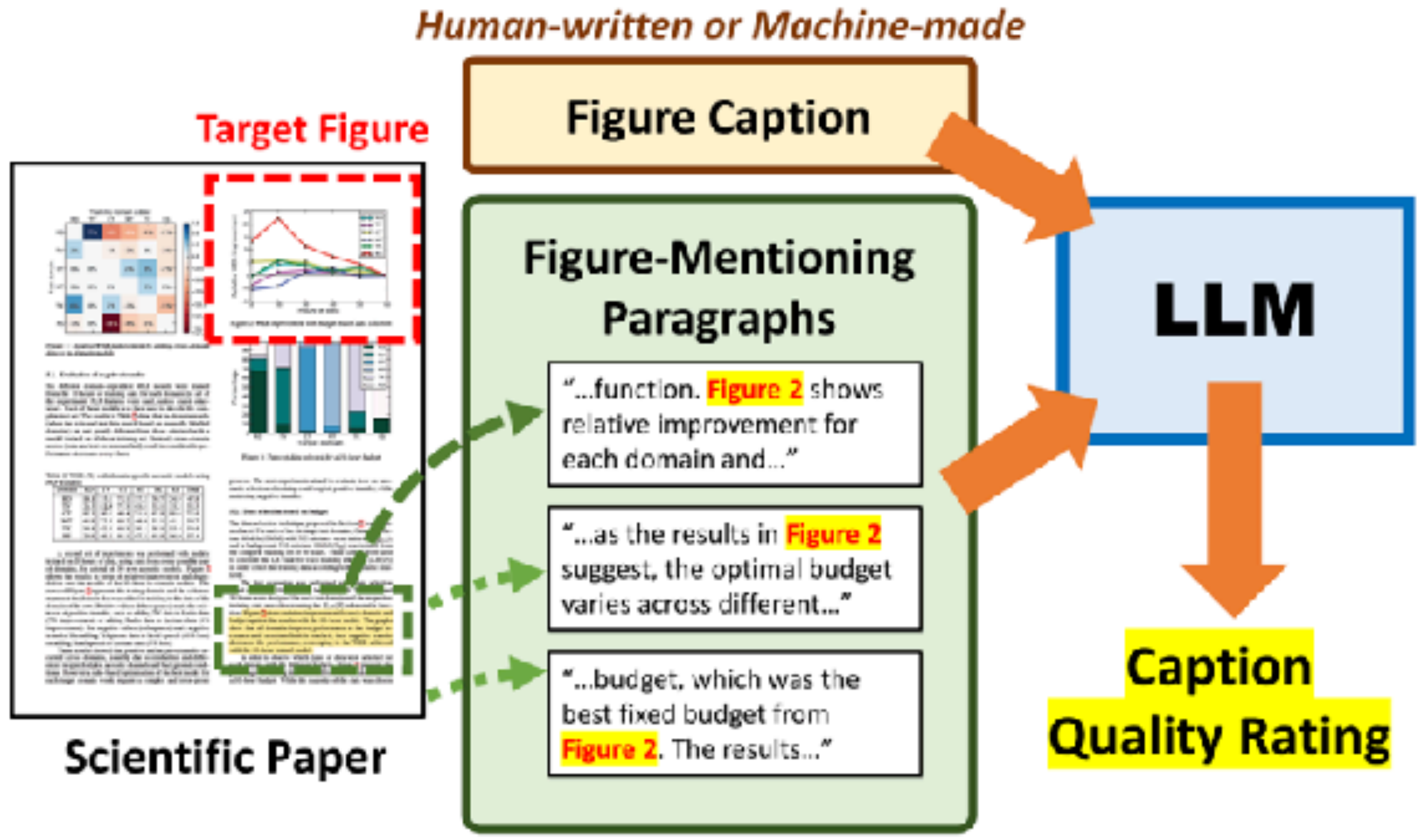
Model-based metrics: LLM as evaluator

- 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.lastnames@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

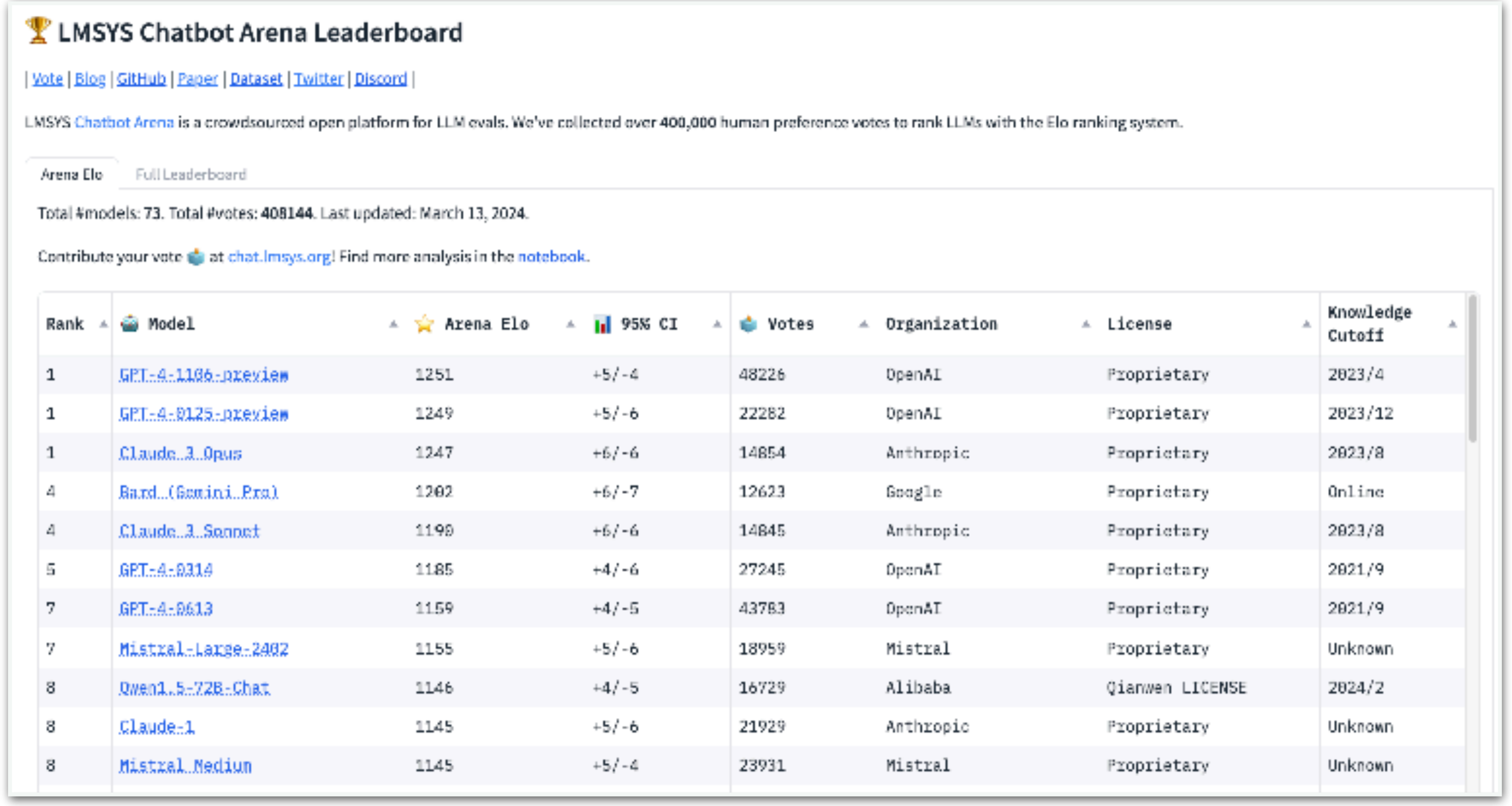
- *Model-based metrics* can be more correlated with human judgment, but often are not interpretable

- Human judgments are critical
  - But humans are inconsistent!

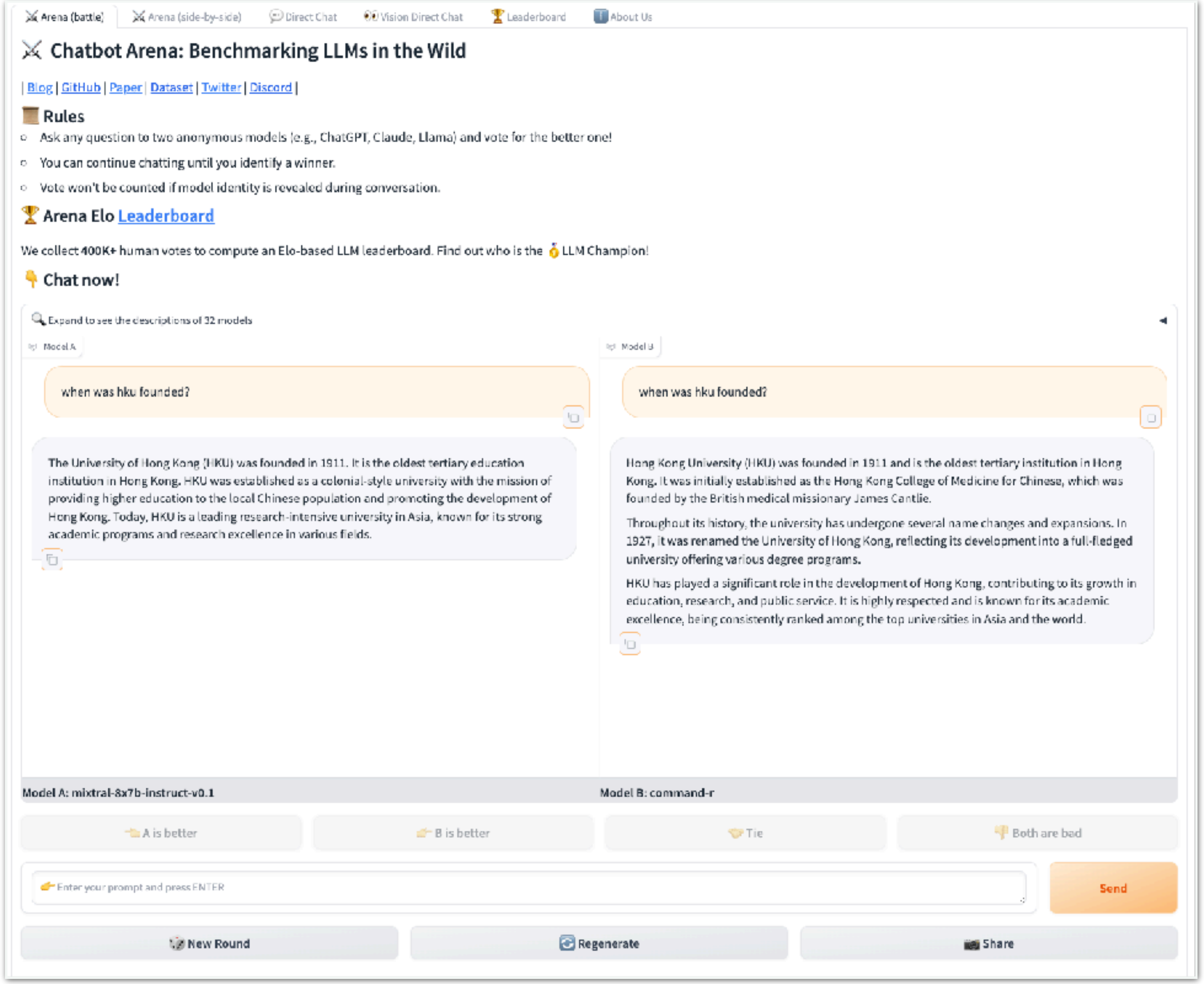
- In many cases, the best judge of output quality is **YOU**!
  - **Look at the actual generations - don't just rely on numbers.**
  - **Publicly release large samples of outputs from your system!**

LLM evaluation: Chatbot Arena

- Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference
- https://arena.lmsys.org



Rank	Model	Arena Elo	95% CI	Votes	Organization	License	Knowledge Cutoff
1	GPT-4-1106-preview	1251	+5/-4	48226	OpenAI	Proprietary	2023/4
1	GPT-4-0125-preview	1249	+5/-6	22282	OpenAI	Proprietary	2023/12
1	Claude 3 Opus	1247	+6/-6	14854	Anthropic	Proprietary	2023/8
4	Bard (Gemini Pro)	1202	+6/-7	12623	Google	Proprietary	Online
4	Claude 3 Sonnet	1190	+6/-6	14845	Anthropic	Proprietary	2023/8
5	GPT-4-0314	1185	+4/-6	27245	OpenAI	Proprietary	2021/9
7	GPT-4-0613	1159	+4/-5	43783	OpenAI	Proprietary	2021/9
7	Mistral-Large-2402	1155	+5/-6	18959	Mistral	Proprietary	Unknown
8	Qwen1.5-72B-Chat	1146	+4/-5	16729	Alibaba	Qianwen LICENSE	2024/2
8	Claude-1	1145	+5/-6	21929	Anthropic	Proprietary	Unknown
8	Mistral Medium	1145	+5/-4	23931	Mistral	Proprietary	Unknown



https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard