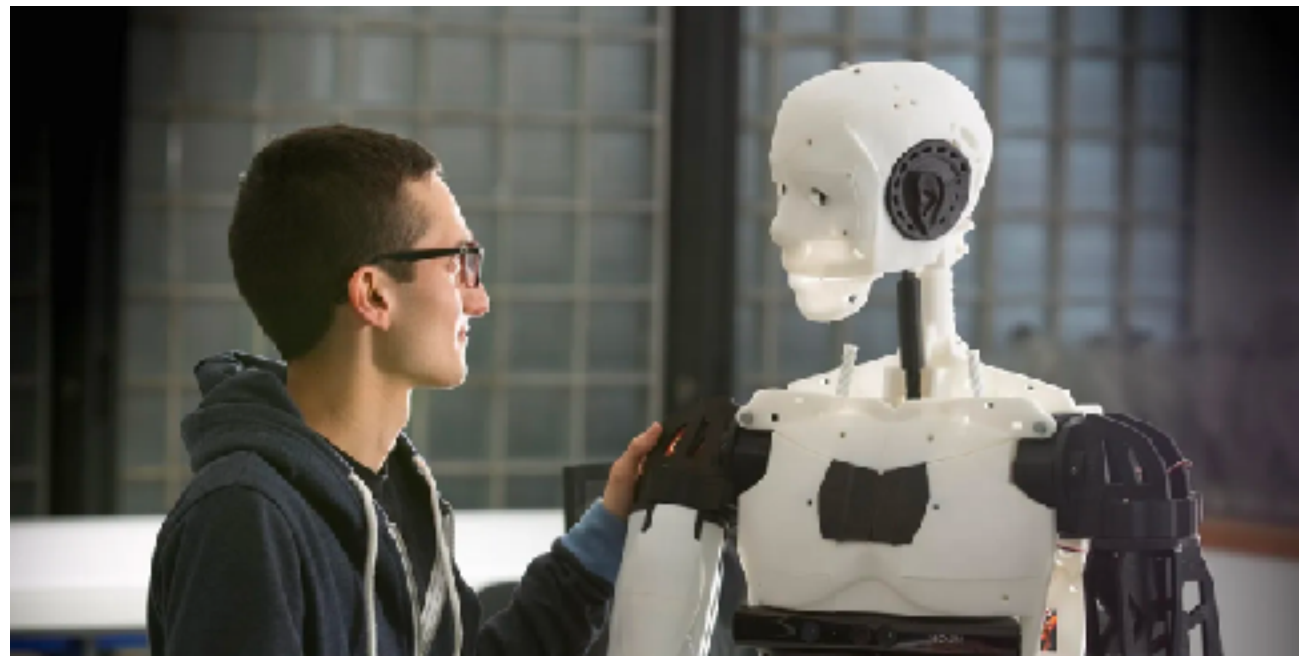
Natural Language Generation Algorithms

CS6120: Natural Language Processing
Northeastern University

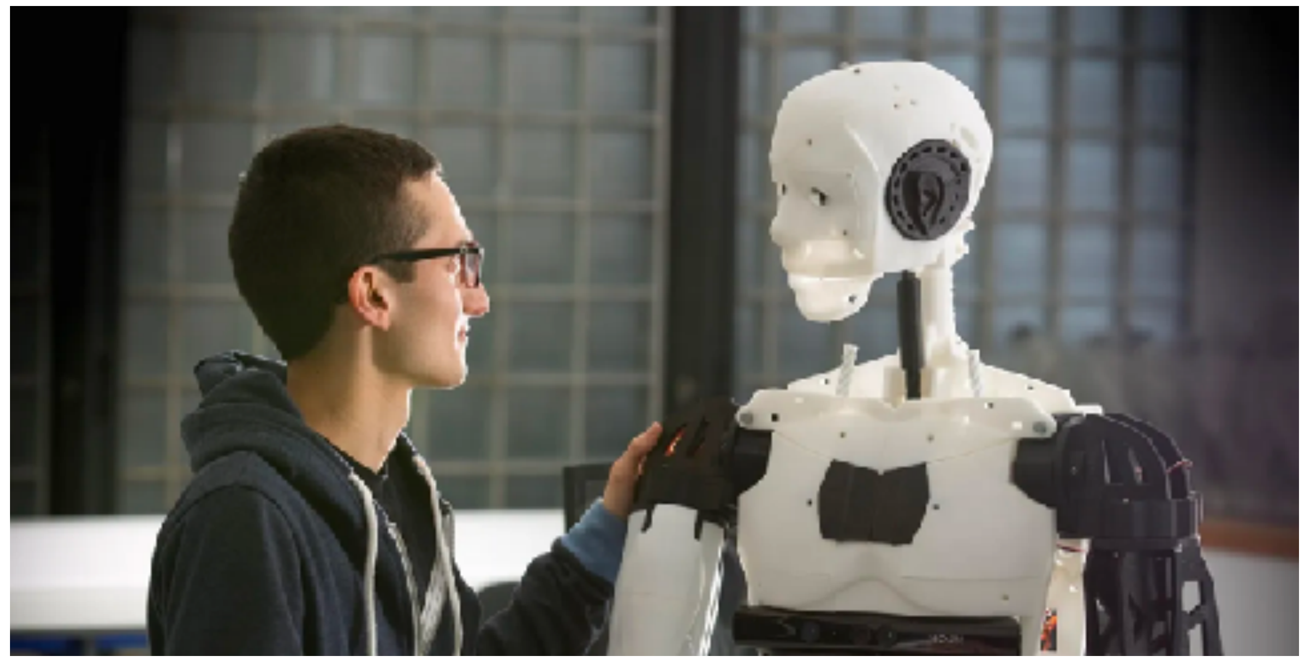
David Smith
with slides from Yejin Choi, Antoine Bosselut, Xiang Lisa Li, Chris Manning

What is natural language generation?



What is natural language generation?

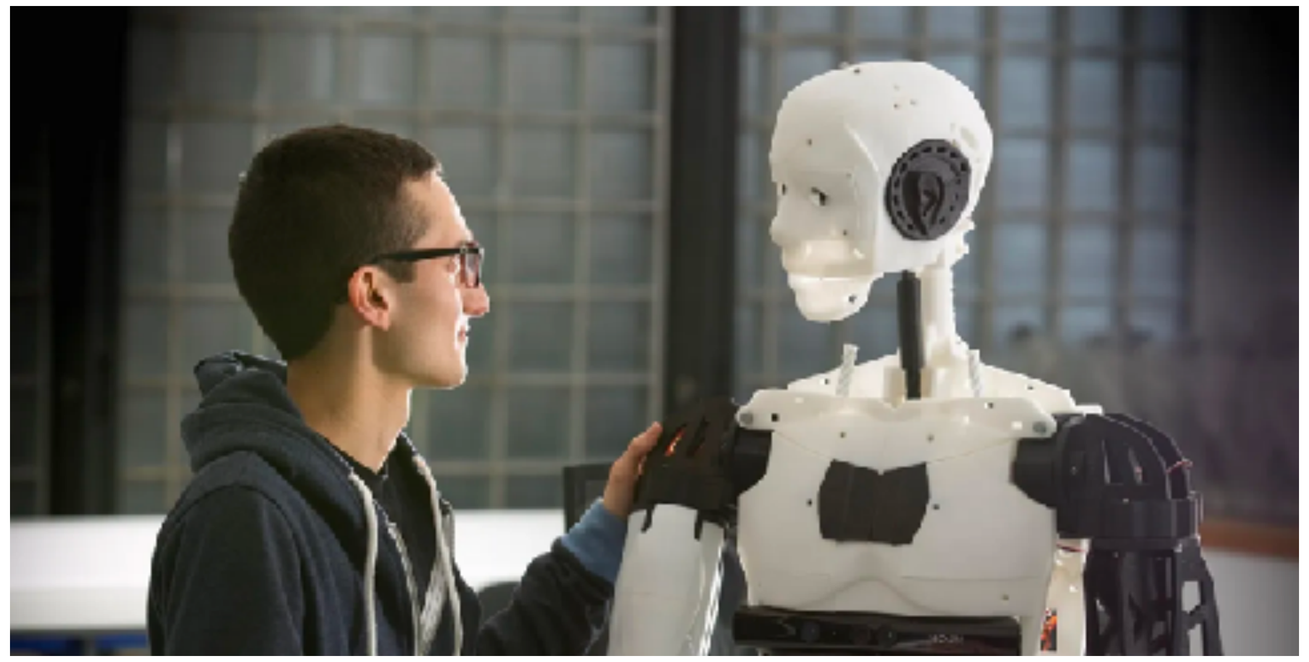
- One view: NLP =
  natural language understanding (NLU) +
  natural language generation (NLG)



What is natural language generation?

- One view: NLP =
  natural language understanding (NLU) +
  natural language generation (NLG)

- Focused on building systems that automatically produce **coherent** and **useful** text for human consumption



© University of Lincoln

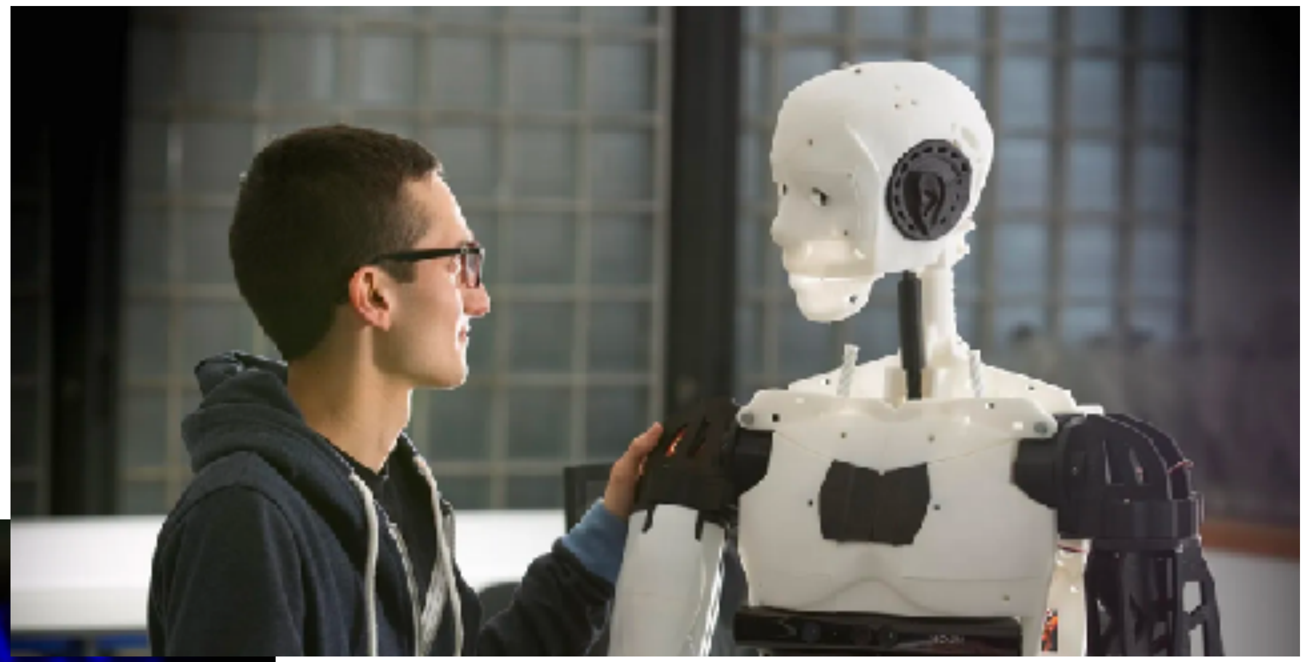
© Marvel Studios

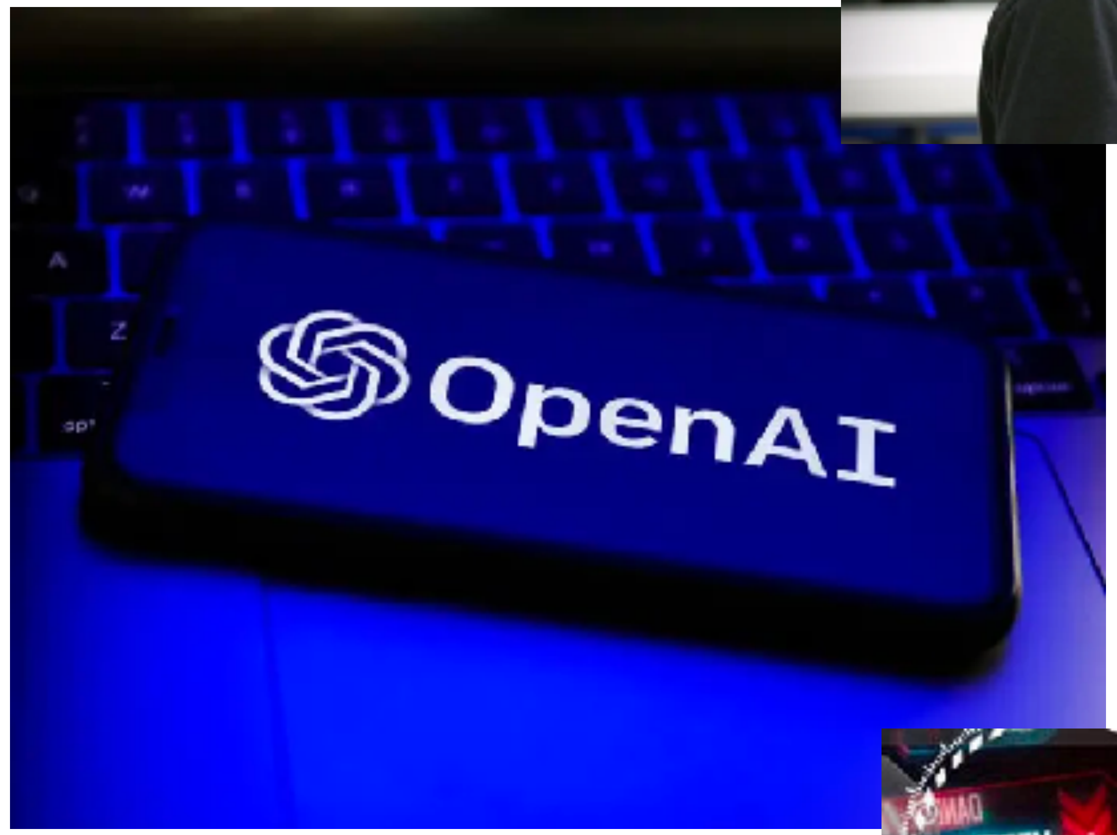
What is natural language generation?

- One view: NLP =
  natural language understanding (NLU) +
  natural language generation (NLG)

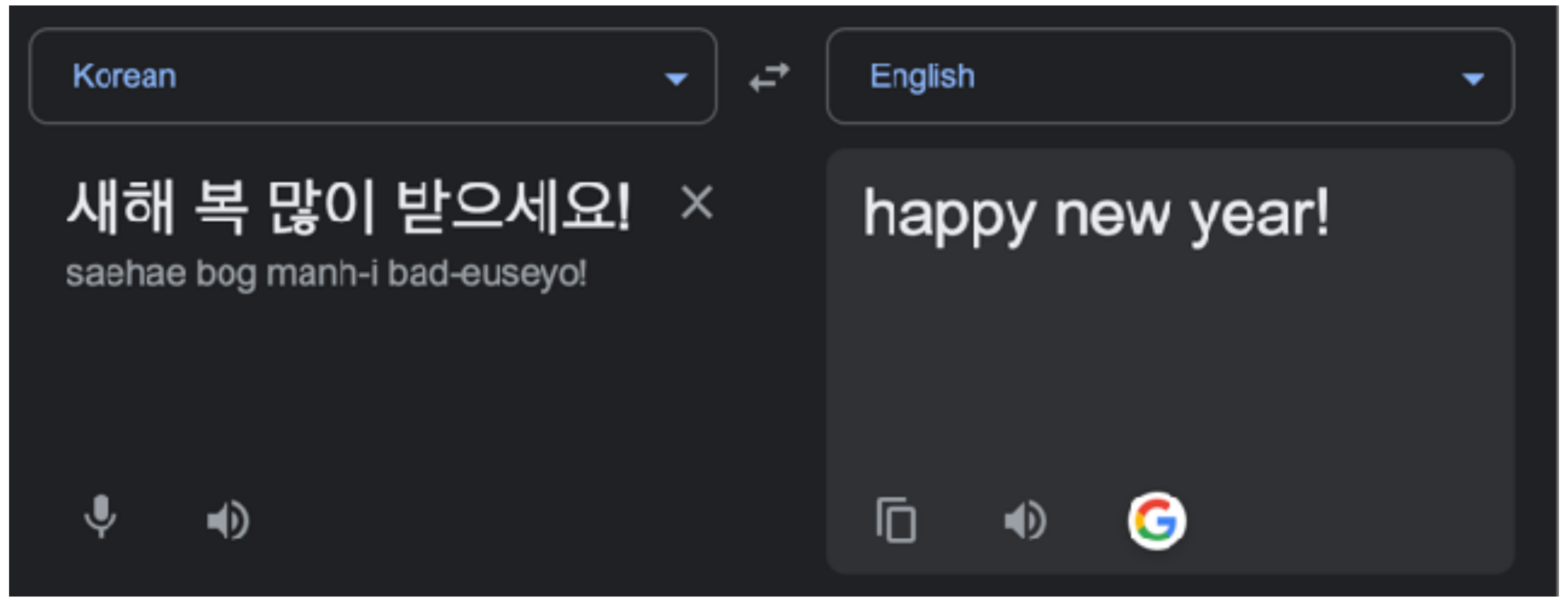
- Focused on building systems that automatically produce **coherent** and **useful** text for human consumption

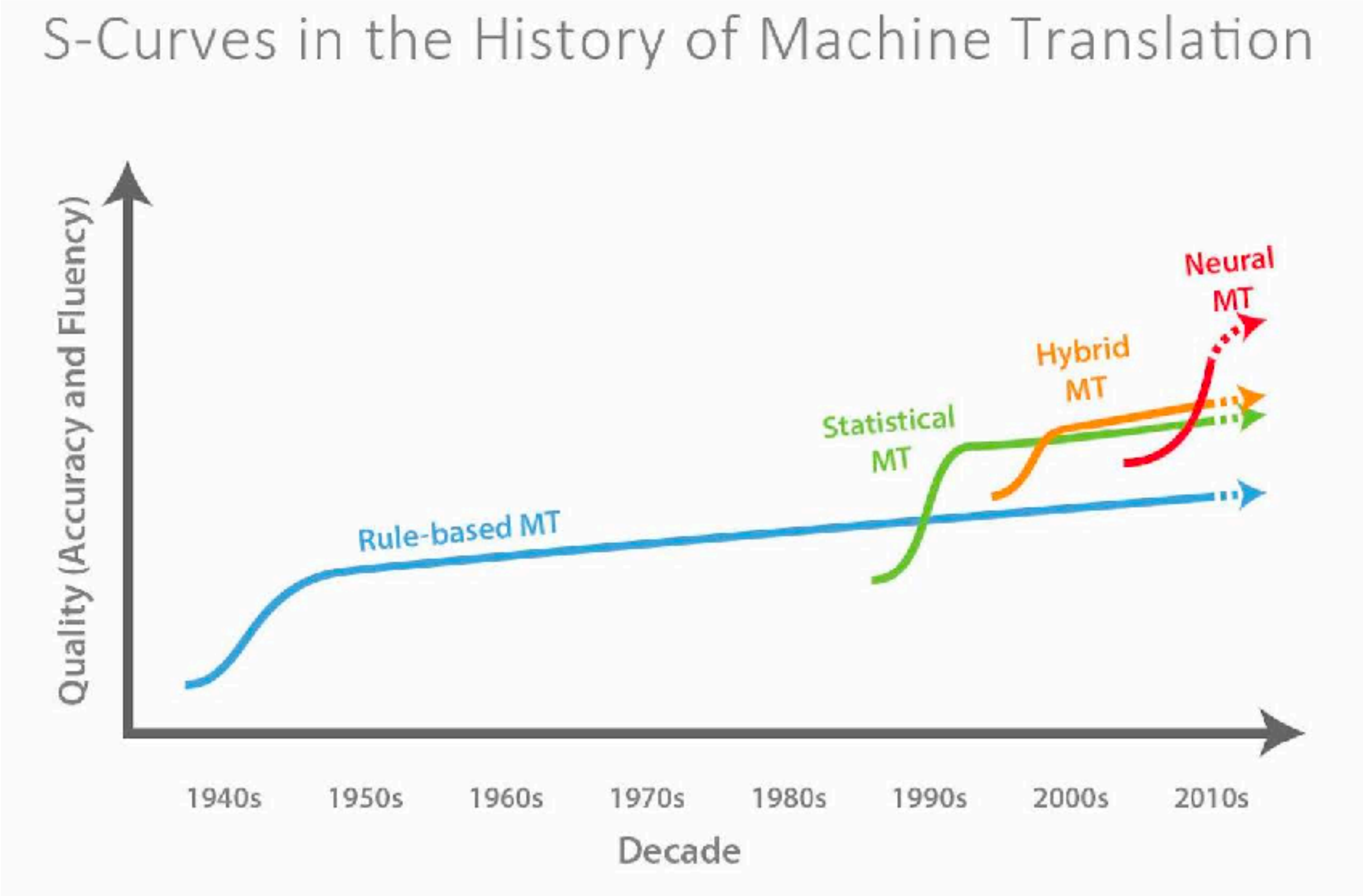
- Large Language Models (the consumer product) are (mostly)
  NLG systems!



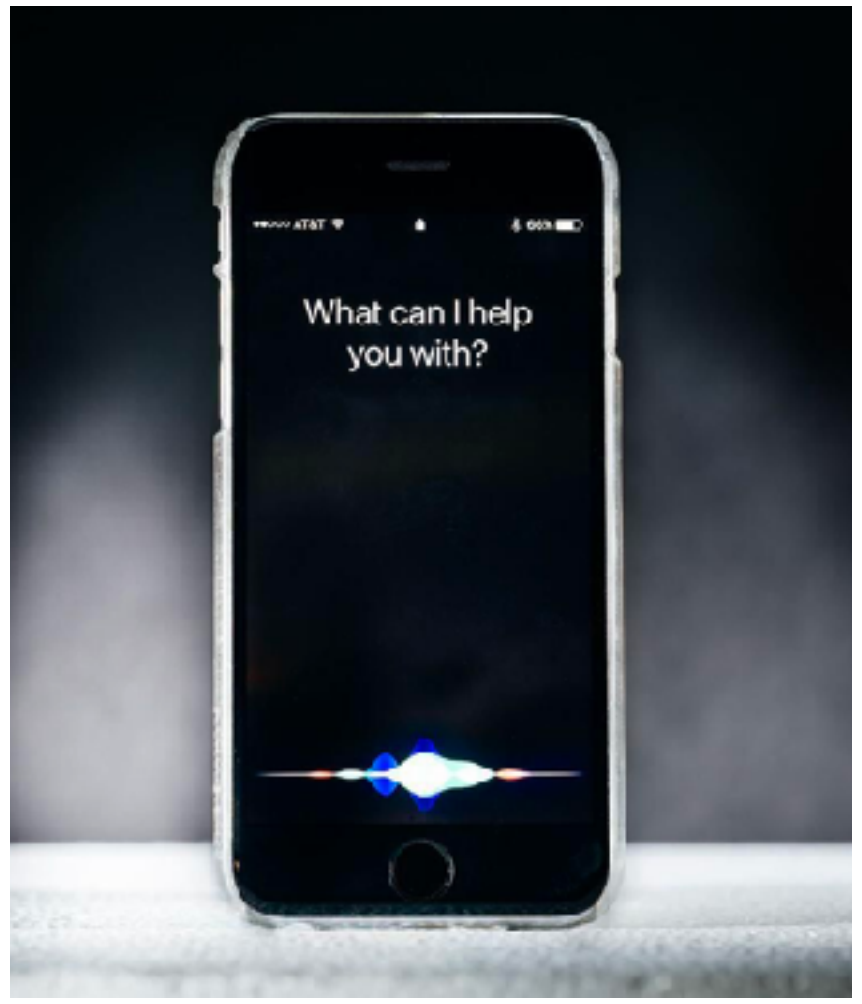


Machine Translation



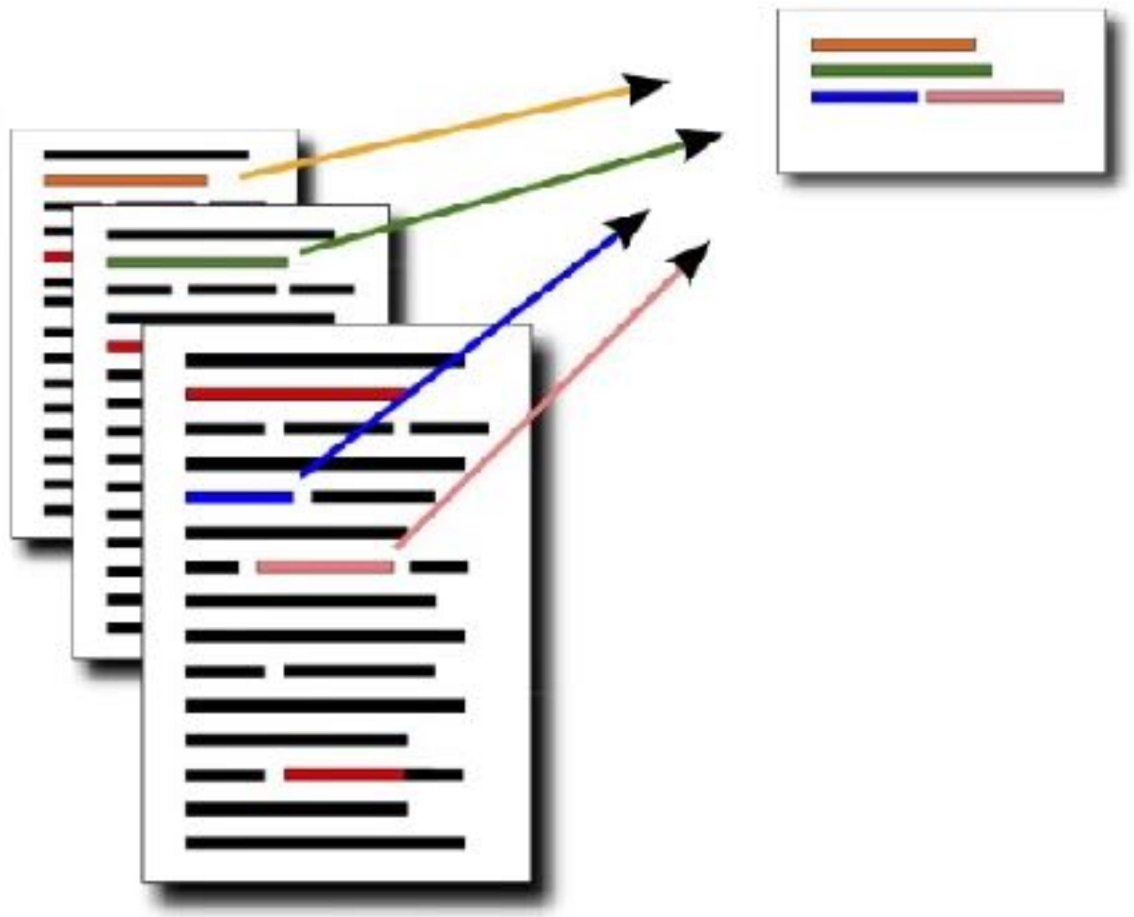


Dialogue Systems



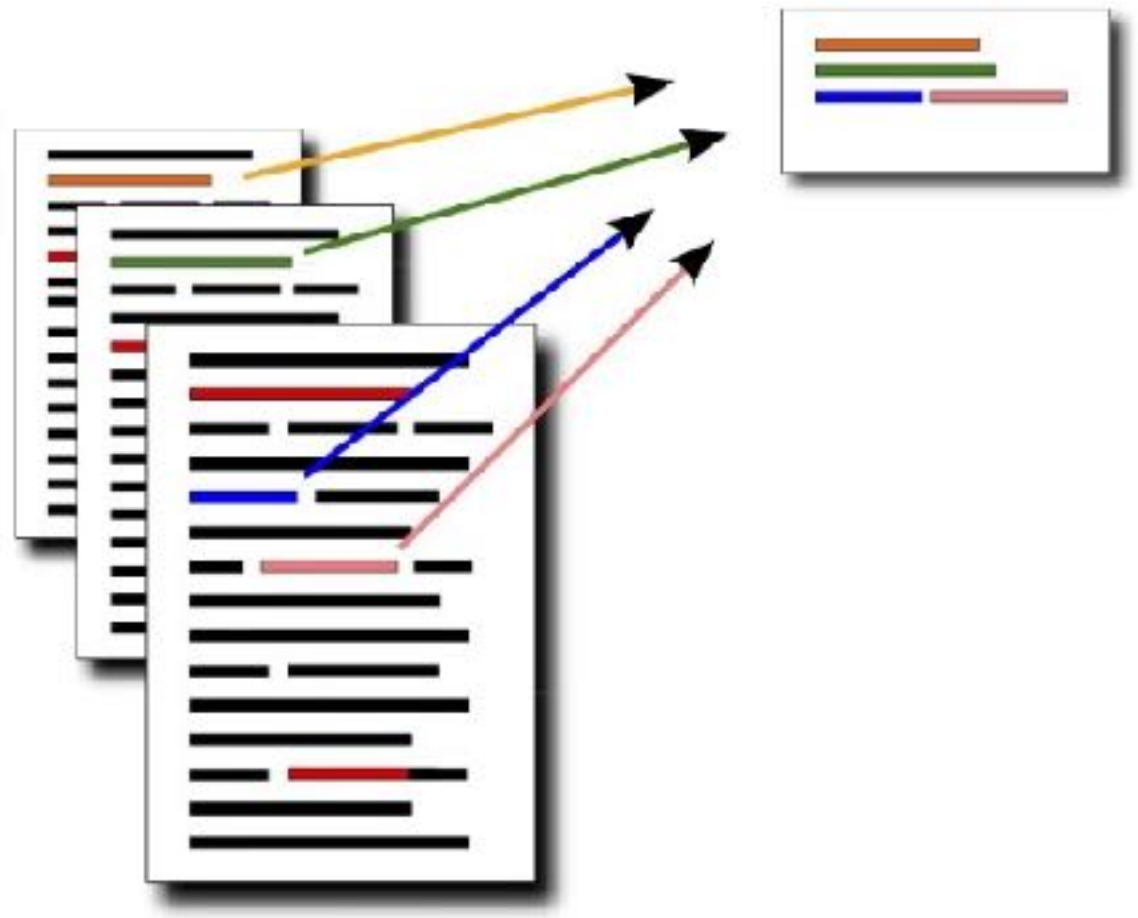
Summarization

Document Summarization



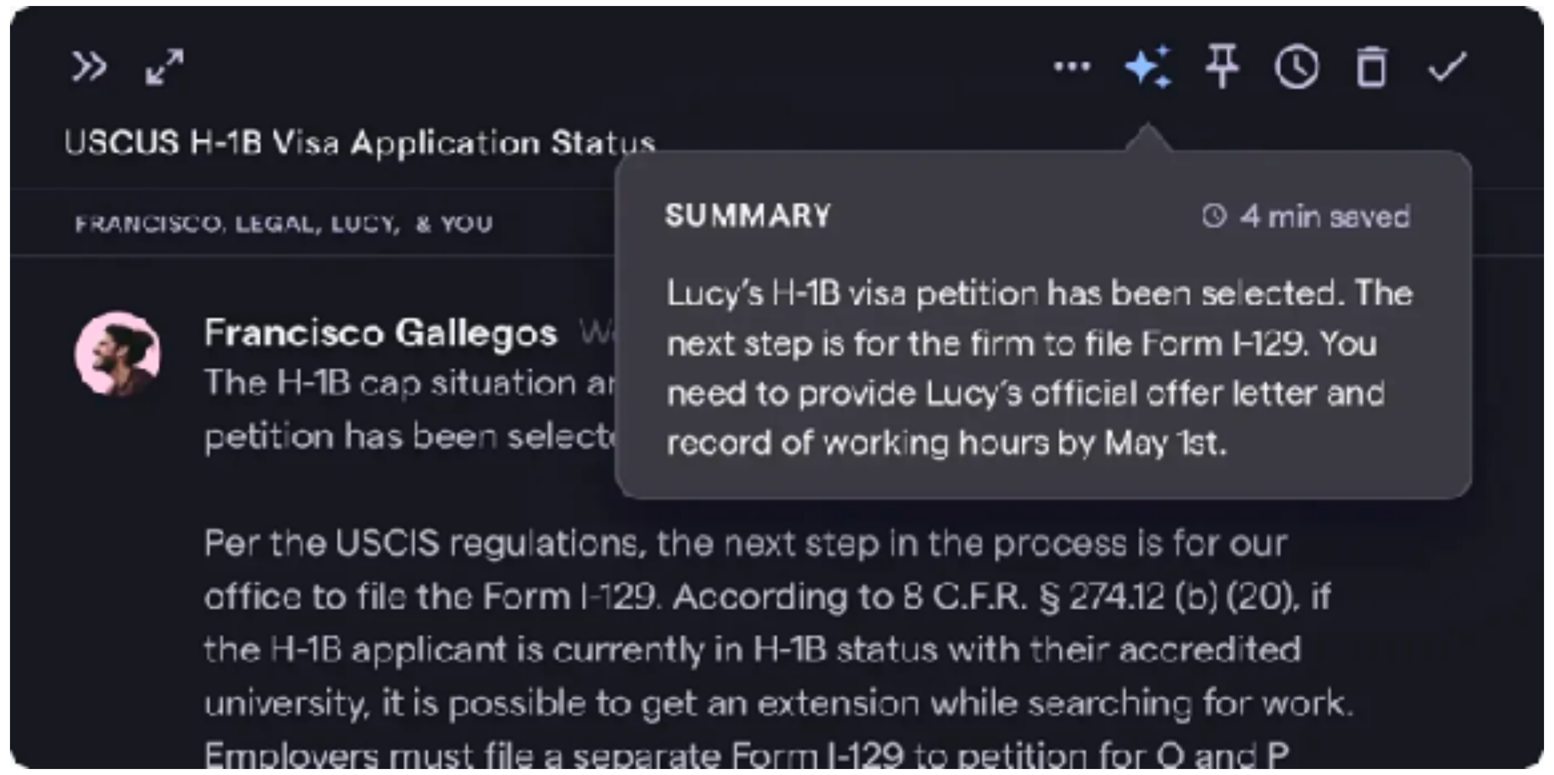
Summarization

Document Summarization



© http://mogren.one/lic/

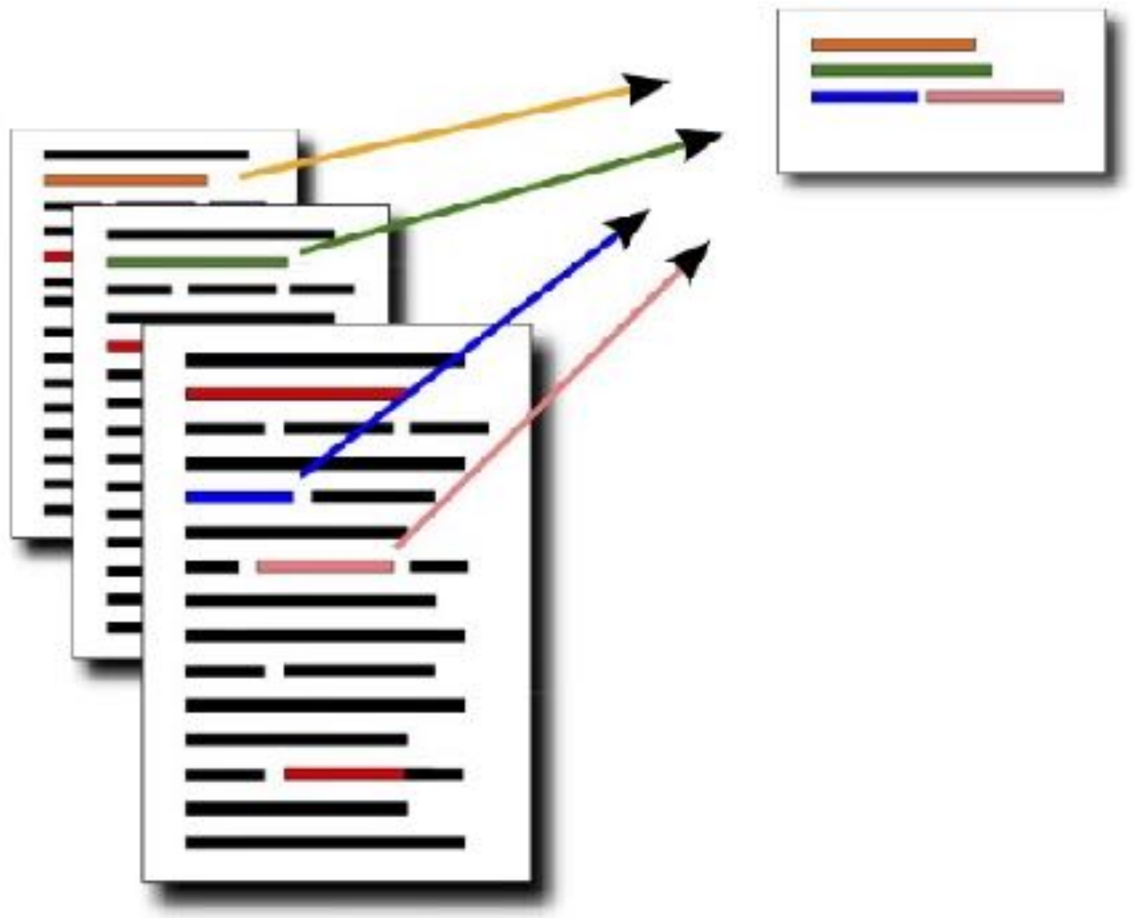
Email Summarization



© techcrunch.com

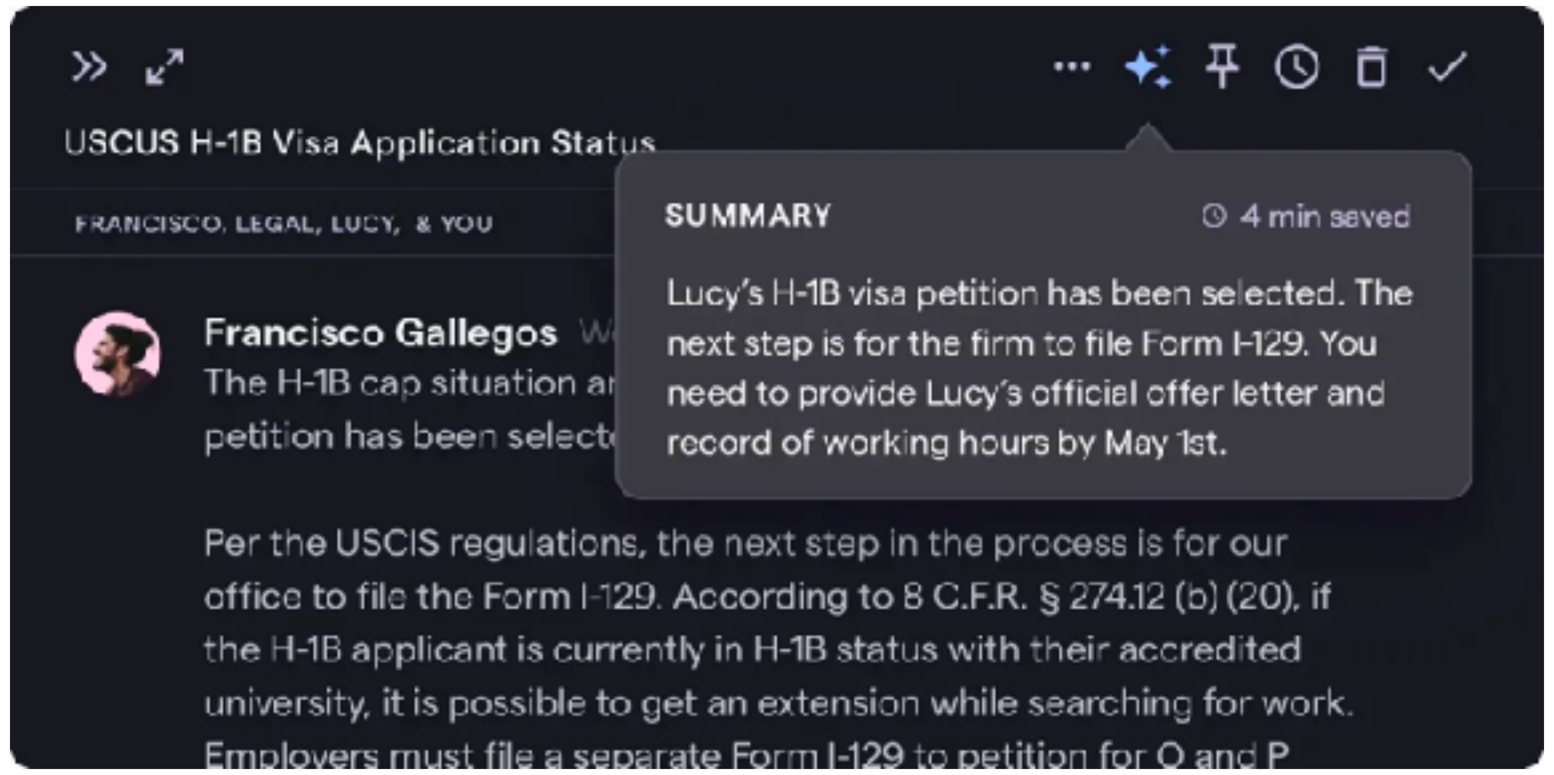
Summarization

Document Summarization



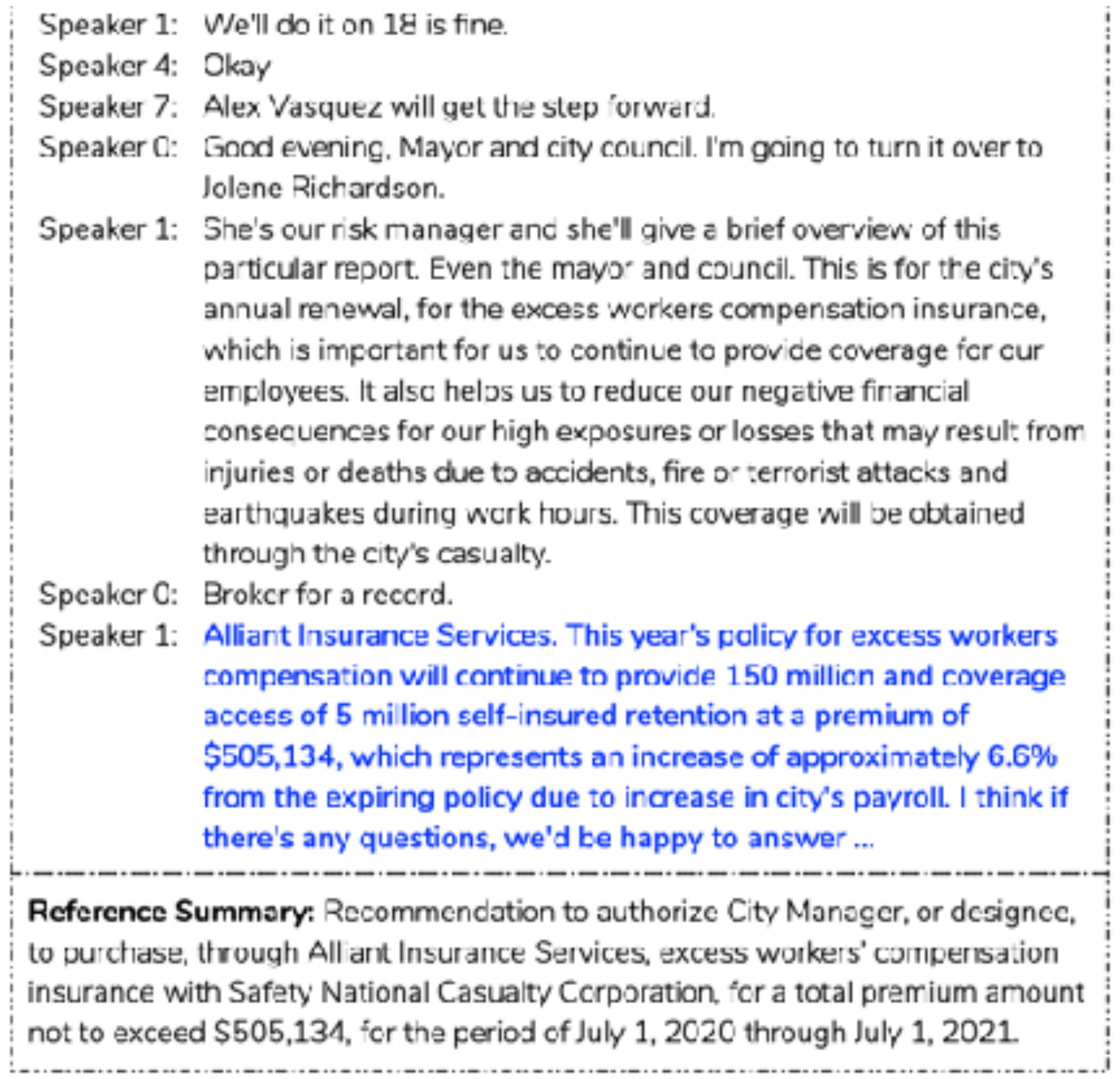
© http://mogren.one/lic/

Email Summarization



© techcrunch.com

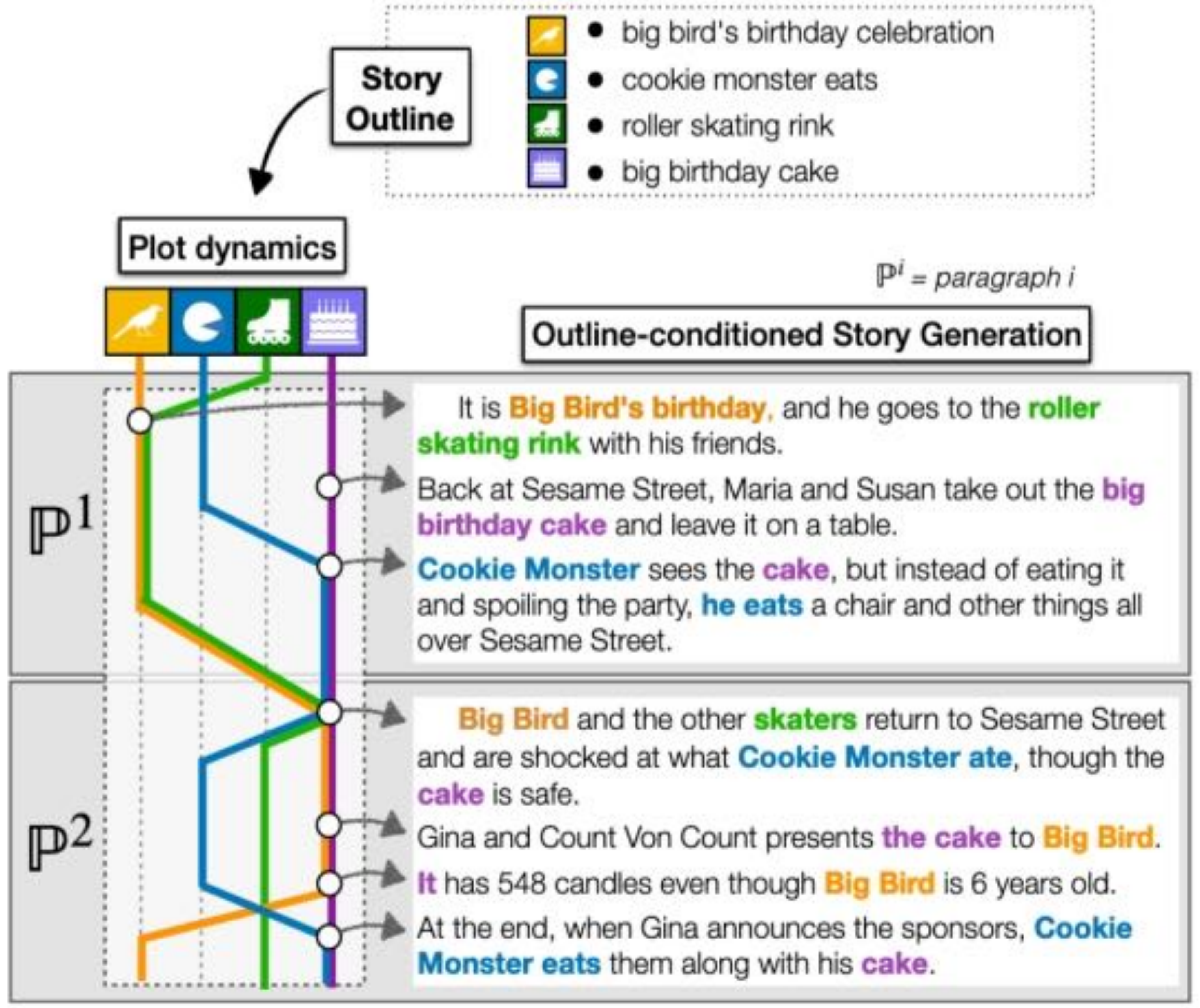
Meeting Summarization



Hu et al., 2023

More interesting NLG uses

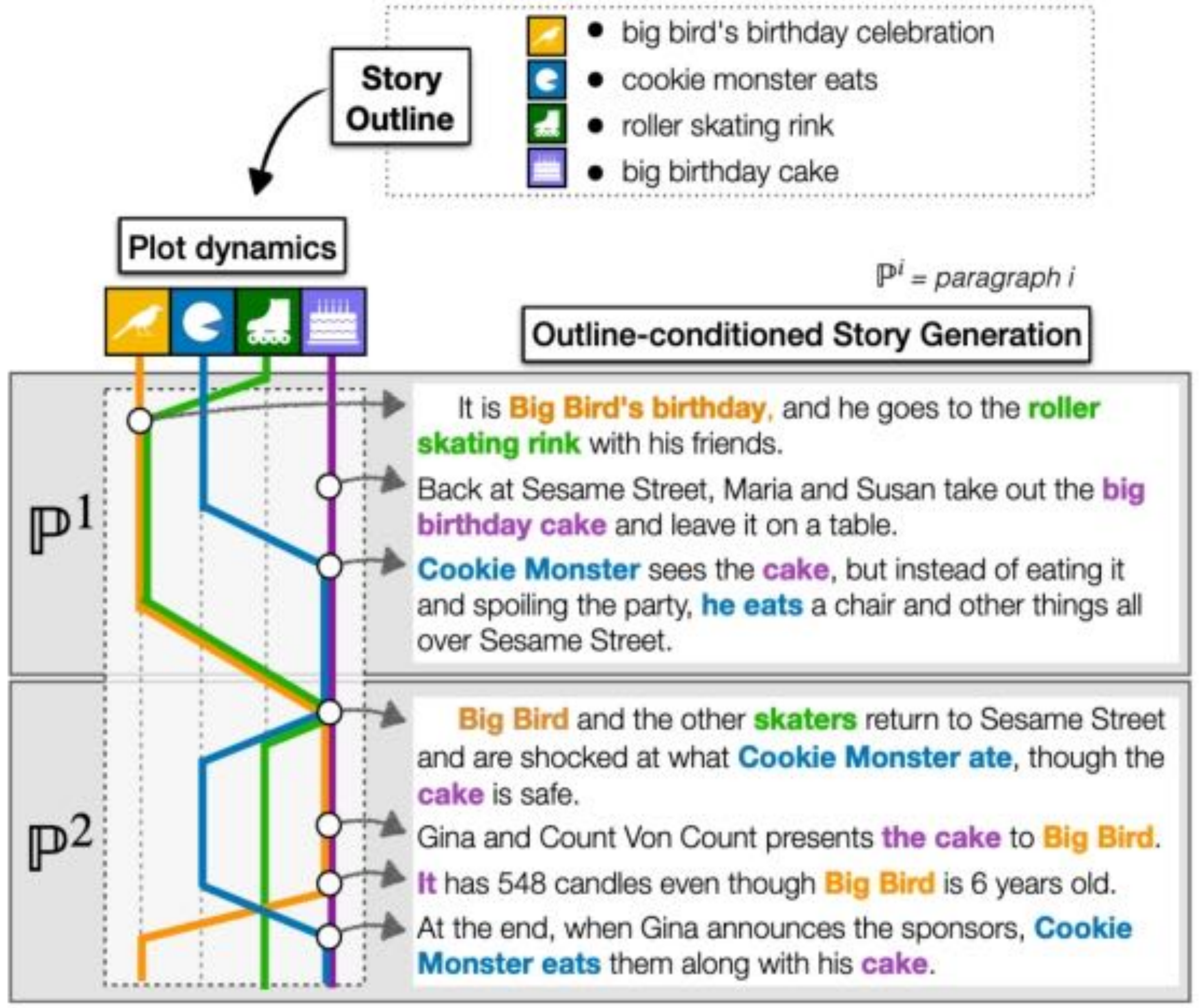
Creative Stories



Rashkin et al., EMNLP 2020

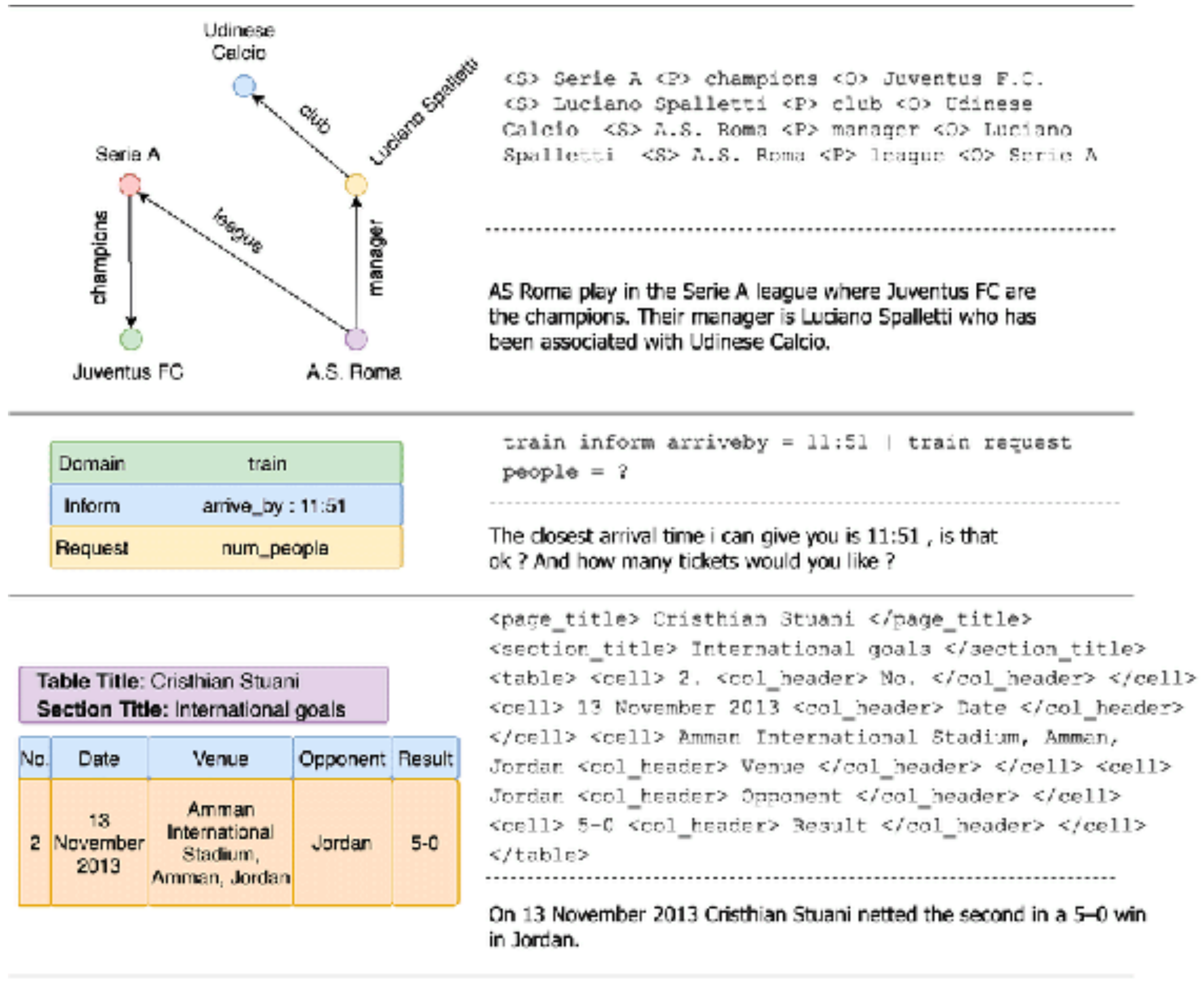
More interesting NLG uses

Creative Stories



Rashkin et al., EMNLP 2020

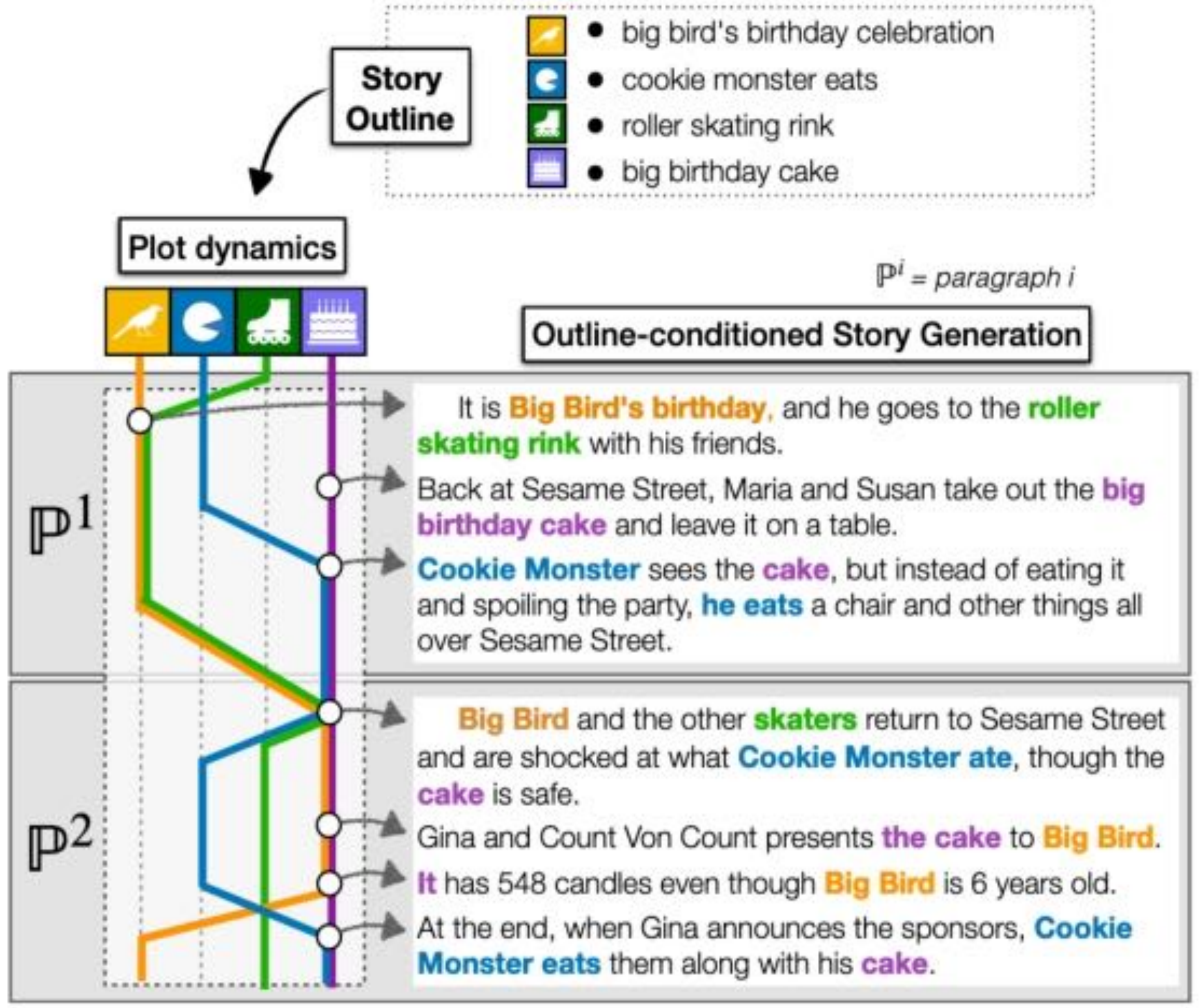
Data-to-Text Generation



Kale et al., INLG 2020

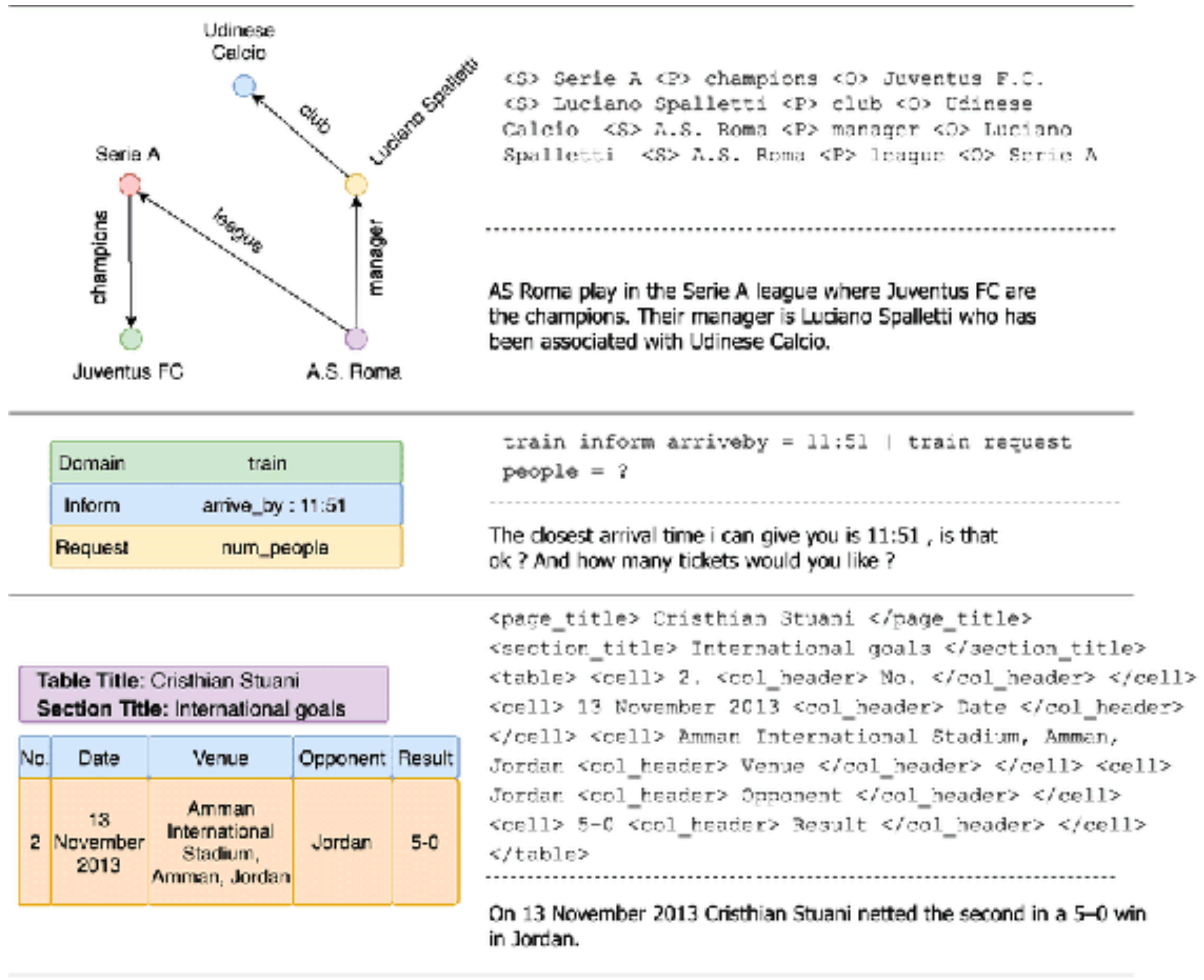
More interesting NLG uses

Creative Stories



Rashkin et al., EMNLP 2020

Data-to-Text Generation



Kale et al., INLG 2020

Visual Description

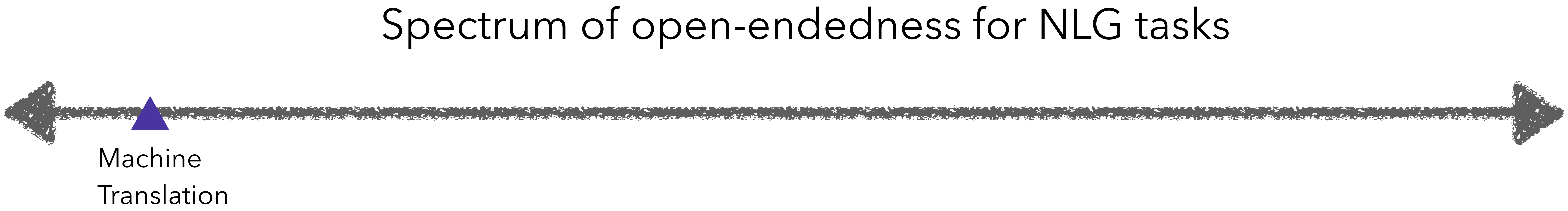
Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy 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

Spectrum of open-endedness for NLG tasks

Categorization of NLG tasks



Categorization of NLG tasks

Spectrum of open-endedness for NLG tasks

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

Categorization of NLG tasks

Spectrum of open-endedness for NLG tasks

Categorization of NLG tasks

Spectrum of open-endedness for NLG tasks

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

Categorization of NLG tasks

Spectrum of open-endedness for NLG tasks

Categorization of NLG tasks

Spectrum of open-endedness for NLG tasks

Input: Write a story about three little pigs?

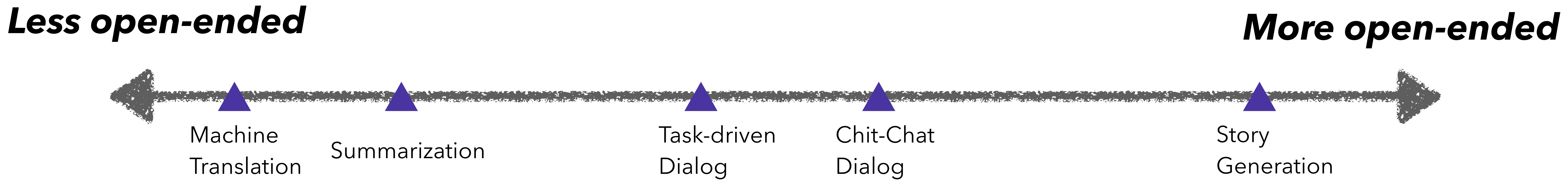
Reference Outputs:

... (so may options)...

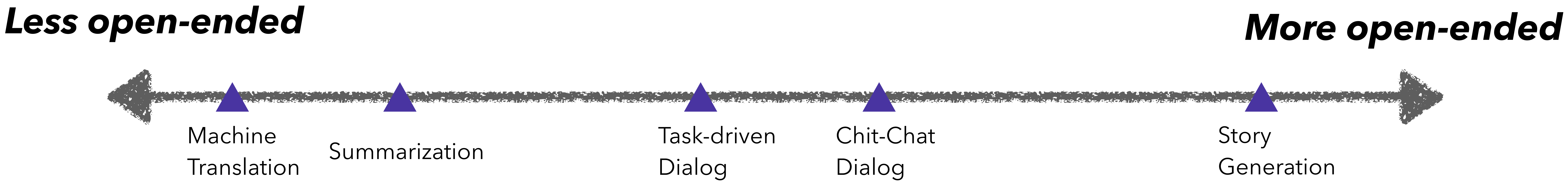
The output space is extremely diverse.

Categorization of NLG tasks

Categorization of NLG tasks



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!

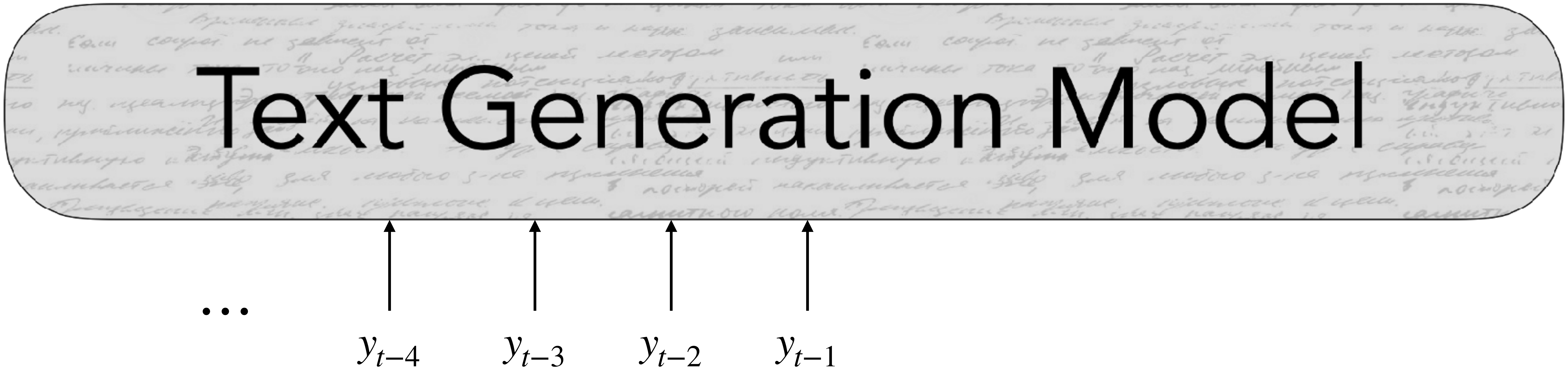
Formalizing Natural Language Generation

Basics of natural language generation

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

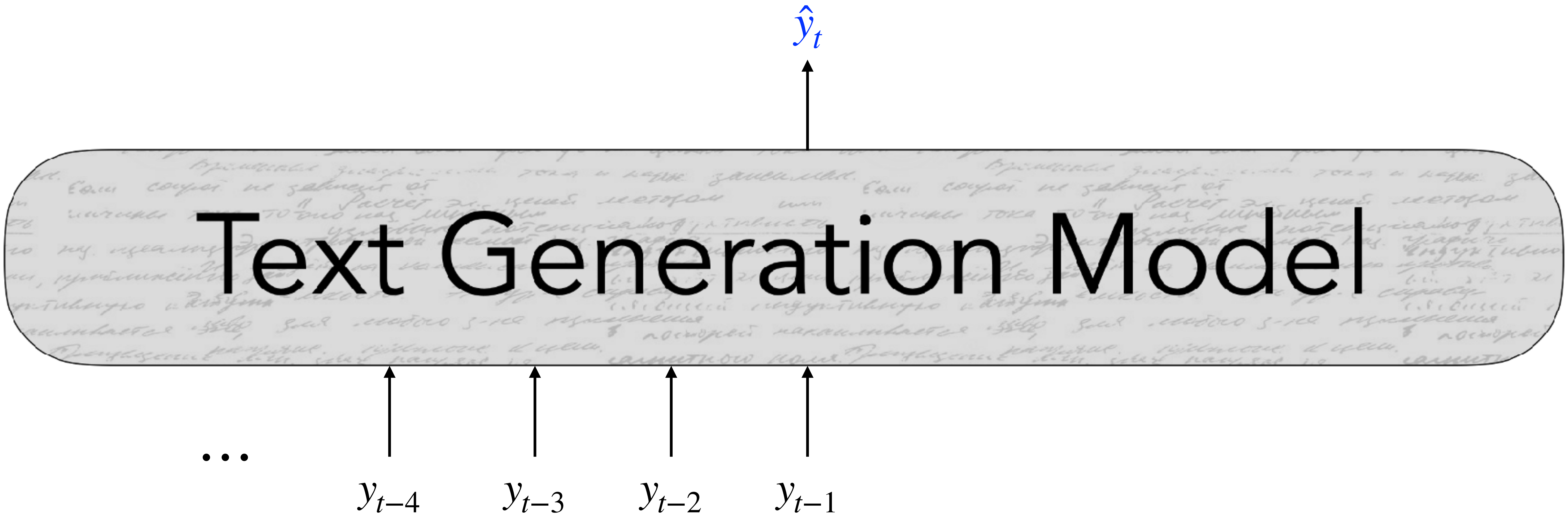
Basics of natural language generation

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



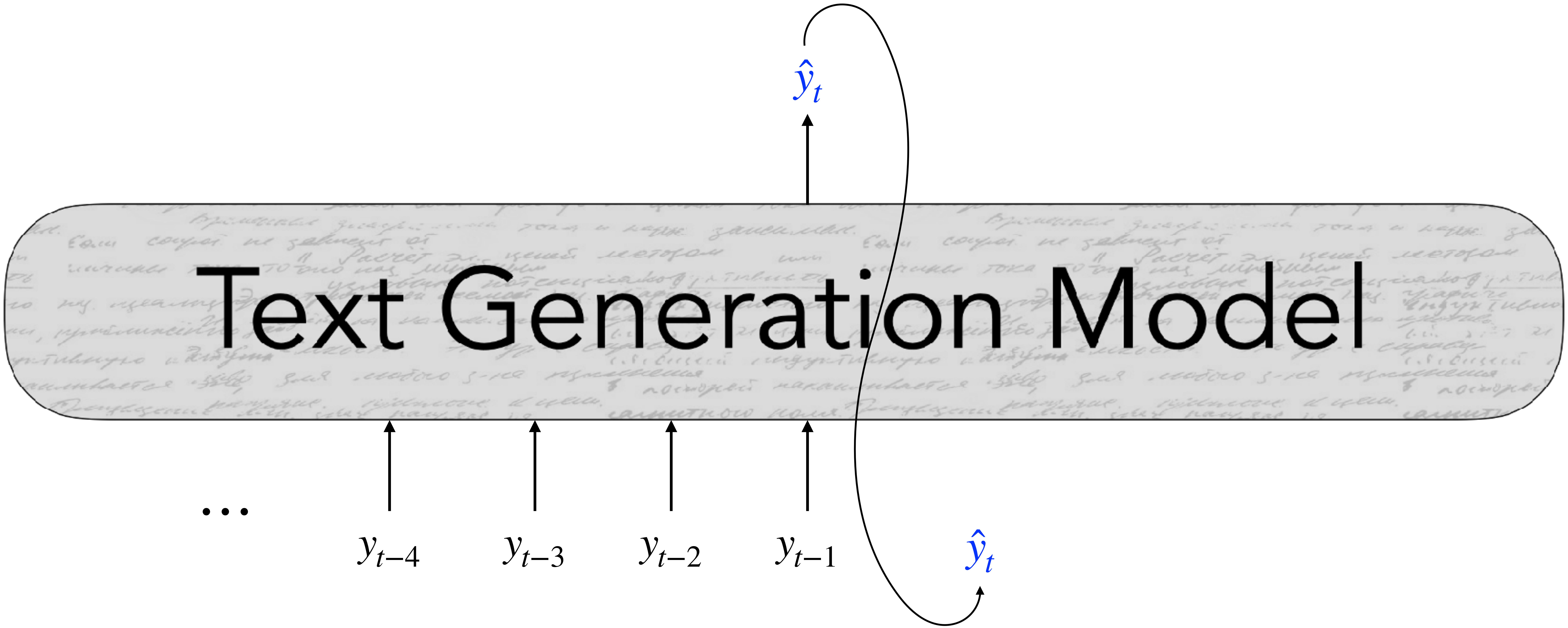
Basics of natural language generation

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



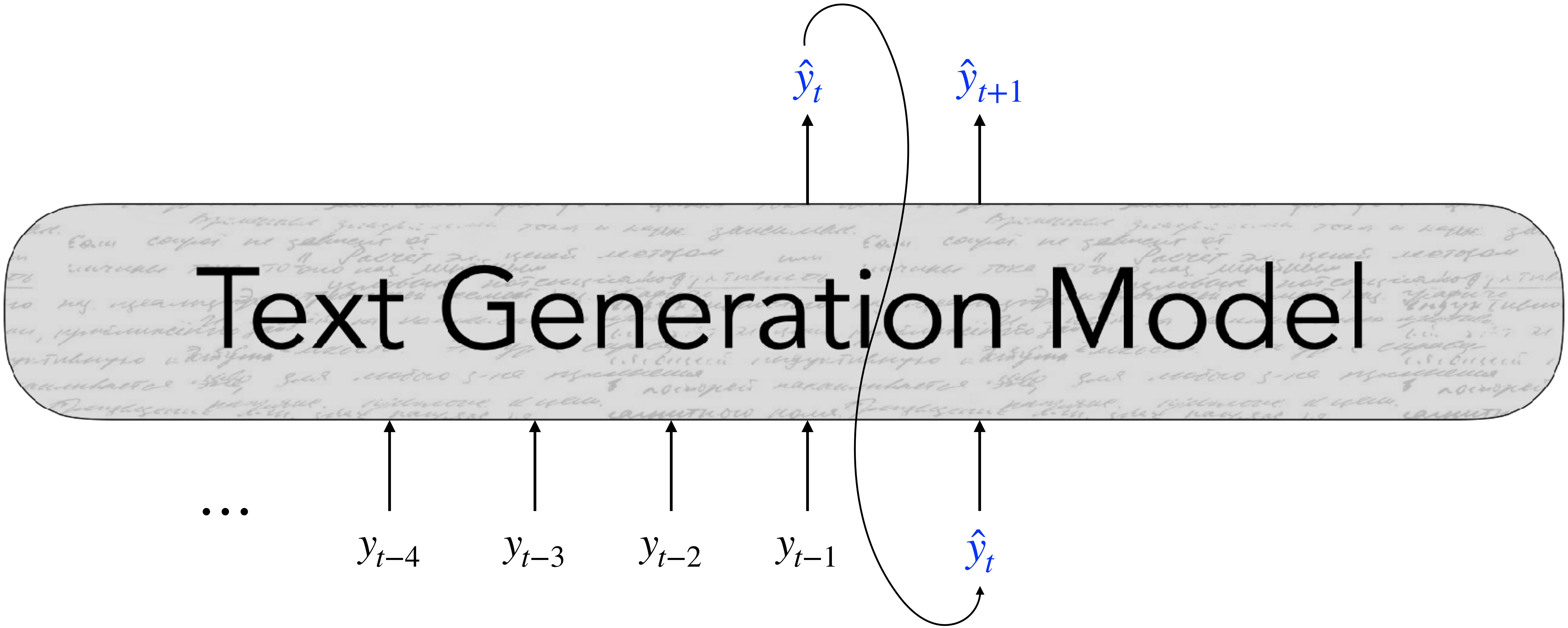
Basics of natural language generation

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



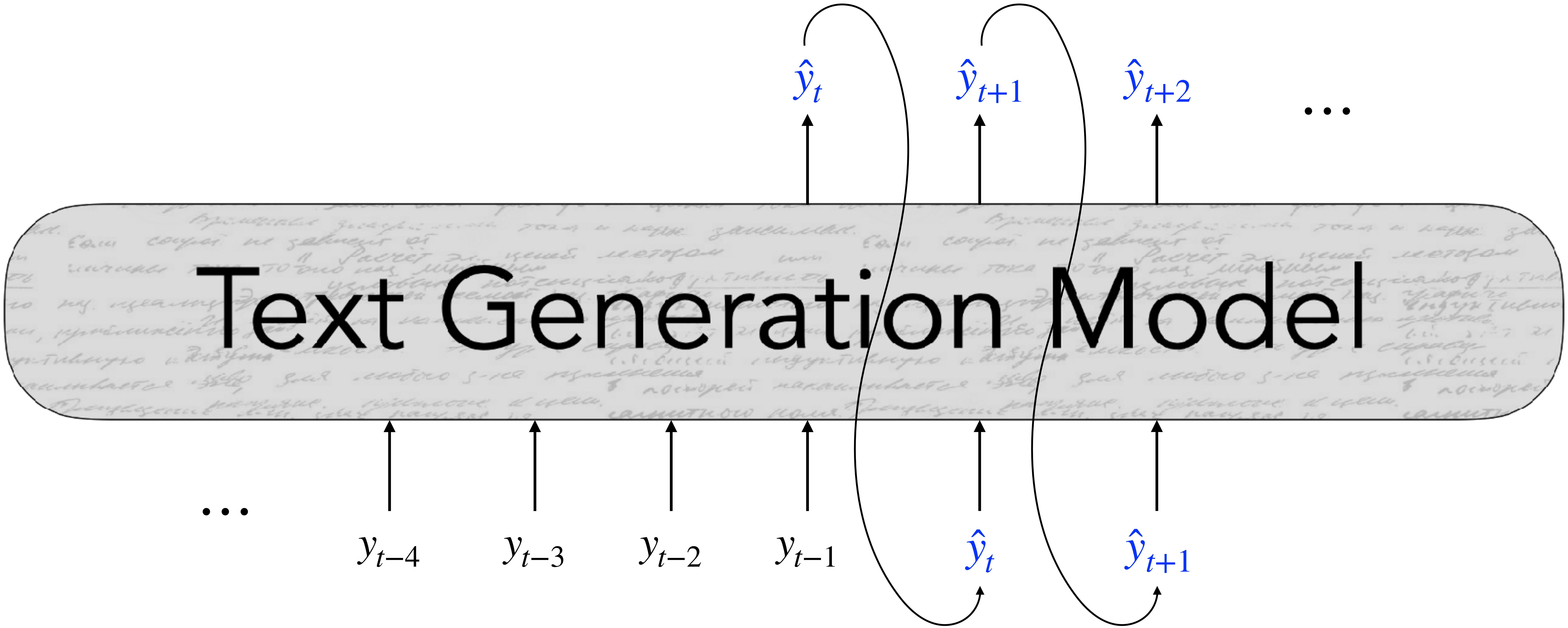
Basics of natural language generation

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



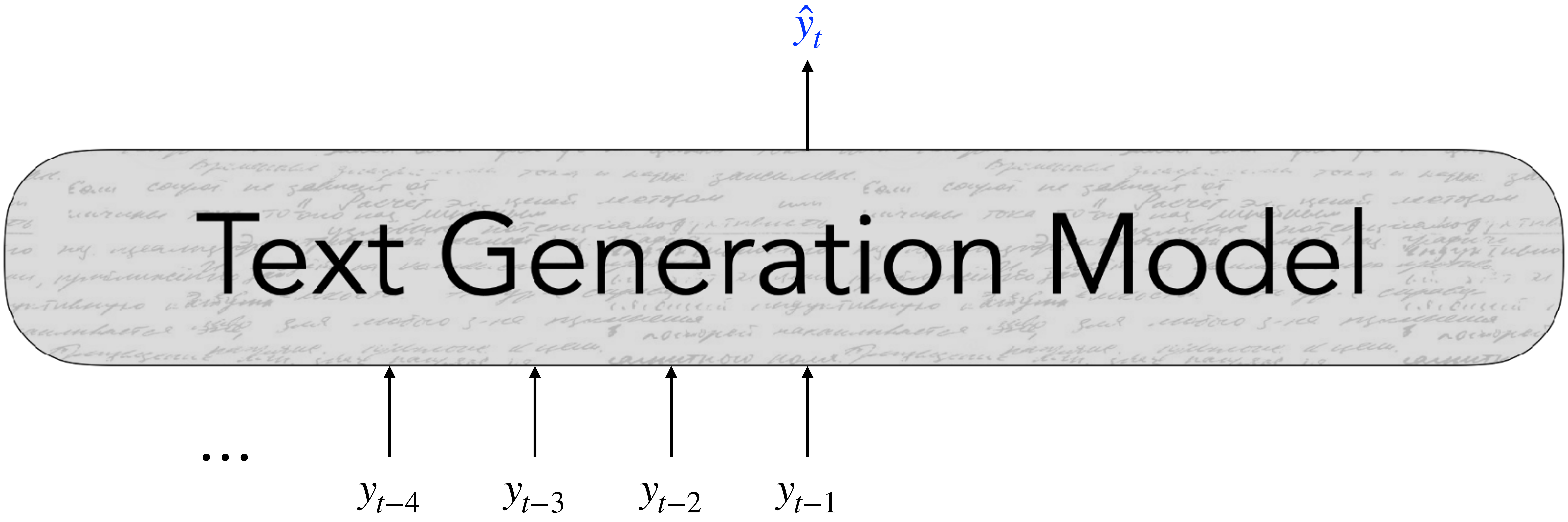
Basics of natural language generation

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



A look at a single step

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



Basics of natural language generation

Basics of natural language generation

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

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

Basics of natural language generation

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

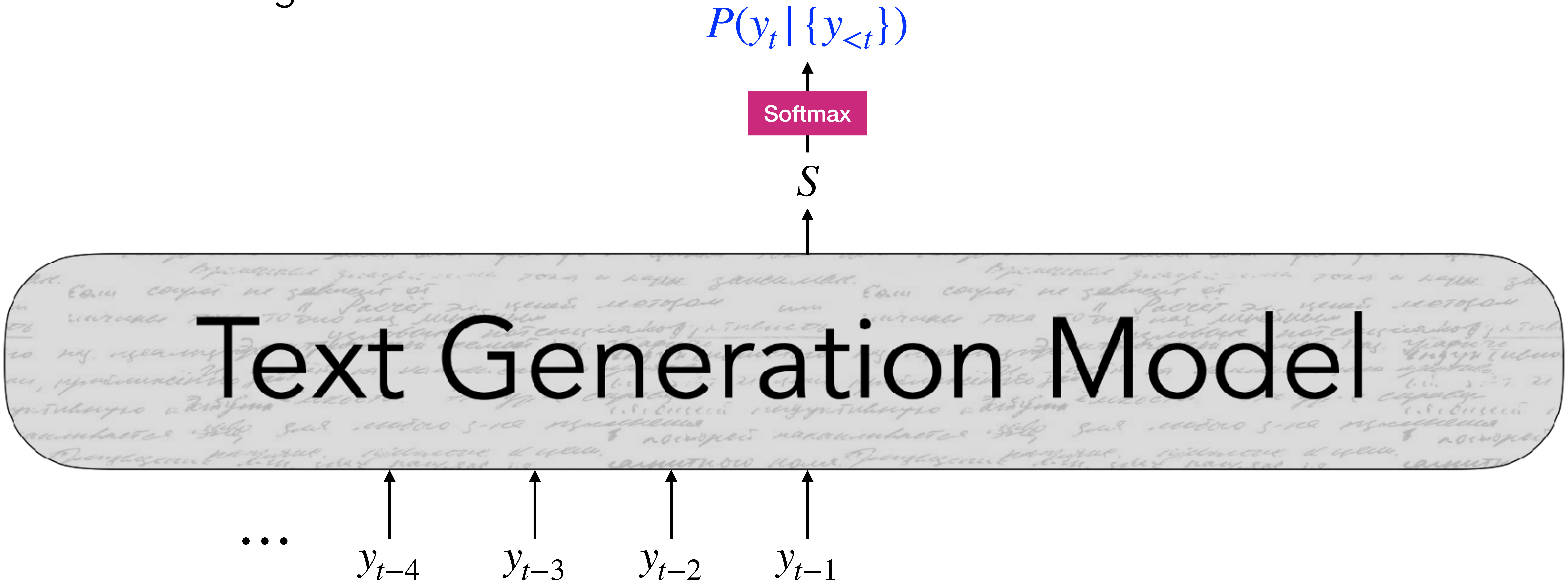
$$S = \underbrace{f(\{y_{<t}\}; \theta)}_{f(\,\cdot\,;\,\theta) \text{ is your model}}$$

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

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

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:



Training and Inference

Training and Inference

- 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

Training and Inference

- 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:

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

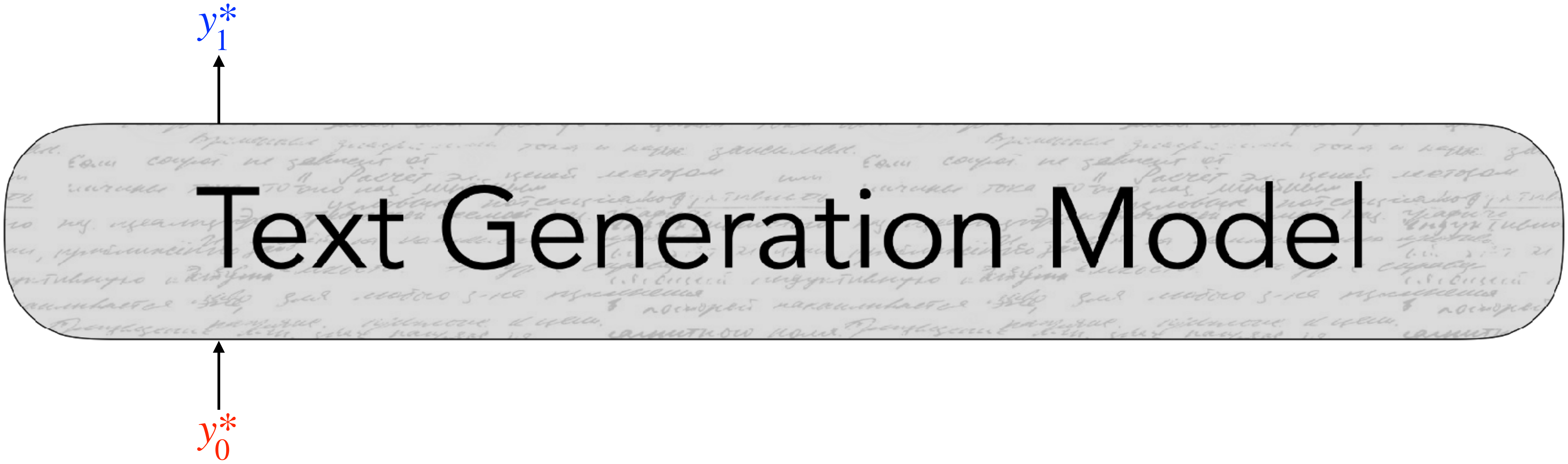
Remark:

- 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".

Maximum Likelihood Training (i.e. teacher-forcing)

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

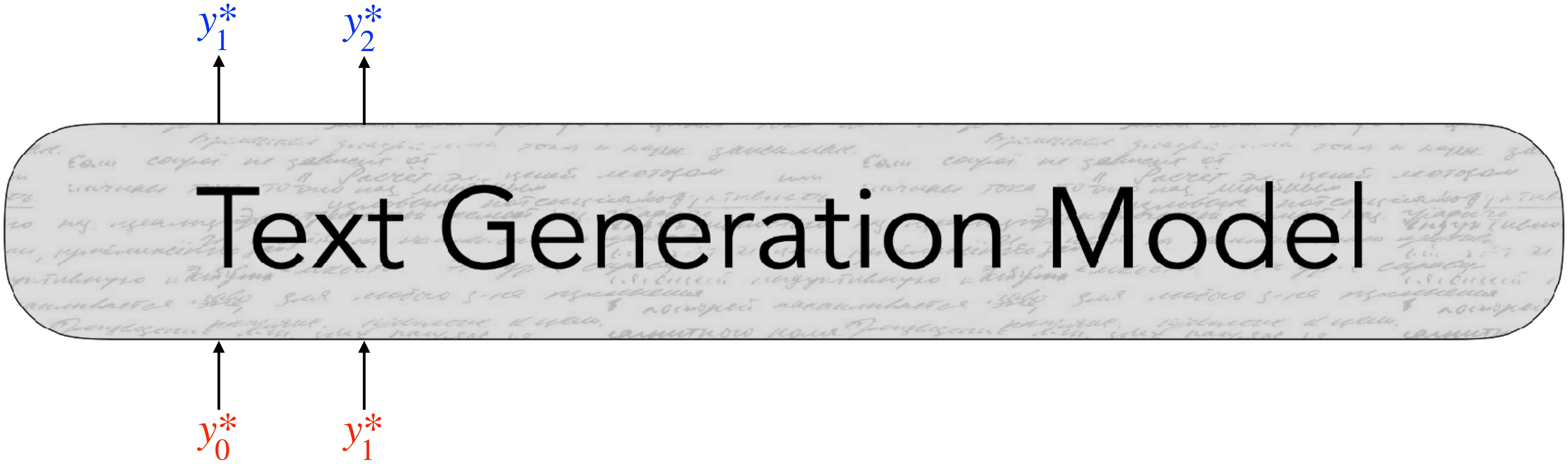
$$L = -\log P(y_1^* \mid y_0^*)$$



Maximum Likelihood Training (i.e. teacher-forcing)

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

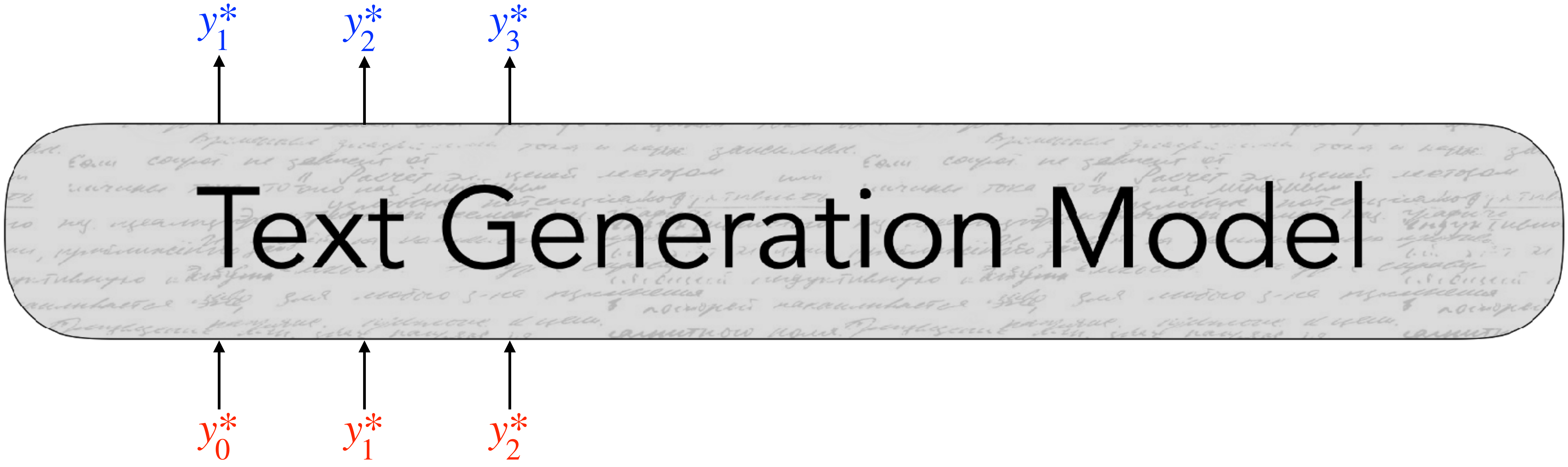
$$L = -\left(\log P(y_1^* \mid y_0^*) + \log P(y_2^* \mid y_0^*, y_1^*)\right)$$



Maximum Likelihood Training (i.e. teacher-forcing)

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

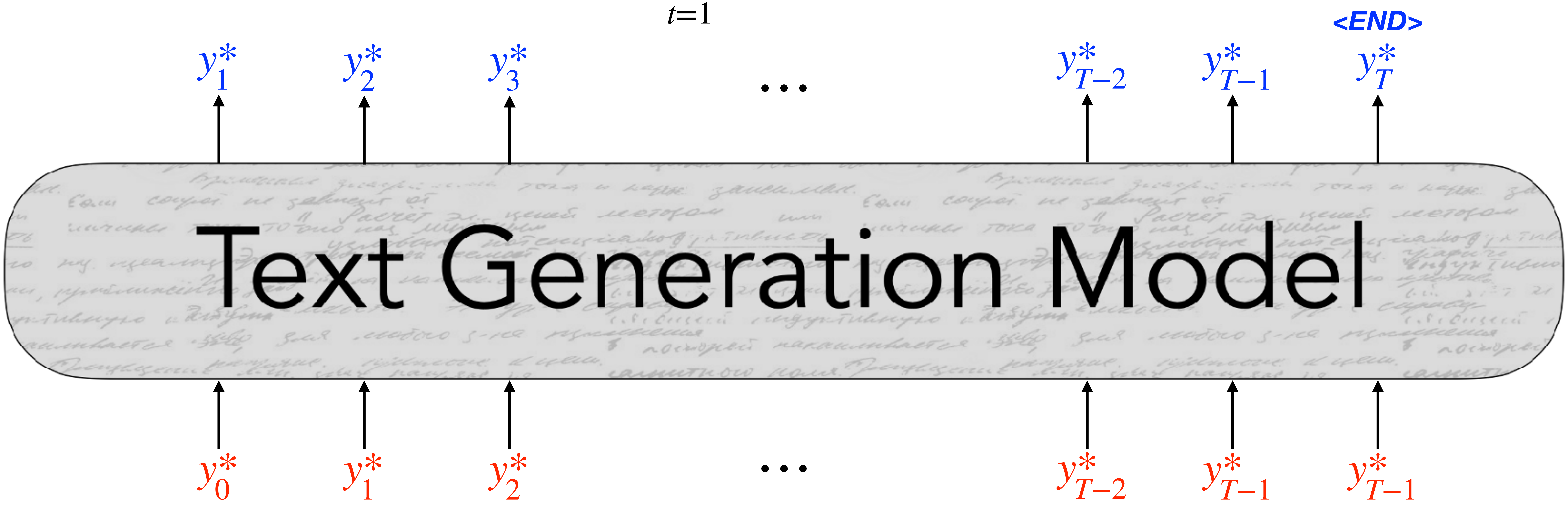
$$L = -\left(\log P(y_1^* \mid y_0^*) + \log P(y_2^* \mid y_0^*, y_1^*) + \log P(y_3^* \mid y_0^*, y_1^*, y_2^*)\right)$$



Maximum Likelihood Training (i.e. teacher-forcing)

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

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



Generation Algorithms

Decoding: What is it all about?

Decoding: What is it all about?

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

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

Decoding: What is it all about?

- 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:

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

Decoding: What is it all about?

- 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:

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

How to find the most likely string?

How to find the most likely string?

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

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

How to find the most likely string?

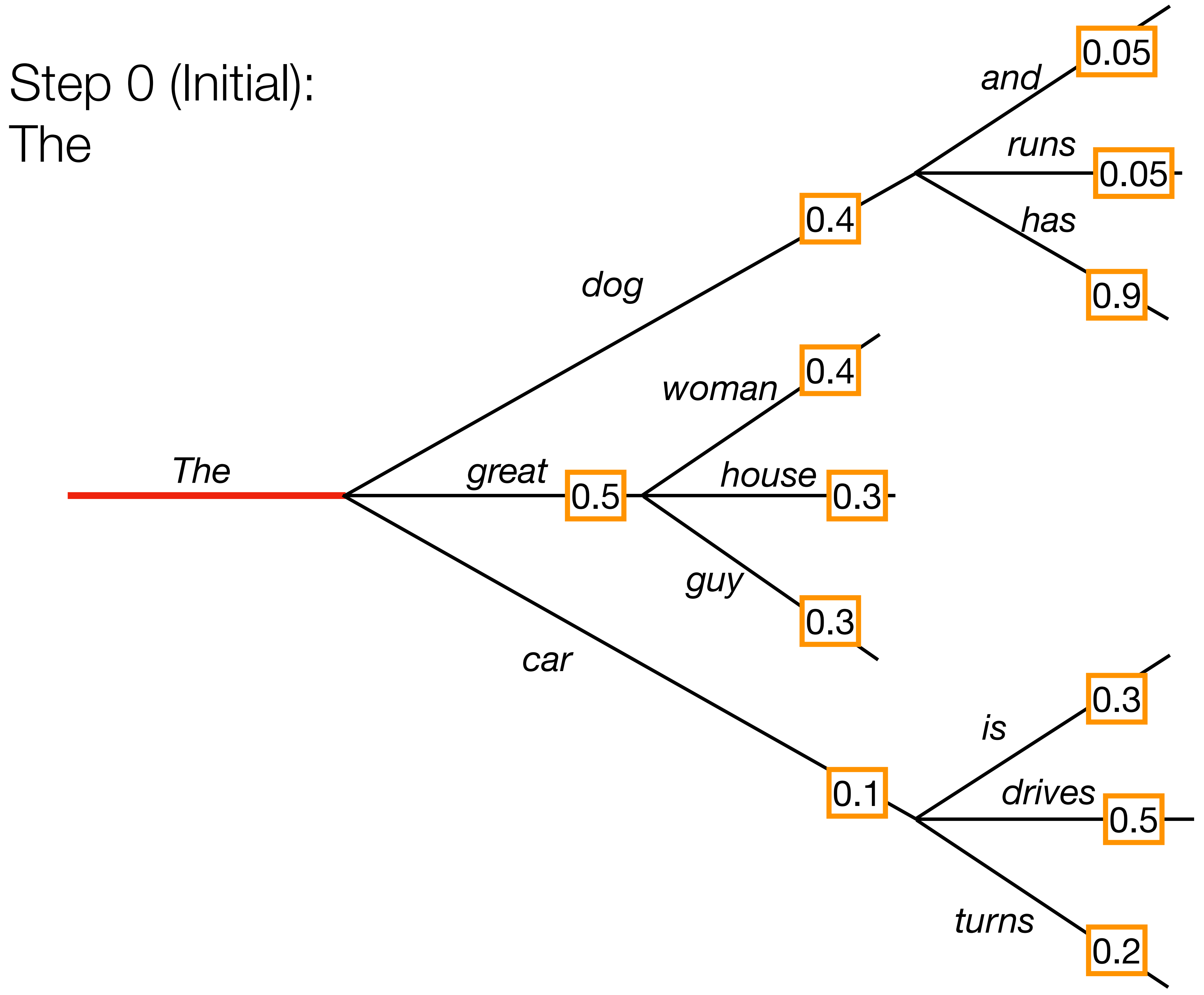
- **Obvious method: Greedy Decoding**
  - Selects the highest probability token according to $P(y_t | y_{<t})$

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

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

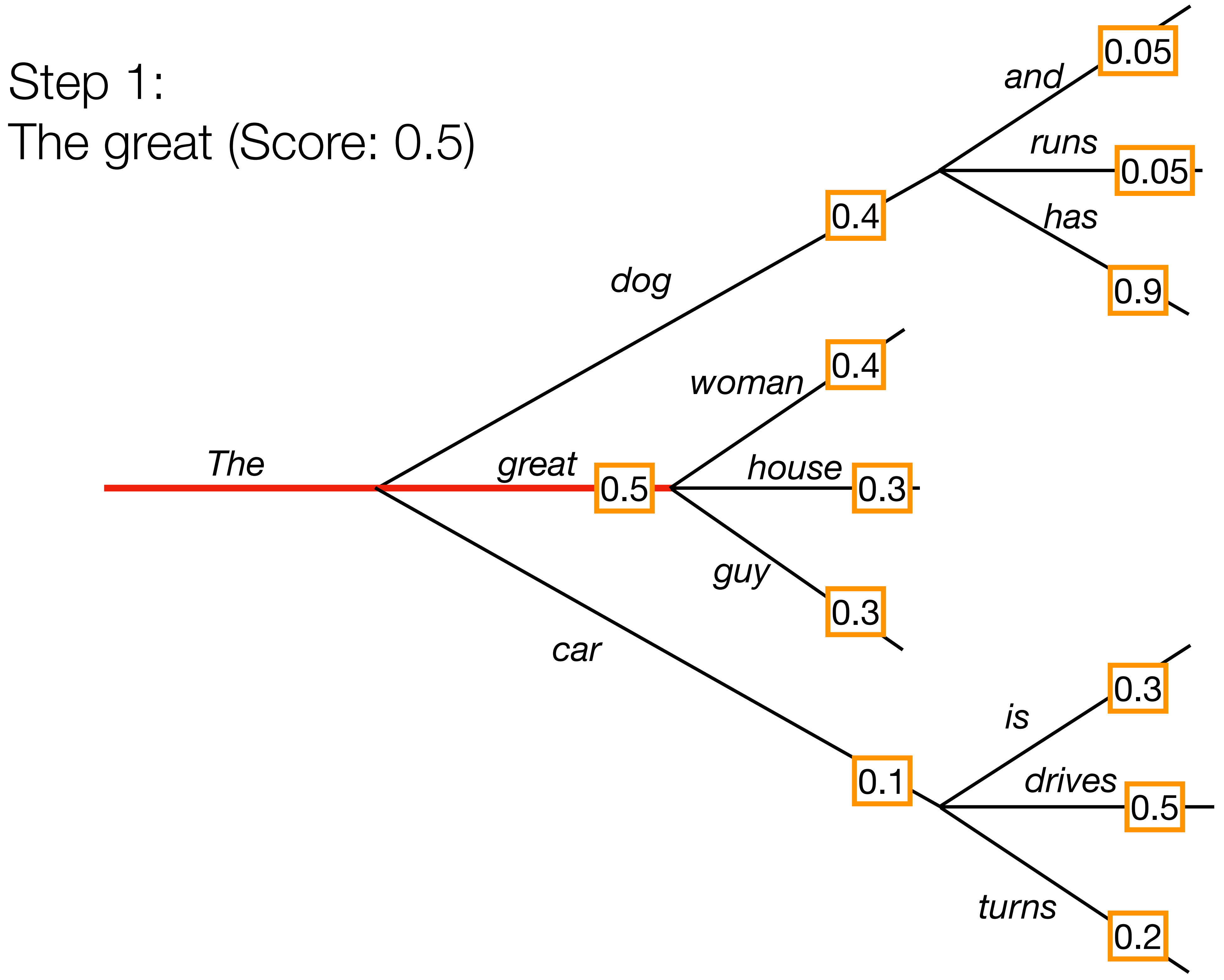
Greedy Decoding vs. Beam Search

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



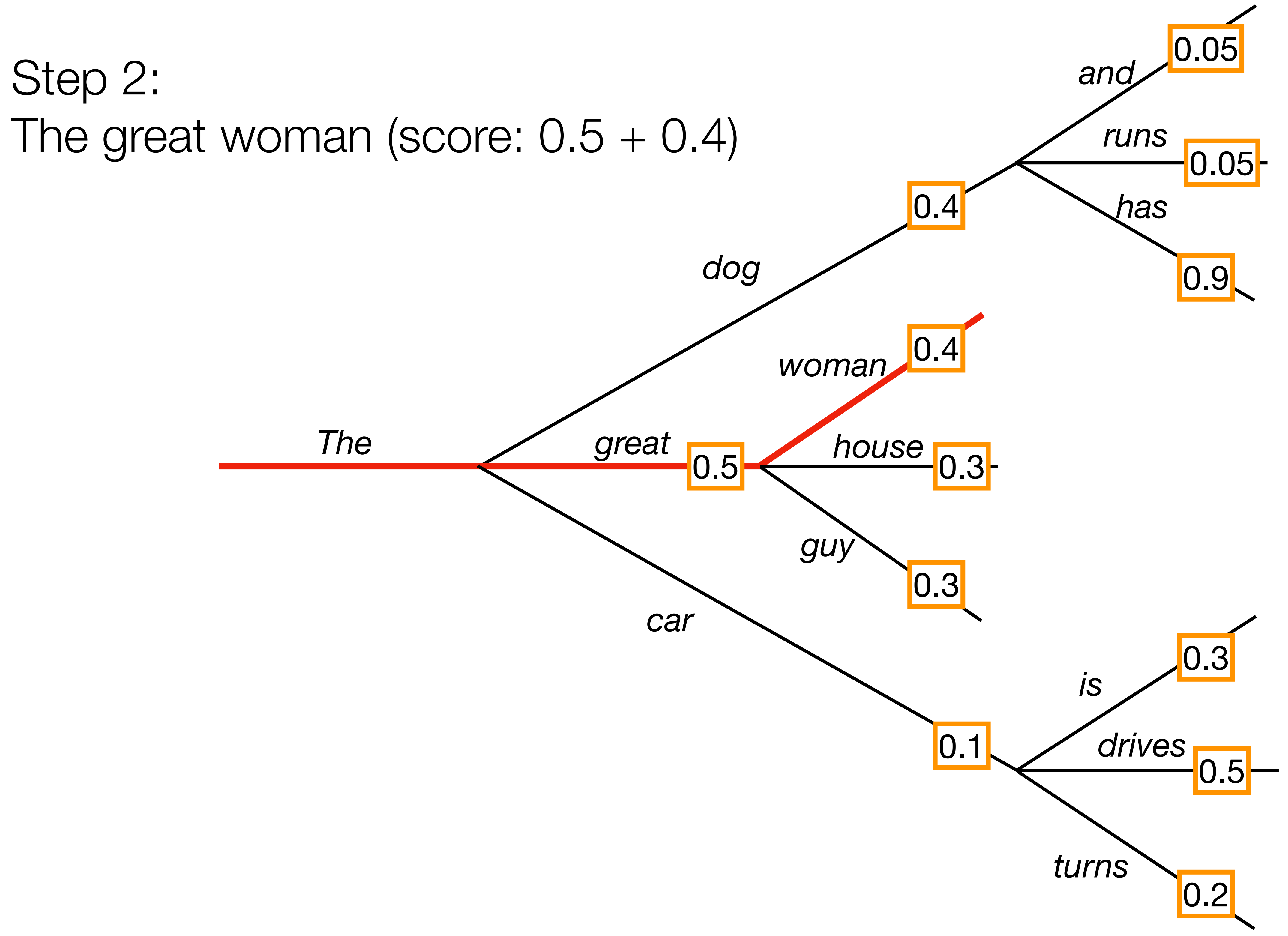
Greedy Decoding vs. Beam Search

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



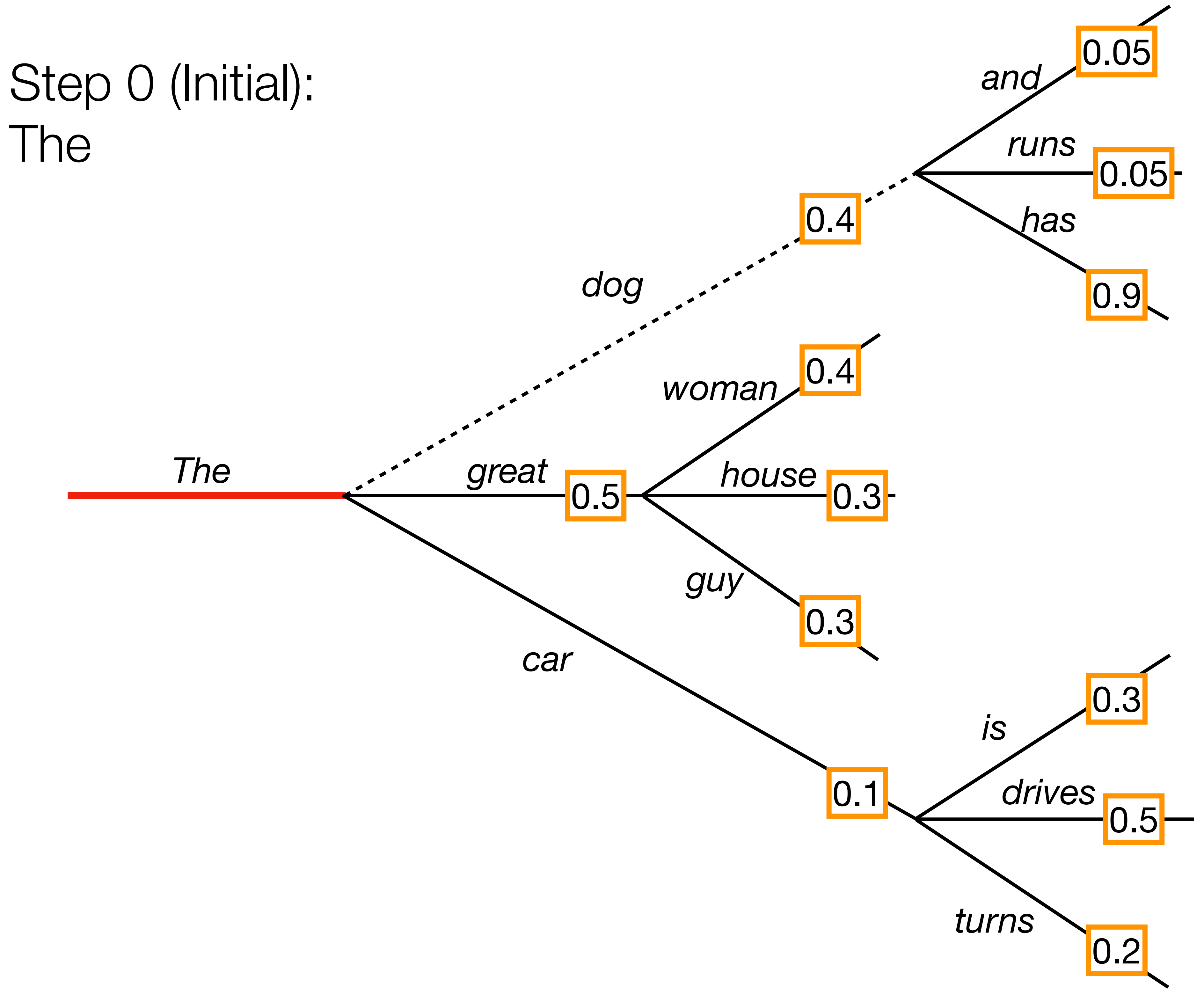
Greedy Decoding vs. Beam Search

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



Greedy Decoding vs. Beam Search

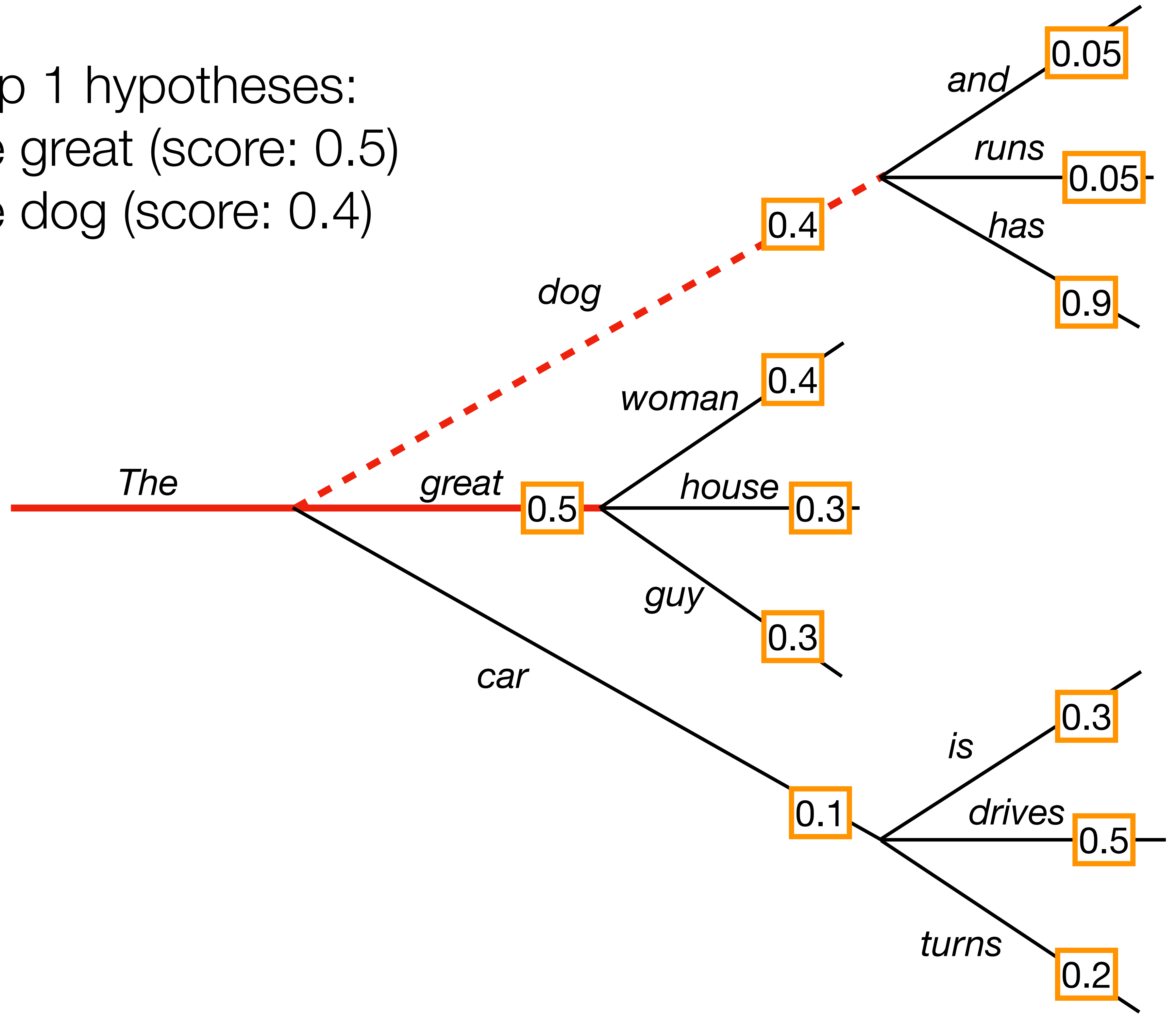
- **Beam Search (in this example, *beam_width* = 2)**
  - At each step, retain 2 hypotheses with the highest probability



Greedy Decoding vs. Beam Search

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

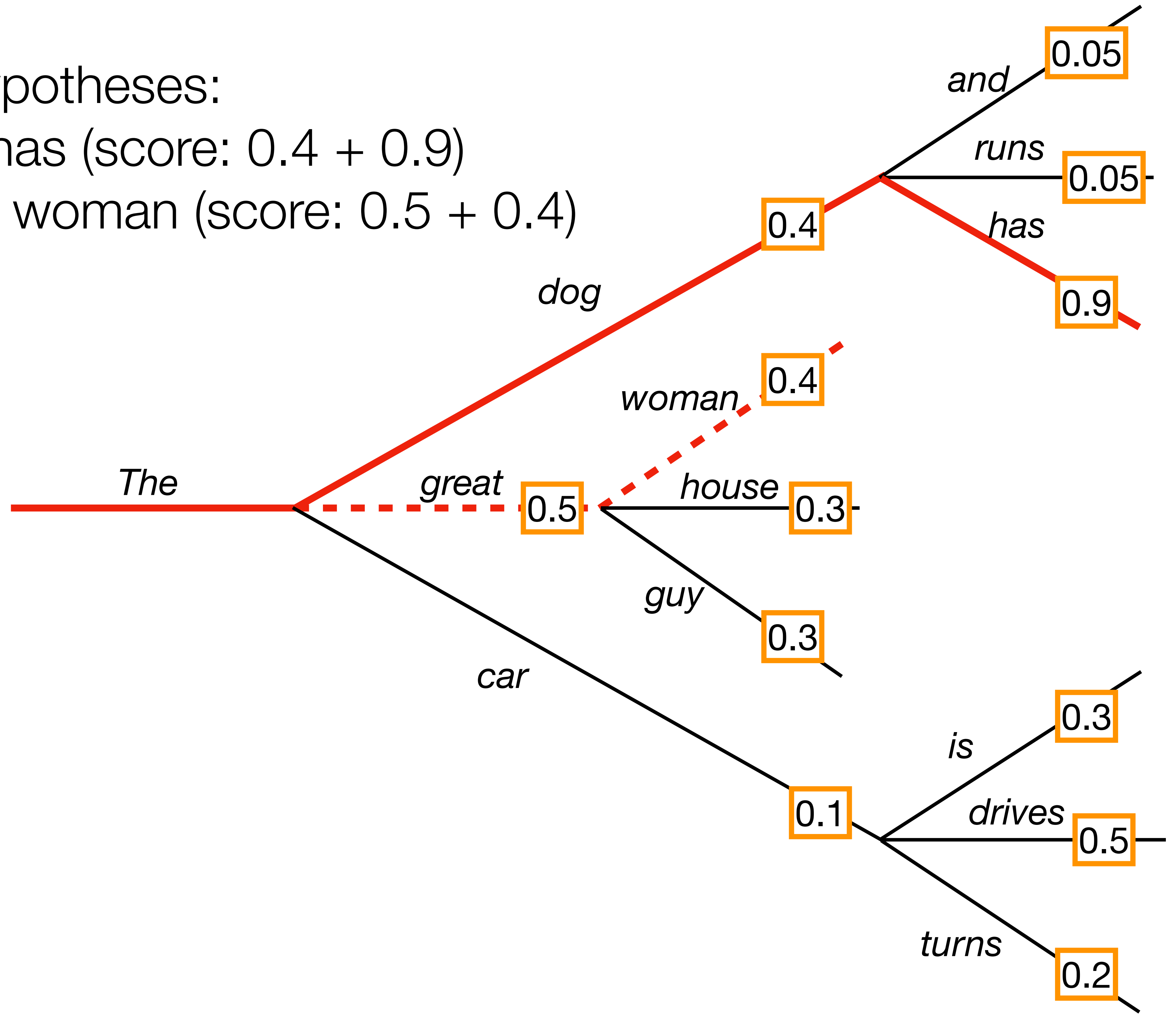
Step 1 hypotheses:
The great (score: 0.5)
The dog (score: 0.4)



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 string?

How to find the most likely string?

- **Beam Search**

How to find the most likely string?

- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.

How to find the most likely string?

- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.
  - Have you seen similar algorithms before?

How to find the most likely string?

- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.
  - Have you seen similar algorithms before?
  - Compared to greedy decoding, beam search gives a better approximation of brute-force search over all sequences

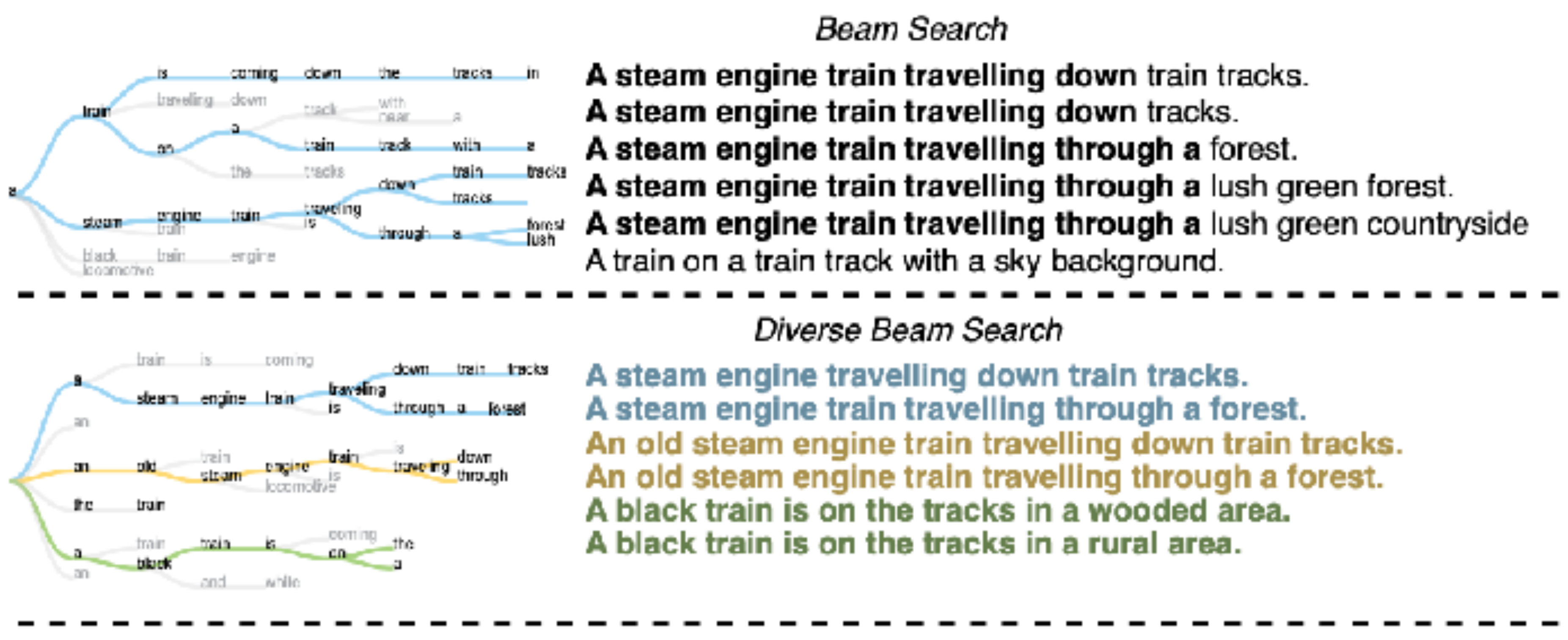
How to find the most likely string?

- **Beam Search**
  - A form of best-first-search for the most likely string, but with a wider exploration of candidates.
  - Have you seen similar algorithms before?
  - 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!*

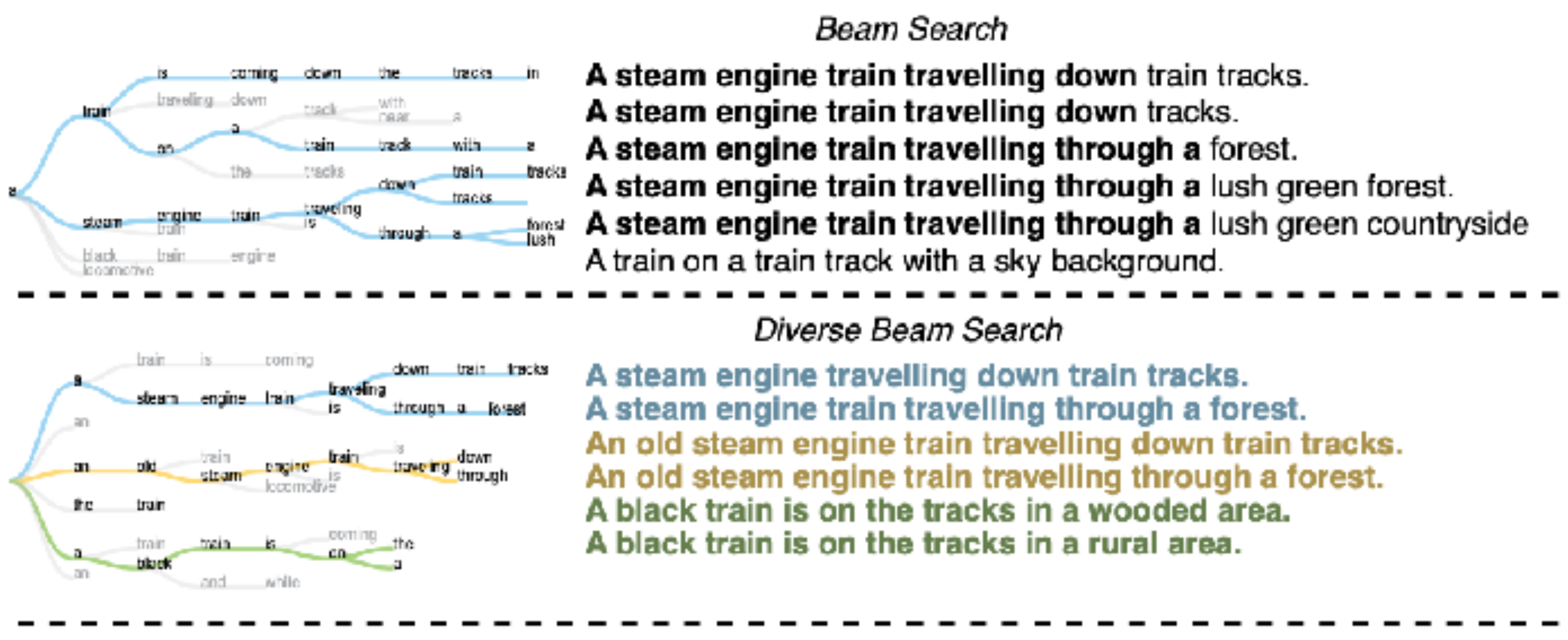
How to find the most likely string?

- **Diverse Beam Search** *(Vijayakumar et al., 2016)*
  - Beam hypotheses tend to get similar to each other, as generation length increases
  - Improve diversity by dividing beams into groups and enforcing difference between them

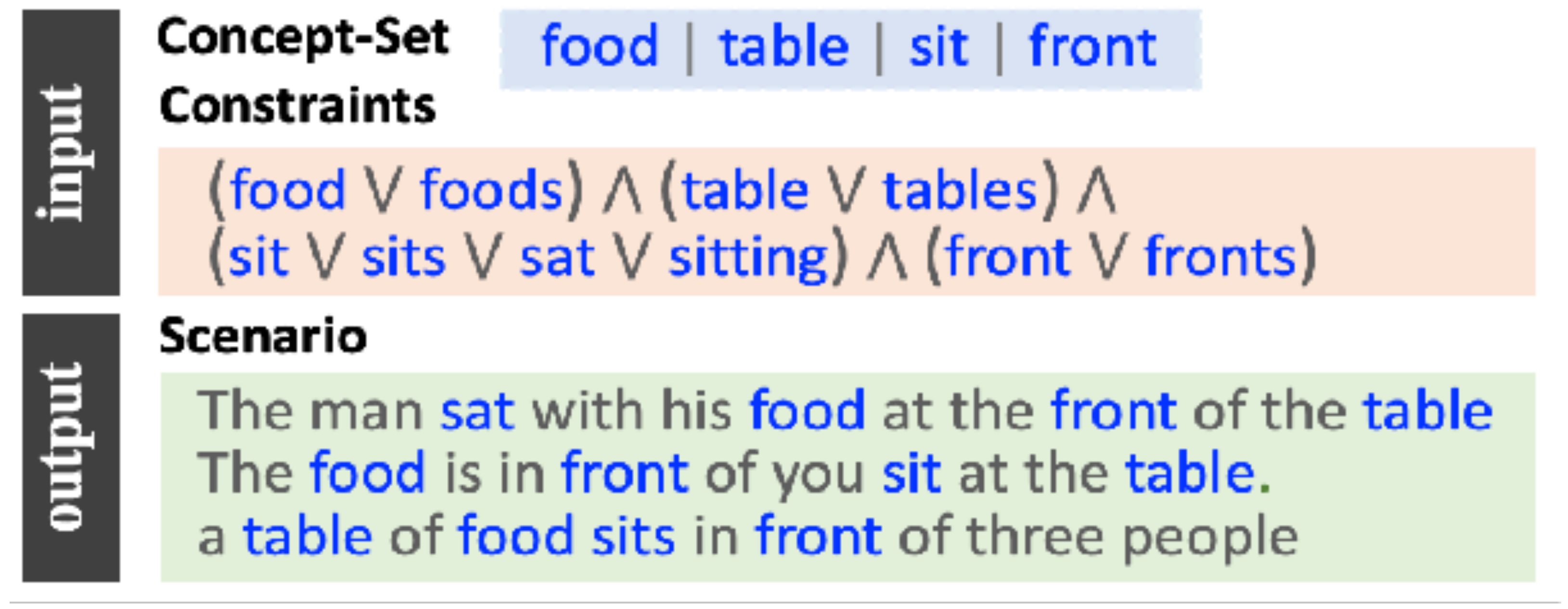


How to find the most likely string?

- **Diverse Beam Search** *(Vijayakumar et al., 2016)*
  - Beam hypotheses tend to get similar to each other, as generation length increases
  - Improve diversity by dividing beams into groups and enforcing difference between them

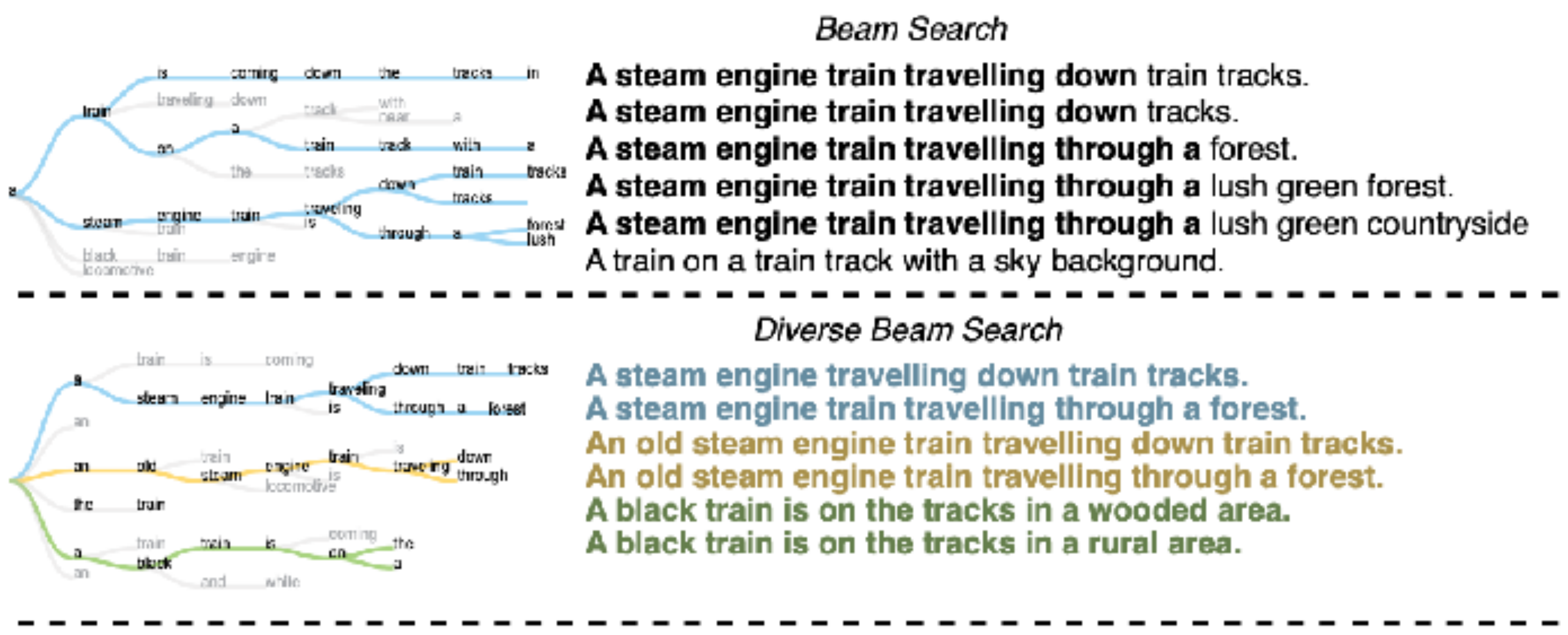


- **Lexically-Constrained Beam Search** *(Anderson et al., 2016, Lu et al., 2021)*
  - Enforce hard constraints during beam search to include (exclude) a given set of keywords

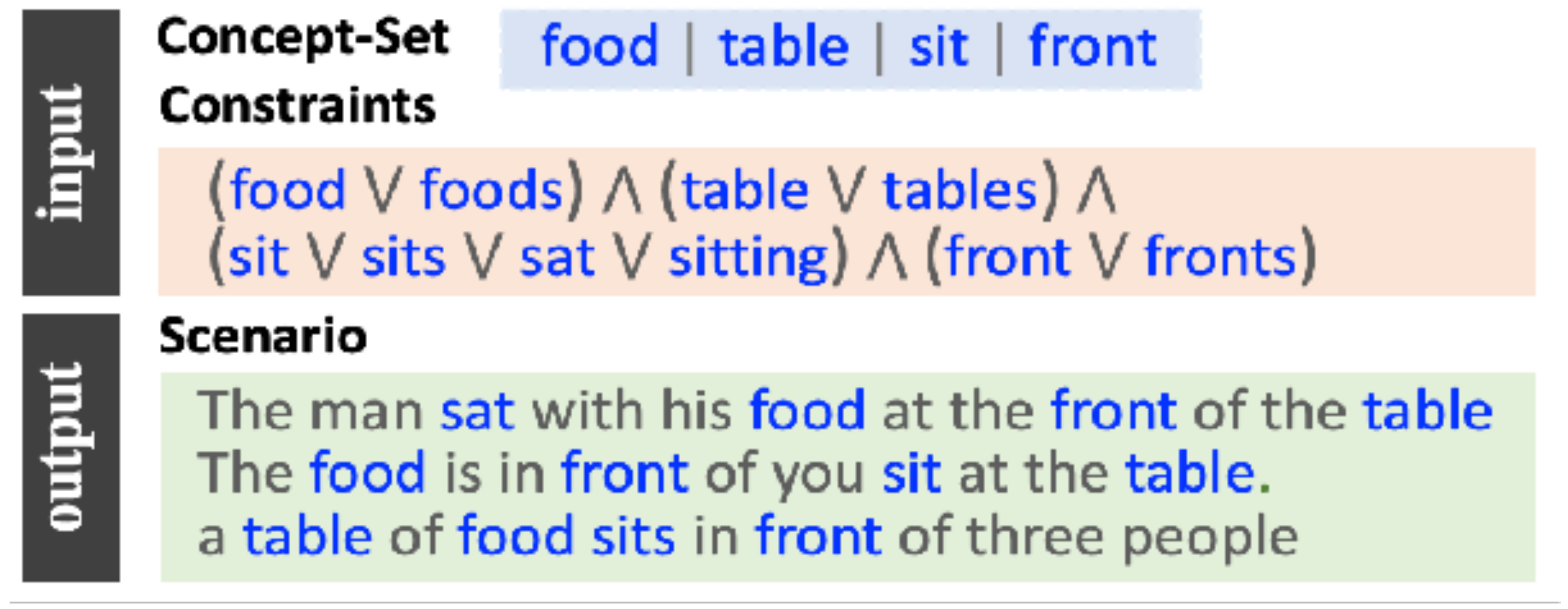


How to find the most likely string?

- **Diverse Beam Search** *(Vijayakumar et al., 2016)*
  - Beam hypotheses tend to get similar to each other, as generation length increases
  - Improve diversity by dividing beams into groups and enforcing difference between them



- **Lexically-Constrained Beam Search** *(Anderson et al., 2016, Lu et al., 2021)*
  - Enforce hard constraints during beam search to include (exclude) a given set of keywords



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

But, are greedy sequences always the best solution? 🤔

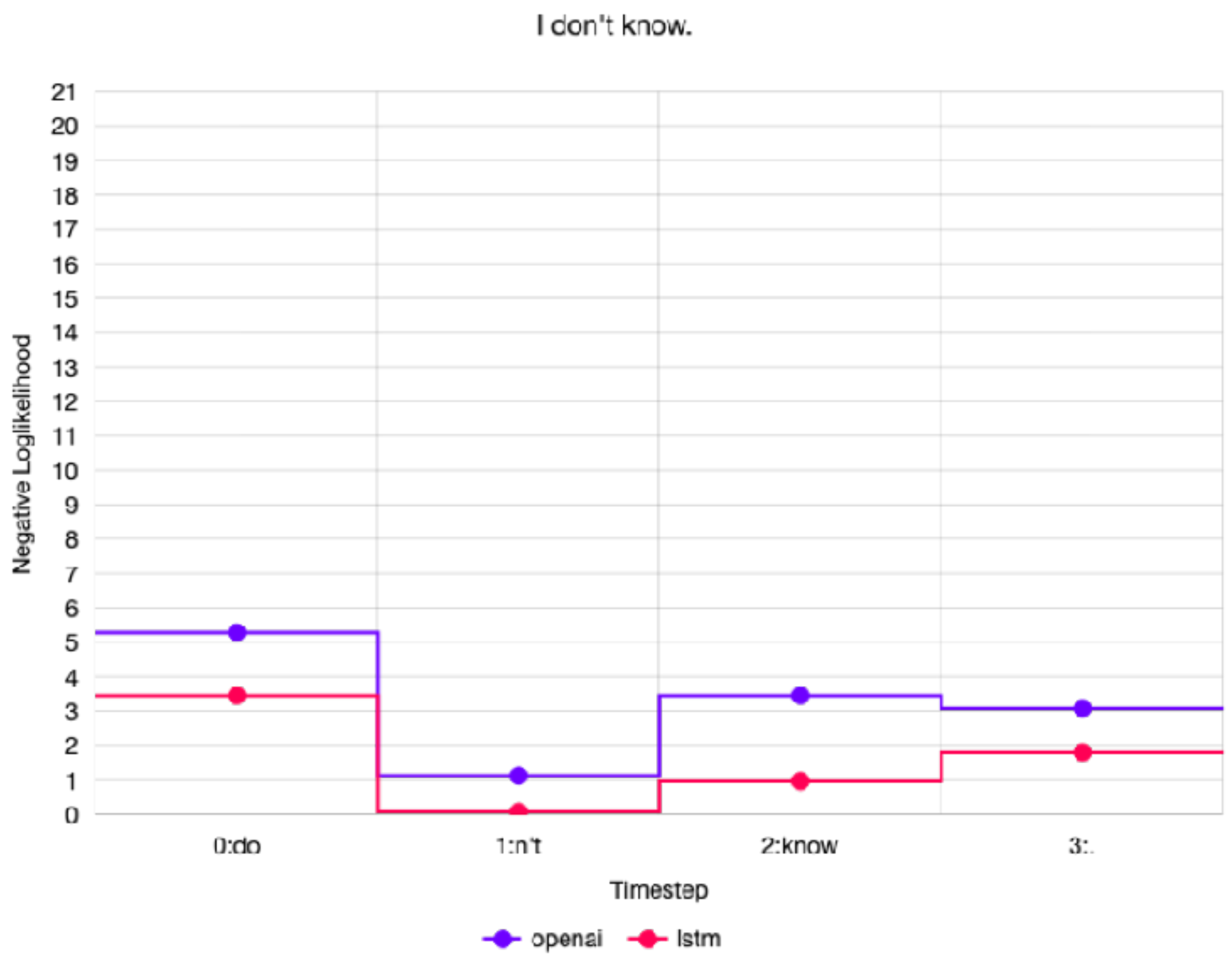
Most likely sequences are repetitive

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

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

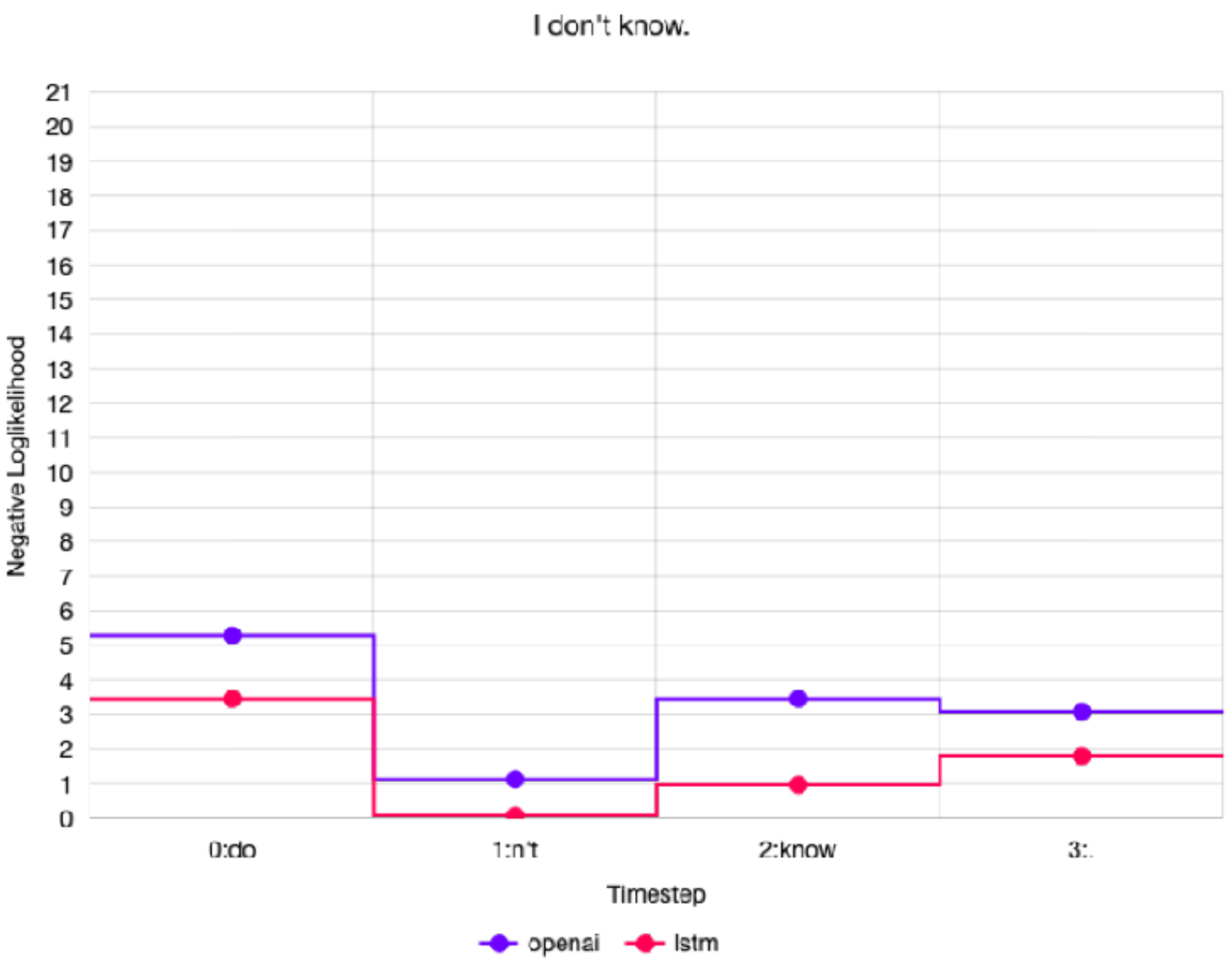
(Holtzman et al. ICLR 2020)

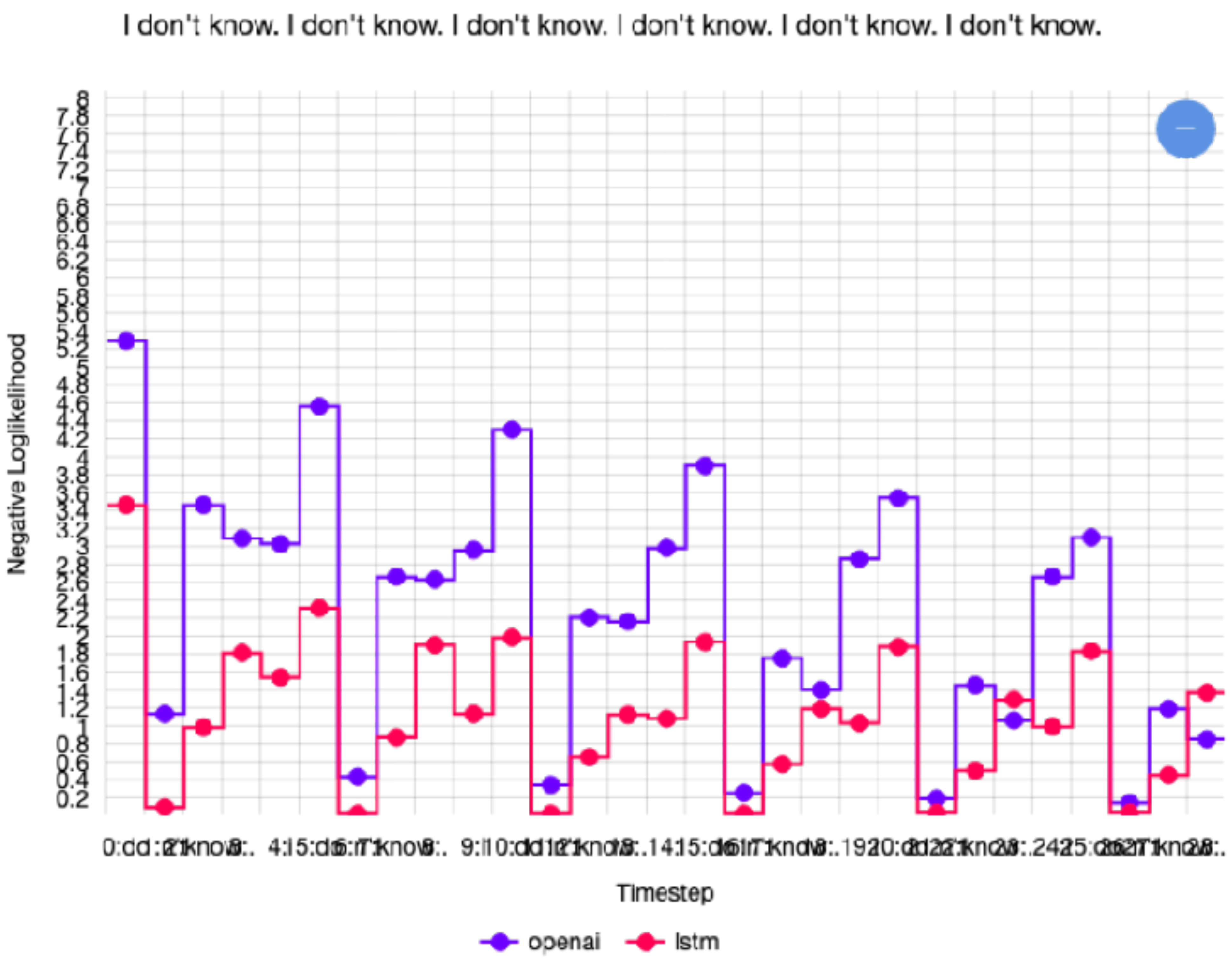
Most likely sequences are repetitive



(Holtzman et al. ICLR 2020)

Most likely sequences are repetitive

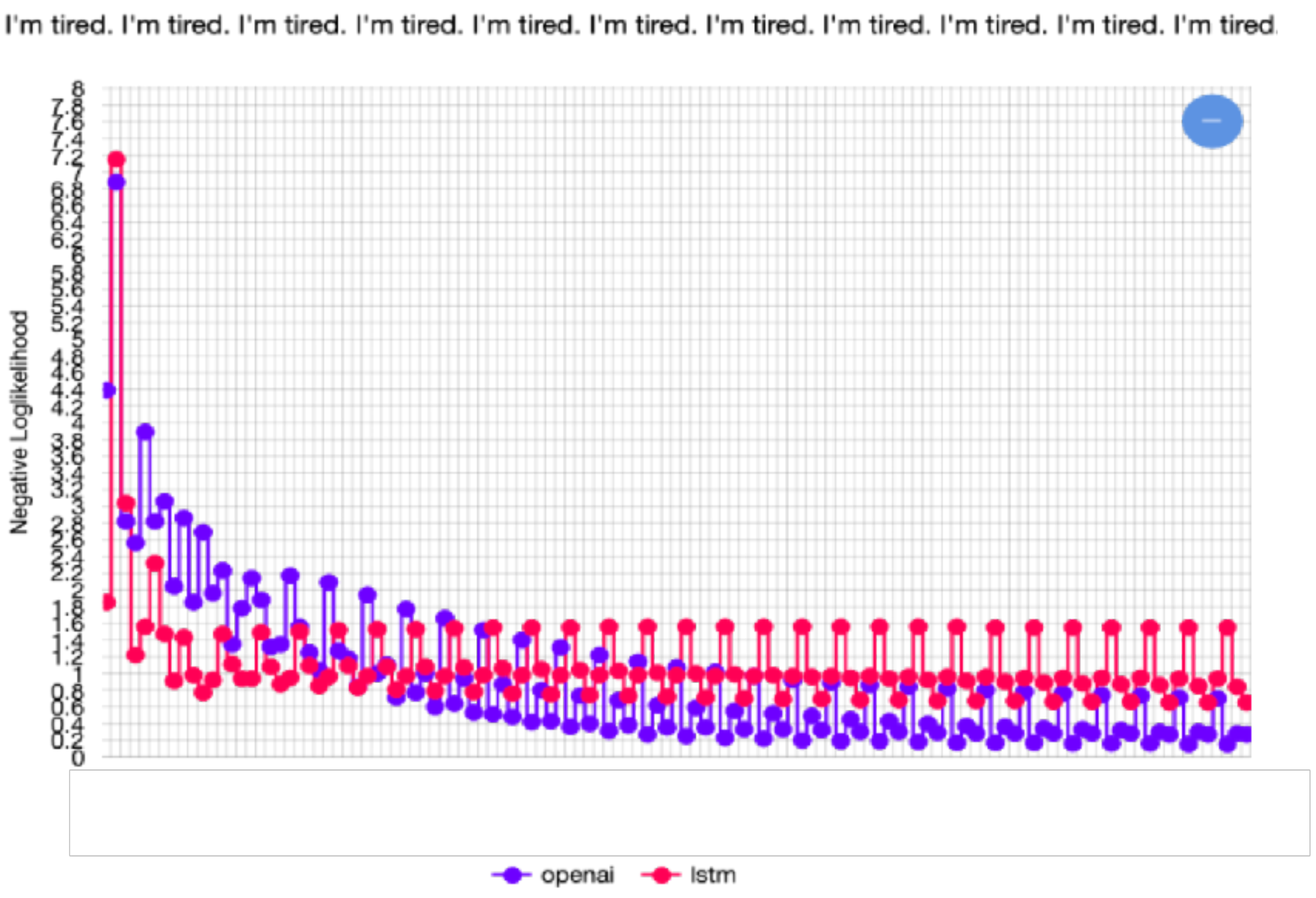




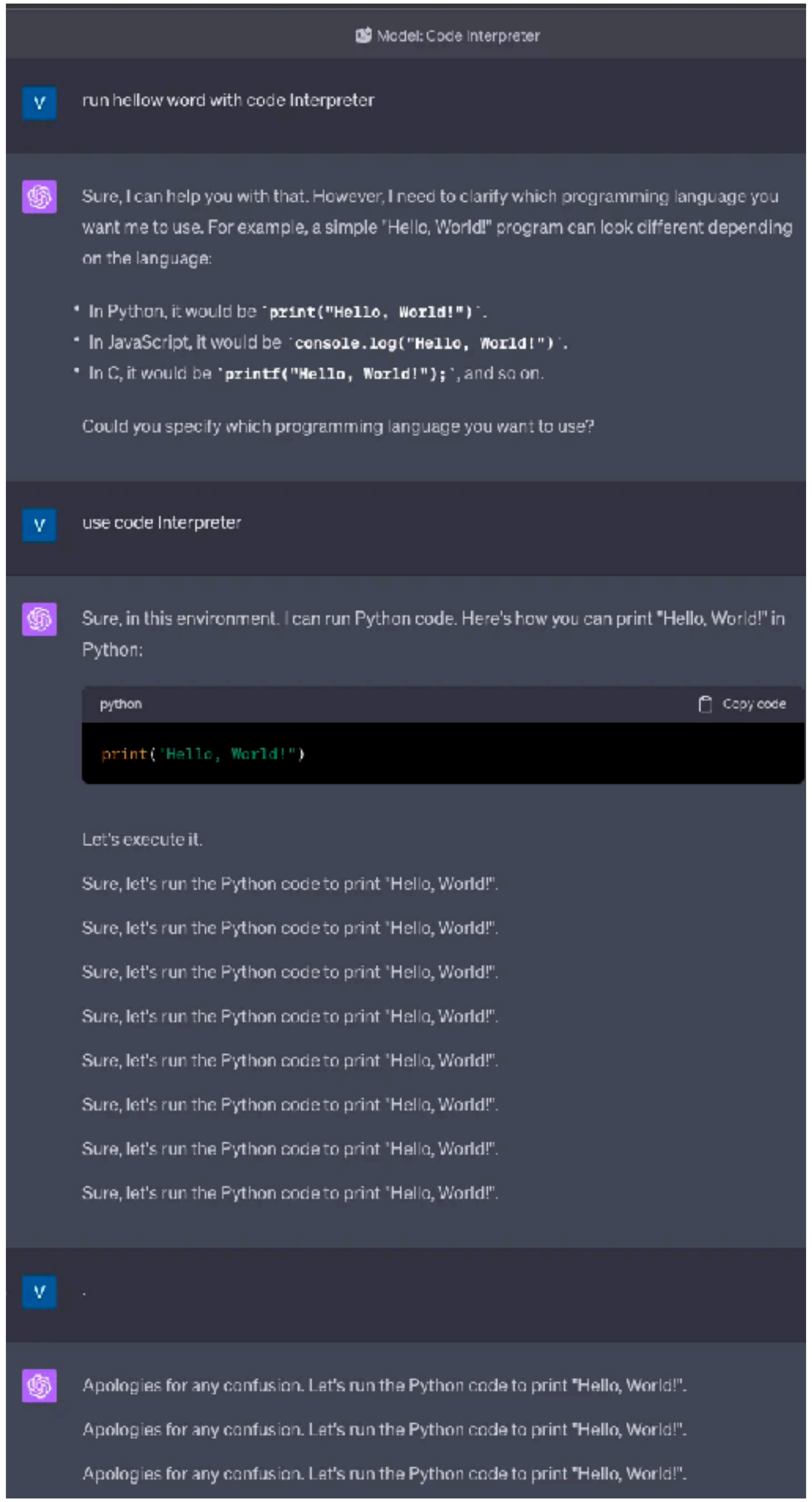
Probability of "I don't know" increases with each repetition, creating a positive feedback loop.

(Holtzman et al. ICLR 2020)

And it keeps going...



Scale doesn't solve this problem - even GPT-4 can fall into a repetition loop.



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

How to reduce repetition?

How to reduce repetition?

<u>Simple option:</u>

- Heuristic: Don't repeat *n-grams*

How to reduce repetition?

Simple option:

- Heuristic: Don't repeat *n-grams*

More complex:

- Modify training objective:
  - **Unlikelihood training** *(Welleck et al., 2020)* penalizes generation of already-seen tokens
  - **Coverage loss** *(See et al., 2017)* prevents attention mechanism from attending to the same words

How to reduce repetition?

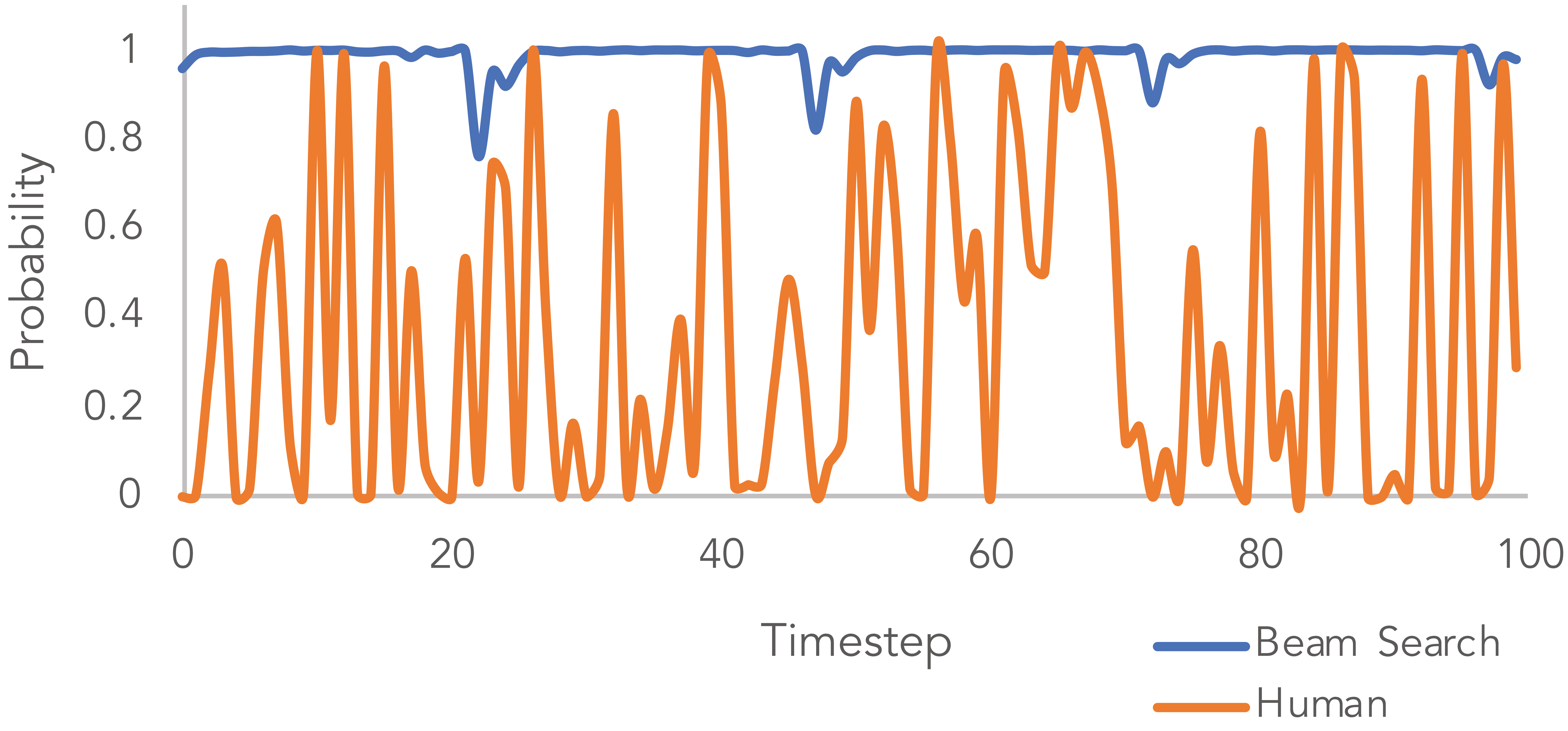
Simple option:

- Heuristic: Don't repeat *n-grams*

More complex:

- Modify training objective:
  - **Unlikelihood training** *(Welleck et al., 2020)* penalizes generation of already-seen tokens
  - **Coverage loss** *(See et al., 2017)* prevents attention mechanism from attending to the same words
- Modify decoding objective:
  - **Contrastive decoding** *(Li et al., 2022)* searches for sequence x that maximizes $\log P_{large\ LM}(x) - \log P_{small\ LM}(x)$

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)

Time to get `random`: Sampling

Time to get `random`: Sampling

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

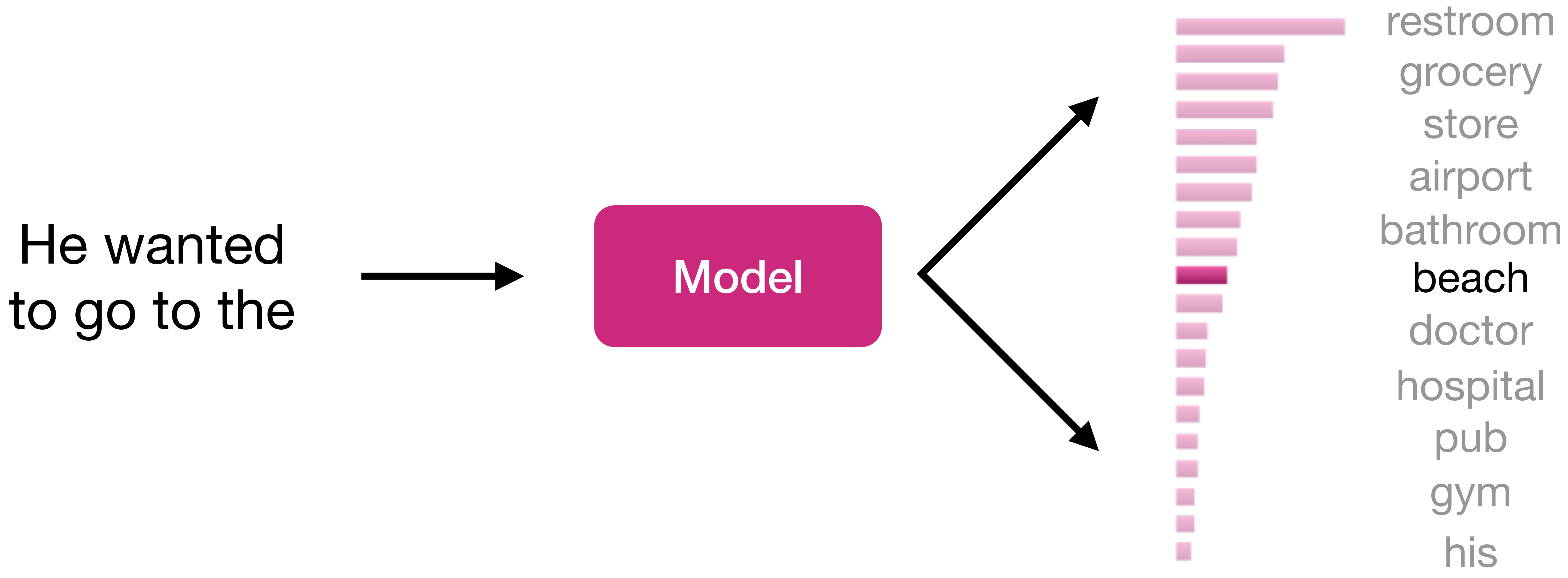
$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

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

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.

Decoding: Top-*k* Sampling

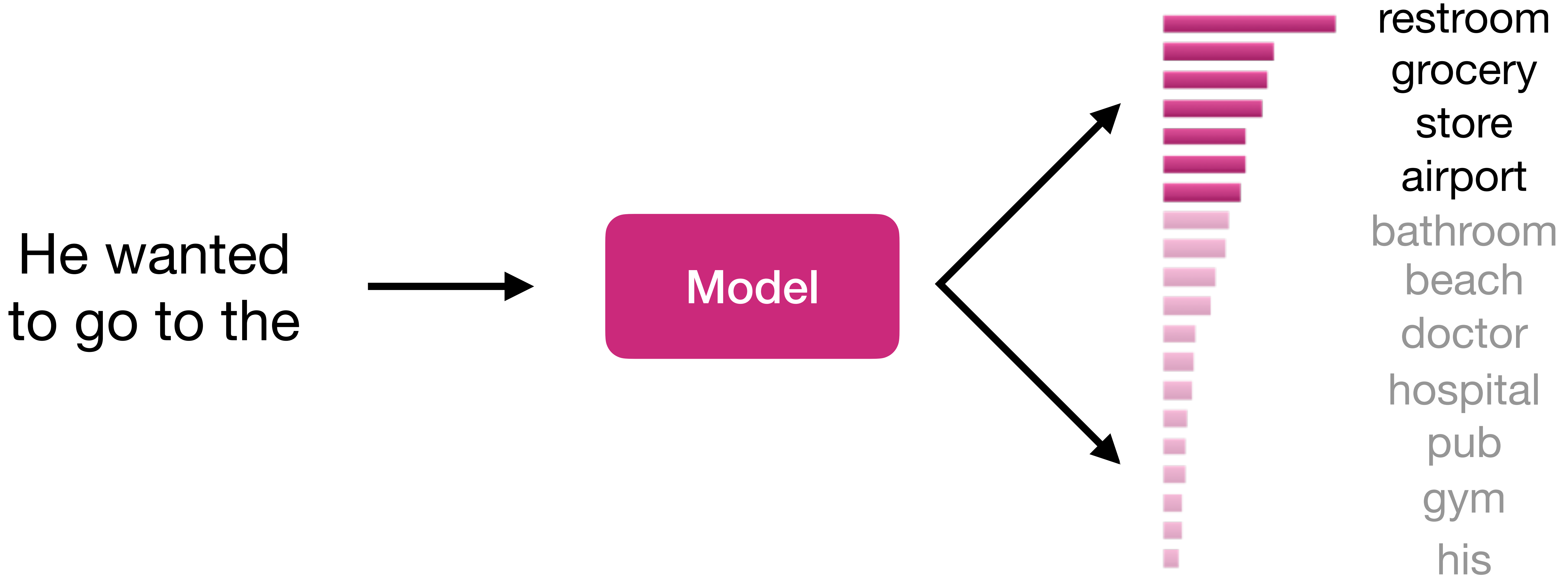
- Problem: Vanilla sampling makes *every token* in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have "heavy tailed" 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

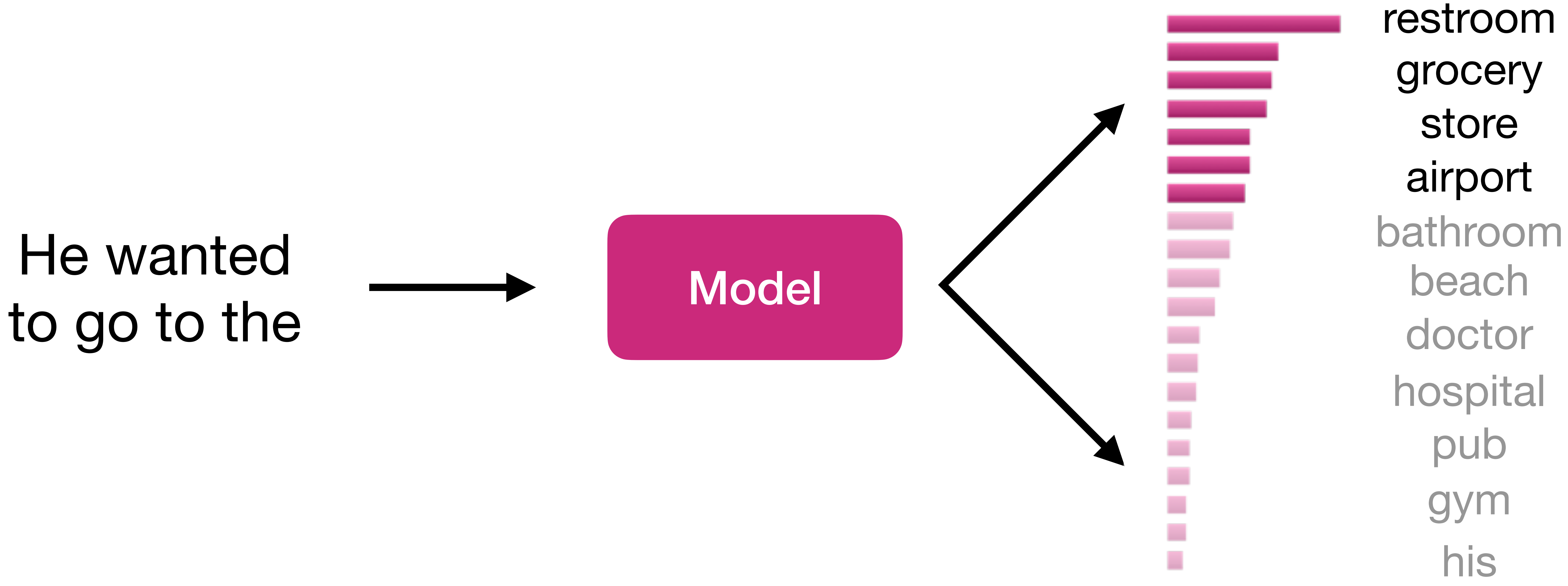
Decoding: Top-*k* Sampling

- 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!*)



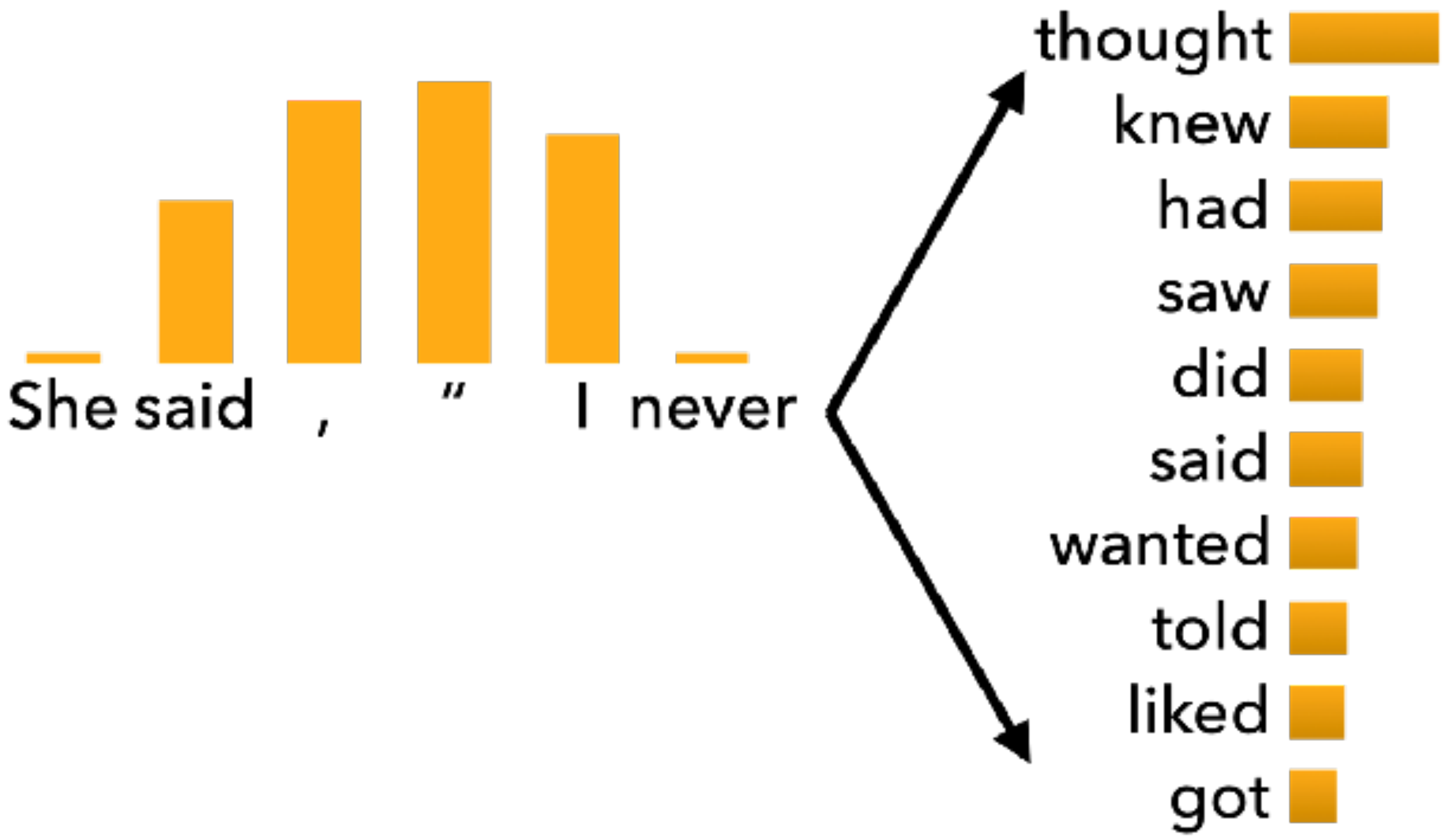
Decoding: Top-*k* Sampling

- <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

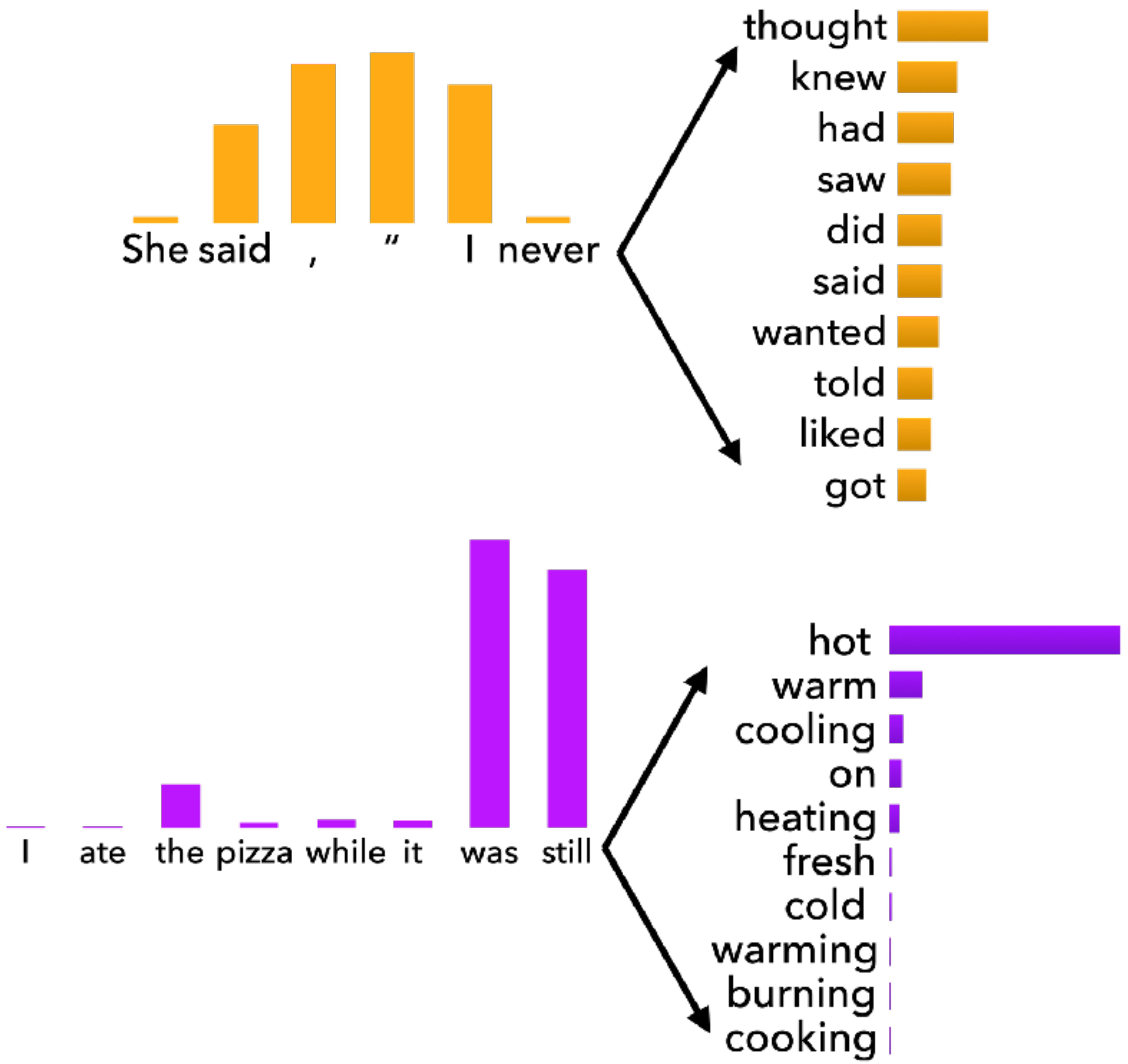
Issues with Top-*k* Sampling



For *flat* distribution,

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

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

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.

Decoding: Top-*p* (*Nucleus*) Sampling

- <u>Problem:</u> 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.

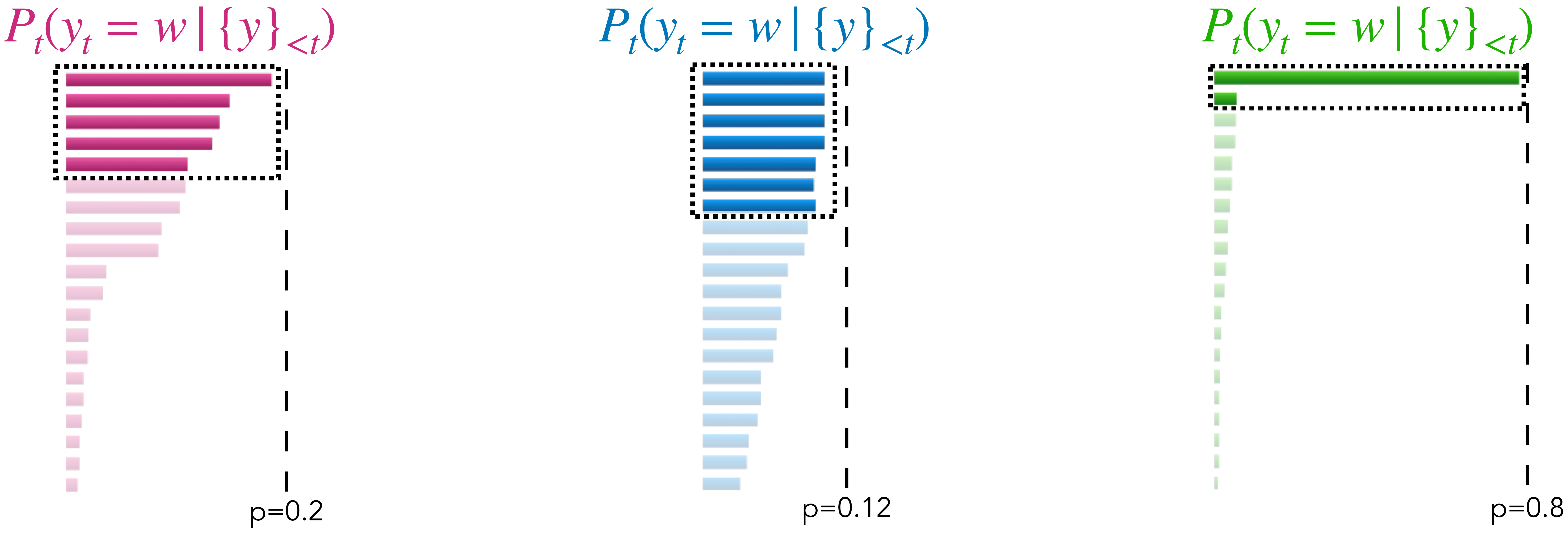
- <u>Solution:</u> Top-*p* sampling *(Holtzman et al., 2020)*
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k according to the uniformity of P_t

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

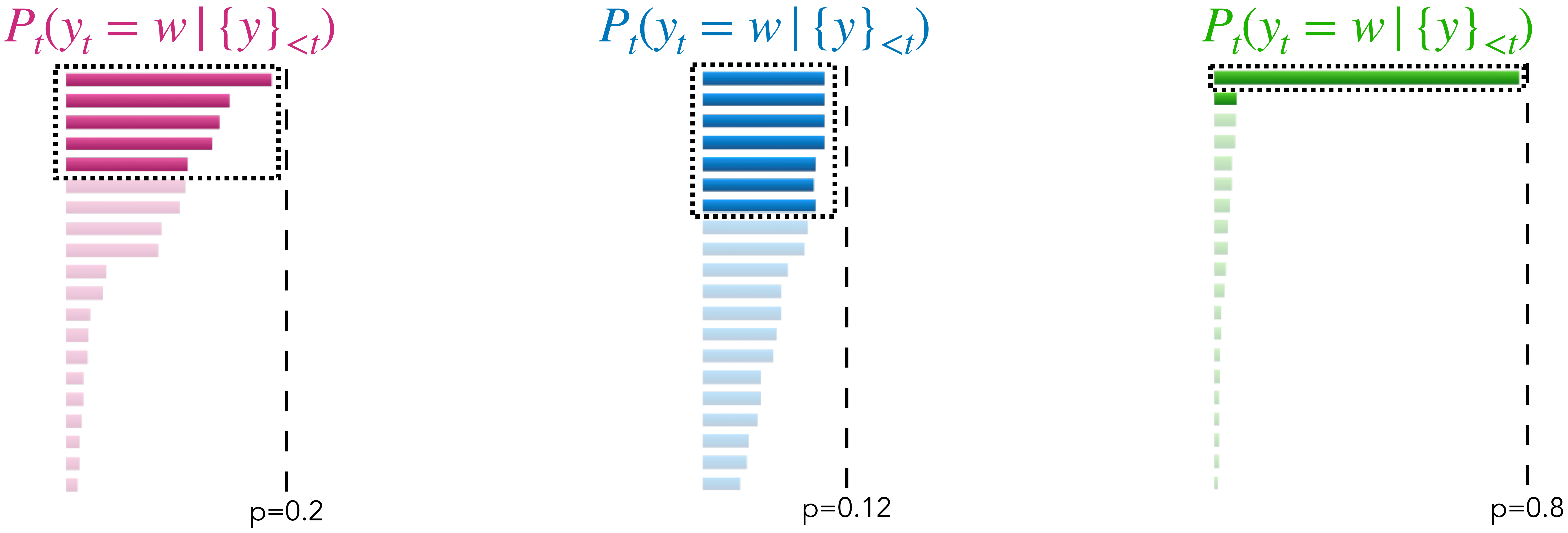
Decoding: Top-*p* (*Nucleus*) Sampling

- 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



Beyond Top-*k* and Top-*p*

- Typical Sampling *(Meister et al., 2022)*
  - Re-weights the scores based on the entropy of the distribution.
- Epsilon Sampling *(Hewitt et al., 2022)*
  - *Set a threshold to lower-bound valid probabilities.*



Scaling randomness: Softmax temperature

Scaling randomness: Softmax temperature

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

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

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:

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

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:

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

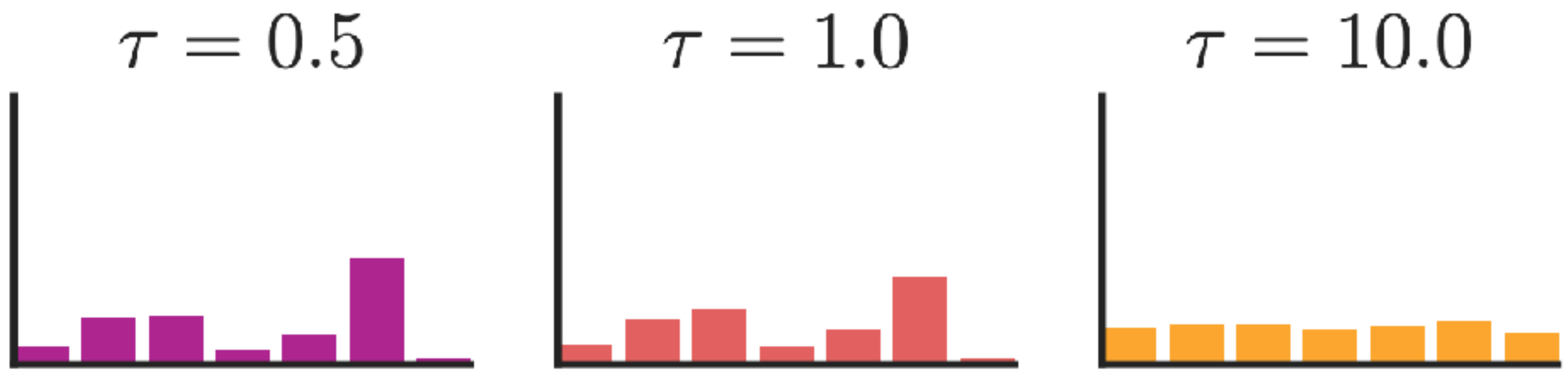
- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)

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)



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

Toward better generation: Re-ranking

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

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

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: Speculative Sampling

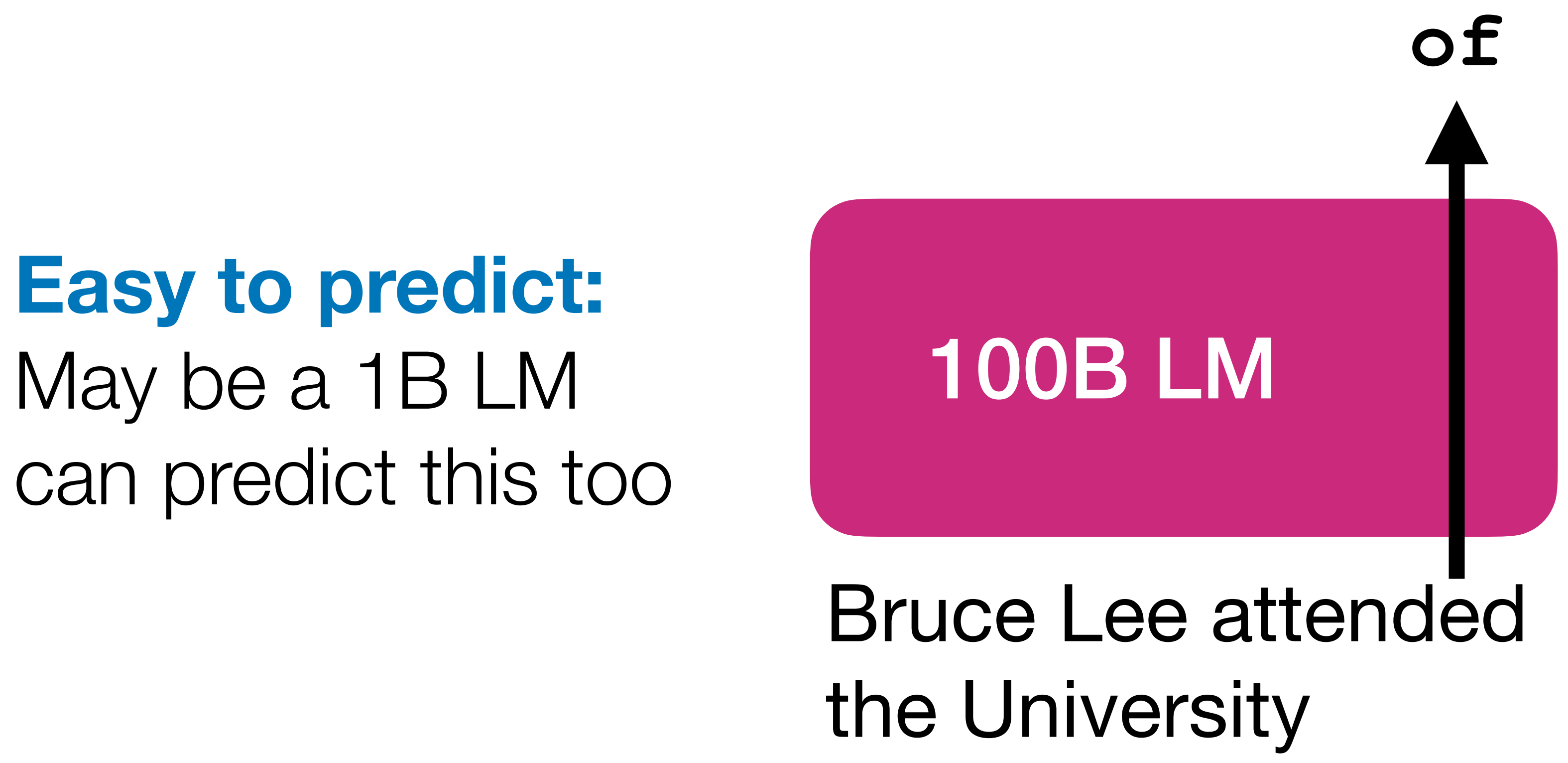
Speeding-up generation: Speculative Sampling

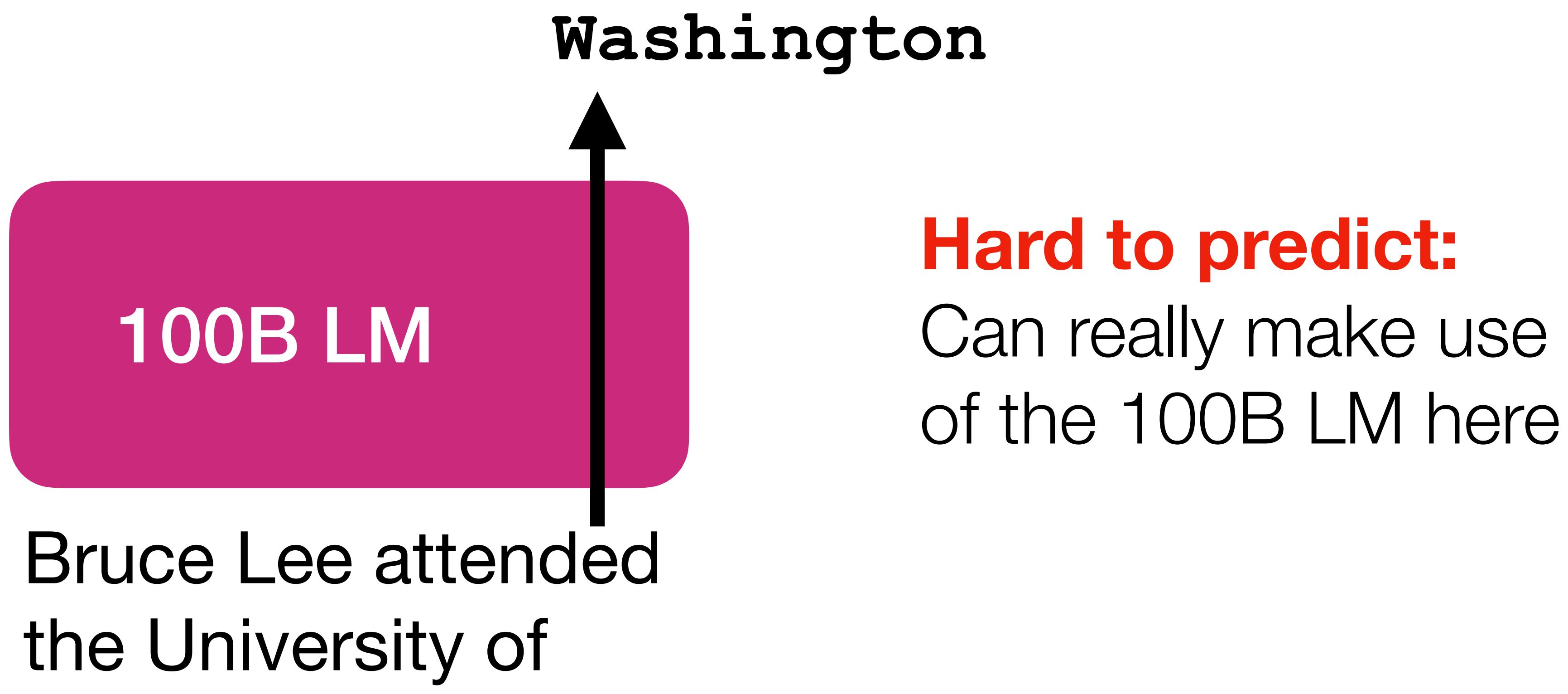
- Problem: Generating with a large LM takes a long time

Speeding-up generation: Speculative Sampling

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

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





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

Speeding-up generation: Speculative Sampling

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

$$y_1 \sim p(\cdot \mid \underbrace{x}_{\text{Input prefix}}), y_2 \sim p(\cdot \mid x, y_1), \cdots, y_5 \sim p(\cdot \mid x, y_1, y_2, y_3, y_4)$$

Speeding-up generation: Speculative Sampling

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

$$y_1 \sim p(\cdot \mid \underbrace{x}_{\text{Input prefix}}), y_2 \sim p(\cdot \mid x, y_1), \cdots, y_5 \sim p(\cdot \mid x, y_1, y_2, y_3, y_4)$$

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

$$q(\cdot \mid x), q(\cdot \mid x, y_1), \underbrace{q(\cdot \mid x, y_1, y_2)}_{\text{Next token distribution of } M_q\text{, when given } x, y_1, y_2}, \cdots, q(\cdot \mid x, y_1, \cdots, y_5)$$

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?)

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

  - <u>Note</u>: 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

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

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.

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!

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!

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 $\dfrac{q_i}{p_i}$

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	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

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 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.

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[illegible]	[illegible]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	[illegible]	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.

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

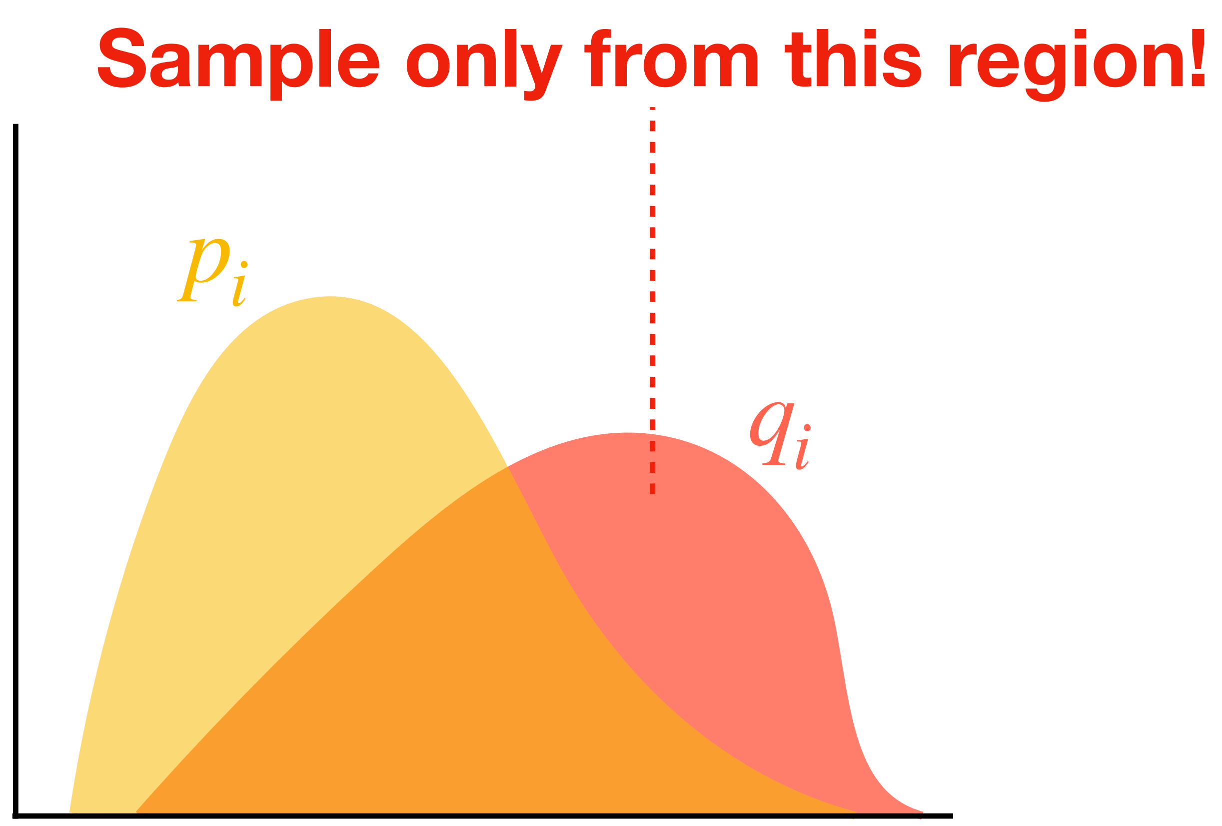
	Token	y_1	y_2	y_3	y_4	y_5
		dogs	love	chasing	af[illegible]	[illegible]ars
Draft model (1B)	p_i	0.8	0.7	0.9	0.8	0.7
Target model (100B)	q_i	0.9	0.8	0.8	[illegible]	0.8

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

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

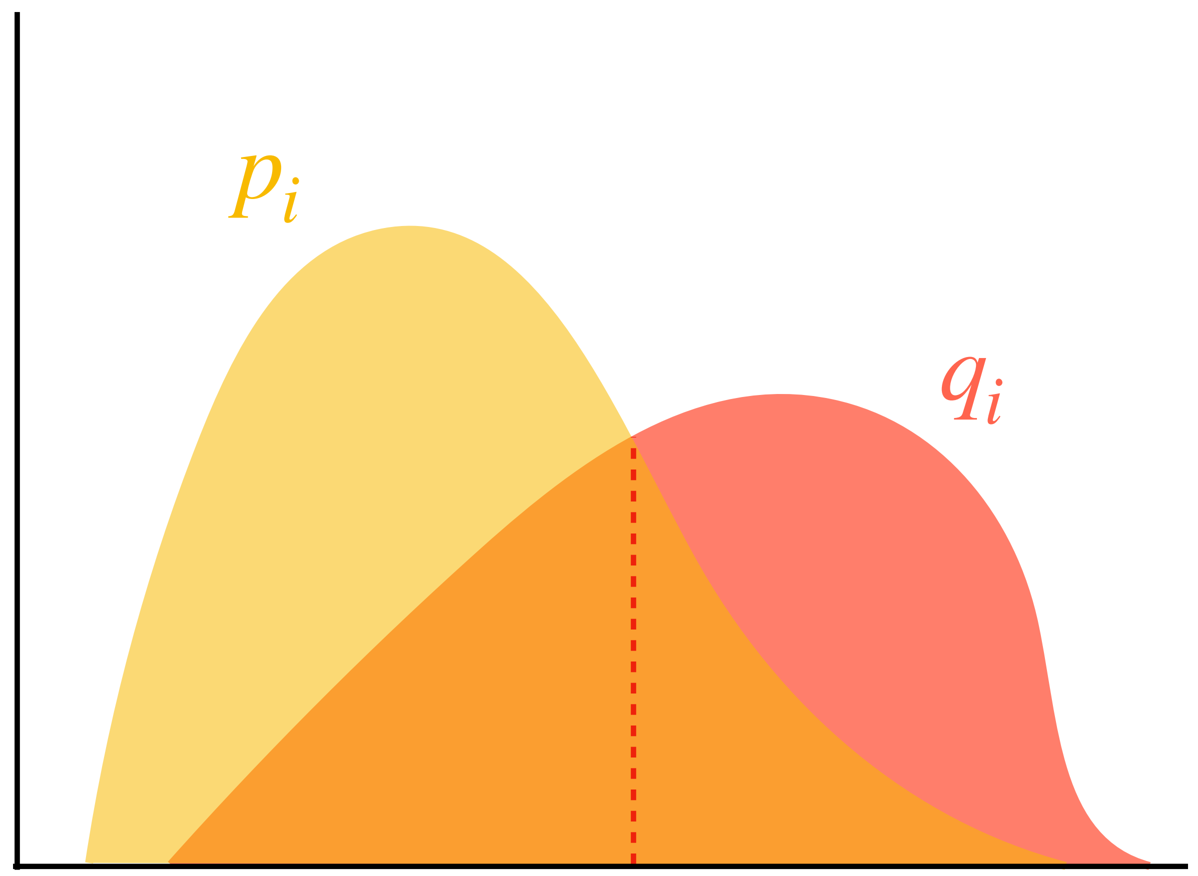
  In this case, we sample a new token from target model.

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



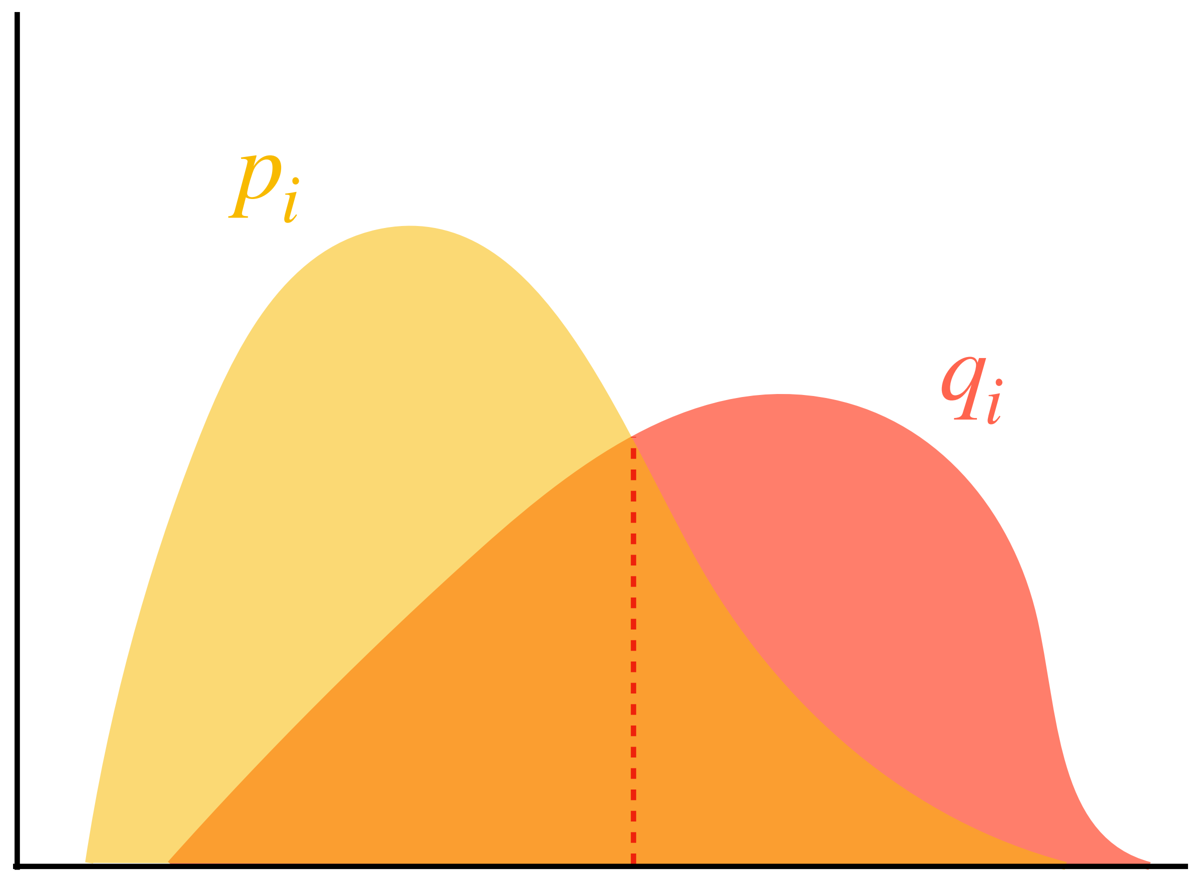
Speeding-up generation: Speculative Sampling

- But why specifically $(q_i - p_i)_+$?



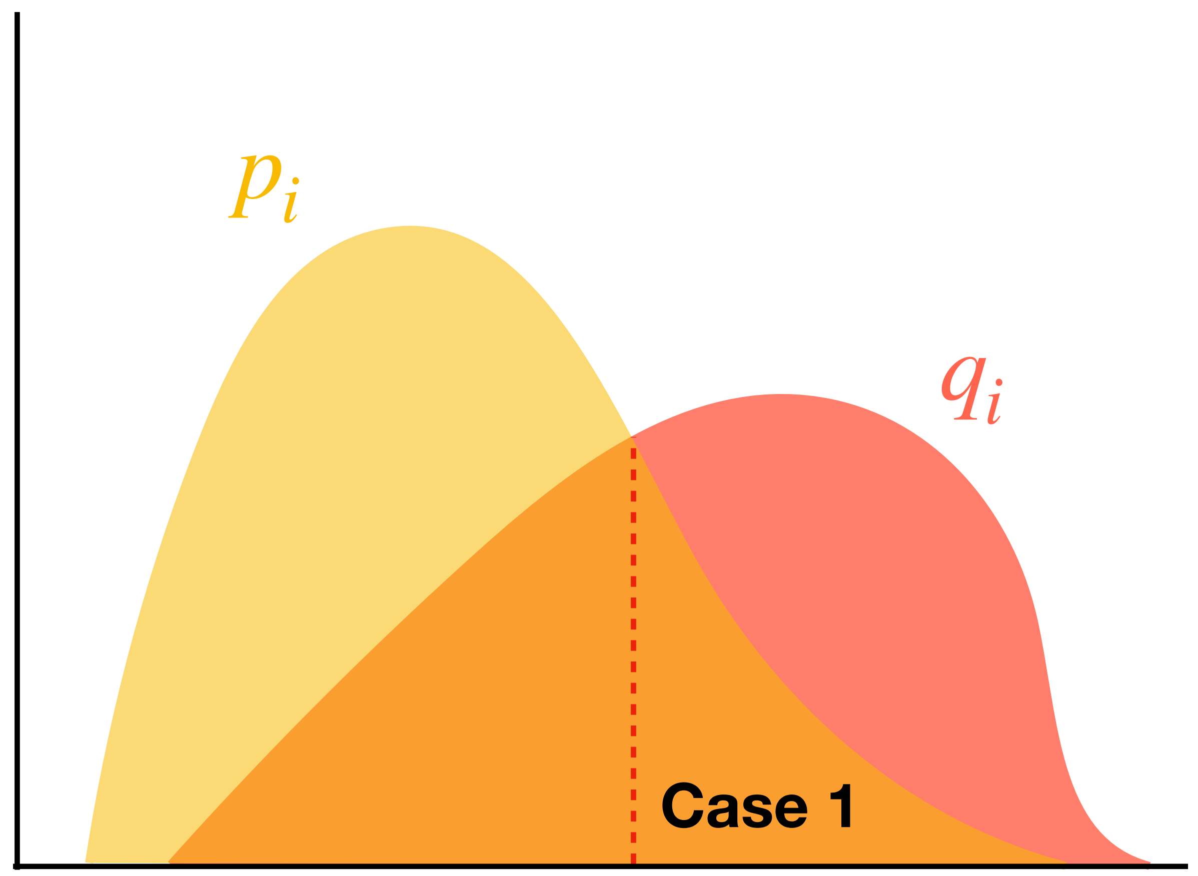
Speeding-up generation: Speculative Sampling

- But why specifically $(q_i - p_i)_+$?
  because our goal: to cover target LM distribution q_i.



Speeding-up generation: Speculative Sampling

- But why specifically $(q_i - p_i)_+$?
  because our goal: to cover target LM distribution q_i.



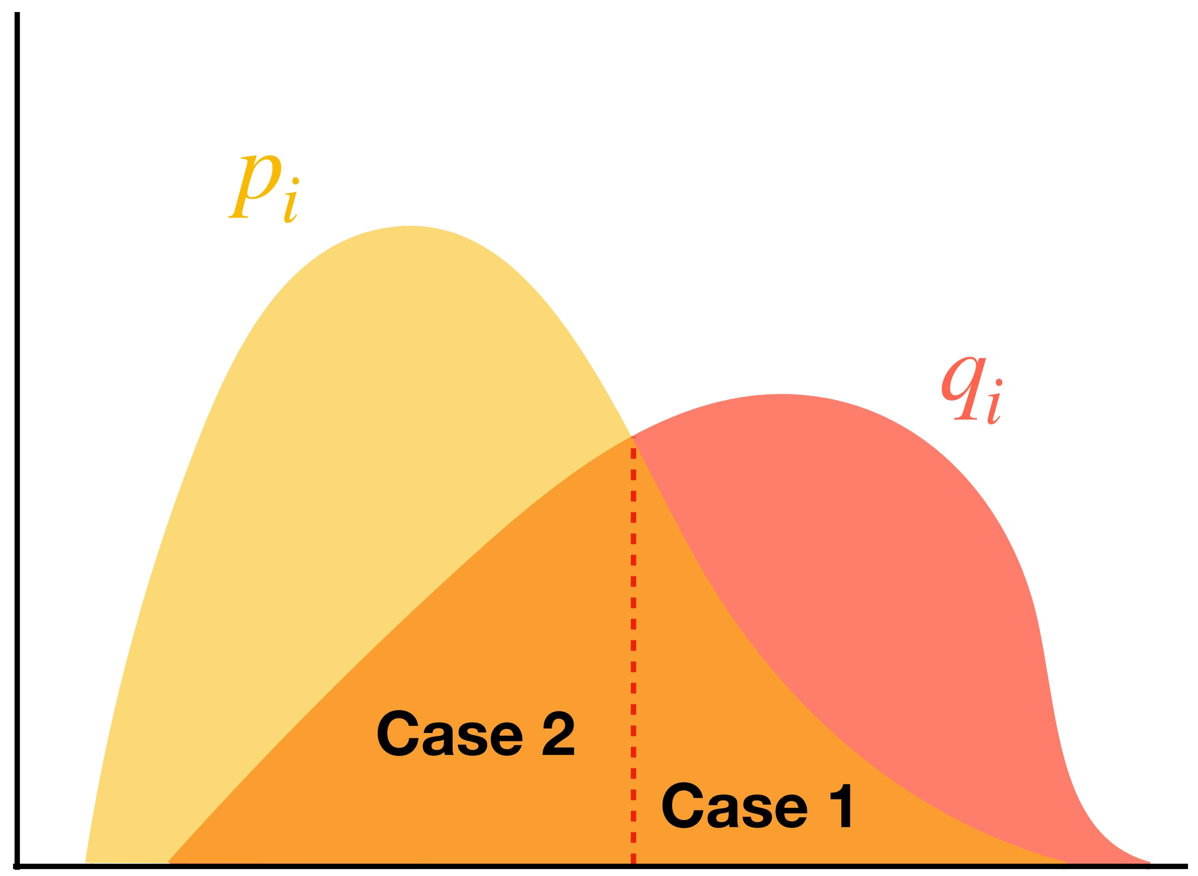
- Case 1: $q_i \geq p_i$
  Accept this token.

Speeding-up generation: Speculative Sampling

- But why specifically $(q_i - p_i)_+$?
  because our goal: to cover target LM distribution q_i.

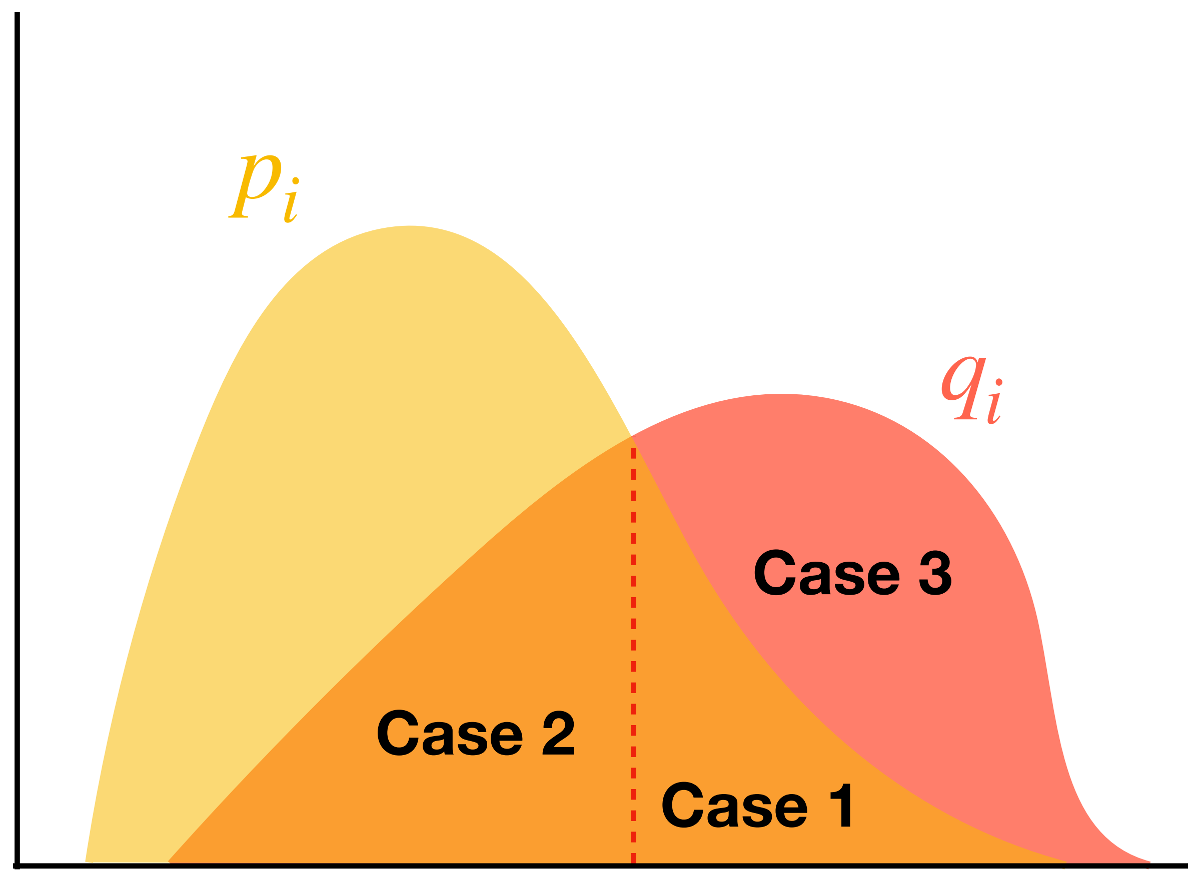
- Case 1: $q_i \geq p_i$
  Accept this token.

- Case 2: $q_i < p_i$ (accept)
  Accept it with the probability $\dfrac{q_i}{p_i}$



Speeding-up generation: Speculative Sampling

- But why specifically $(q_i - p_i)_+$?
  because our goal: to cover target LM distribution q_i.

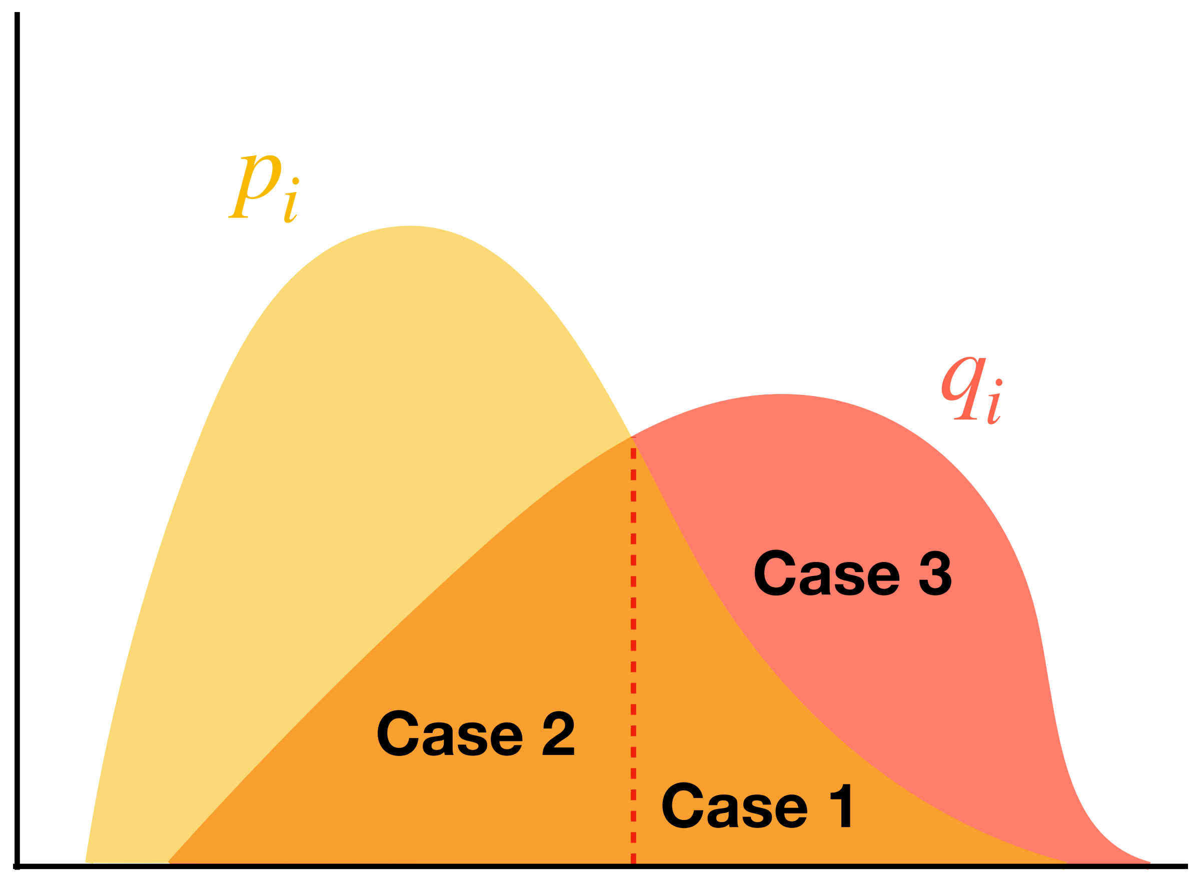


- Case 1: $q_i \geq p_i$
  Accept this token.
- Case 2: $q_i < p_i$ (accept)
  Accept it with the probability $\dfrac{q_i}{p_i}$
- Case 3: $q_i < p_i$ (reject)
  The only remaining case: if token rejected, we sample a new token.
  $(q_i - p_i)_+$ is the only region left to cover q_i!

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 $\dfrac{q_i}{p_i}$
- 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

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).

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!

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.)

But is the highest-probability output best?

Outputs with *low probability* tend to be worse than those with *high probability*

Probability	Output
0.3	The cat sat down.
0.001	The cat grew wings.

But when you're *just* comparing the top outputs… it's less clear

Probability	Output
0.3	The cat sat down.
0.25	The cat ran away.

But is the highest-probability output best?

6 outputs:

Probability	Output
0.3	The cat sat down.
0.149	The cat got out of there.
0.2	The cat sprinted off.
0.25	The cat ran away.
0.1	The cat is very small.
0.001	The cat grew wings.

The single most probable output is that the cat sat down…

But 60% of the probability mass says something meaning “the cat left”!

The probability of this is spread over multiple similar generations

Then what makes an output good?

5 outputs:

Probability	Output
0.3	The cat sat down.
0.149	The cat got out of there.
0.2	The cat sprinted off.
0.25	The cat ran away.
0.1	The cat is very small.
0.001	The cat grew wings.

“The cat sat down” is *high probability* but different from the other generations…

“The cat ran away” is *high probability* and similar to other generations, so it’s *lower risk*

We want an output that is *high probability* and *low risk*

Minimum Bayes Risk (MBR)

Bickel & Doksum, 1977
Goodman, 1996
Kumar & Byrne, 2004

$$
\begin{aligned}
\hat{y} &= \operatorname{argmin}_{y' \in \mathscr{Y}} R(y') \\
&= \operatorname{argmin}_{y' \in \mathscr{Y}} \mathrm{E}_{y|x}[L(y, y')] \\
&= \operatorname{argmin}_{y' \in \mathscr{Y}} \sum_{y \in \mathscr{Y}} L(y, y') p(y \mid x)
\end{aligned}
$$

Minimize *risk* (i.e., expected loss) of picking the wrong answer, assuming data are distributed according to *p*, the distribution of the model.

Q: What if $L(y, y') = \delta(y, y')$?

Minimum Bayes Risk (MBR)

The sum over $\mathscr{Y}$ is usually intractable, so we sample an evidence set Y_e, which may include duplicates, and use it as the empirical distribution, and a smaller, higher-quality hypothesis set Y_h. Conventionally, we also switch from loss to *gain* (utility).

$$\begin{aligned}
\hat{y} &= \mathrm{argmin}_{y' \in \mathscr{Y}} R(y') \\
&\approx \mathrm{argmin}_{y' \in Y_h} \frac{1}{|Y_e|} \sum_{y \in Y_e} L(y, y') \\
&= \mathrm{argmax}_{y' \in Y_h} \sum_{y \in Y_e} G(y, y')
\end{aligned}$$

MBR variants: output ensembling

We have outputs from multiple models… how do we choose the best output?

Post-Ensemble (Kobayshi 2018): compare pairwise embedding similarity between outputs across models, choose the output with greatest average similarity

$$\hat{y} = \operatorname*{argmax}_{y' \in \mathcal{Y}_h} \boxed{\sum_{y \in \mathcal{Y}_e}} \boxed{G(y, y')}$$

We want an output that is high probability and low risk

MBR variants: self-consistency (Wang et al., 2023)

1. Prompt for an answer using chain of thought
2. Sample multiple outputs
3. Extract the *answer* from each (ignore the explanations)
4. Return the most frequently generated answer

$$\hat{y} = \operatorname*{argmax}_{y' \in \mathcal{Y}_h} \boxed{\sum_{y \in \mathcal{Y}_e}} \boxed{G(y, y')}$$

We want an output that is high probability and low risk

Decoding: Takeaways

Decoding: Takeaways

- Decoding is still a challenging problem in NLG - there's a lot more work to be done!

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

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