CS11-711 Advanced NLP

Decoding Algorithms

Sean Welleck

https://cmu-l3.github.io/anlp-fall2025/

https://github.com/cmu-l3/anlp-fall2025-code

Slides adapted from:
Matthew Finlayson (NeurIPS 2024 Tutorial) and Amanda Bertsch (Spring 2025 Guest Lecture)

Recap

- **Modeling/parameterization**
  - Classification or generation?
  - Autoregressive?
  - Which architecture?
- **Learning**
  - Maximum likelihood or other?
  - Pre-train first?
  - What data or supervision can I leverage?
- **Today**: ***Inference***
  - Using a model after learning

Today: generating outputs with a language model

Today’s lecture

- Basic setup
- Decoding objectives and algorithms
- Speeding up decoding

Basic setup

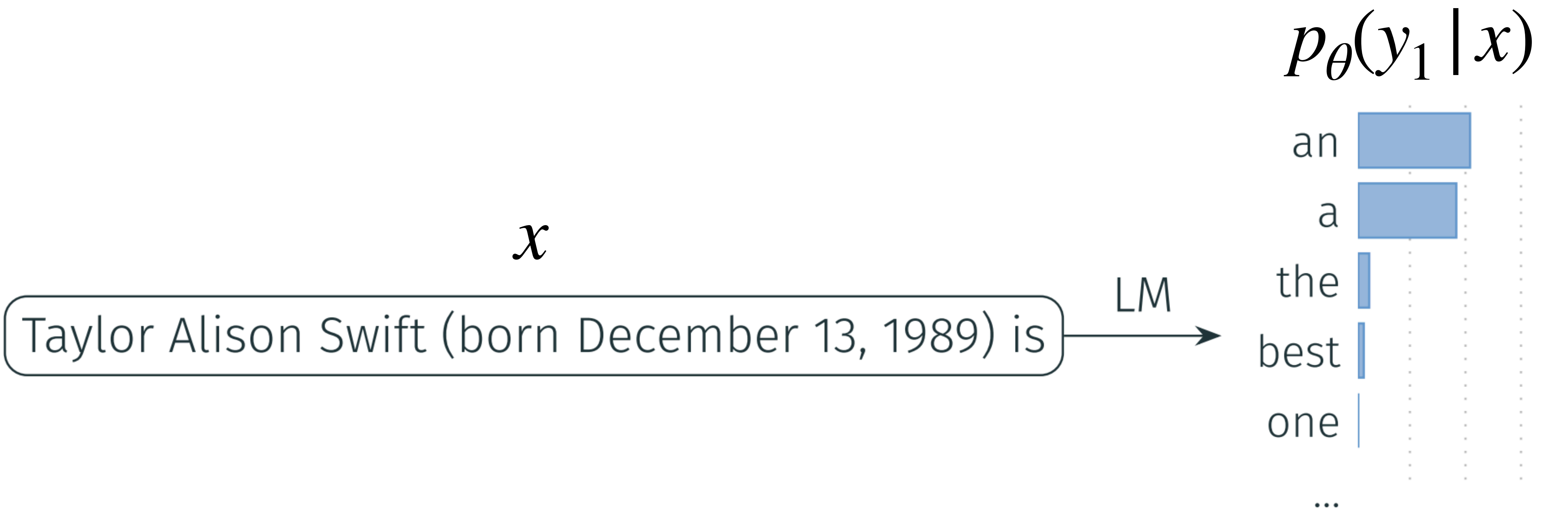
- With an autoregressive language model, we have:

$$p_\theta(y_{1:T}|x) = \prod_{t=1}^{T} p_\theta(y_t|y_{<t}, x)$$

- Note: we'll use y to refer to a full sequence $y_{1:T}$.

Basic setup

- Each term $p_\theta(y_t \mid y_{<t}, x)$ gives us a probability distribution over next-tokens



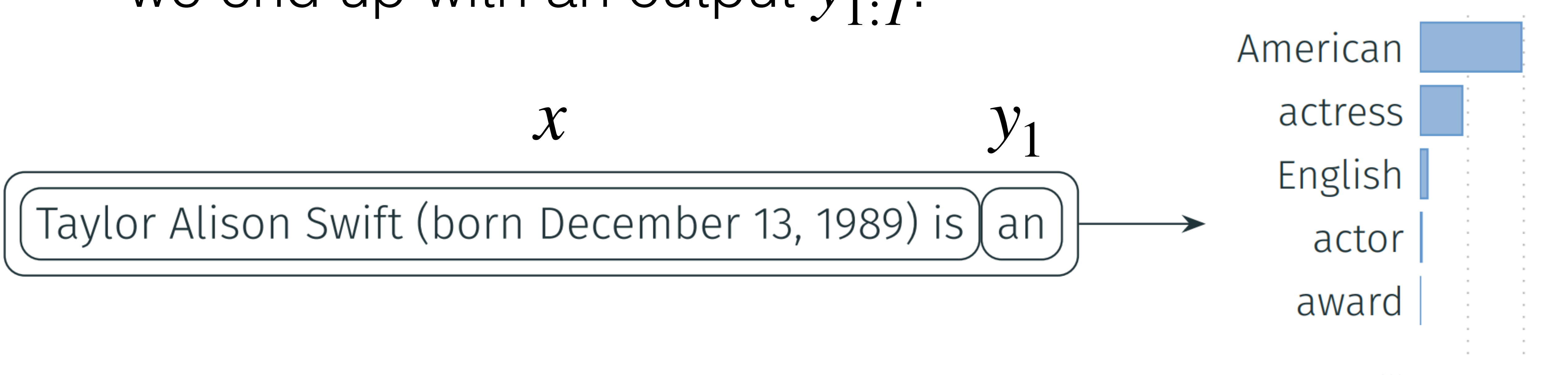
Basic setup

- Each term $p_\theta(y_t \mid y_{<t}, x)$ gives us a probability distribution over next-tokens

- We can choose a next token, add it to the context, and get a new distribution over next-tokens

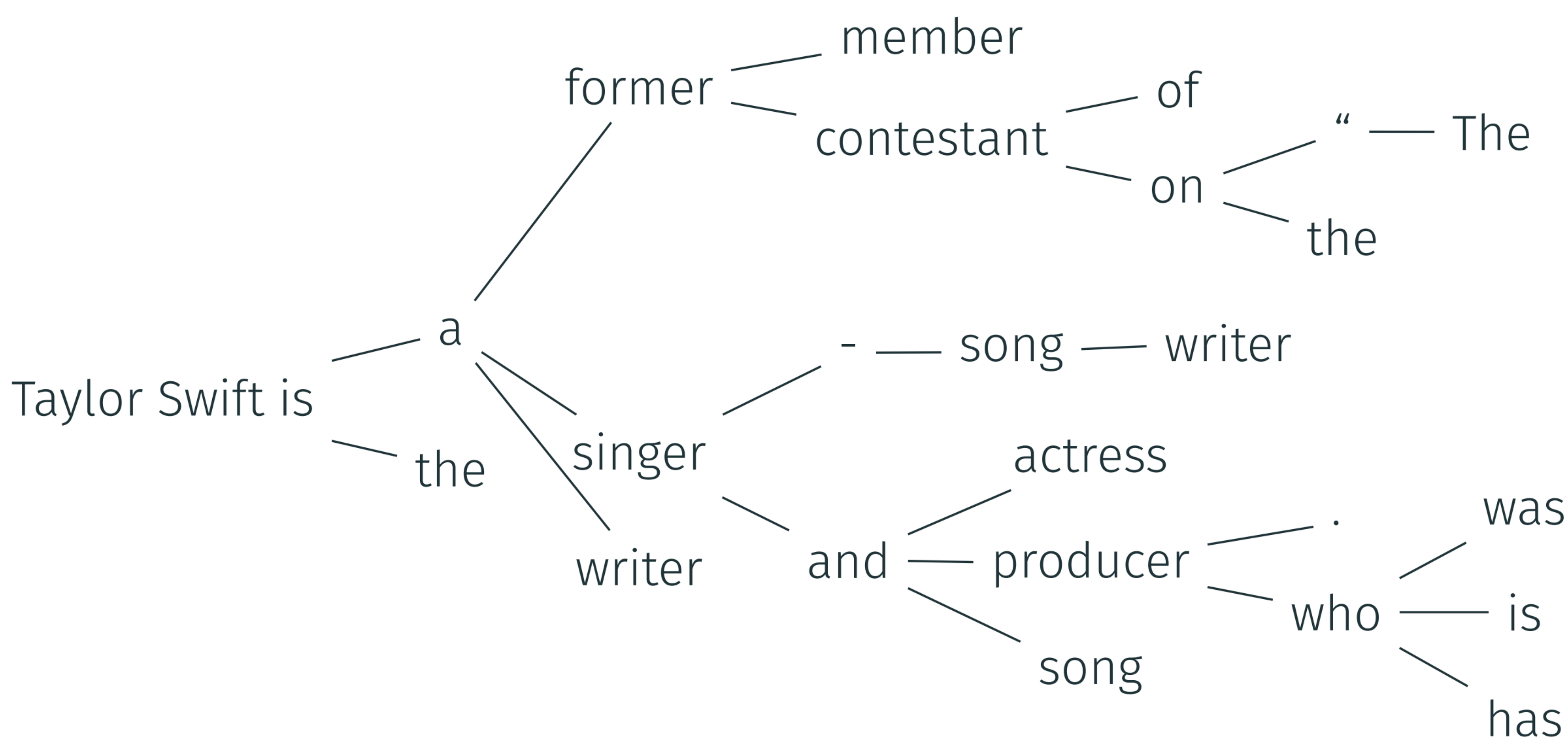
- **Decoding**: choose next tokens so that we end up with an output $y_{1:T}$.

$$p_\theta(y_2 \mid x, y_1)$$



Decoding

- Each time-step of decoding requires a choice



- What is the *objective*? How do we make *local choices* that achieve the objective?

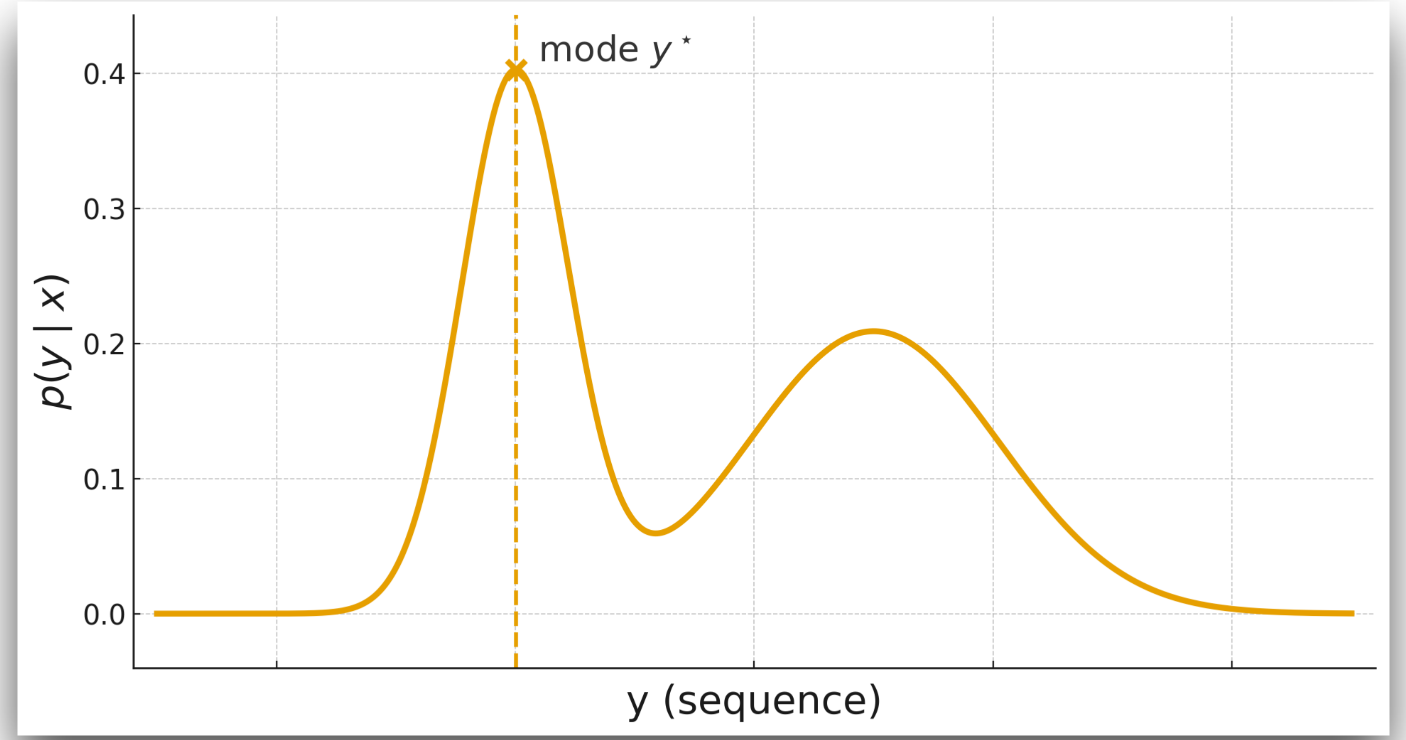
Today’s lecture

- Basic setup
- Decoding objectives and algorithms
    - Optimization
    - Sampling

Decoding as optimization

- **Goal**: find a single most likely output

$$\hat{y} = \text{argmax}_{y \in \mathcal{Y}} p_\theta(y \mid x)$$



Decoding as optimization

- **Goal**: find a single most likely output

$$\hat{y} = \text{argmax}_{y \in \mathcal{Y}} p_\theta(y \mid x)$$

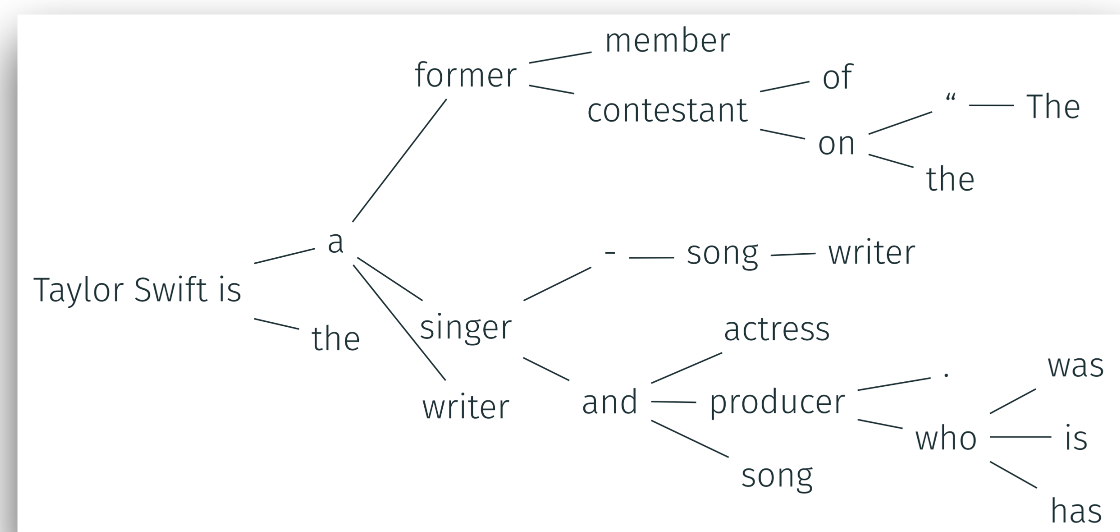
- Referred to as:
    - *Mode-seeking*: finds a mode of the distribution
    - *Maximum a-posteriori (MAP)*: given a *prior* θ and *evidence* x, find a mode of the *posterior* $p_\theta(y \mid x)$

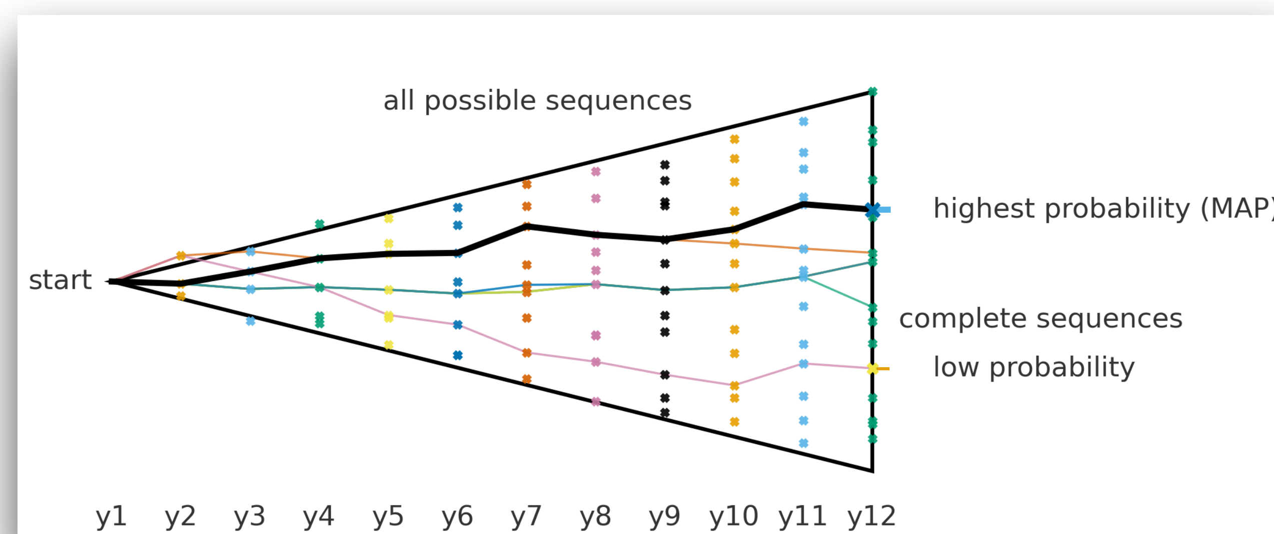
Decoding as optimization

- **Goal**: find a single most likely output

$$\hat{y} = \text{argmax}_{y \in \mathcal{Y}} p_\theta(y \mid x)$$

- Key challenge: output space $\mathcal{Y}$ is very large



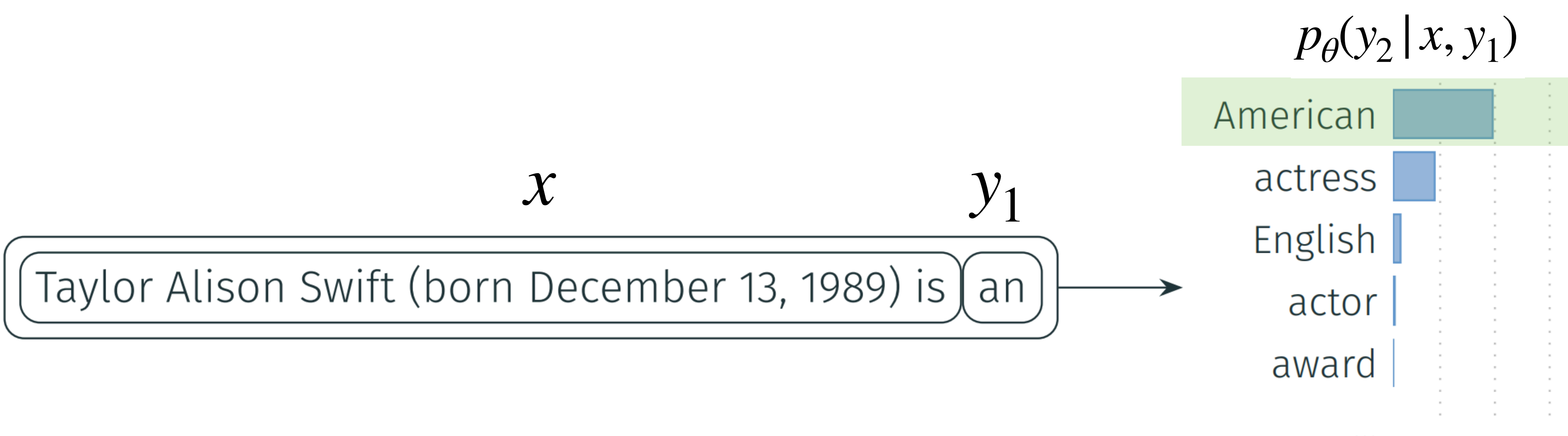


Approach 1: greedy decoding

- Choose the most likely token at each step:

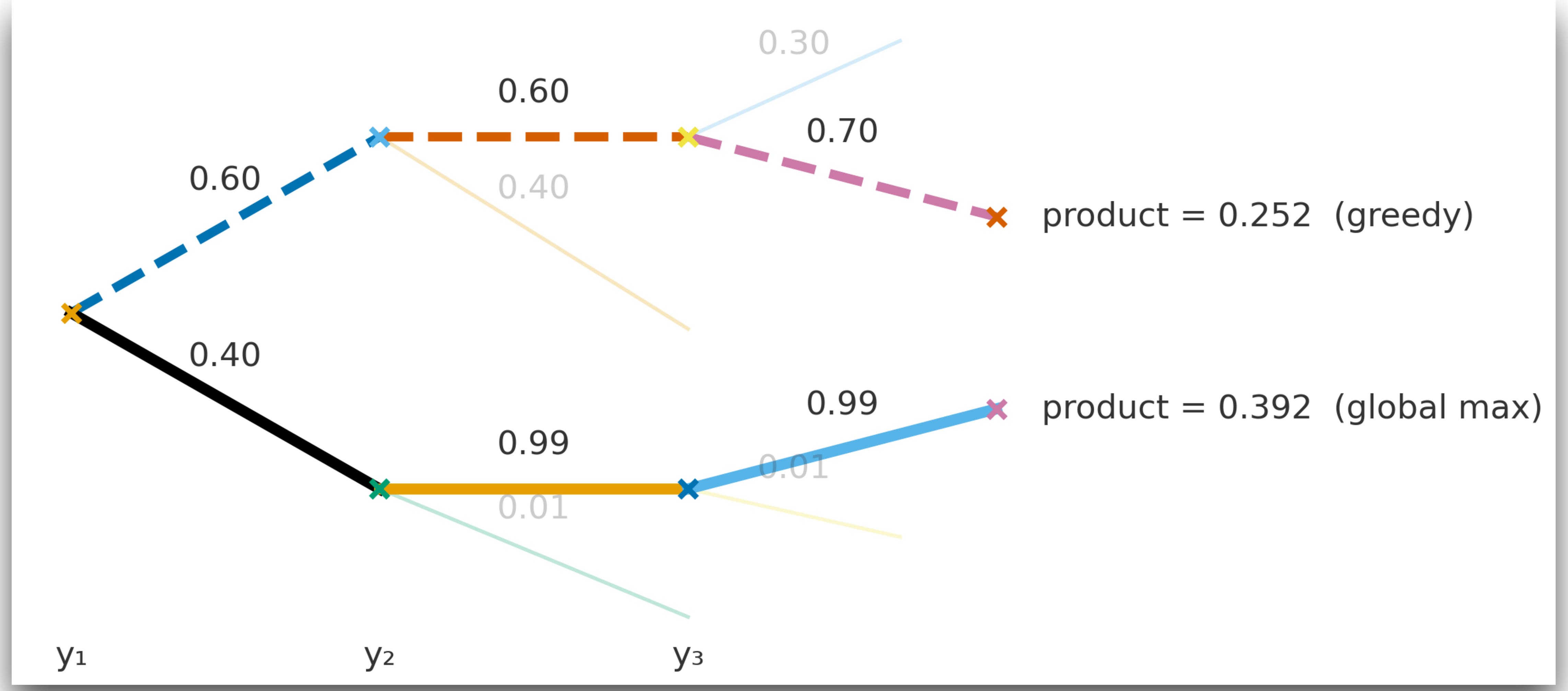
For t = 1…End:

$$\hat{y}_t = \text{argmax}_{y_t \in V} p_\theta(y_t \mid \hat{y}_{<t}, x)$$



Approach 1: greedy decoding

- Does not guarantee the most-likely sequence:



Approach 1: greedy decoding

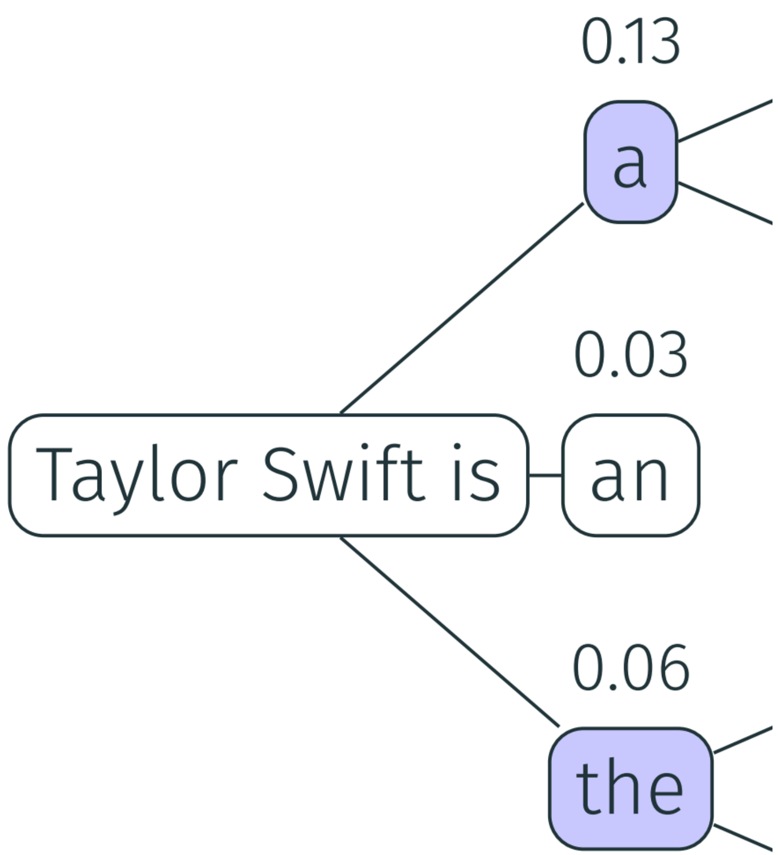
- Does not guarantee the most-likely sequence:

	Prefix	Continuation			Prob.
Greedy	Taylor Swift is a	former	contestant	on	
Token prob.		0.023	0.022	0.80	0.0004
Non-greedy	Taylor Swift is a	singer	,	song	
Token prob.		0.012	0.26	0.21	**0.0007**

Approach 2: beam search

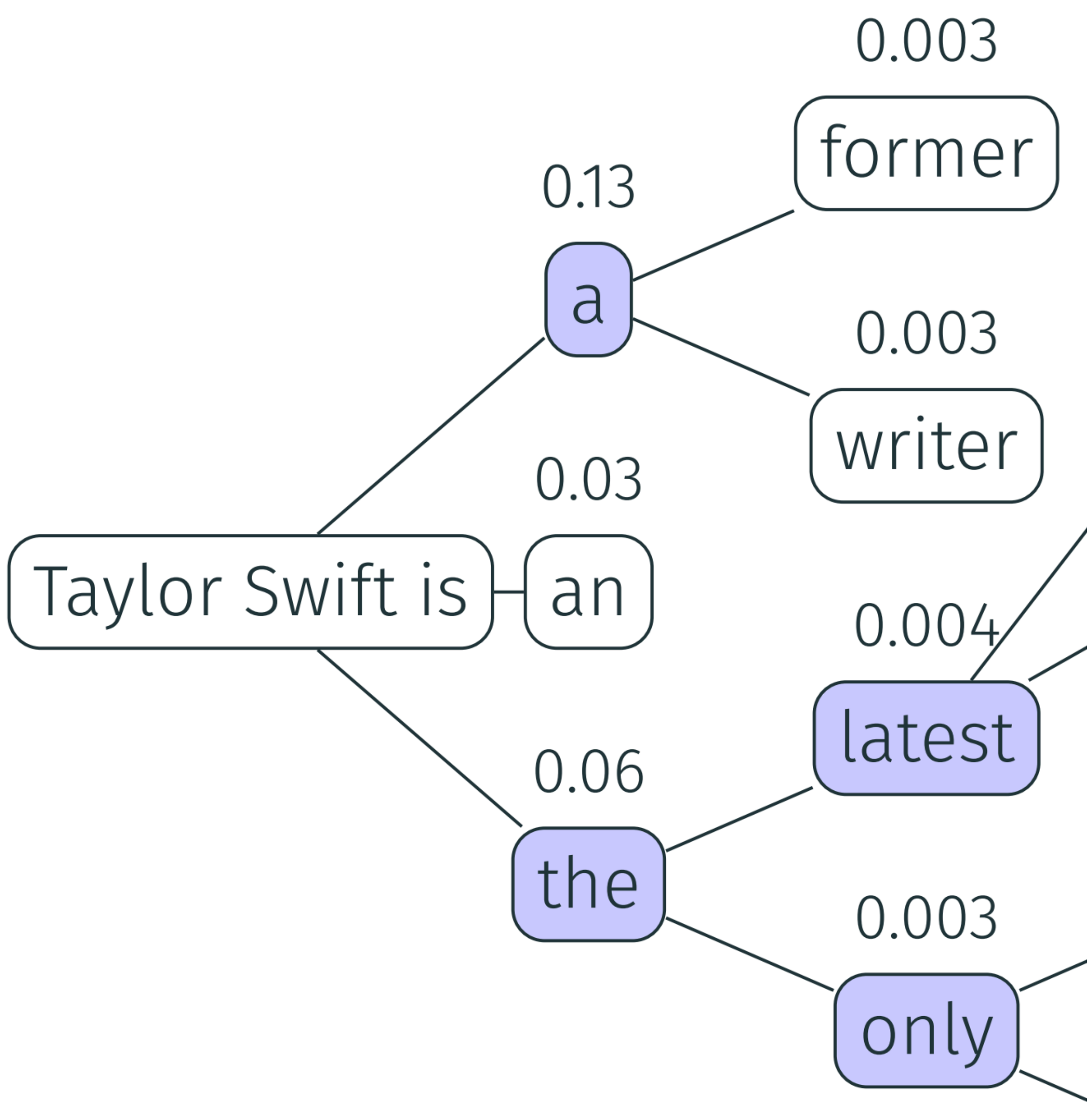
- Beam search is a width-limited breadth-first search
  - Key idea: maintain several likely paths

Approach 2: beam search



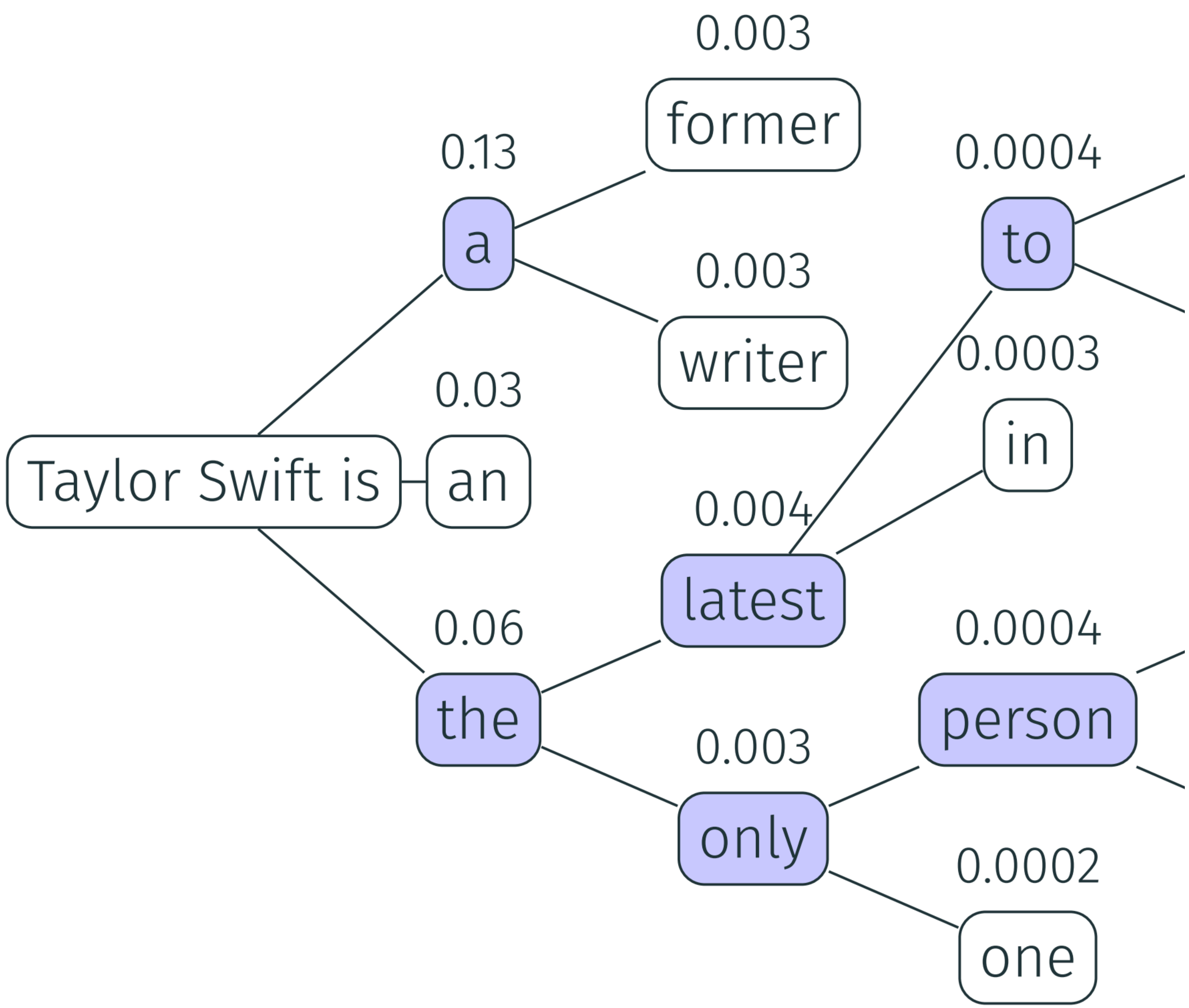
GPT2, beam size 2

Approach 2: beam search



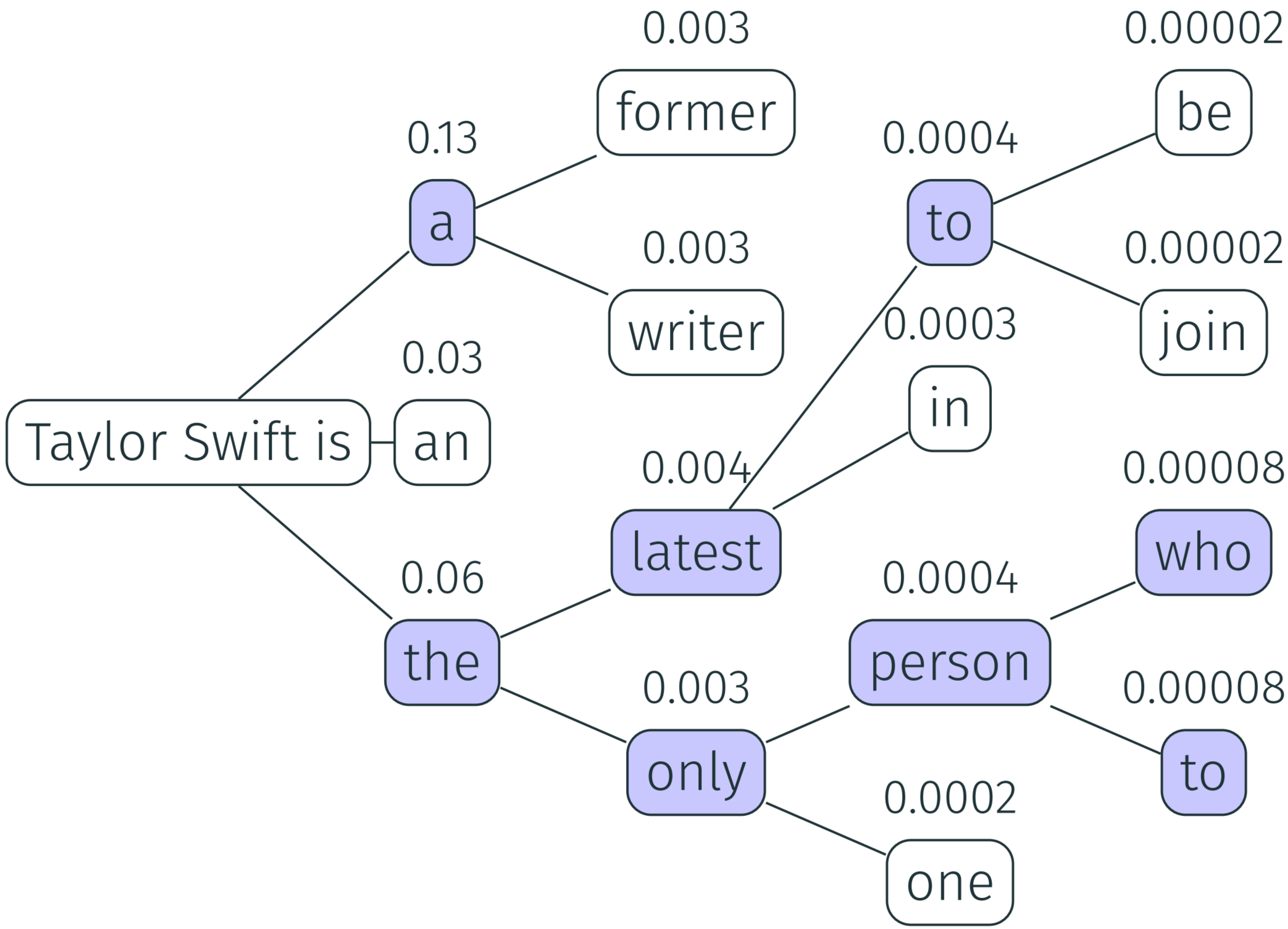
GPT2, beam size 2

Approach 2: beam search



GPT2, beam size 2

Approach 2: beam search



GPT2, beam size 2

Approach 2: beam search

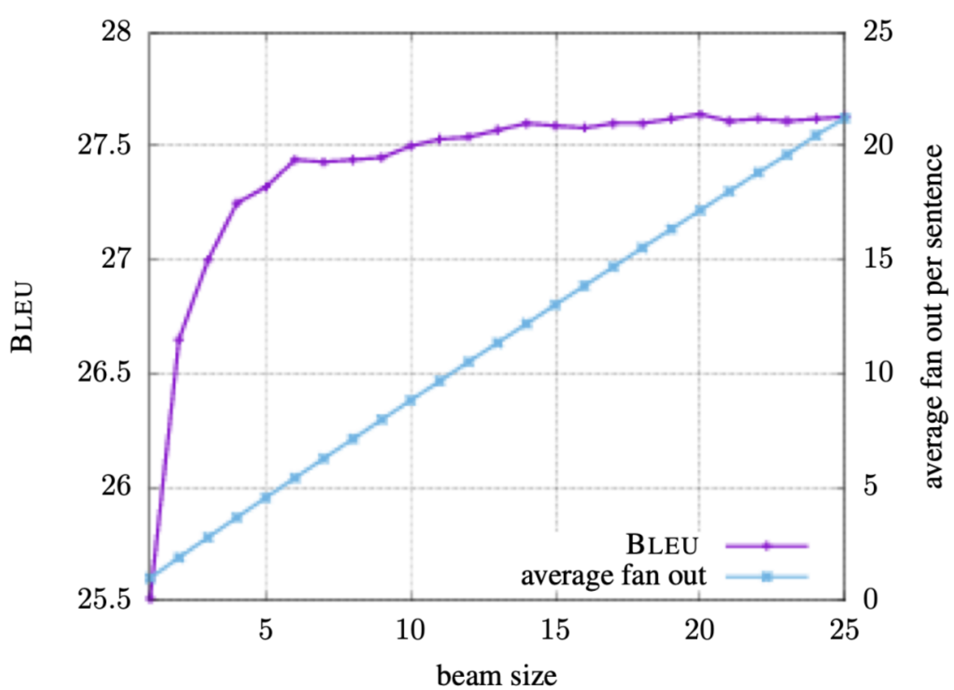
- Beam search is a width-limited breadth-first search
  - B = 1: greedy decoding
  - B = $|V|^{T_{max}}$: exact MAP
    - Example: $50000^{128} = \text{very big}$
- In practice, we use B = smaller number, e.g. 16, treated as a hyper-parameter

Huggingface interface

- Greedy decoding
  - `model.generate(do_sample=False, num_beams = 1)`
- Beam search
  - ```
    b=16
    model.generate(do_sample=False, num_beams = b)
    ```
    ```

MAP decoding

- Traditionally widely used in closed-ended tasks like translation or summarization



[Freitag and Al-Onaizan, 2017]

Model	Dataset	Metric	Greedy	BS
Llama2-7B	HumanEval	Pass@1	12.80	15.24
	MBPP		17.80	19.40
	GSM8K	Acc	13.87	17.21
	XSUM	R-L	27.21	21.88
	CNN/DM		23.43	20.69
	De⇒En	B-4	28.80	30.14
	En⇒De		22.63	23.99
	Zh⇒En		19.44	20.11
	En⇒Zh		15.15	14.50
	CQA	Acc	62.90	64.37
	SQA		60.76	62.25

[Shi et al., 2024]

Pitfalls of MAP decoding

- 1. Degeneracy: repetition traps, short sequences
- 2. Is the highest probability the “best”?

Degeneracy: repetition traps

- MAP decoding (greedy search) with SmolLM2-135M:

```
Greedy:
The weather today is very cold and windy.

The weather is very cold and windy.

The weather is very cold and windy.

The weather is very cold and windy.

The weather is very cold and windy.

The weather is
```

- Models tend to assign high probability to repetitive loops
  - Mitigations: repetition penalty, modify the loss function

Degeneracy: short sequences

- [Stahlberg and Byrne, 2019]: the highest-probability sequence might be the *empty sequence!*

Pr[Taylor Swift is <eos>] > **Pr**[Taylor Swift is an American singer-...]

- Remedy: length normalization

Degeneracy: atypicality

- Biased coin $\Pr[\text{H}] = 0.6, \Pr[\text{T}] = 0.4$
- What is the most likely outcome of 100 flips?
  - All heads! H H H H H H H H H H …
  - This outcome is *atypical*
  - Similarly, the *most likely generation* may also be atypical
- Remedy: sampling

[Meister et al, 2022]

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.

Is the highest-probability output best?

- When there are multiple ways to say the same thing, probability is **spread** across the multiple ways

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

Total 0.6

Pitfalls of MAP decoding

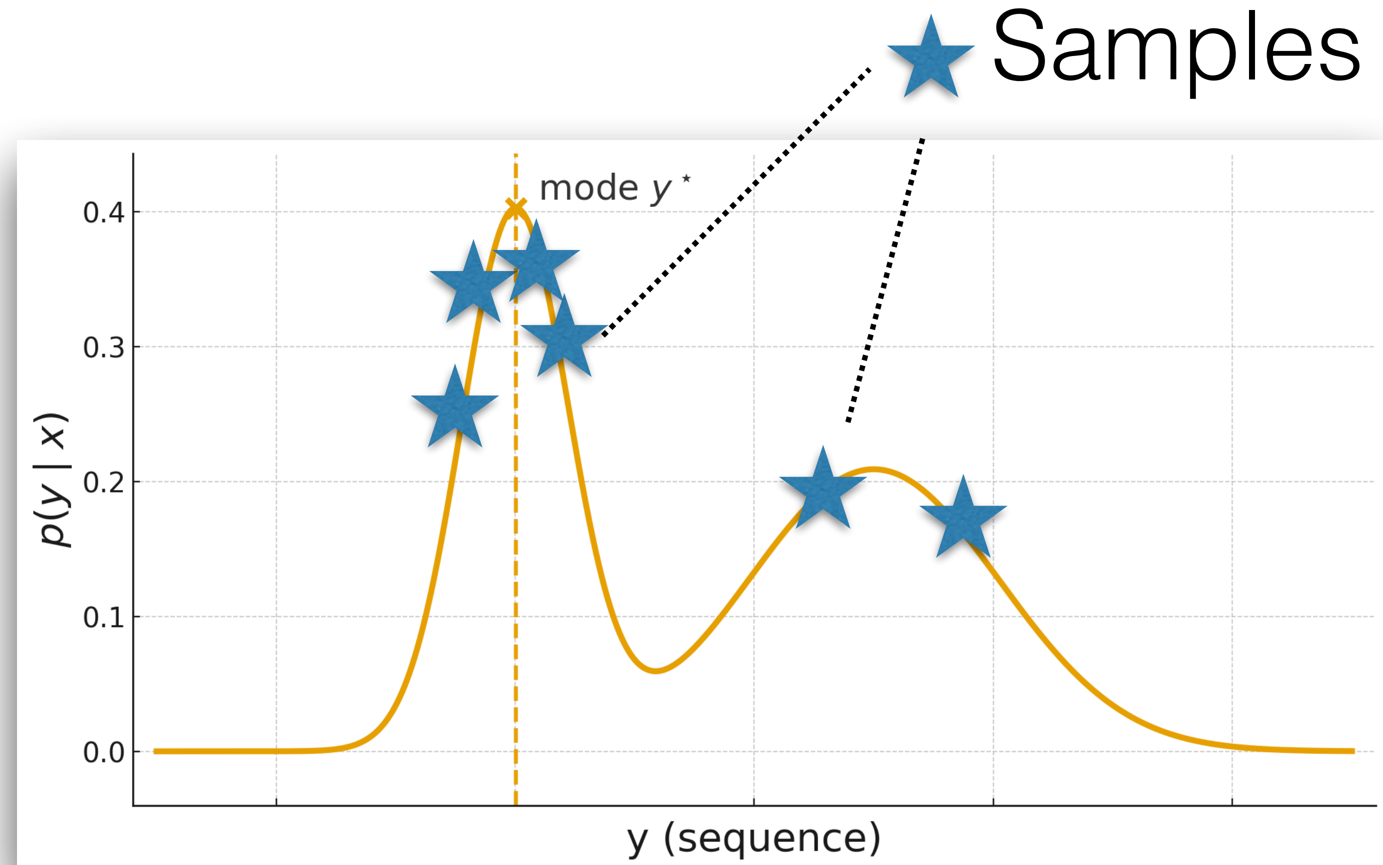
- As a result, we often want outputs that are “likely” but not “maximally likely”

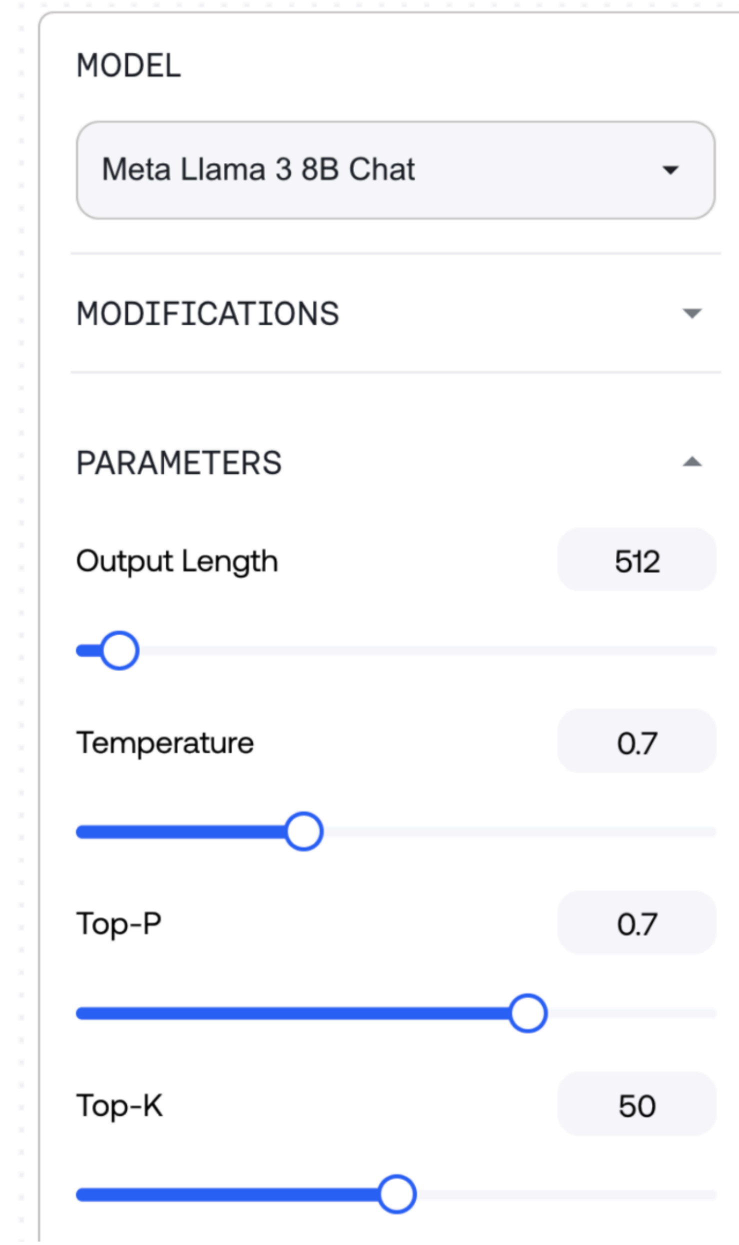
Today's lecture

- Basic setup
- Objectives
    - Optimization
    - *Sampling*

Sampling

- Modern LLM APIs offer settings for *sampling*





Together.ai playground.

Basic sampling ("ancestral sampling")

- Simply sample from the model's next-token distribution at each step

For t = 1…End:

$$\hat{y}_t \sim p_\theta(y_t \,|\, \hat{y}_{<t}, x)$$

$$p_\theta(y_2 \,|\, x, y_1)$$

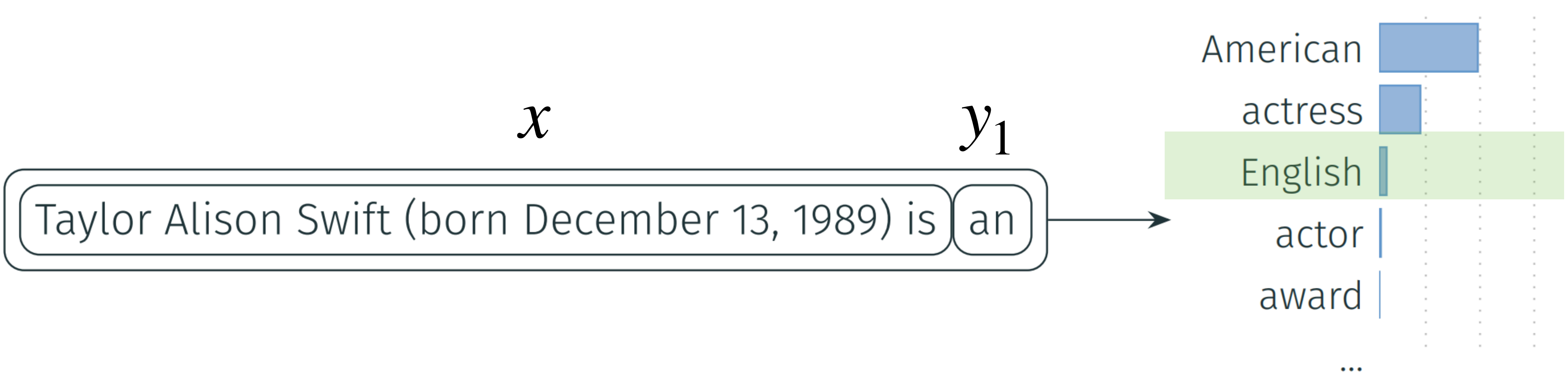
Basic sampling ("ancestral sampling")

- Simply sample from the model's next-token distribution at each step

For t = 1…End:

$$\hat{y}_t \sim p_\theta(y_t \mid \hat{y}_{<t}, x)$$

$$p_\theta(y_2 \mid x, y_1)$$

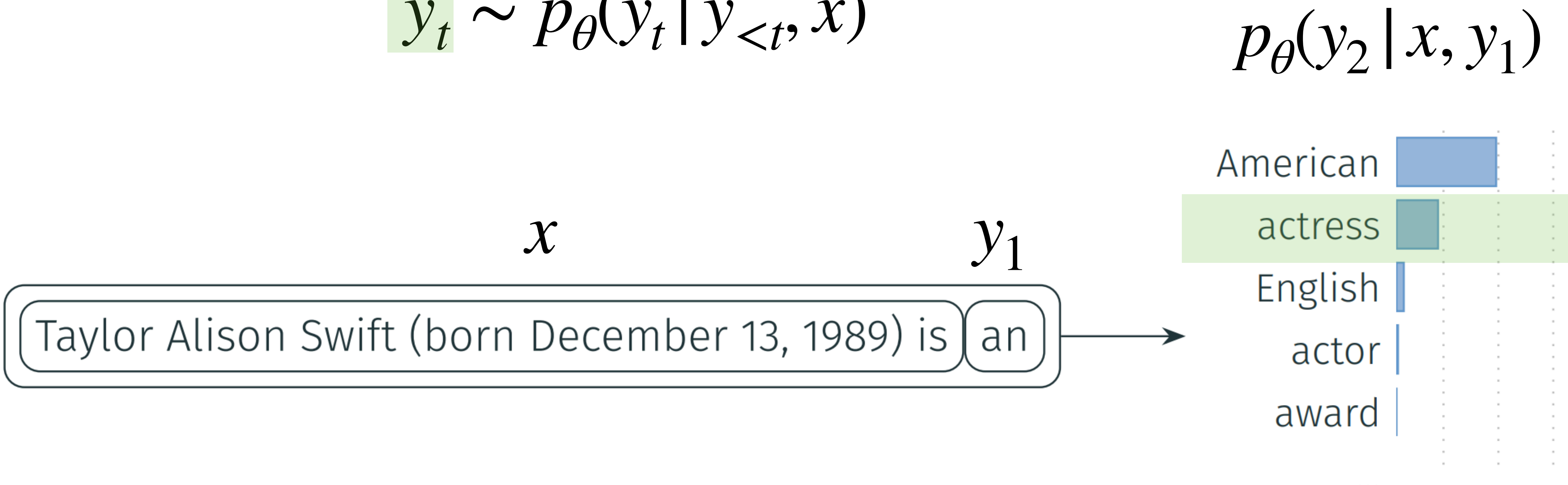


Basic sampling ("ancestral sampling")

- Simply sample from the model's next-token distribution at each step

For t = 1…End:

$$\hat{y}_t \sim p_\theta(y_t \mid \hat{y}_{<t}, x)$$



Basic sampling ("ancestral sampling")

- Simply sample from the model's next-token distribution at each step

For t = 1…End:

$$\hat{y}_t \sim p_\theta(y_t \mid \hat{y}_{<t}, x)$$

- Equivalent to sequence sampling, $y_{1:T} \sim p_\theta(y_{1:T} \mid x)$

Aside: categorical sampling

- Each next-token distribution is a categorical distribution over V (vocab size) items
  - Easy/fast to sample from
  - Categorical sampling is implemented in common libraries such as PyTorch

```
vocab = ['a', 'b', 'c', 'd', 'e']
probs = np.array(
        [0.1, 0.2, 0.1, 0.4, 0.2]
)
```

```
import torch

# Sample 100 times using PyTorch
torch_probs = torch.tensor(probs)
categorical = torch.distributions.Categorical(probs=torch_probs)
categorical.sample((100,))
```

✓ 0.0s

```
tensor([3, 4, 1, 1, 3, 2, 0, 0, 1, 0, 1, 4, 3, 3, 3, 4, 3, 3, 1, 4, 1, 3, 4, 3,
        3, 0, 2, 4, 4, 4, 3, 1, 1, 3, 4, 0, 1, 2, 3, 4, 4, 4, 1, 2, 1, 3, 3, 0,
        2, 4, 1, 0, 3, 3, 3, 0, 3, 2, 3, 3, 0, 0, 3, 3, 1, 4, 0, 4, 4, 3, 0, 1,
        1, 4, 3, 3, 4, 1, 0, 1, 4, 3, 1, 0, 4, 2, 3, 1, 4, 1, 3, 4, 0, 3, 3, 3,
        1, 2, 3, 3])
```

What is wrong with ancestral sampling?

- Often leads to *incoherence*

```
Greedy:
The weather today is very cold and windy.

The weather is very cold and windy.

The weather is very cold and windy.

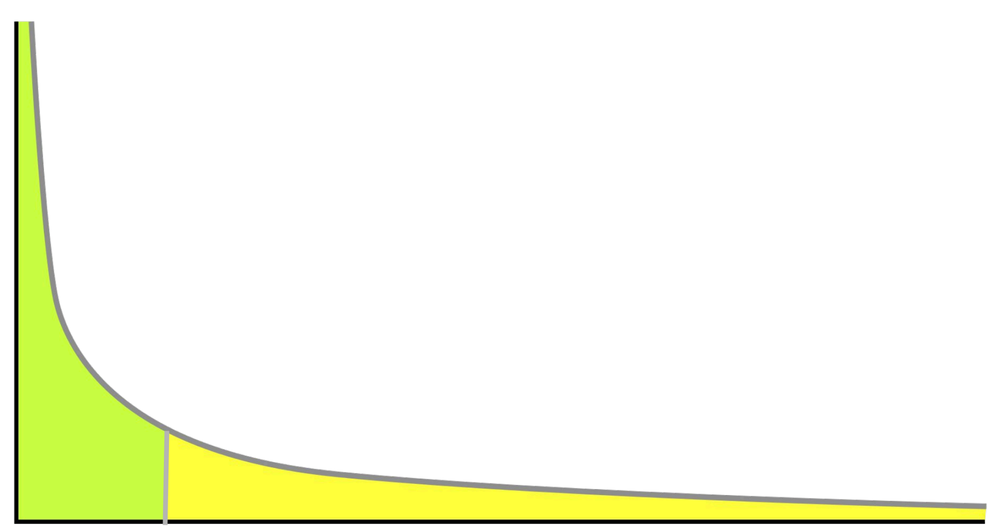
The weather is
```

```
Temperature=1.0:
The weather today is very cold outside as it got cold the night before.
14.  The teacher is going to give a card tomorrow.
```

What is wrong with ancestral sampling?

- Often leads to *incoherence*

- **Heavy tail:** there are many choices for the next-token (e.g., 50,000). Even if each ‘bad’ token has a small probability, the sum of bad tokens has a nontrivial probability



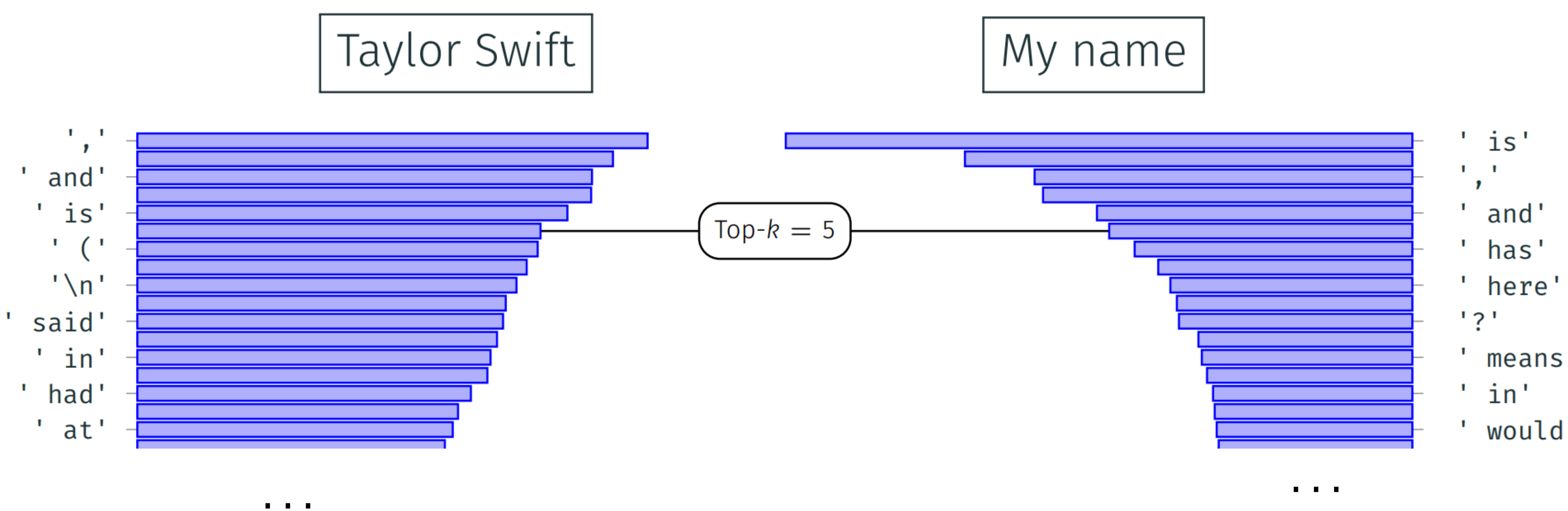
What is wrong with ancestral sampling?

- **Compounding error**: Suppose the total probability of sampling a bad token is ϵ.
  - Then for a length-T sequence, the probability of sampling no bad tokens is $(1-\epsilon)^T$
    - $\epsilon = 0.01,\ T = 128$: p(no bad tokens): 0.276
    - $\epsilon = 0.05,\ T = 128$: p(no bad tokens): 0.0014
    - $\epsilon = 0.01,\ T = 1024$: p(no bad tokens): 0.000033

Workaround: *truncate* the tail

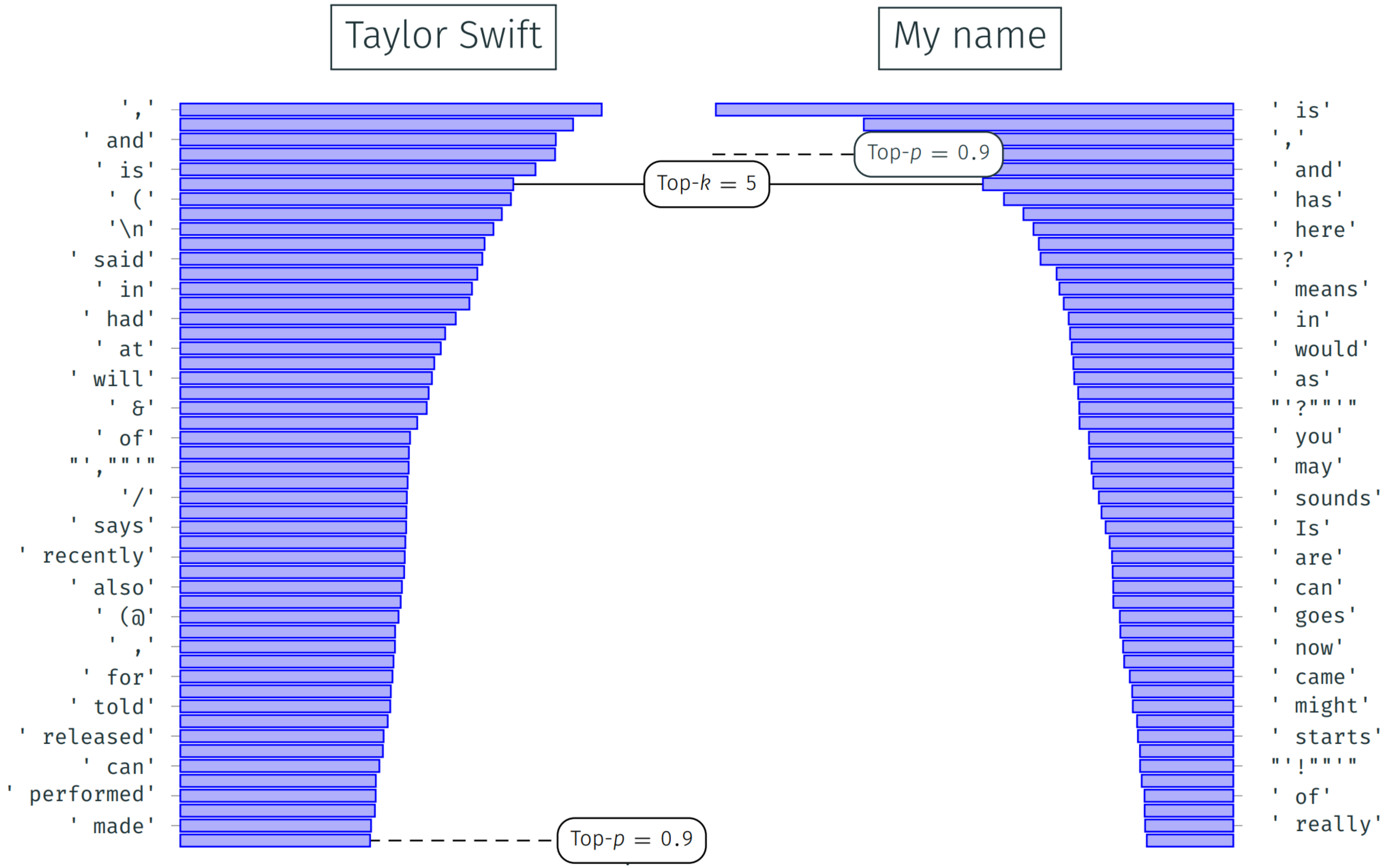
- Top-*k* sampling: sample only from the *k* most-probable tokens at each step

$$\hat{y}_t \sim \begin{cases} p_\theta(y_t \mid y_{<t}, x)/Z_t & \text{if yt in top k} \\ 0 & \text{otherwise} \end{cases}$$



Workaround: *truncate* the tail

- Top-*p* sampling: sample only from the top *p* probability mass



Workaround: *truncate* the tail

```
Temperature=1.0:
The weather today is very cold outside as it got cold the night before.
14.  The teacher is going to give a card tomorrow.
```

```
Top-k=20:
The weather today is very cold with low temperature of 30 C, but there is still some rain
```

```
Top-p=0.9:
The weather today is clear and I know it is going to rain soon. I’m not in a hurry so I’m heading
```

Huggingface interface

- Ancestral sampling
  - `model.generate(do_sample=True)`
- Top-k sampling
  - ```
    k=20
    model.generate(do_sample=True, top_k=k)
    ```
- Top-p sampling
  - ```
    p=0.9
    model.generate(do_sample=True, top_p=p)
    ```

Workaround: *truncate* the tail

- Several strategies have been developed, e.g.:

Method	Threshold strategy
Top-k	Sample from k-most-probable
Top-p	Cumulative probability at most p
ϵ	Probability at least ϵ
η	Min prob. proportional to entropy
Min-p	Prob. at least p_{min} scaled by max token prob.

Temperature sampling

- Instead of truncation, make distribution more "peaked"

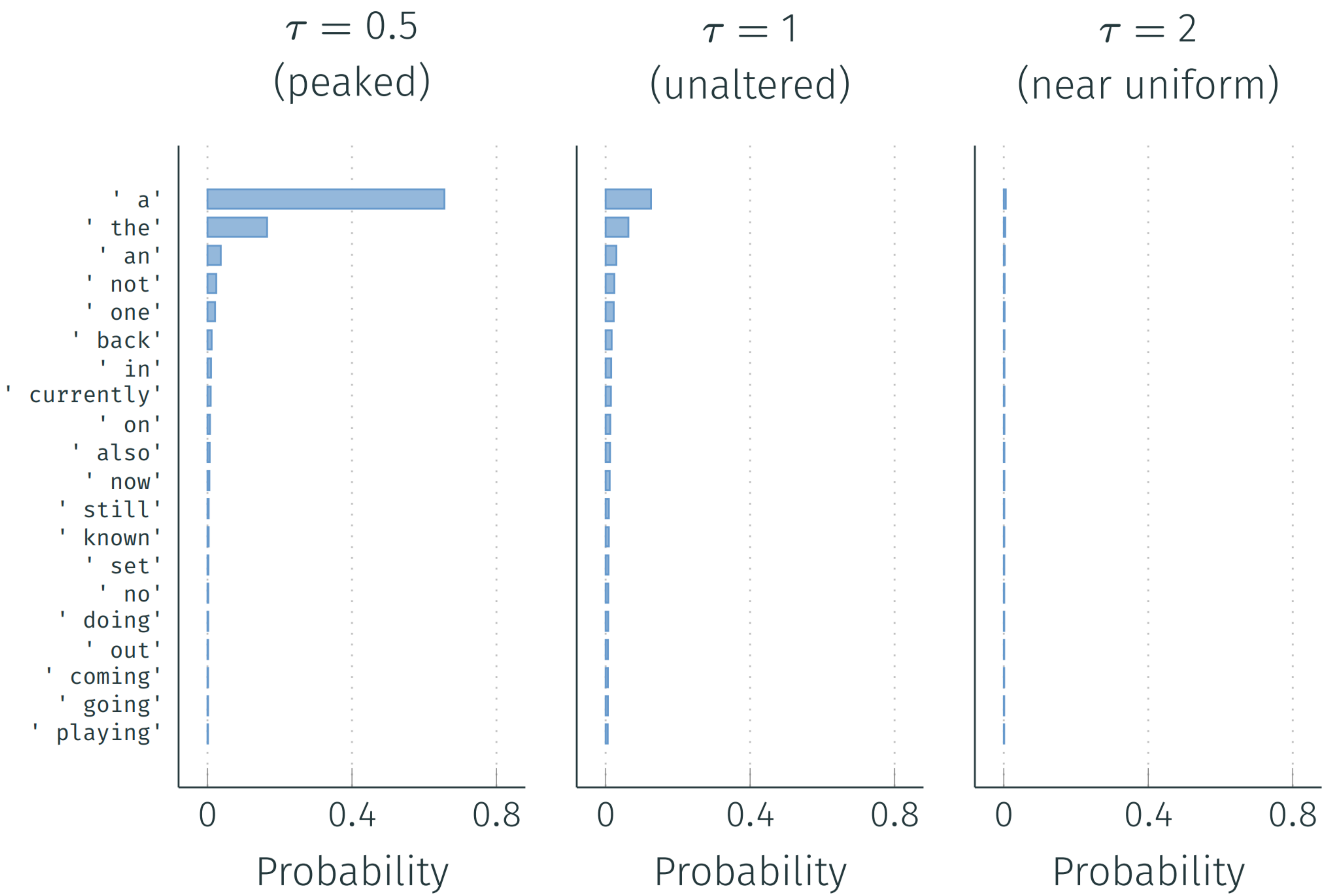
$$\text{softmax}(x, \tau) = \frac{\exp(x/\tau)}{\sum_i (x_i/\tau)}$$

Temperature	Parameter	Pro	Con
High	$\tau \geq 1$	Diverse	Incoherent
Low	$\tau < 1$	Coherent	Repetitive

Temperature sampling

Taylor Swift is...

$\mathrm{softmax}(\boldsymbol{x}/\tau)$



Temperature sampling

```
Temperature 0.5:
The weather today is very cold. The wind is blowing from the north.

The weather is not very cold, but there is a lot of ice on the ground

The driver has to stop and take the car into the
```

```
Temperature 1.0:
The weather today is very nice, some water and snow. It's only 2ft. high at the real level

It
```

```
Temperature 1.5:
The weather today is: Low in the Treasure Nevada at Mosquittle Examinerare] Emergence Outreach
```

Today's lecture

- Decoding as optimization
- Sampling
- *Speeding up decoding*

Speeding up decoding

- We will have a more comprehensive discussion in a later lecture (*Advanced Inference*)
- Today: *key-value caching*

Key value cache

- During decoding, each new token at time *t* attends to positions $\leq t$
- The attention for step *t* needs the keys and values for *all past tokens* $1 : t$
  - If we recomputed those keys and values for every step, we would redo $O(T^2)$computations:
    - k_1, v_1
    - k_1, v_1, k_2, v_2
    - $k_1, v_1, k_2, v_2, k_3, v_3$
    - …
- **KV caching:** store the previously computed keys/values
  - Due to masking future tokens, caching is equivalent to recomputing!

Key value cache

- Consider 1 transformer layer with 1 attention head. At step t of decoding:
  - $q_t = h_t W_q \in \mathbb{R}^{1 \times d_k}$
  - $k_t = h_t W_K \in \mathbb{R}^{1 \times d_k}$
  - $v_t = h_t W_V \in \mathbb{R}^{1 \times d_v}$
- We have the previous keys and values cached:
  - $K_{1:t-1} \in \mathbb{R}^{(t-1) \times d_k}$
  - $V_{1:t-1} \in \mathbb{R}^{(t-1) \times d_v}$
- We append k_t to $K_{1:t-1}$ and v_t to $V_{1:t-1}$ and compute attention:
  - $z_t = \text{softmax}\left(\frac{q_t K_{1:t}^T}{\sqrt{d_k}}\right) V_{1:t}$

Without caching, we recompute:
$K_{1:t} = [h_1; h_2; \ldots; h_t] W_k$
$V_{1:t} = [h_1; h_2; \ldots; h_t] W_v$

Key value cache

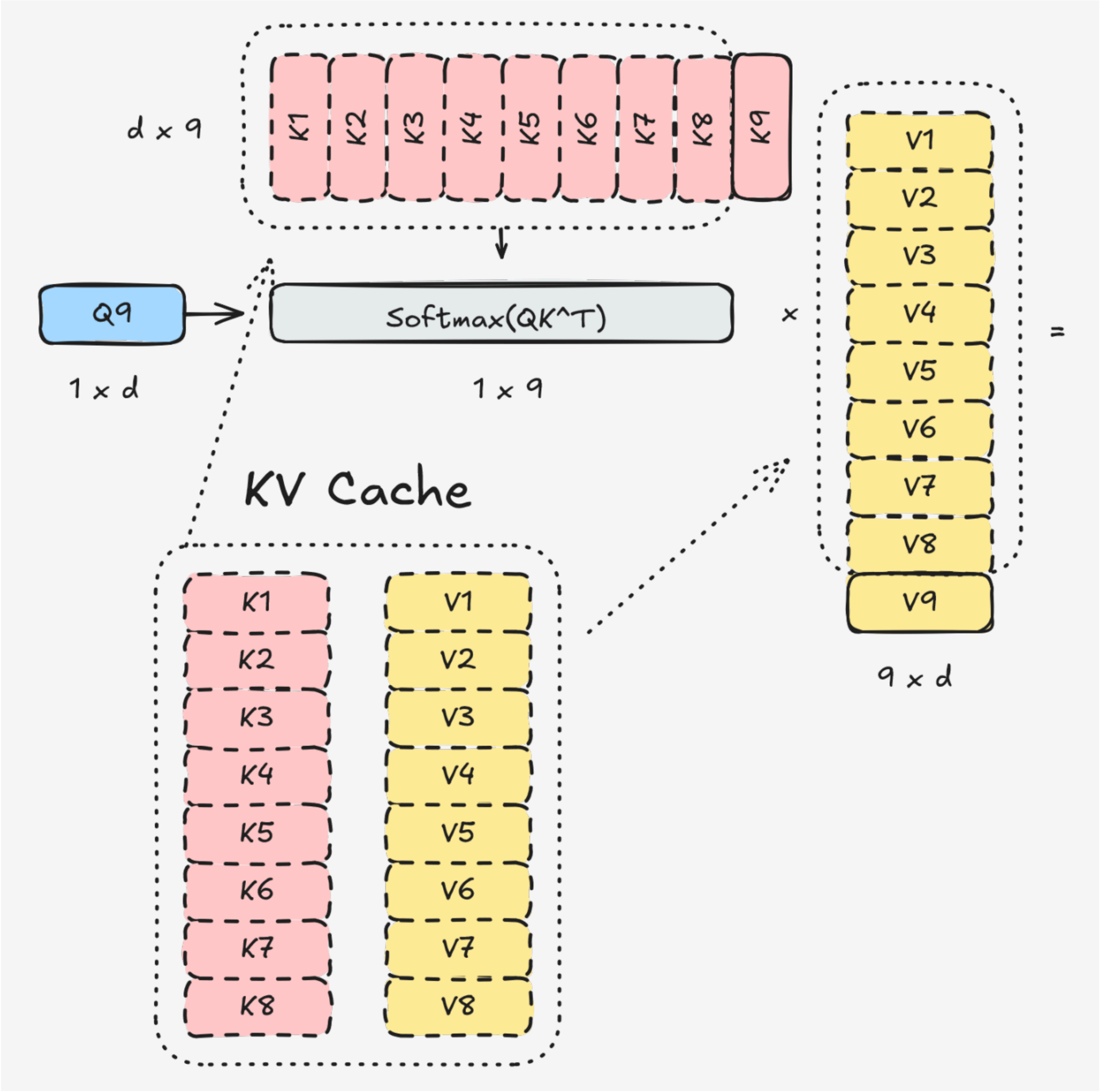


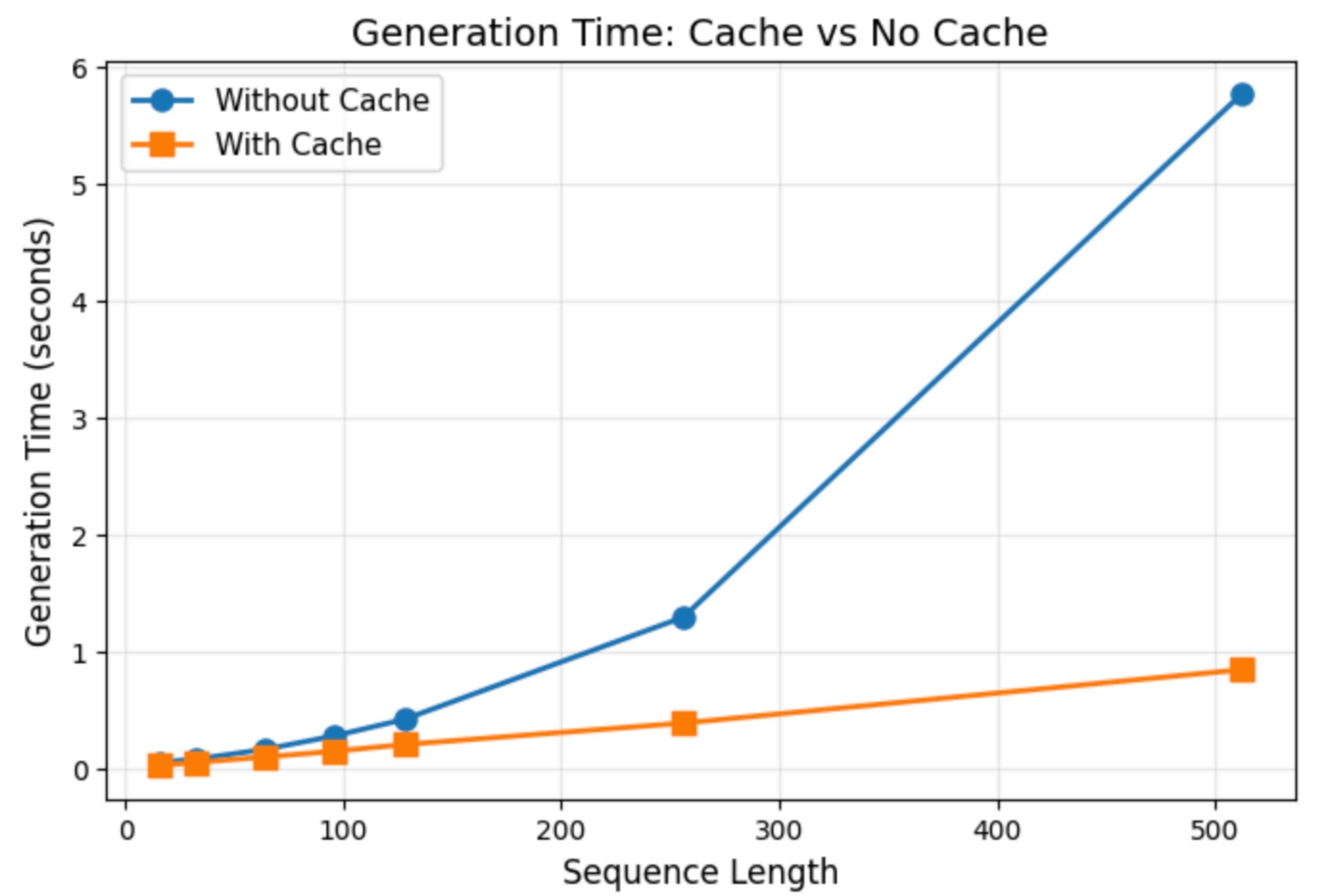
Diagram by Hailey Schoelkopf

Code example

```python
if use_cache and self.cache_k is not None:
    # Only compute K, V for the new token(s)
    K_new = self.k_proj(x_norm)
    V_new = self.v_proj(x_norm)

    # Append to cache
    K = torch.cat([self.cache_k, K_new], dim=1)
    V = torch.cat([self.cache_v, V_new], dim=1)

    # Update cache
    self.cache_k = K
    self.cache_v = V
```



Recap

- Decoding as optimization
- Sampling
- Speeding up decoding

Thank you