Decoding from Neural Text Generation Models

Antoine Bosselut

Generation Model Basics

1. At each time step, model computes a score o_n for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

$f(\,.\,)$ is your model

Generation Model Basics

1. At each time step, model computes a score o_n for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

$f(.)$ is your model

2. Compute a probability distribution over these scores (usually softmax)

$P(.)$ is your distribution over tokens

$$P(y_t = w_n \mid \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

Generation Model Basics

1. At each time step, model computes a score o_n for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

$f(\,.\,)$ is your model

2. Compute a probability distribution over these scores (usually softmax)

$P(\,.\,)$ is your distribution over tokens

$$P(y_t = w_n | \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

3. Define a function to select a token from this distribution

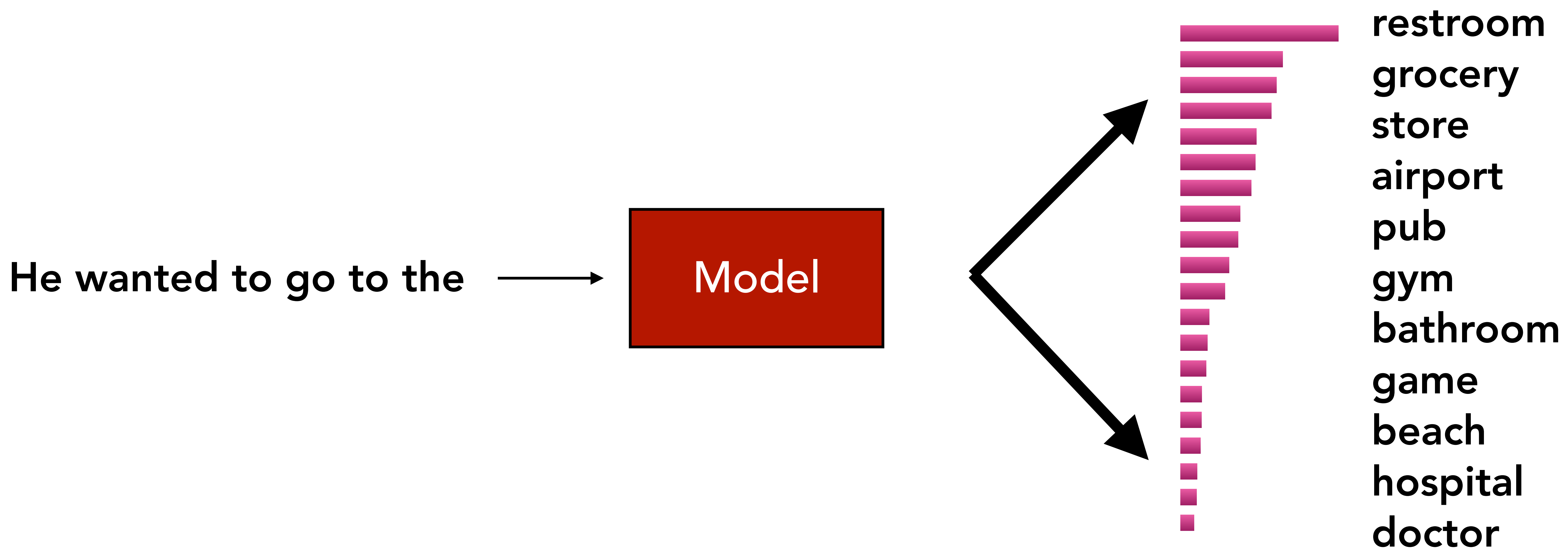
$$\hat{y}_t = g(P(y_t | \{y\}_{<t}))$$

$g(\,.\,)$ is your decoding algorithm

Simplest approach: Argmax Decoding

- g = select the token with the highest probability:

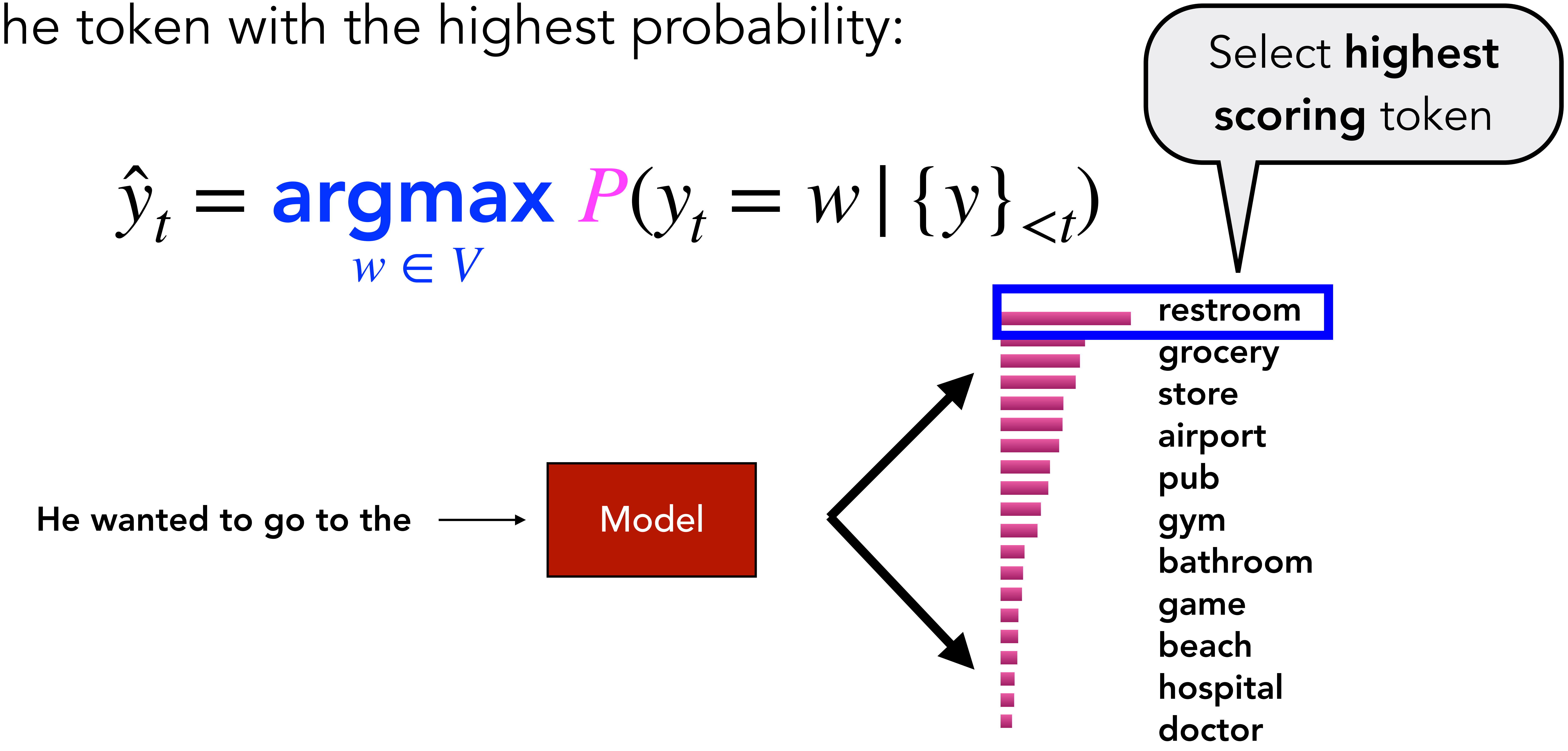
$$\hat{y}_t = \underset{w \in V}{\textbf{argmax}}\, P(y_t = w \mid \{y\}_{<t})$$



Simplest approach: Argmax Decoding

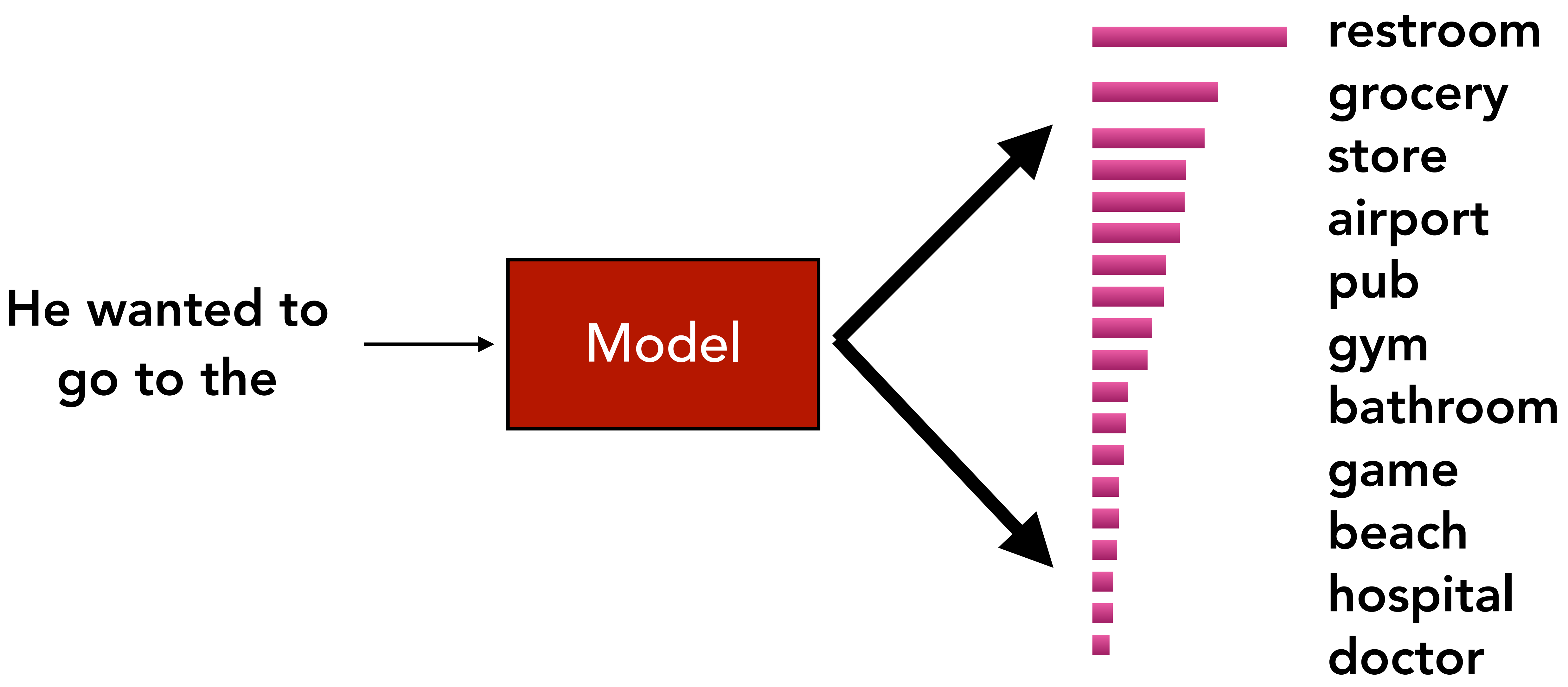
- g = select the token with the highest probability:

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



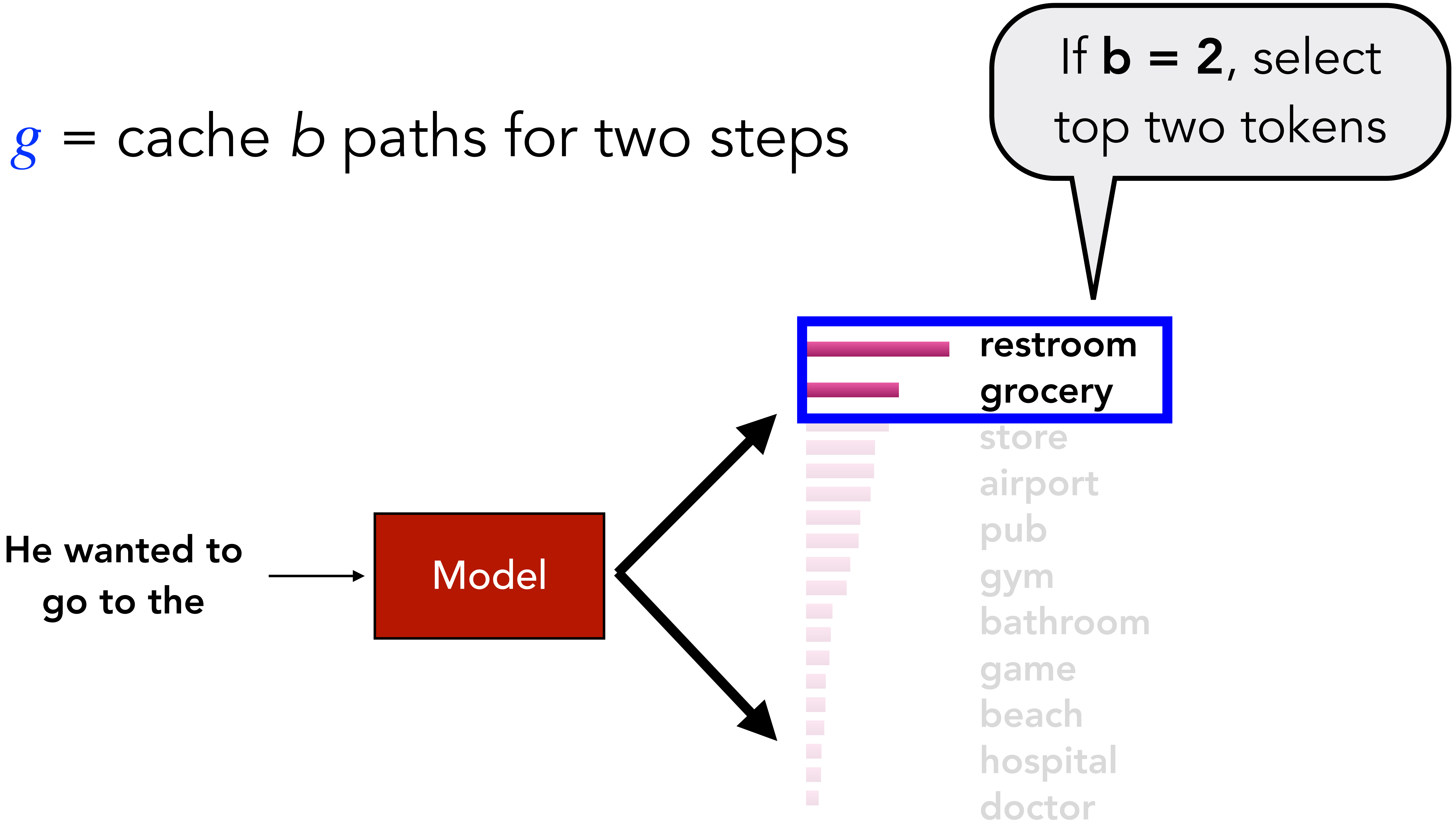
Maybe we need more options: Beam Search

- g = cache b paths for two steps



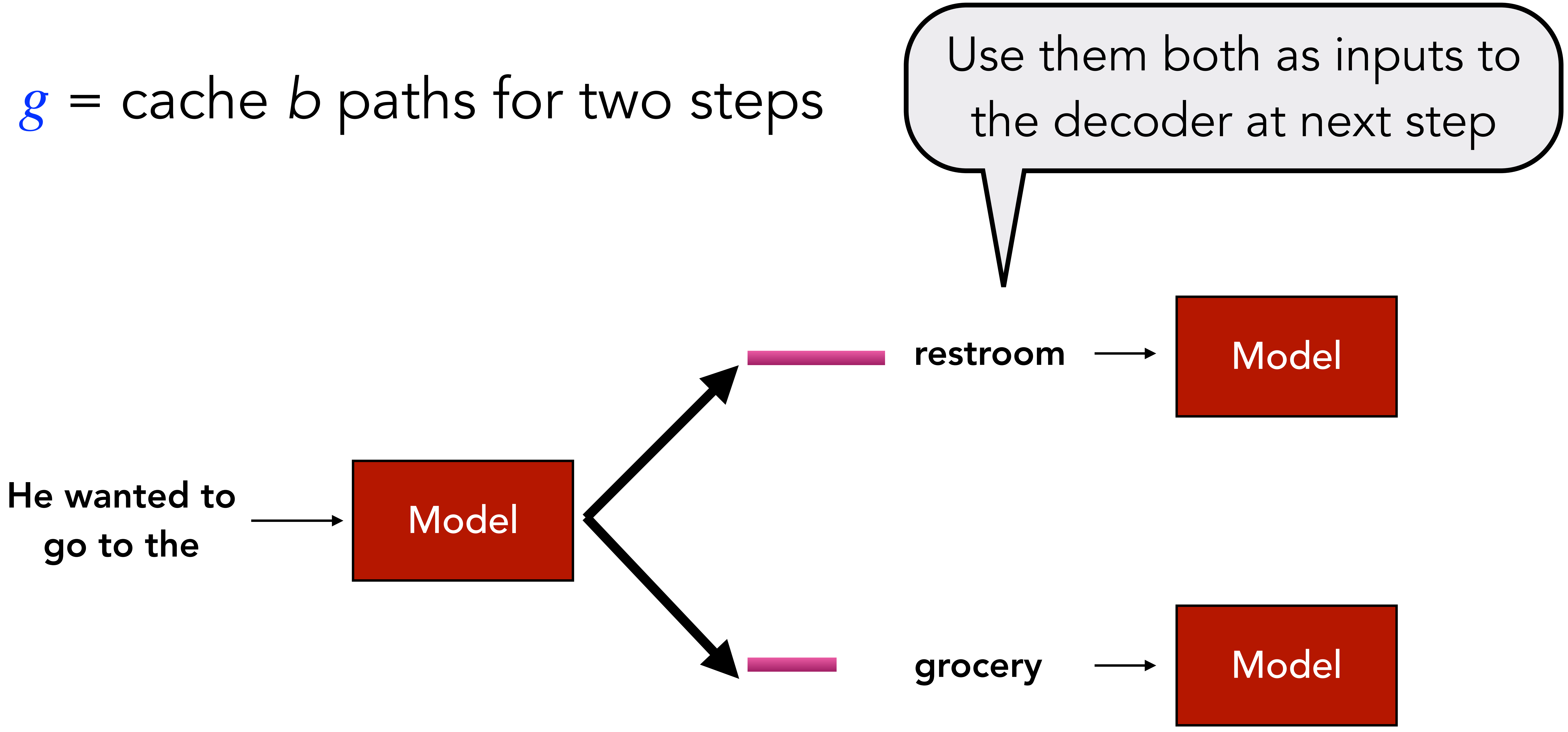
Maybe we need more options: Beam Search

- g = cache b paths for two steps



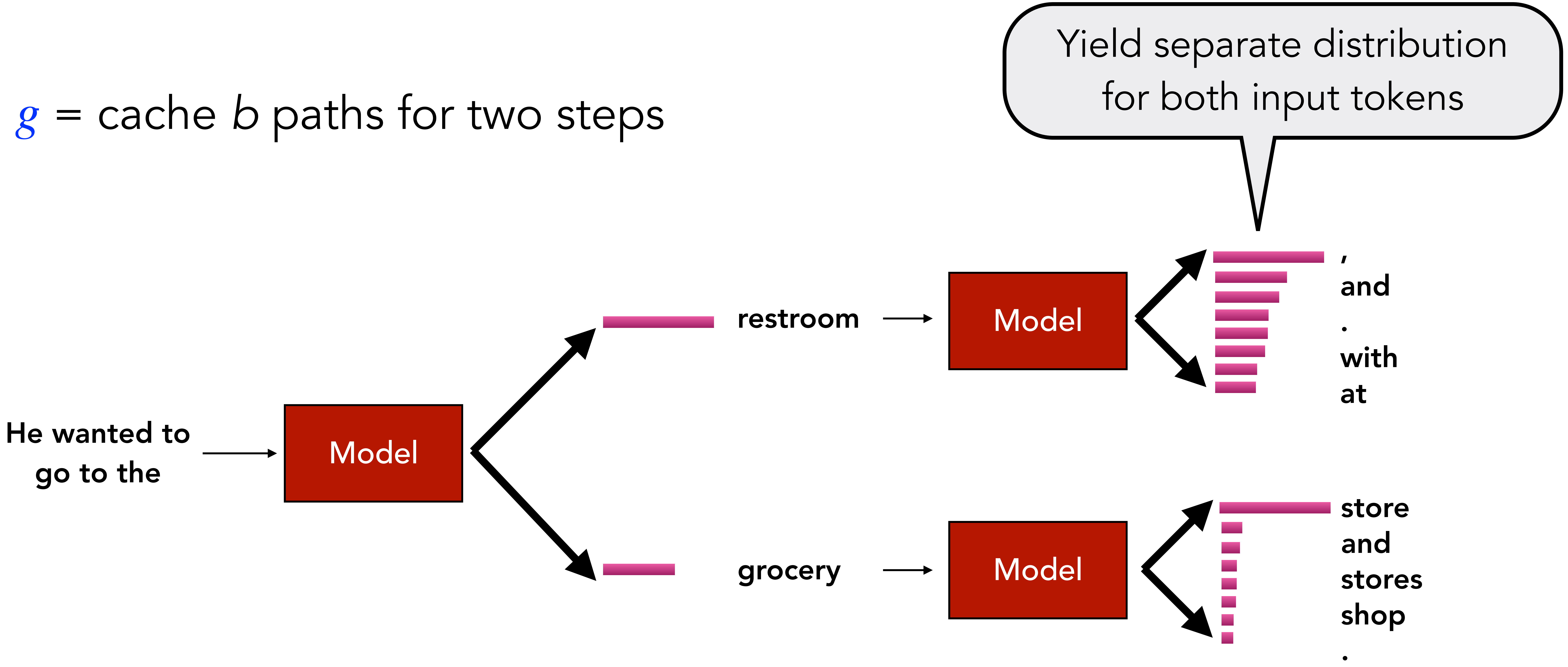
Maybe we need more options: Beam Search

- g = cache b paths for two steps



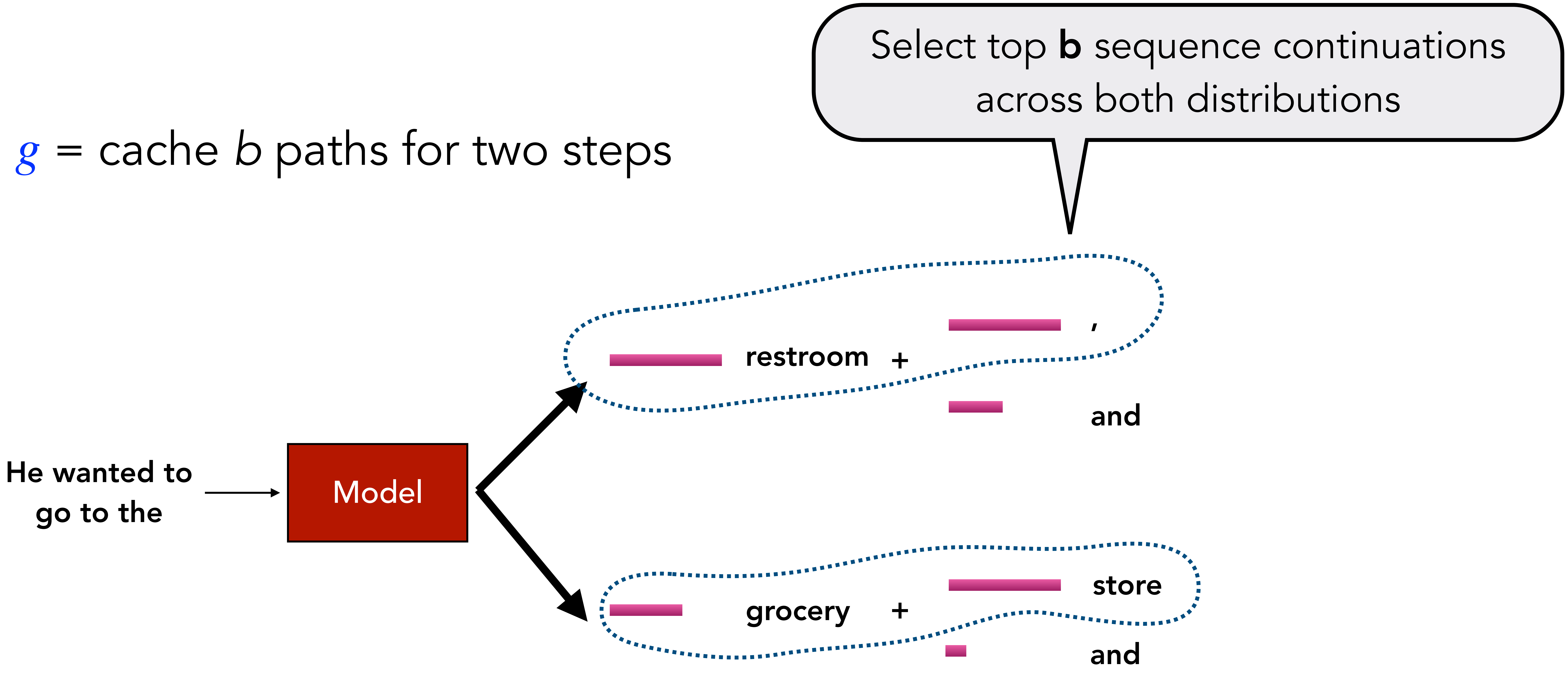
Maybe we need more options: Beam Search

- g = cache b paths for two steps



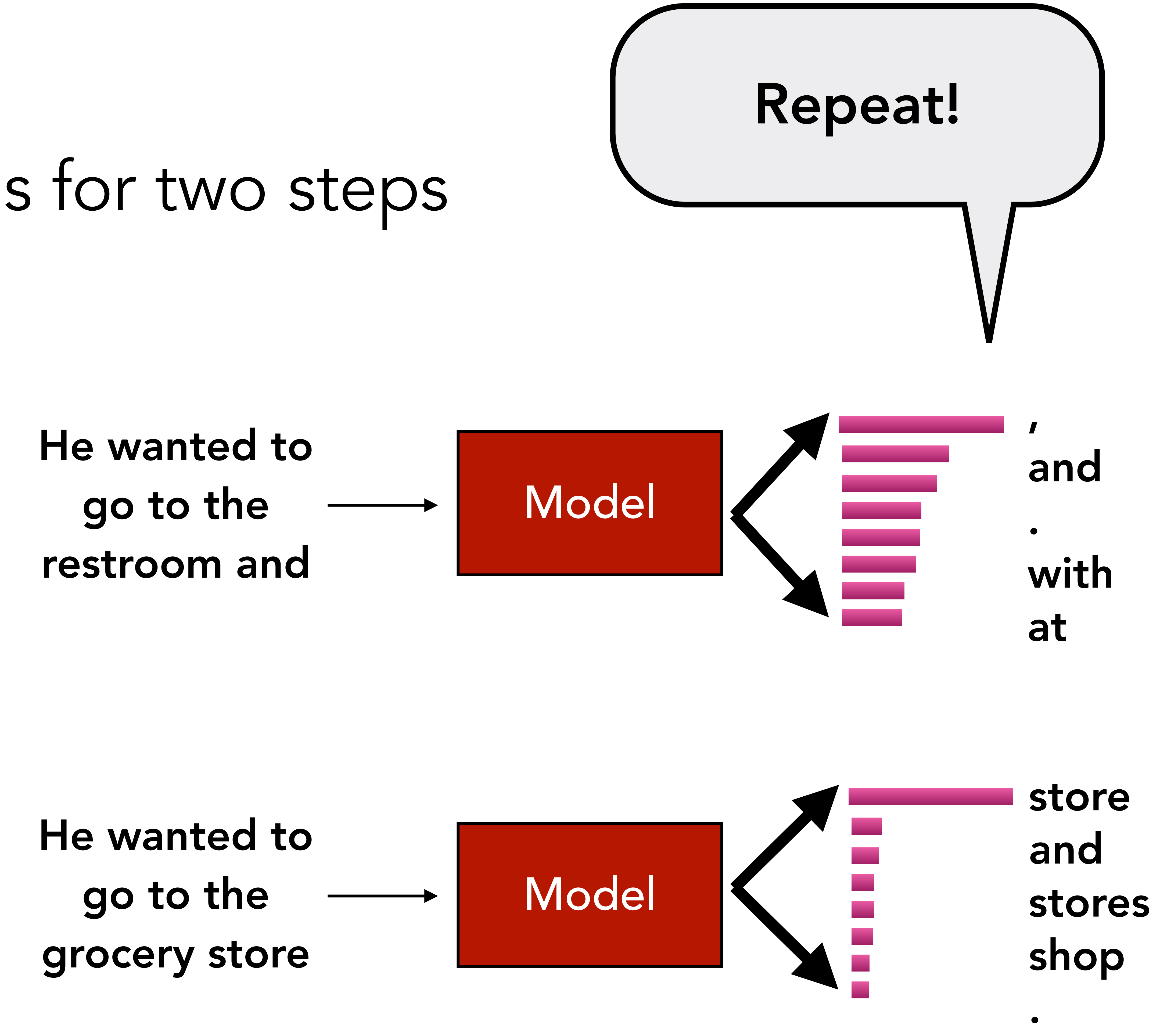
Maybe we need more options: Beam Search

- g = cache b paths for two steps



Maybe we need more options: Beam Search

- g = cache b paths for two steps



Does this penalize longer sequences?

$$s(Y) = \sum_{t=1}^{T} \log P\left(y_t \mid \{y\}_{<t}\right)$$

Shorter sequences will score better!

Does this penalize longer sequences?

- **Solution**: Normalize by token length of sequence

$$s(Y) = \frac{1}{|Y|} \sum_{t=1}^{|Y|} \log P\left(y_t \mid \{y\}_{<t}\right)$$

- **Solution:** Normalize by token length relative to reference sequence

$$s(Y) = \frac{1}{lp(Y)} \sum_{t=1}^{|Y|} \log P\left(y_t \mid \{y\}_{<t}\right) \qquad lp(Y) = \frac{(5 + |Y|)^{\alpha}}{(5+1)^{\alpha}}$$

Wu et. al., 2016

Beam search gets repetitive and 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

Beam search gets repetitive and repetitive

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even

Repetition is a big problem
in text generation!

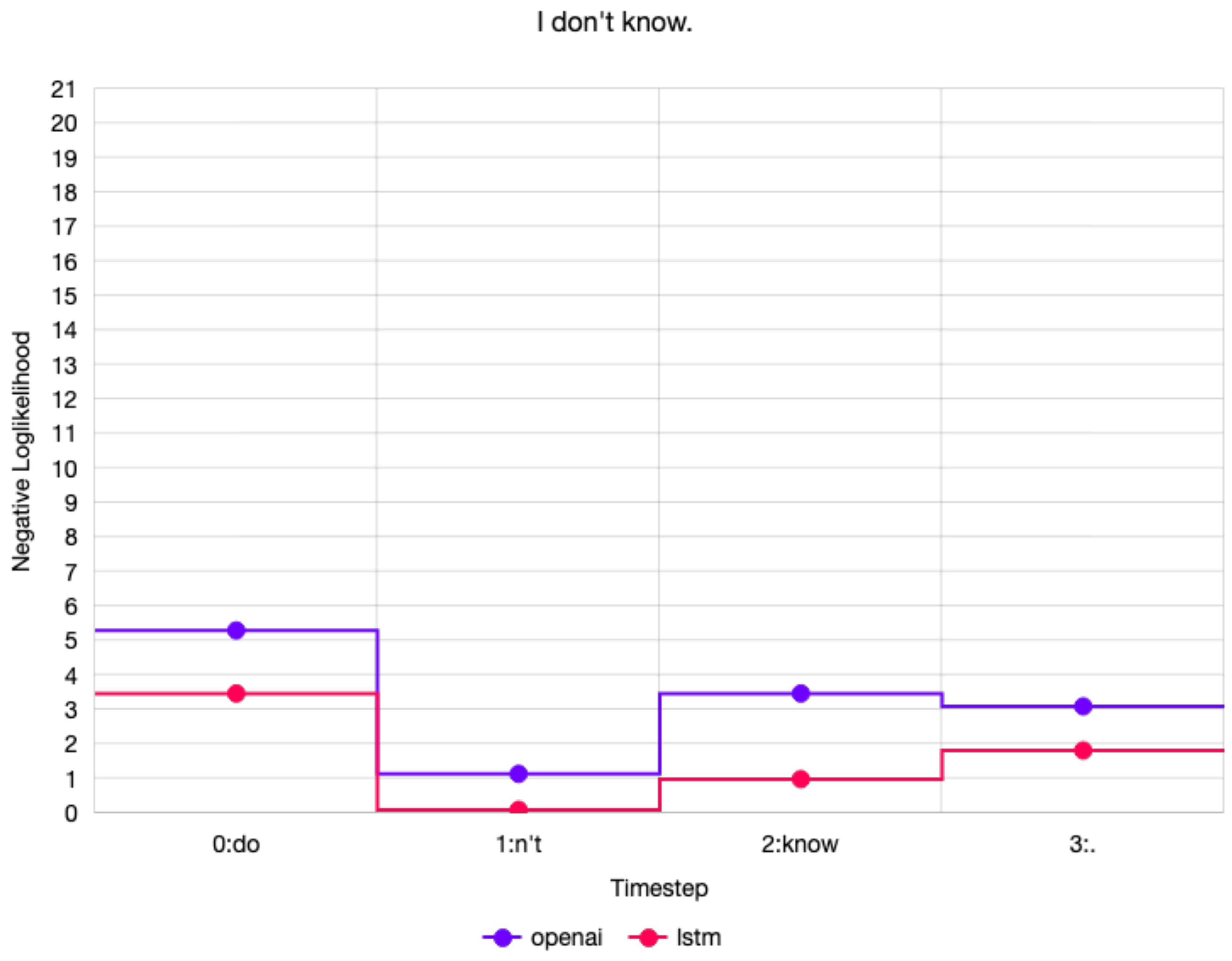
Continuatio

e

Universidad Nacional Autonoma de Mexico (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...**

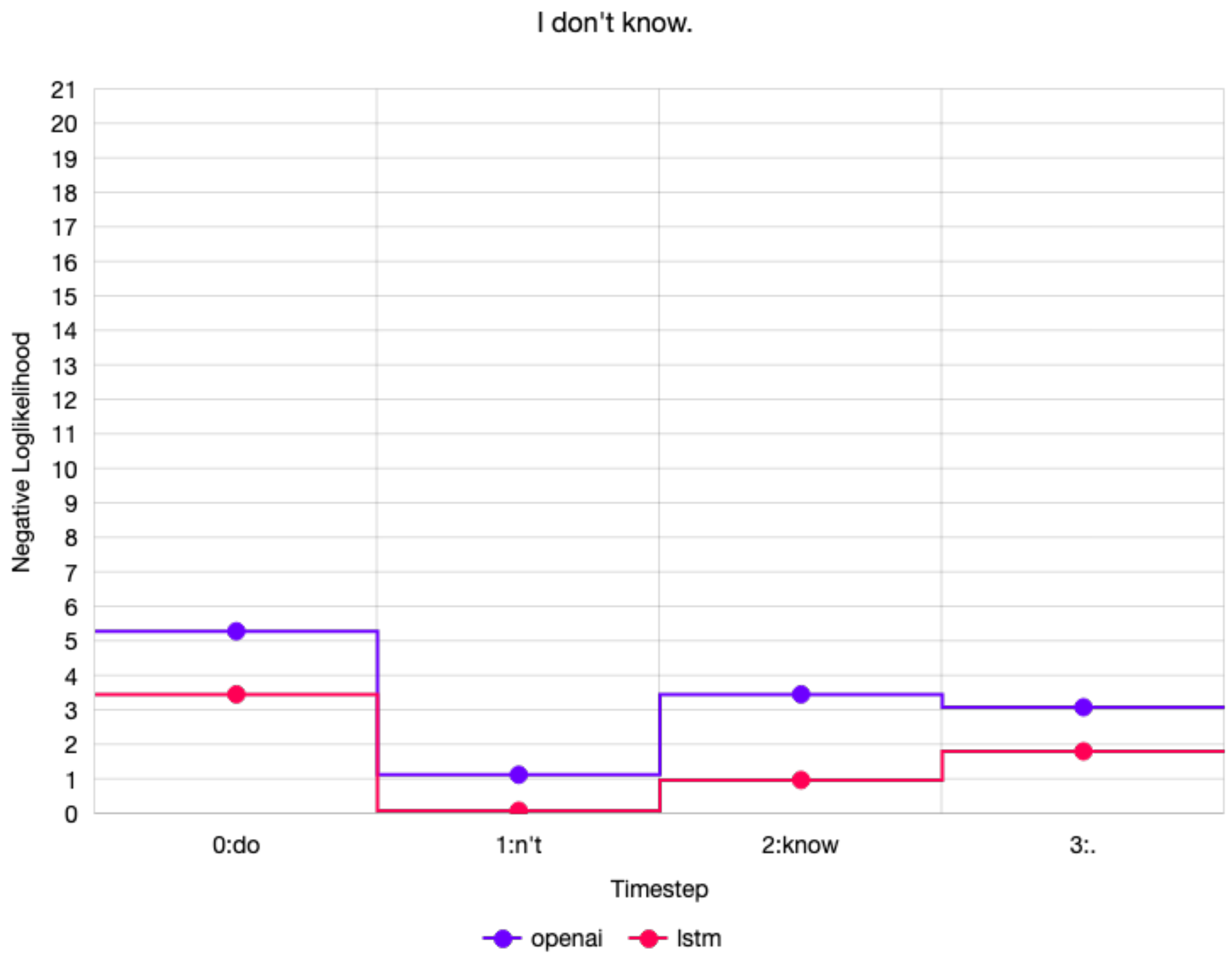
Holtzman et. al., ICLR 2020

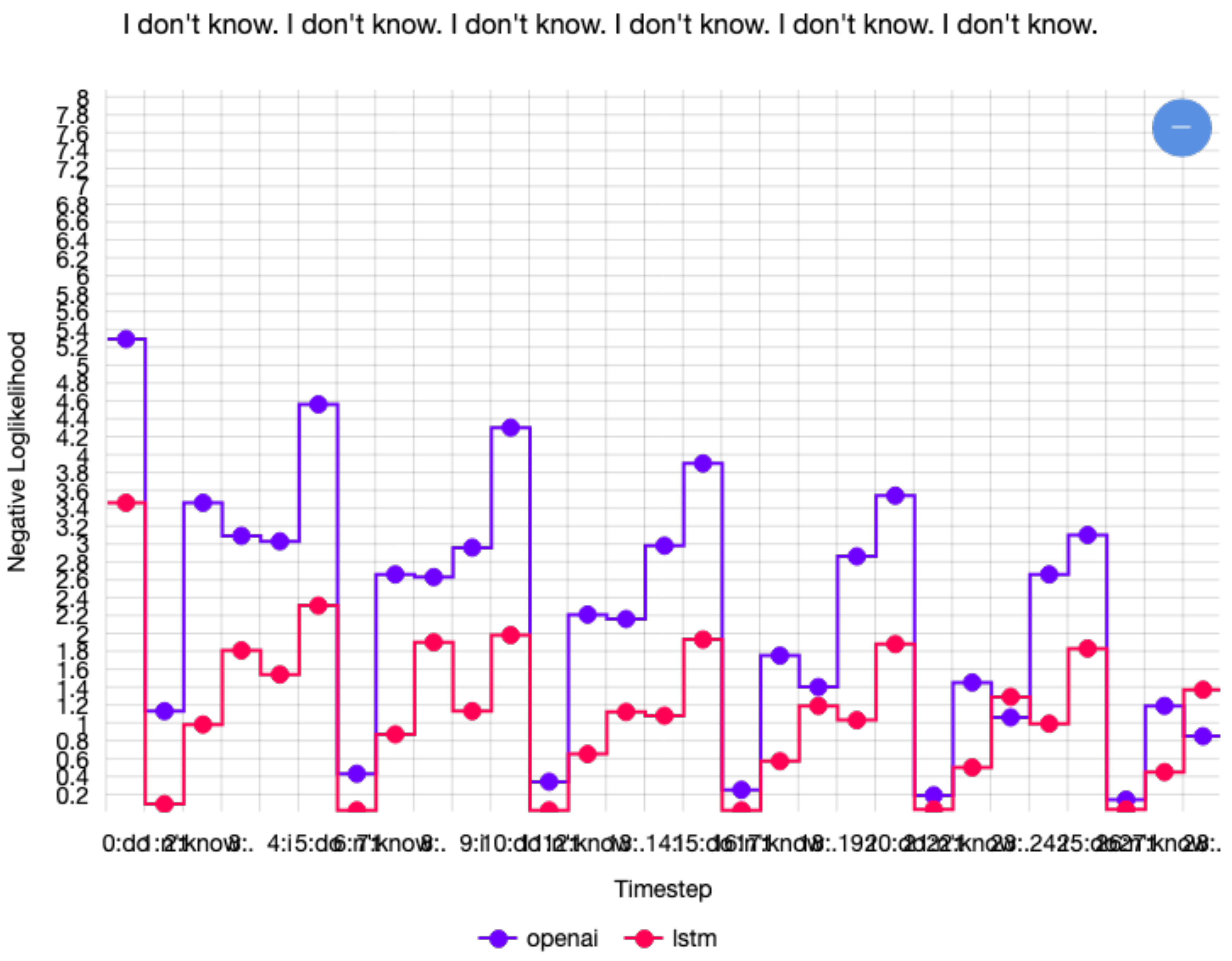
Why does this happen?



Holtzman et. al., ICLR 2020

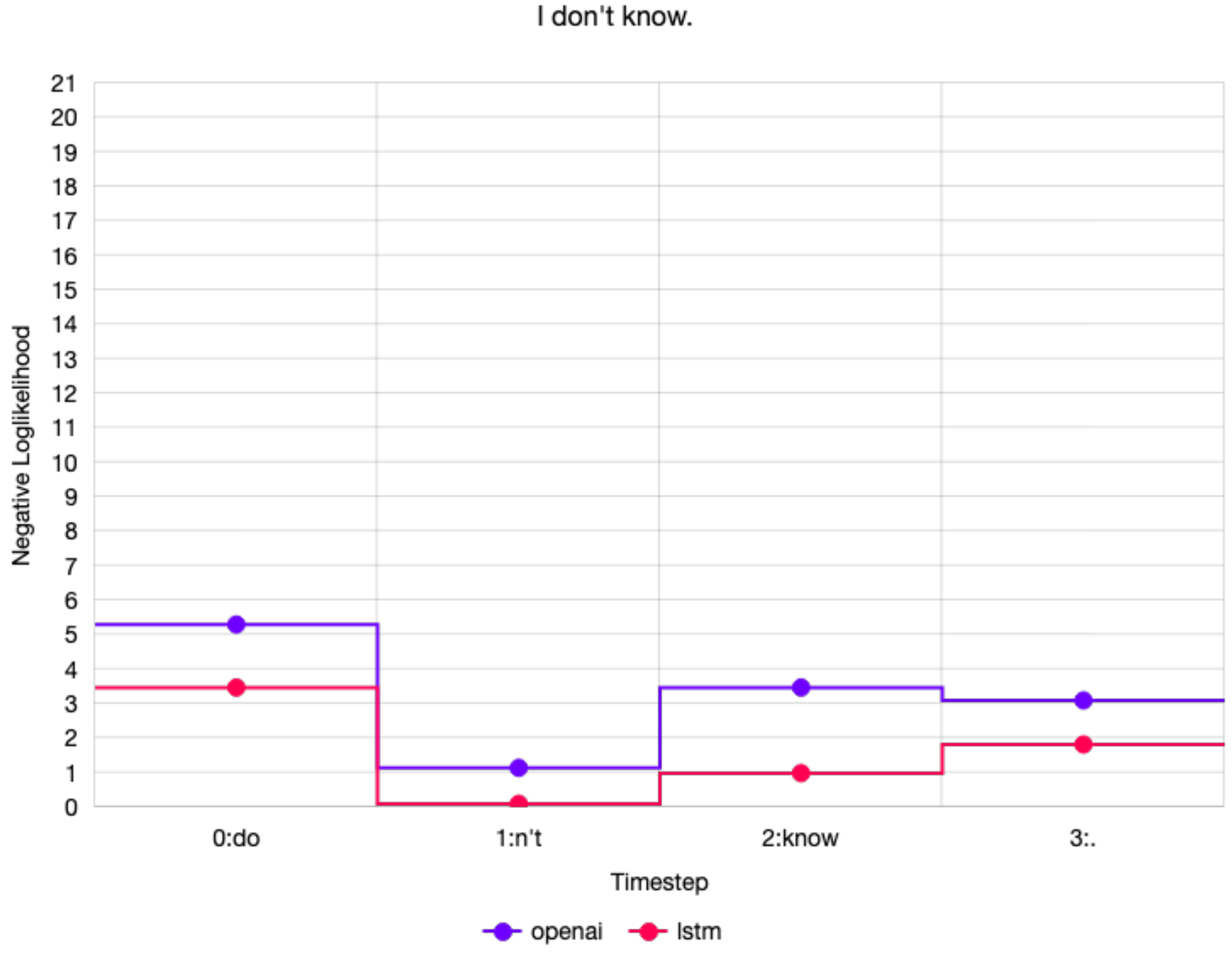
Artifact of Maximum Likelihood Training

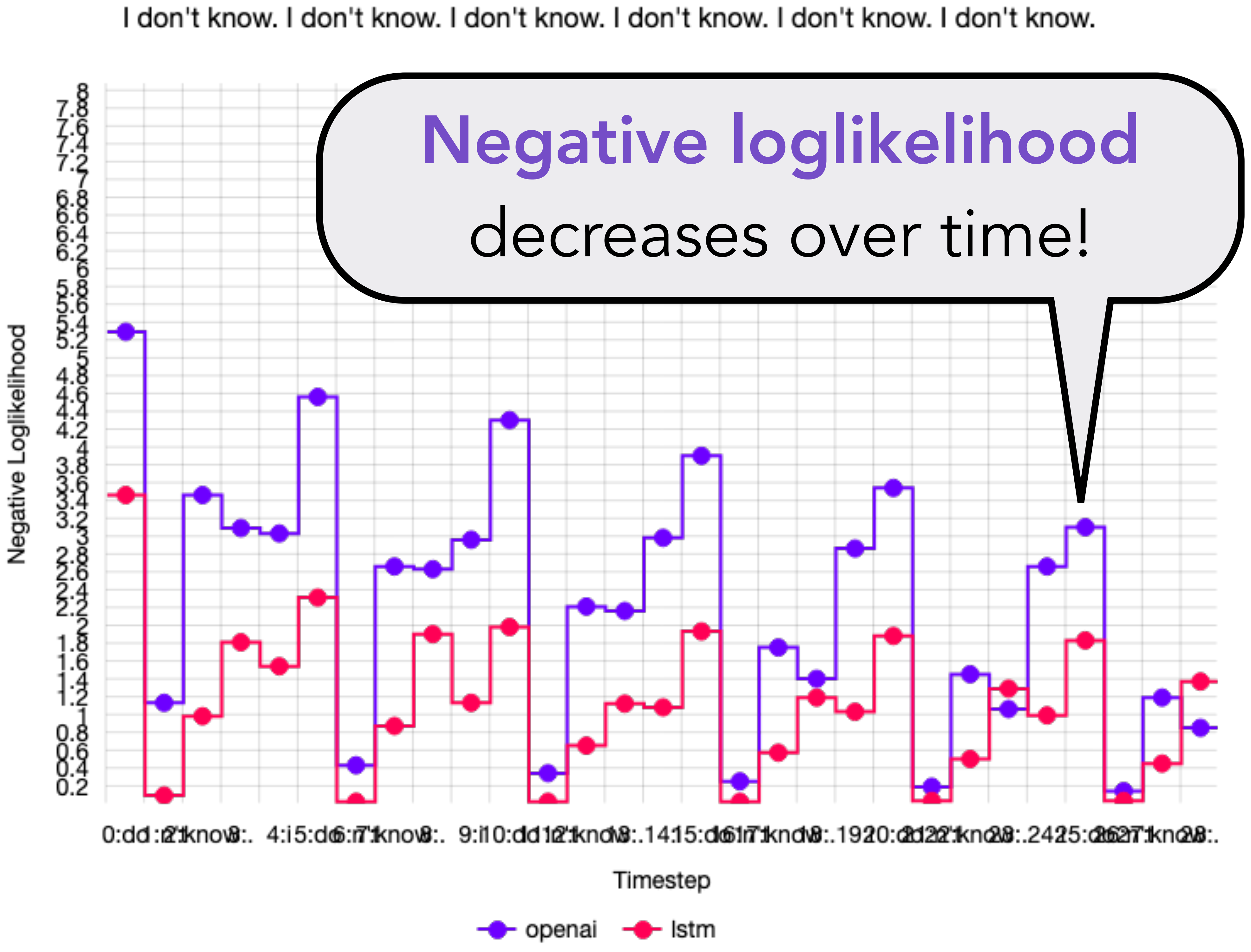




Holtzman et. al., ICLR 2020

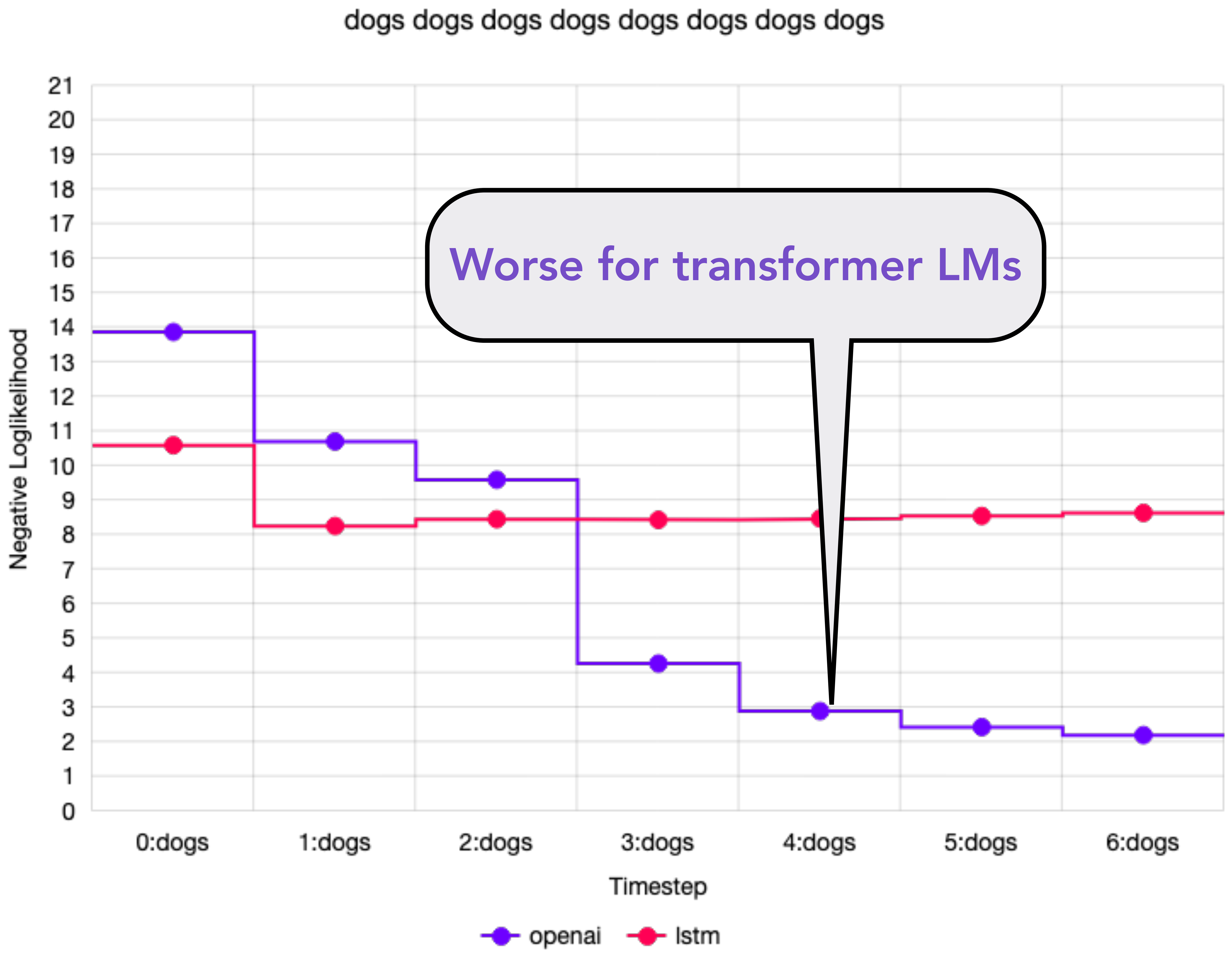
Artifact of Maximum Likelihood Training





Holtzman et. al., ICLR 2020

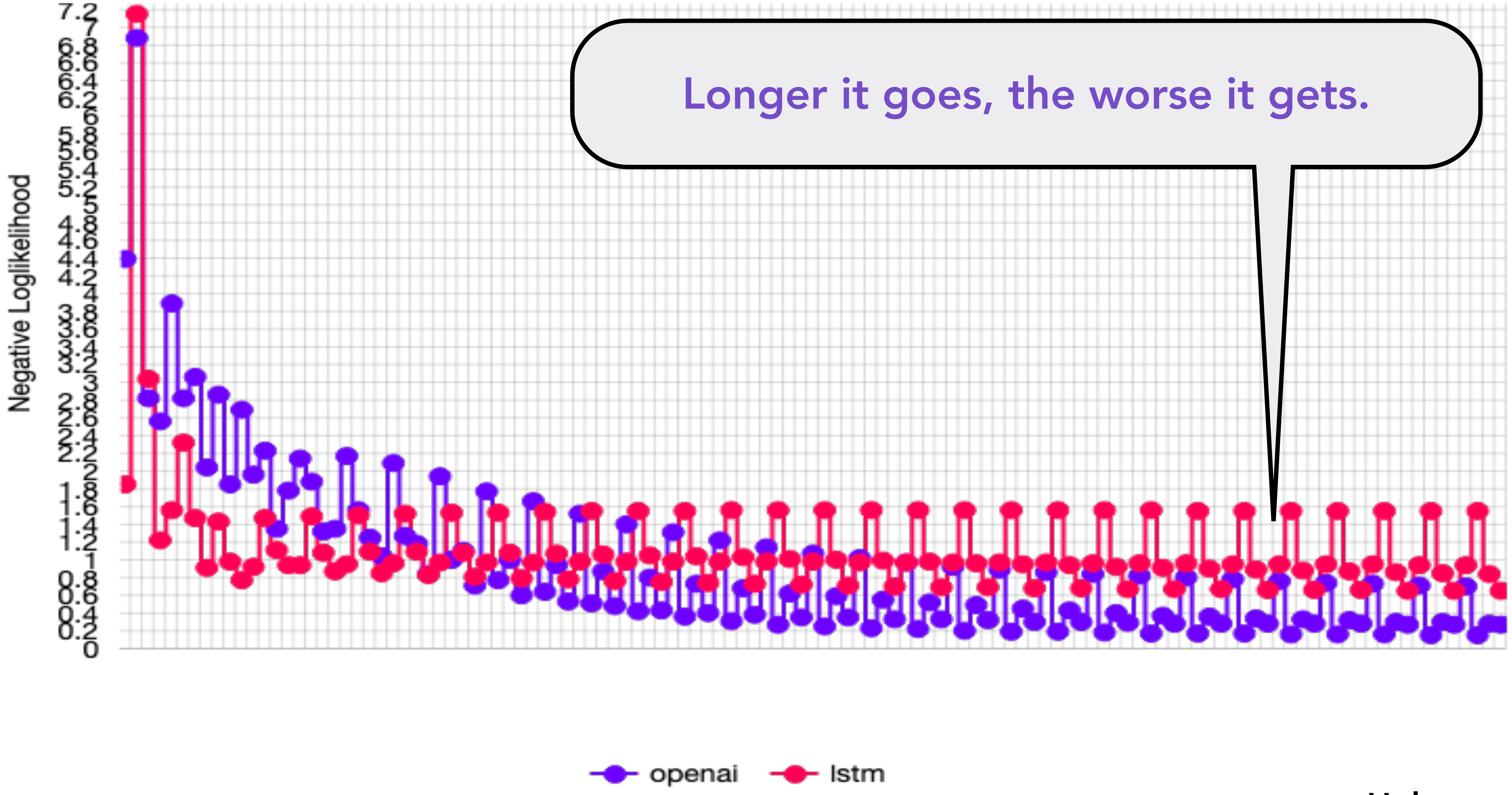
Beam search gets repetitive and repetitive



Holtzman et. al., ICLR 2020

Beam search gets repetitive and repetitive

I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired.



Holtzman et. al., ICLR 2020

How can we reduce repetition?

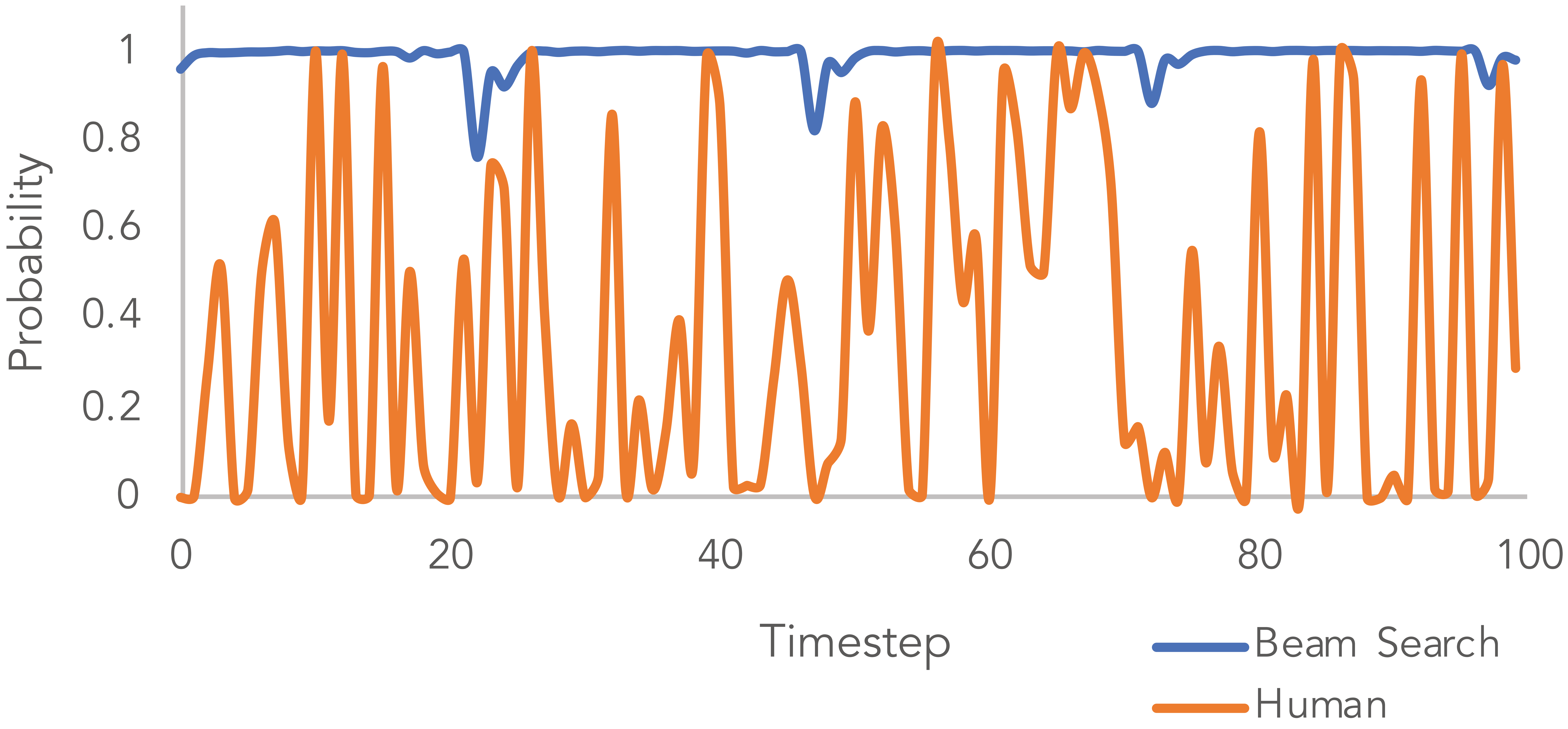
- Don't repeat *n*-grams (Hacky, but works!)

- Minimize additional loss term for minimizing hidden state similarity (LSTMs)

$$\hat{y}_t = g\left(\log P(y_t \mid \{y\}_{<t}) - s(h_t, h_{t-m})\right)$$

(Celikyilmaz et al., NAACL 2018, Paulus et al., ICLR 2018)

Step-by-step Maximization

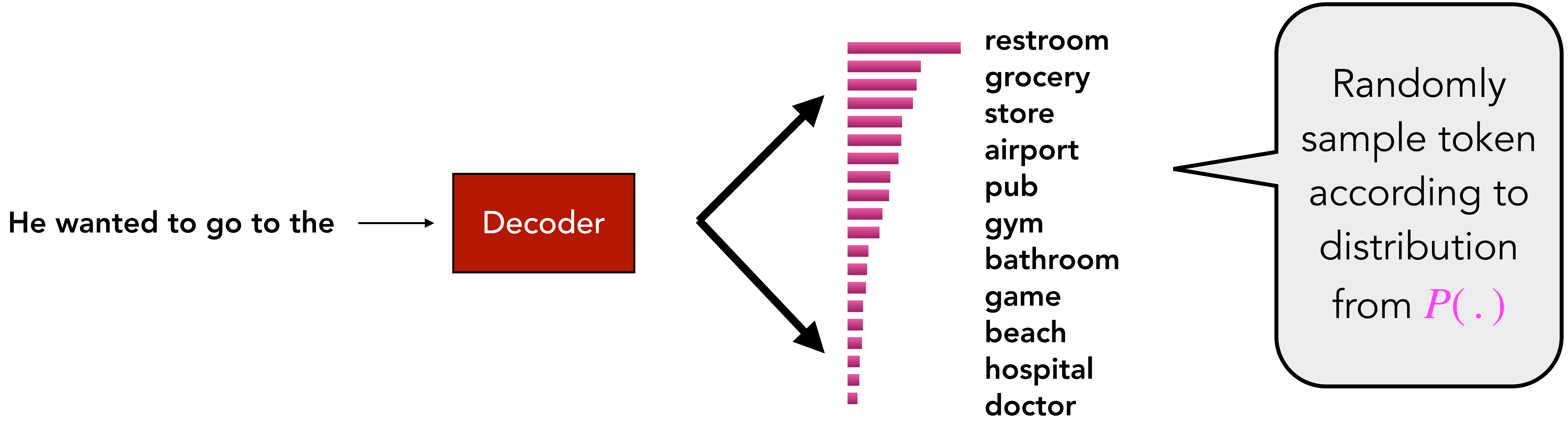


Holtzman et. al., ICLR 2020

Time to get *random*: Sampling

- g = sample a token from the distribution of tokens

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



Whoa, too *random*: Temperature Scaling

- Too much randomness: distribution has too much entropy

Whoa, too *random*: Temperature Scaling

- Too much randomness: distribution has too much entropy

- **Solution:** Make the distribution more "peaky" with temperature scaling

Recall: $$P(y_t \mid \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

Whoa, too *random*: Temperature Scaling

- Too much randomness: distribution has too much entropy
- **Solution:** Make the distribution more "peaky" with temperature scaling

Recall: $P(y_t \mid \{y\}_{<t}) = \dfrac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$

$$\Rightarrow \quad P(y_t \mid \{y\}_{<t}) = \frac{e^{o_n / \tau}}{\sum_{m=1}^{M} e^{o_m / \tau}}$$

Whoa, too *random*: Temperature Scaling

- Too much randomness: distribution has too much entropy
- **Solution:** Make the distribution more "peaky" with temperature scaling

$$P(y_t \mid \{y\}_{<t}) = \frac{e^{o_n / \tau}}{\sum_{m=1}^{M} e^{o_m / \tau}}$$

$\tau > 1$ "flatter" distribution

$\tau < 1$ "peakier" distribution

Maybe we need fewer options: Top-*k* sampling

- The entire distribution over tokens is not needed at every step
- Many token choices should have no chance of being selected

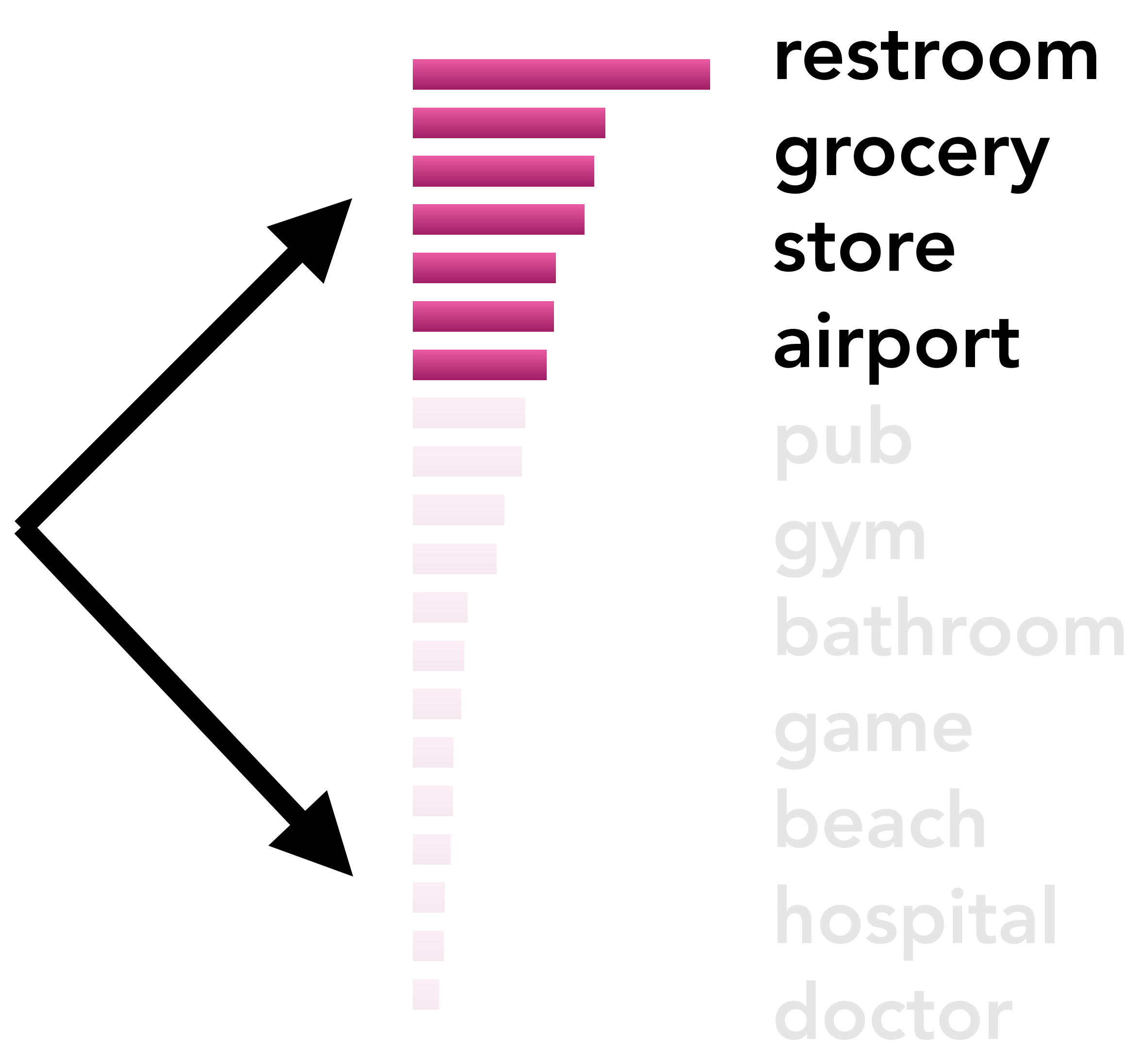
(Holtzman et. al., ACL 2018; Fan et al., ACL 2018)

Maybe we need fewer options: Top-*k* sampling

- The entire distribution over tokens is not needed at every step
- Many token choices should have no chance of being selected
- Only sample from the top *k* tokens in the distribution

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

He wanted to go to the → Decoder

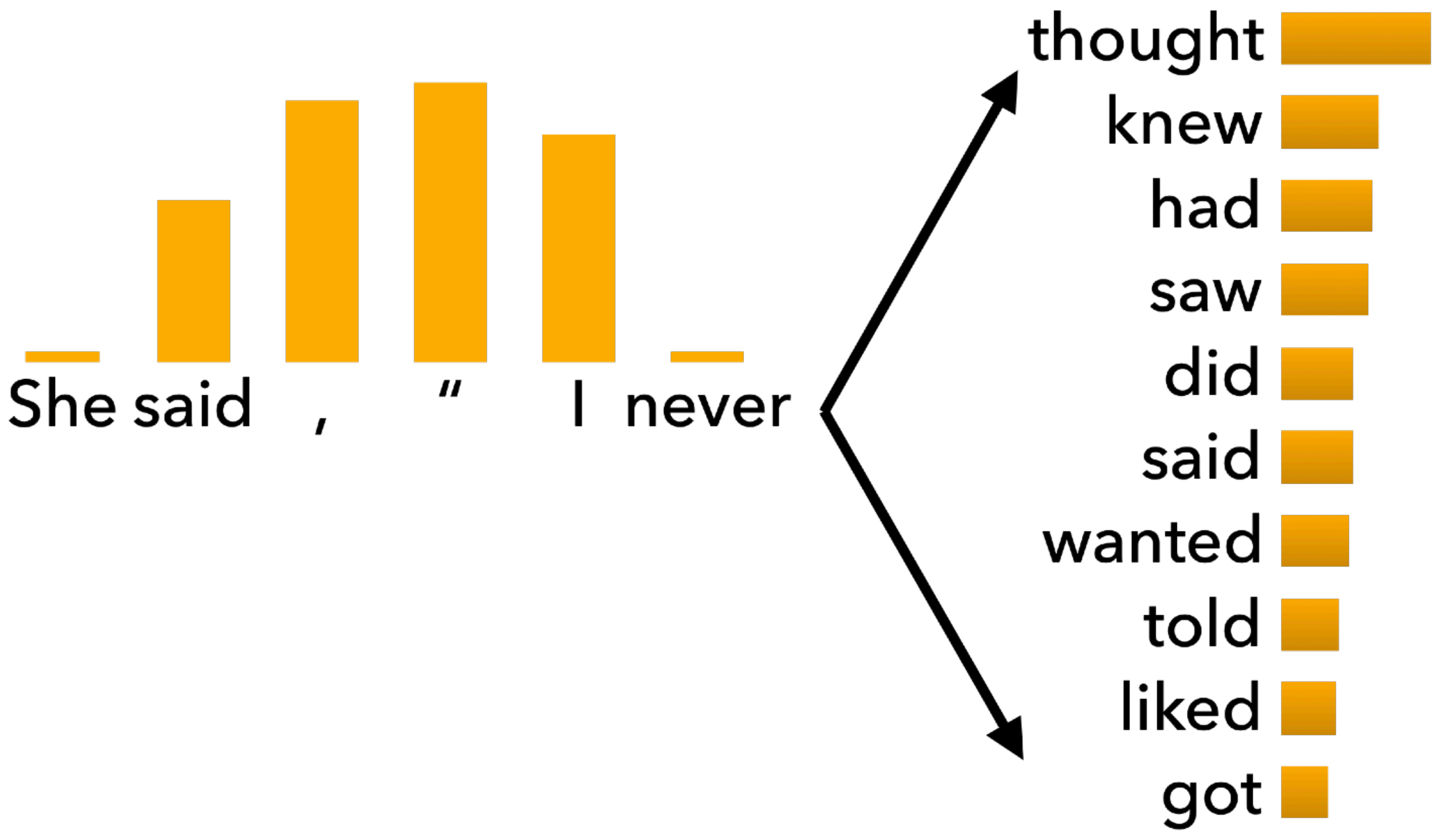


Randomly sample token from top *k* highest probability tokens in $P(\,.\,)$

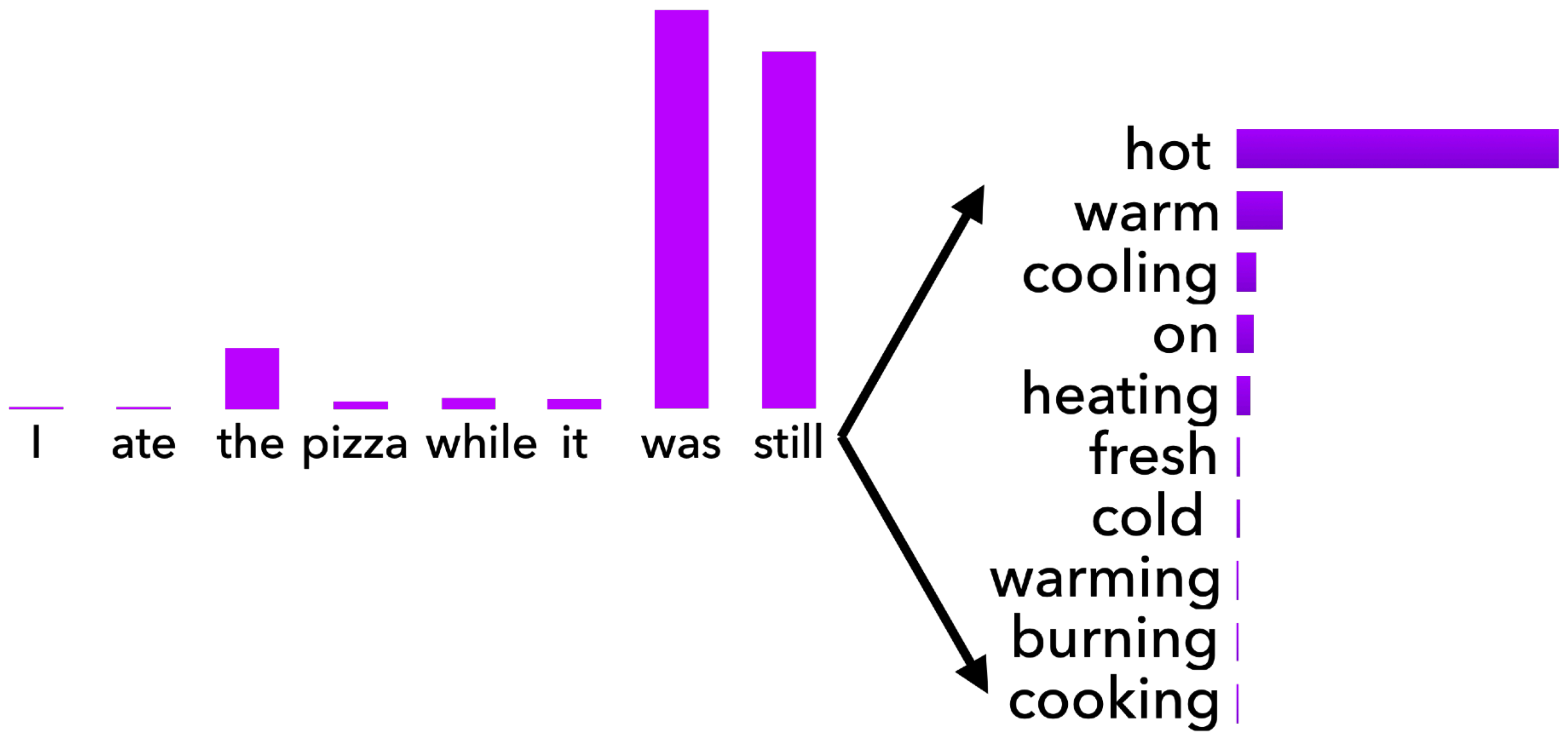
(Holtzman et. al., ACL 2018; Fan et al., ACL 2018)

Issues with top-*k* sampling

Top-*k* can cut-off too ***quickly***



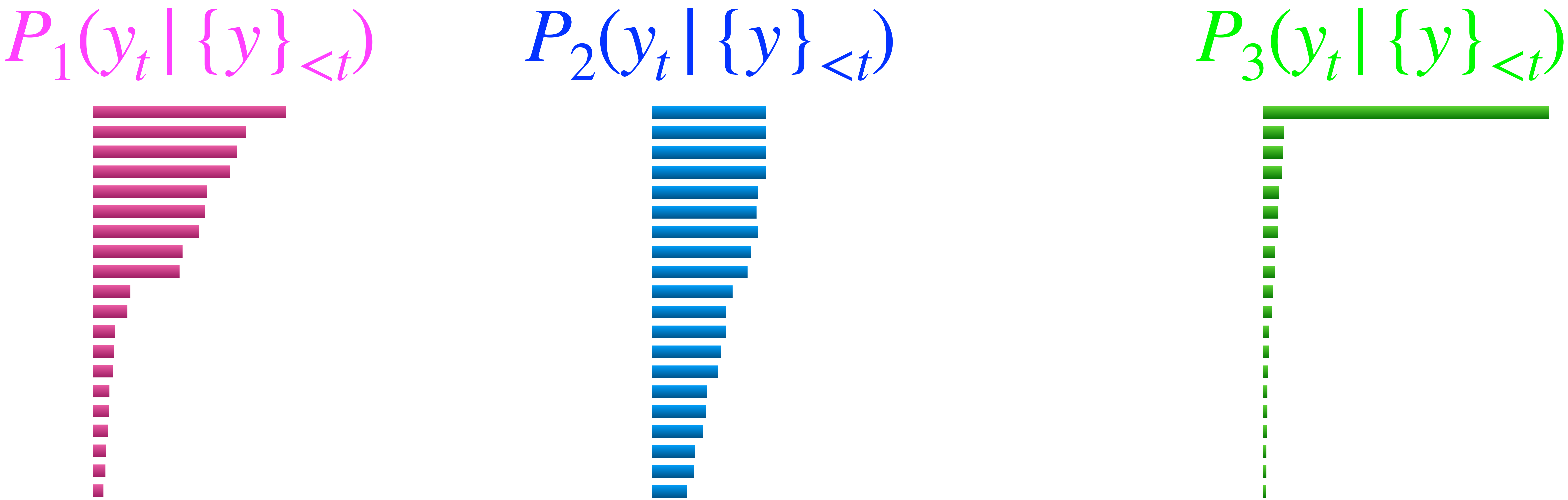
Top-*k* can cut-off too ***slowly***



(Holtzman et. al., ICLR 2020)

I don't know how many options I need: Top-*p* sampling

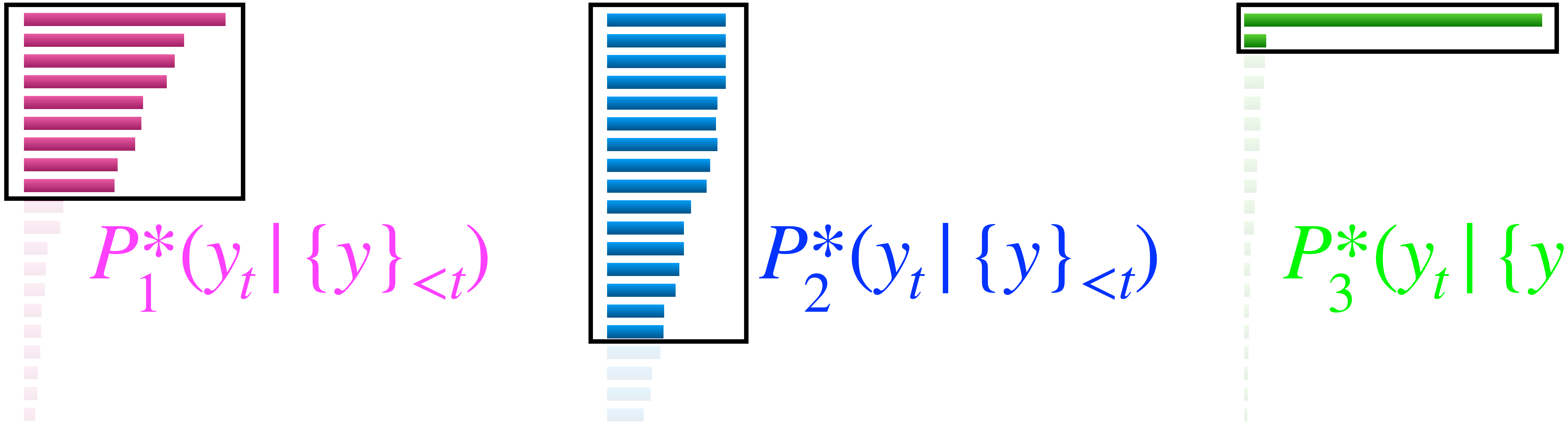
- Also known as **nucleus** sampling
- Sample from subset of vocabulary where probability mass is concentrated



(Holtzman et. al., ICLR 2020)

I don't know how many options I need: Top-*p* sampling

- Also known as **nucleus** sampling:
- Sample from subset of vocabulary where probability mass is concentrated
- Probability mass has a dynamically changing **nucleus**



$P_1^*(y_t \mid \{y\}_{<t})$ $P_2^*(y_t \mid \{y\}_{<t})$ $P_3^*(y_t \mid \{y\}_{<t})$

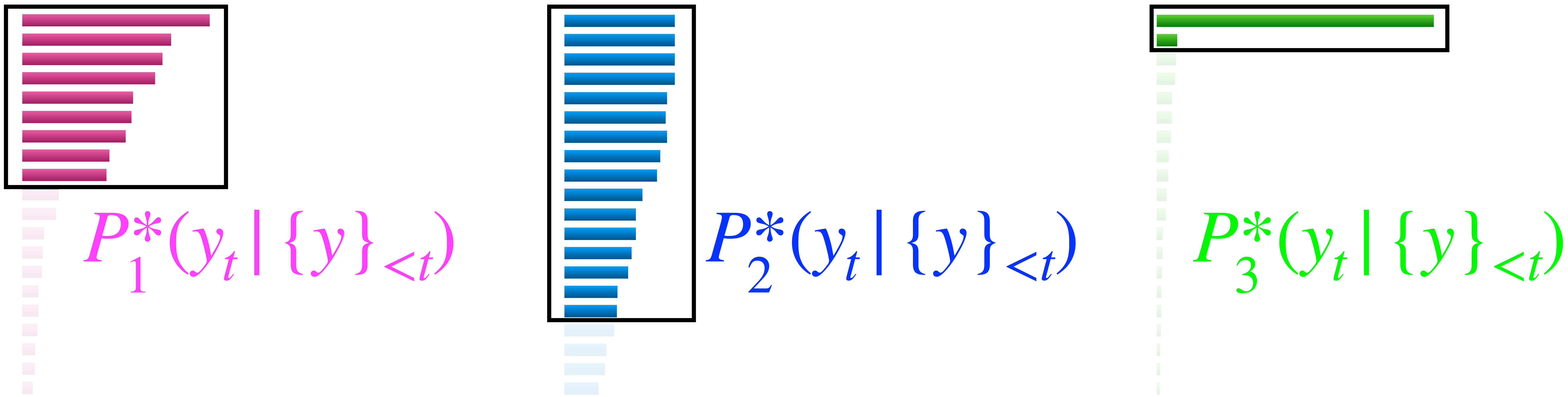
*nuclei not to scale

(Holtzman et. al., ICLR 2020)

I don’t know how many options I need: Top-*p* sampling

- Also known as **nucleus** sampling:
- Sample from subset of vocabulary where probability mass is concentrated
- Probability mass has a dynamically changing **nucleus**

Only sample from the **nucleus** of the **distribution**



*nuclei not to scale

(Holtzman et. al., ICLR 2020)

This all sounds a bit *risky*

- What if my sequence just isn't very good?

Optimize other sequence-level scores: Re-ranking

- What if my sequence just isn’t very good?
- Sample a bunch of sequences
- Define a score to approximate the quality of your sequence.
- Simplest is to just use perplexity!

Optimize other sequence-level scores: Re-ranking

- What if my sequence just isn't very good?
- Sample a bunch of sequences
- Define a score to approximate the quality of your sequence.
- Simplest is to just use perplexity!
- However, re-rankers can be used to score a variety of properties: style (Holtzman et al., 2018), discourse (Gabriel et al., 2019), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more…

Optimize other sequence-level scores: Re-ranking

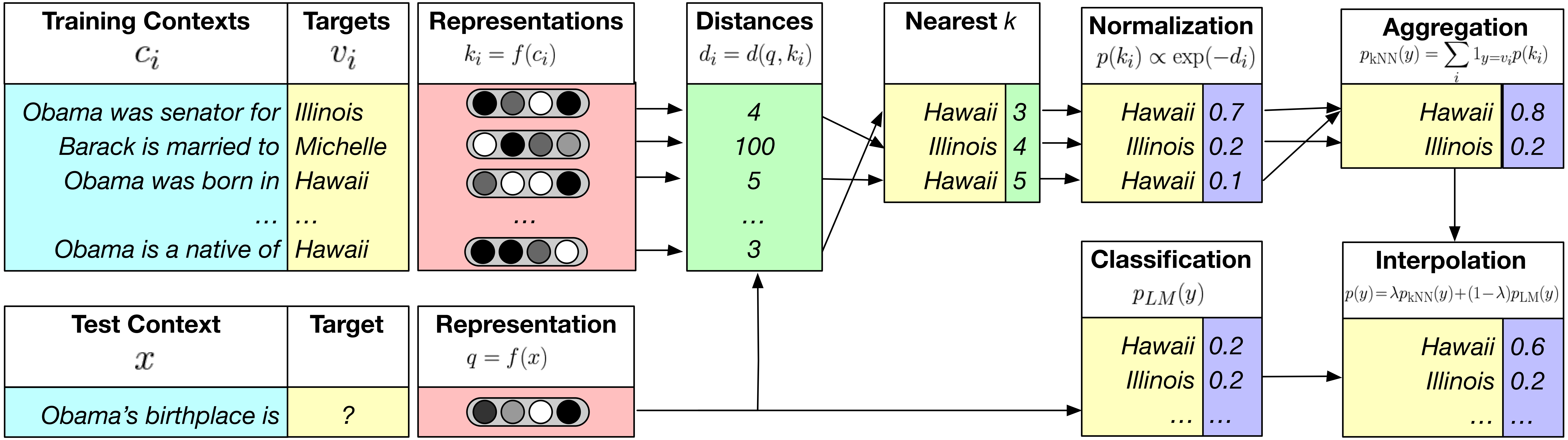
- What if my sequence just isn't very good?
- Sample a bunch of sequences
- Define a s
- Simplest is
- However, re-rankers can be used to score a variety of properties: style (Holtzman et al., 2018), discourse (Gabriel et al., 2019), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more…

Change your distribution
at inference time!

kNN Language Models

- Don't just rely on your trained model to generate a distribution over tokens

- Use knowledge of similar contexts from another corpus

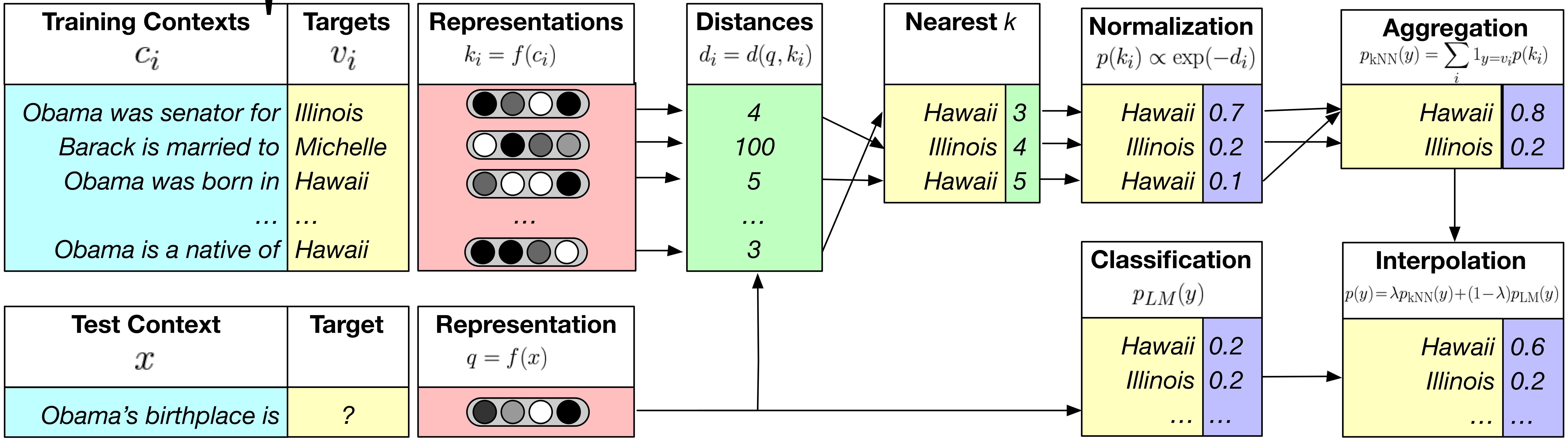


(Khandelwal et. al., ICLR 2020)

kNN Language Models

Initialize a **database** of contexts

- Don't just rely on your trained model to generate a distribution over
- Use knowledge of similar contexts from another corpus



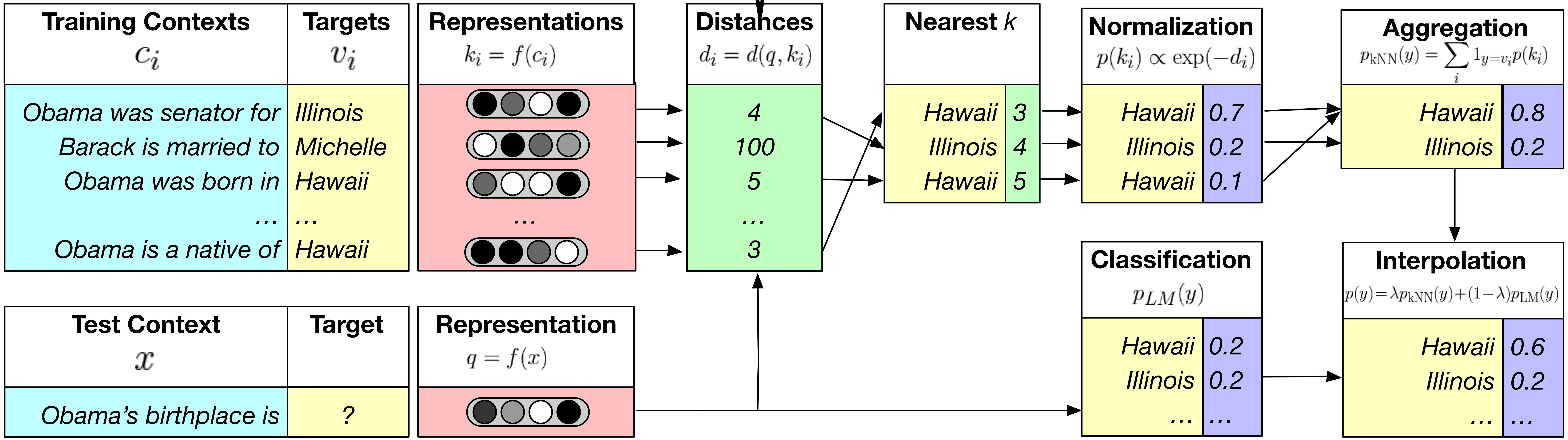
(Khandelwal et. al., ICLR 2020)

kNN Language Models

- Don't just rely on your trained model to generate a distribution over to

Efficiently compute **distance** between each context in DB and current sequence

- Use knowledge of similar contexts from another corpus



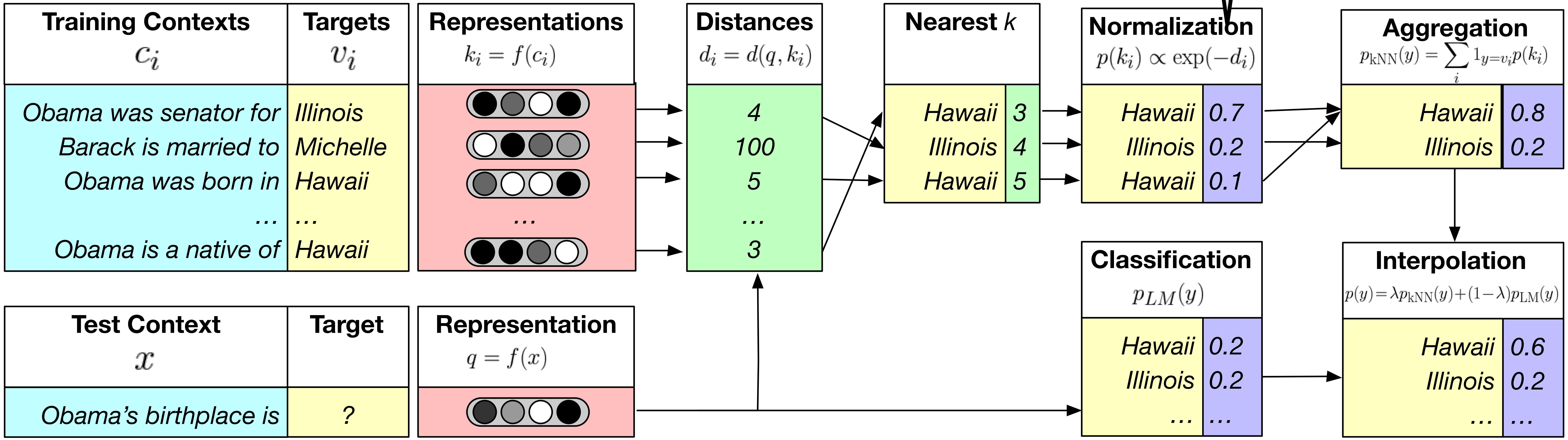
(Khandelwal et. al., ICLR 2020)

kNN Language Models

- Don't just rely on your trained model to generate a distribution over tokens

Compute **distribution** over possible **targets** from context DB sequences using **distance of history**

- Use knowledge of similar contexts from another corpus



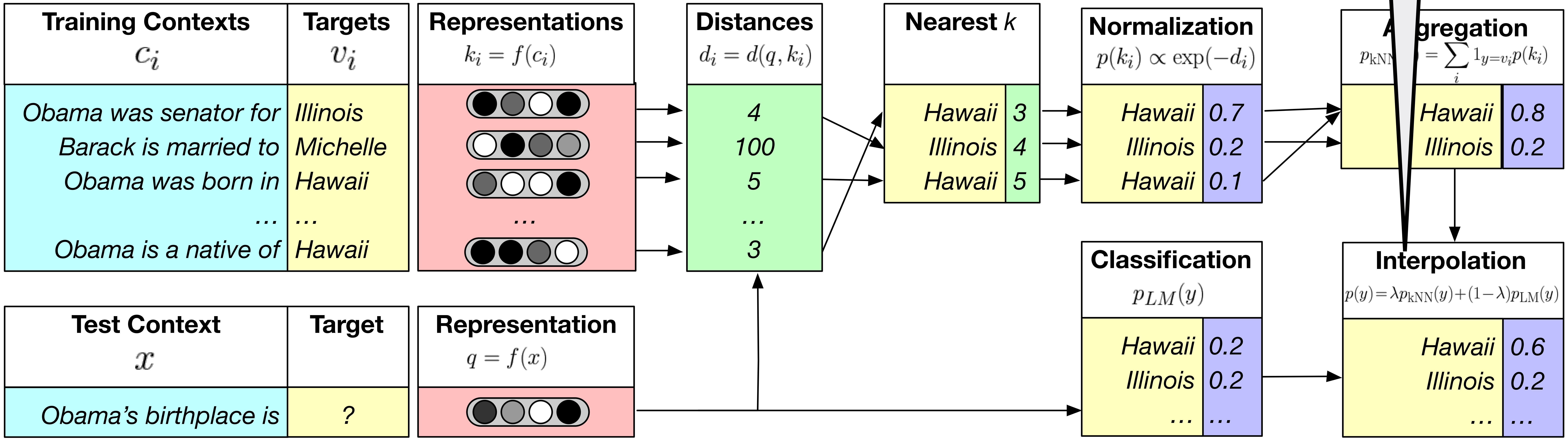
(Khandelwal et. al., ICLR 2020)

kNN Language Models

- Don't just rely on your trained model to generate a distribution over tokens

Interpolate distribution from nearest neighbor search with model distribution

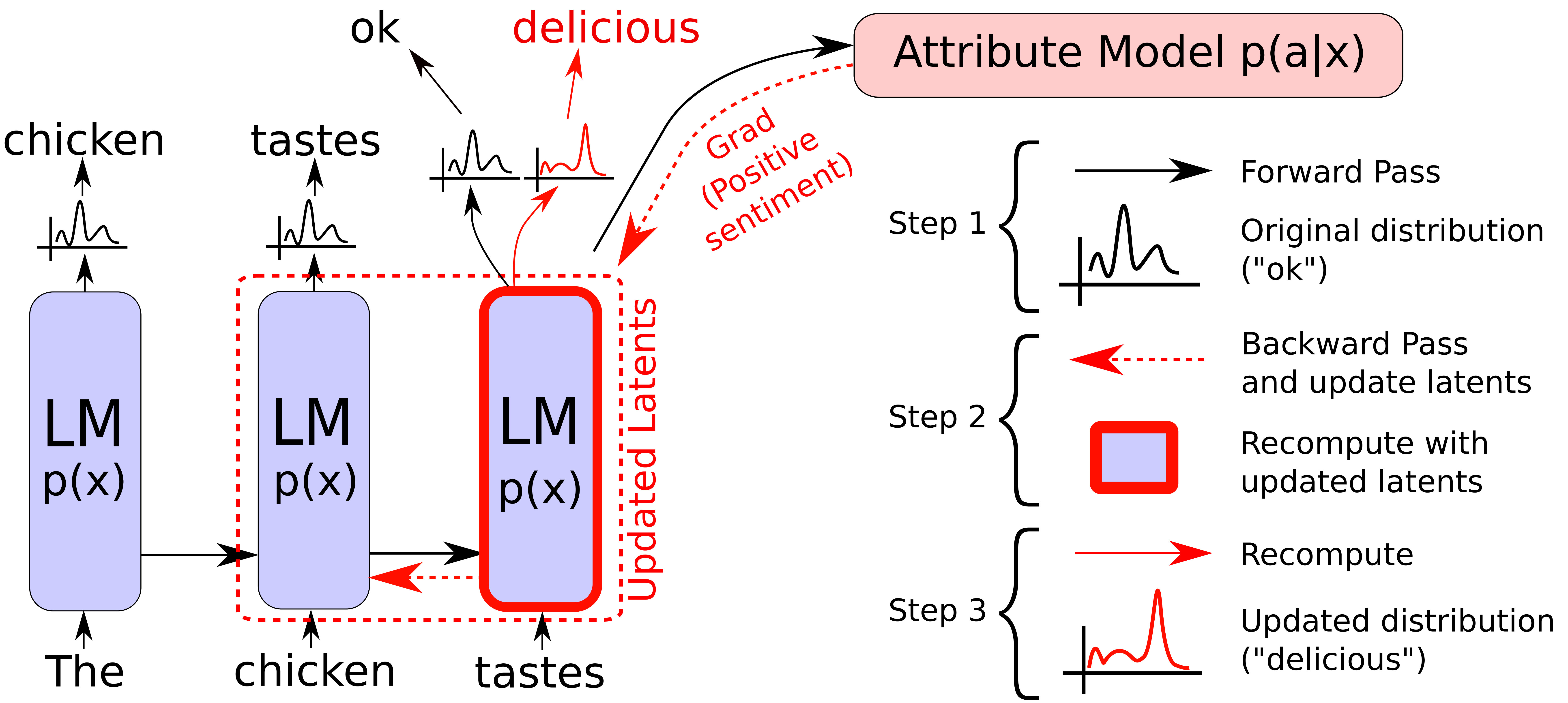
- Use knowledge of similar contexts from another corpus



(Gulcehre et al., 2015; Venugopalan et al., EMNLP 2016; Khandelwal et. al., ICLR 2020)

Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?

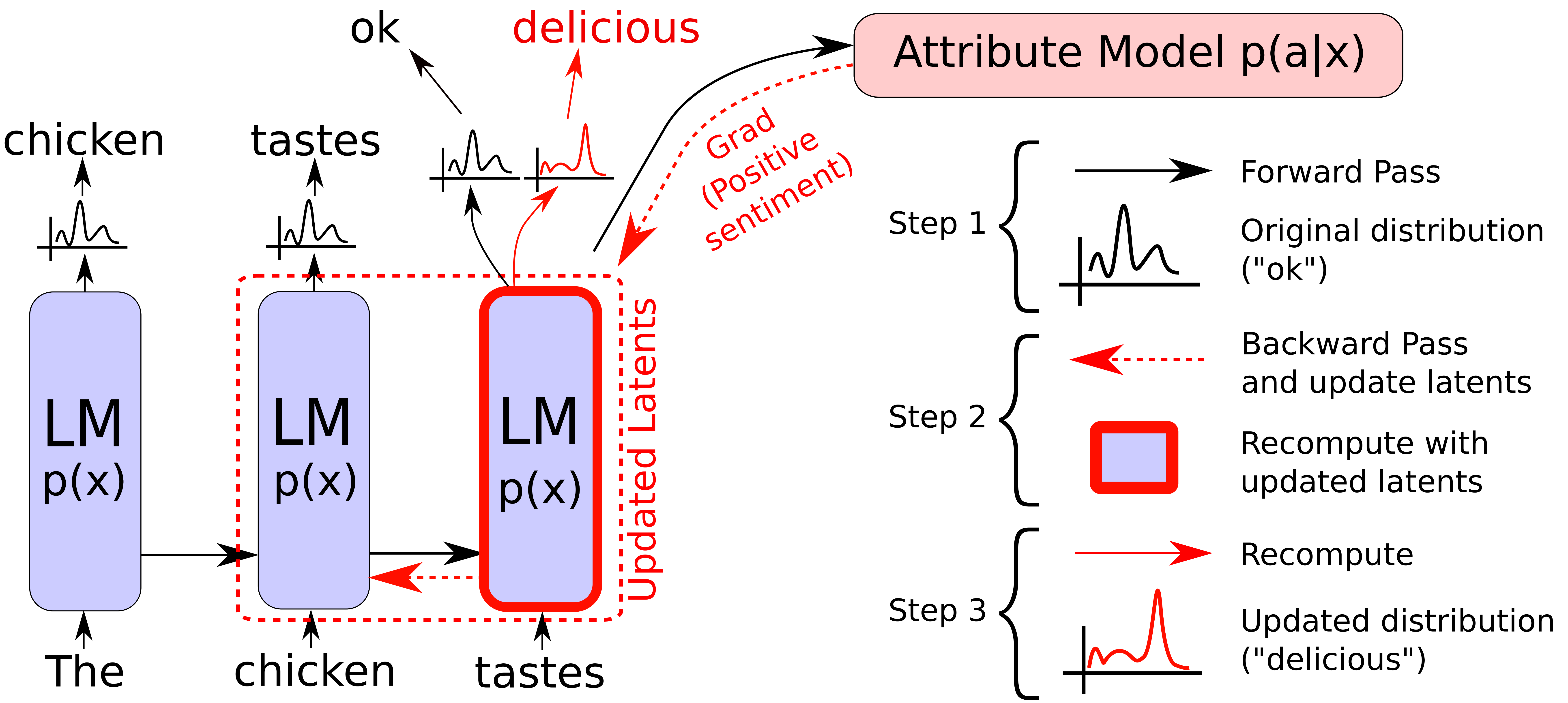


(Dathathri et al., ICLR 2020)

Plug and Pl s!

Define an attribute model that scores the generated sequence. Each generated token must try to increase the score given to the sequence by the attribute model

- What if I want to encou
  time?

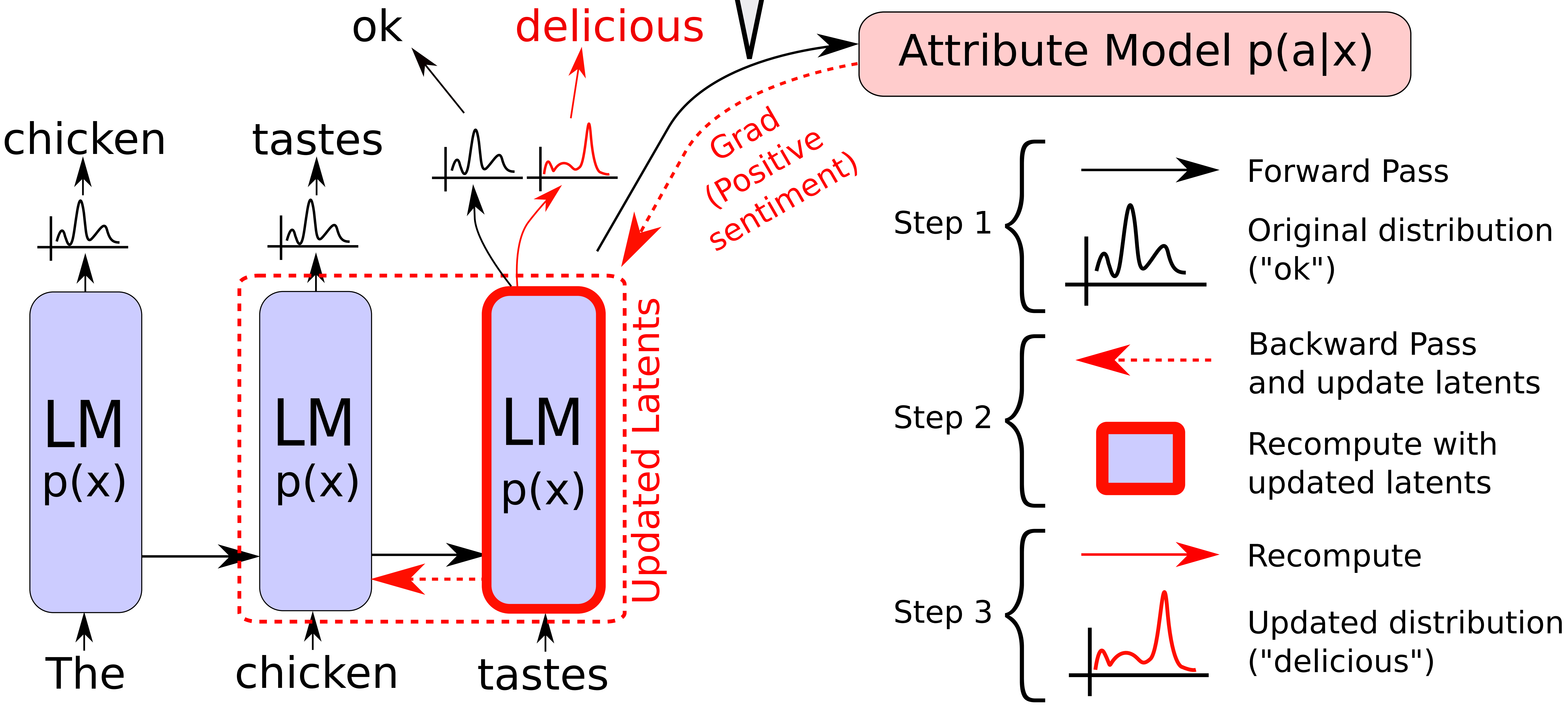


(Dathathri et al., ICLR 2020)

Plug and Play Language Models!

- What if I wa rmalize behavior at inference time?

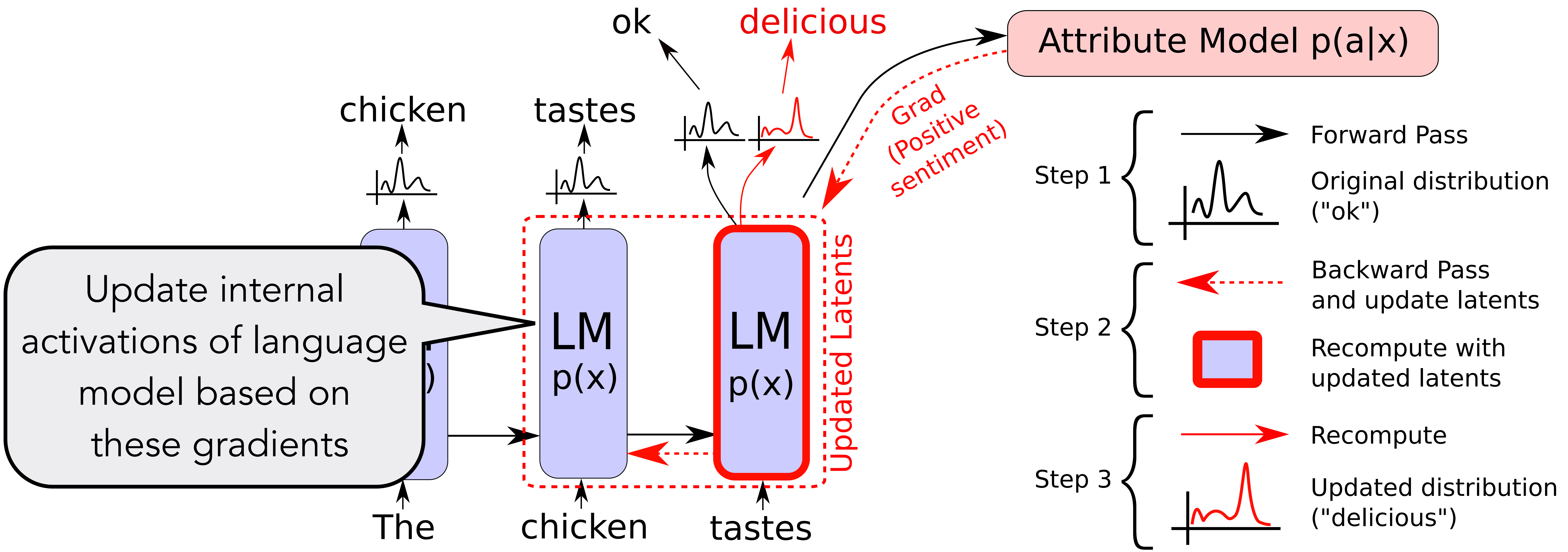
Backpropagate loss with respect to attribute score



(Dathathri et al., ICLR 2020)

Plug and Play Language Models!

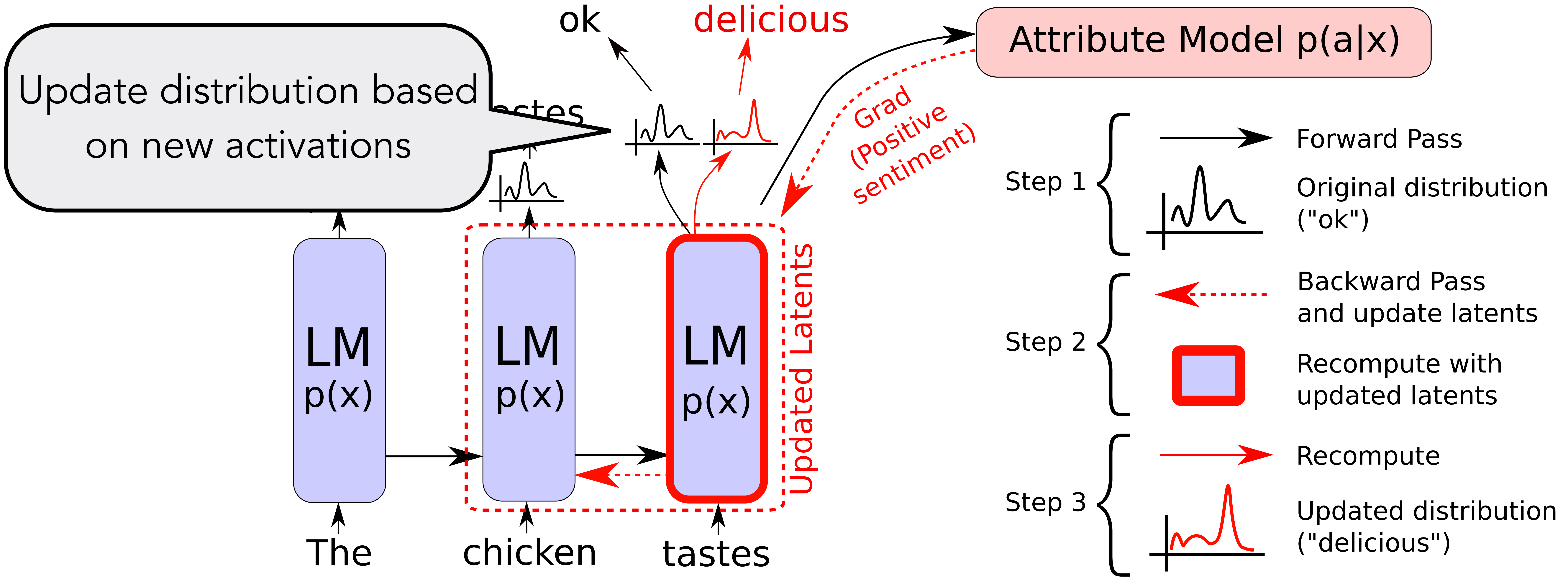
- What if I want to encourage a tough to formalize behavior at inference time?



(Dathathri et al., ICLR 2020)

Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?



(Dathathri et al., ICLR 2020)

(Qin et. al., EMNLP 2020)

Takeaways

- Decoding is a challenging problem in natural language generation
- Human language distribution is noisy and doesn't reflect simple properties (i.e., *maximization)*
- Decoding algorithms can allow us to interject inductive biases that encourage properties of coherent NLG
- A lot more work to be done!

Training Neural Text Generation Models

Antoine Bosselut

Generation Model Basics

1. At each time step, model computes a score o_n for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

$f(\,.\,)$ is your model

2. Compute a probability distribution over these scores (usually softmax)

$P(\,.\,)$ is your distribution over tokens

$$P(y_t = w_n \mid \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

3. Define a loss function to select a token from this distribution

Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

Text Generation Model

Allen

y_0^*

Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

Maximizing Likelihood

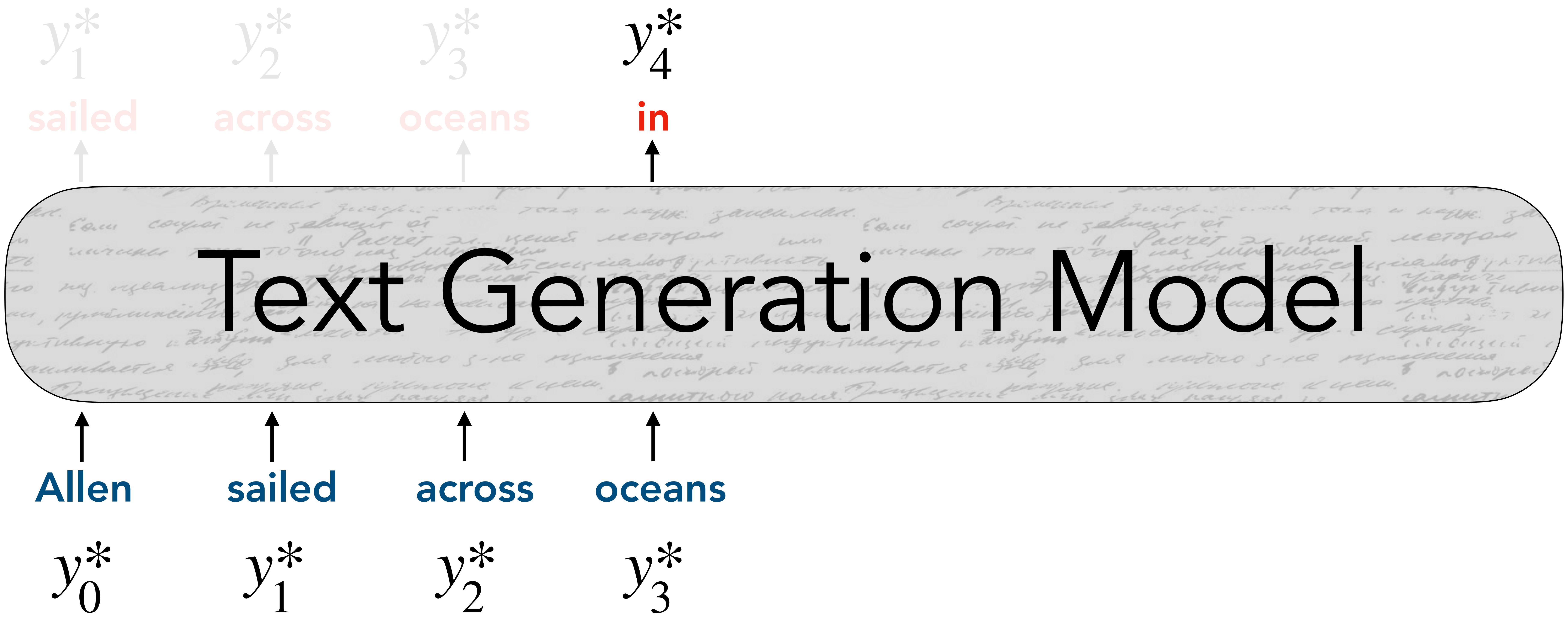
- Trained to generate the next word given a set of preceding words

Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

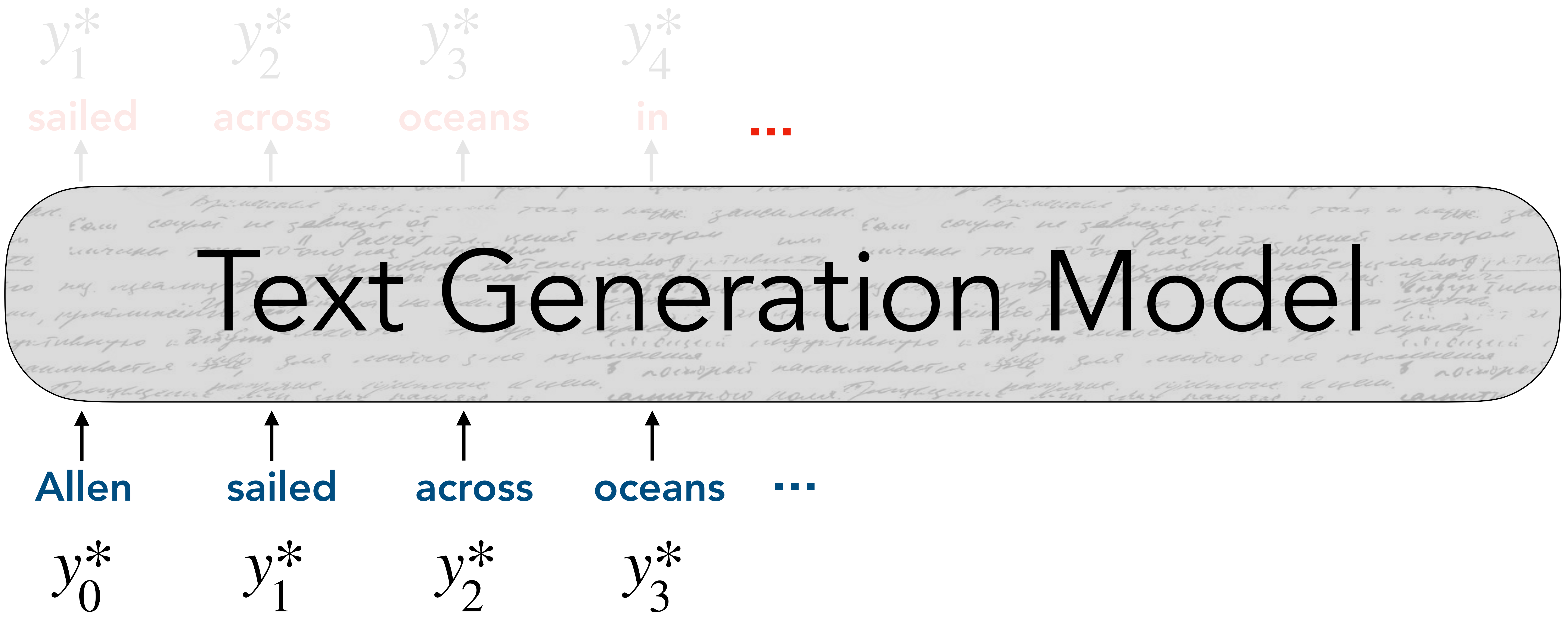
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words



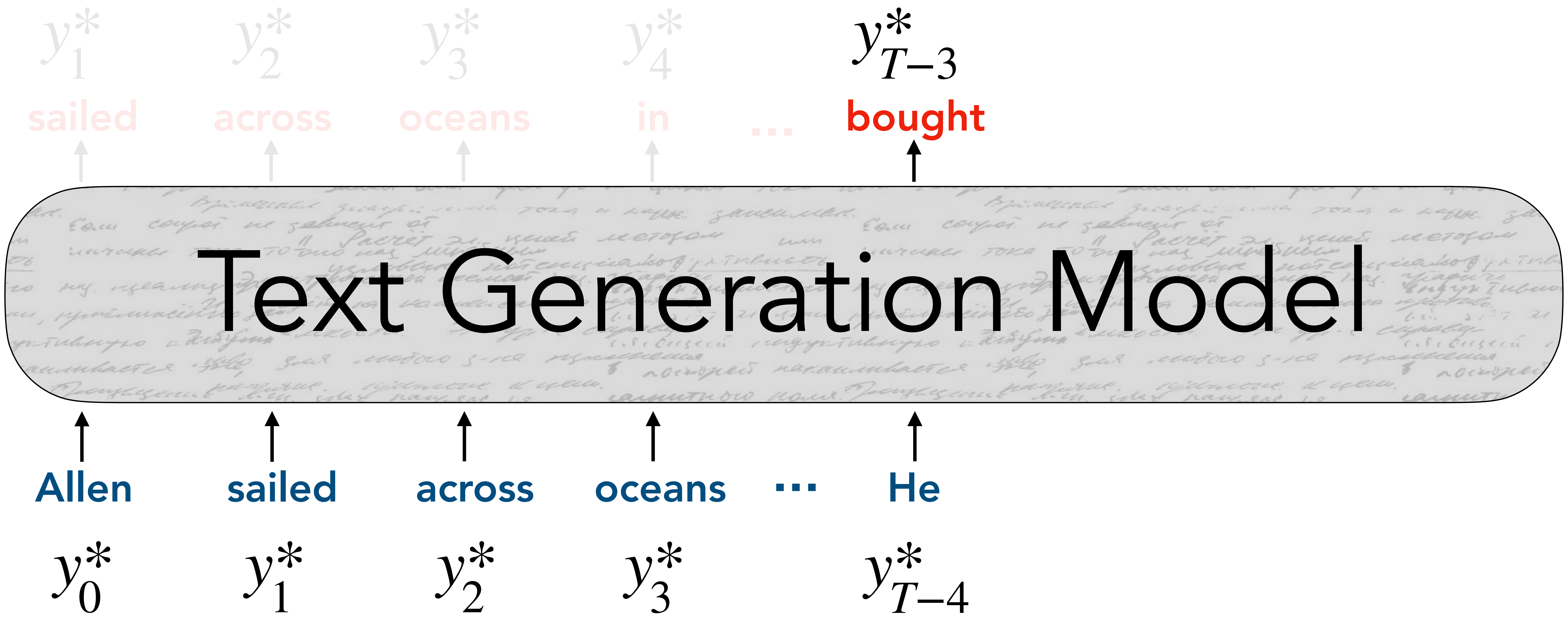
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words



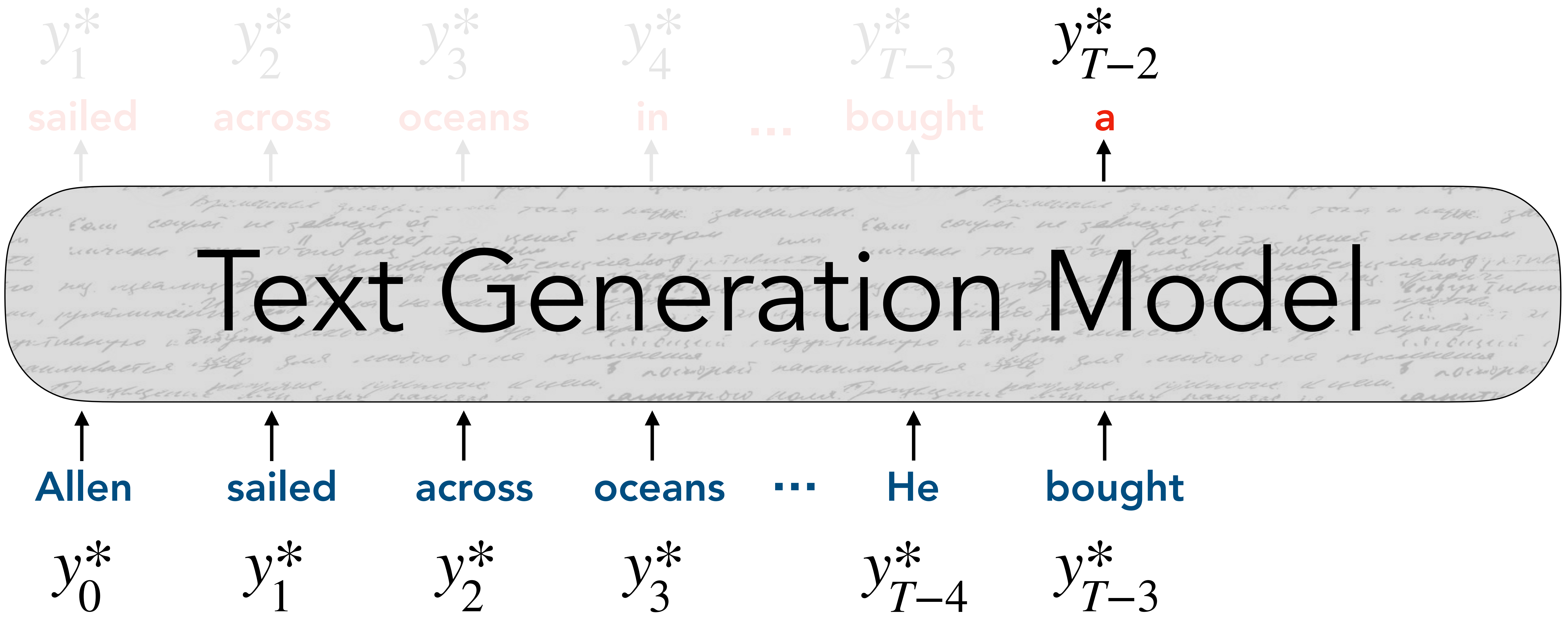
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words



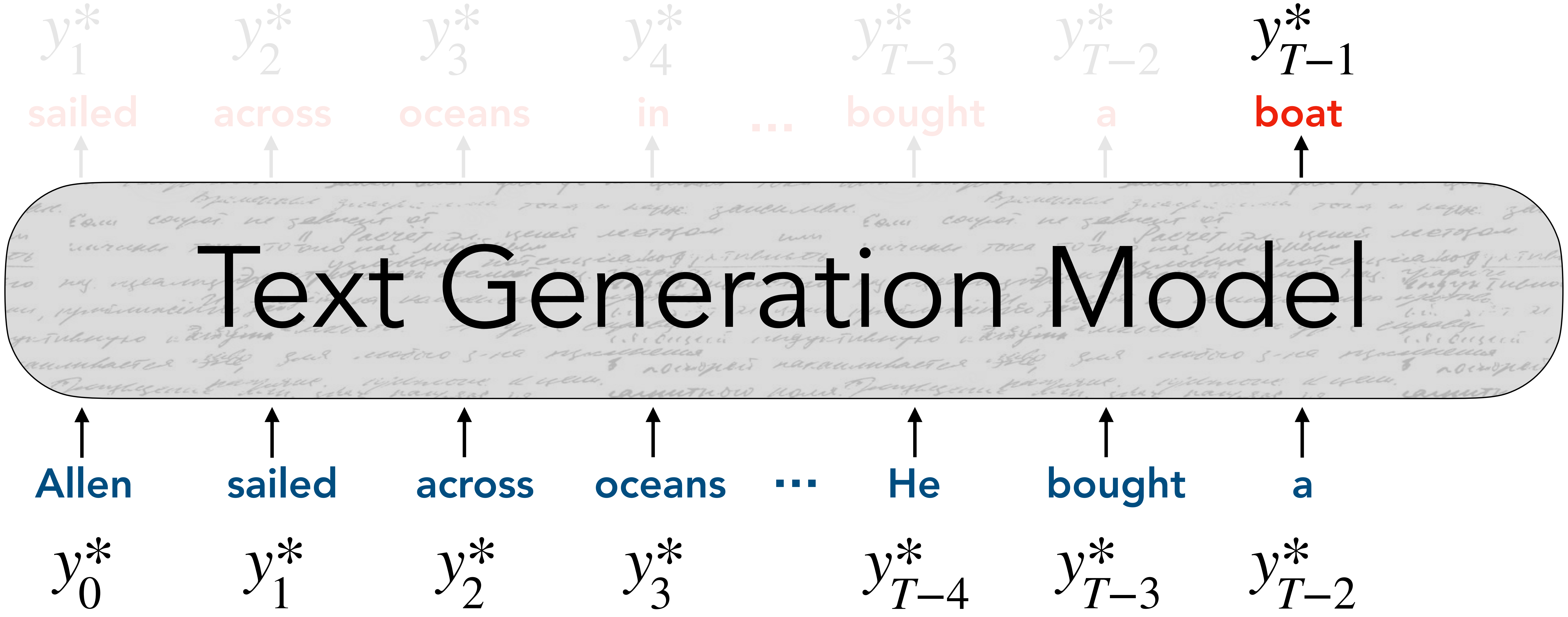
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words



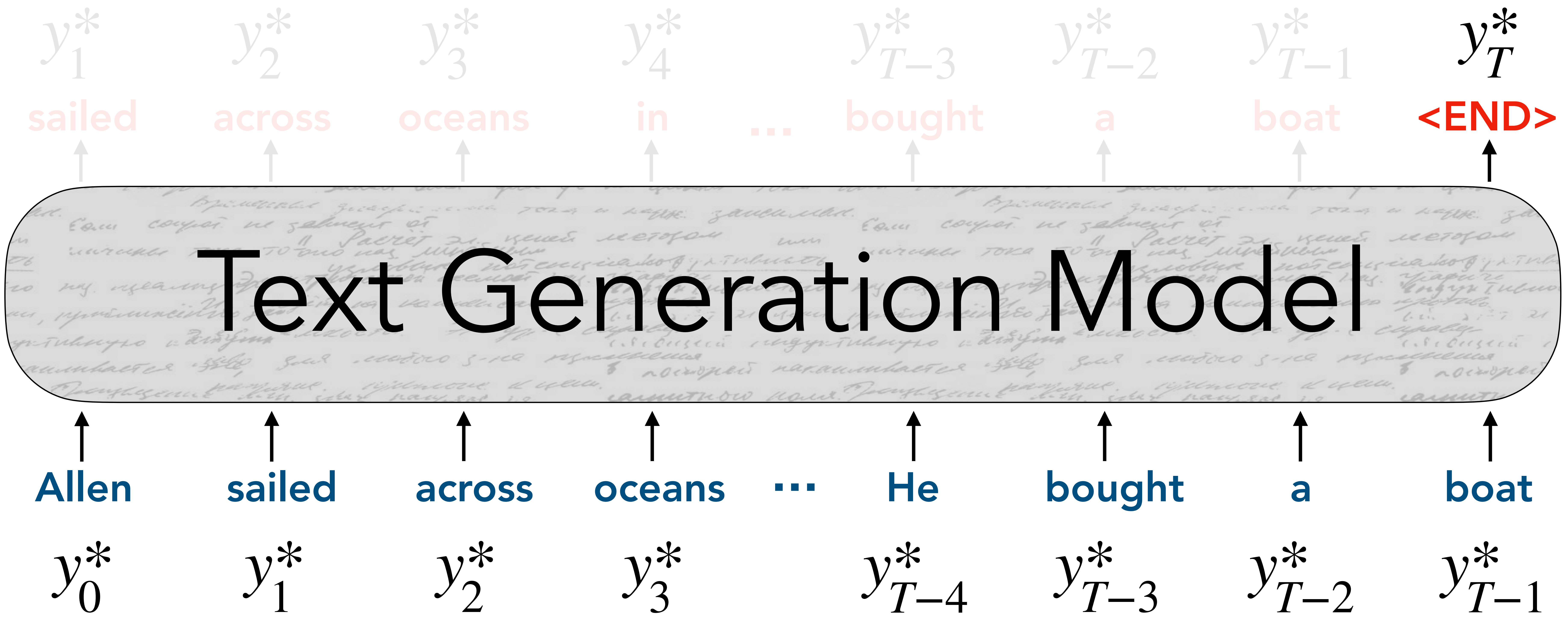
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words



Maximizing Likelihood

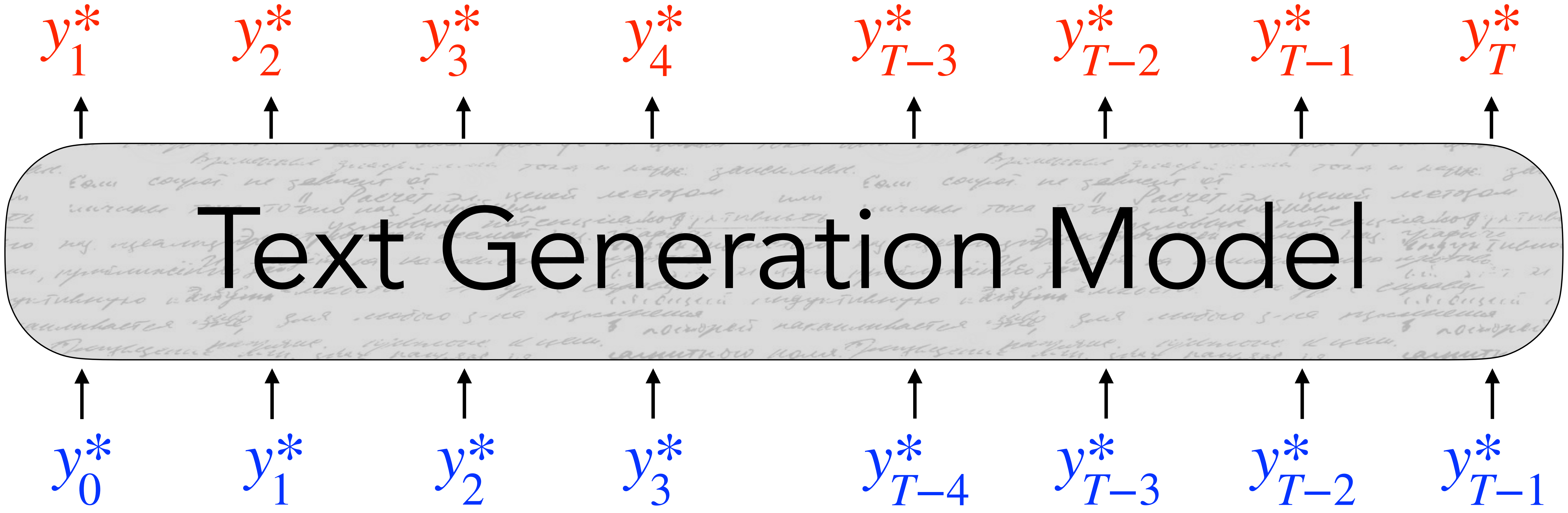
- Trained to generate the next word given a set of preceding words



Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

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



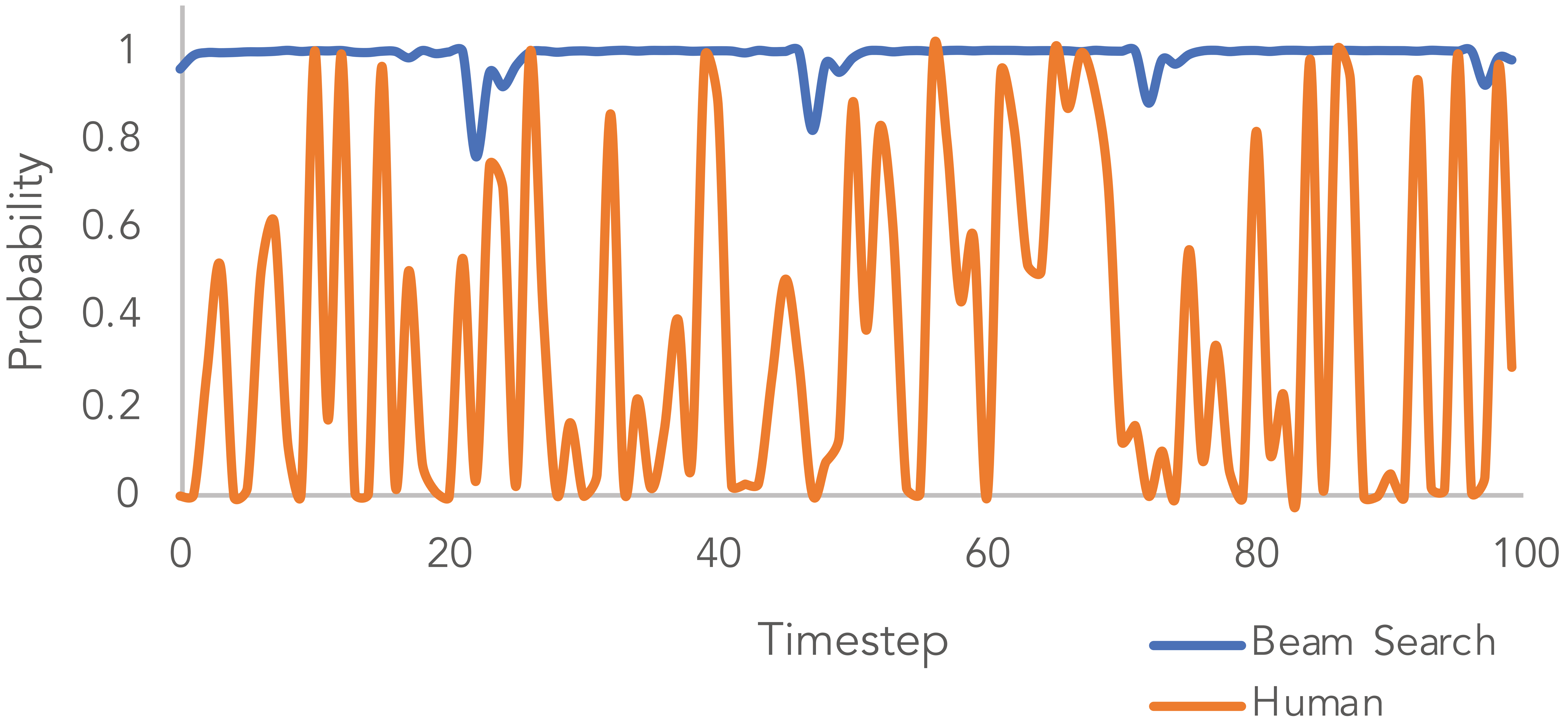
Issue #1: MLE *discourages* diversity

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

Issue #1: MLE *discourages* diversity



Holtzman et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t}) \qquad \mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

Welleck et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t})$$

$$\mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

Typical Negative Loglikehood objective

Welleck et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t})$$

$$\mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

Typical Negative Loglikehood objective

Unlikelihood objective lowers the probability of certain tokens

Welleck et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training
- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t})$$

$$\mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

$$\mathscr{L}^t_{ULE} = \mathscr{L}^t_{MLE} + \alpha \mathscr{L}^t_{UL}$$

Combine them for full unlikelihood training

Welleck et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t})$$

$$\mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

$$\mathscr{L}^t_{ULE} = \mathscr{L}^t_{MLE} + \alpha \mathscr{L}^t_{UL}$$

But wait, what's $\mathscr{C}$?

Welleck et. al., ICLR 2020

Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathscr{C}$, lower their likelihood in context

$$\mathscr{L}^t_{MLE} = -\log P(y^*_t \mid \{y^*\}_{<t})$$

$$\mathscr{L}^t_{UL} = -\sum_{y_{neg} \in \mathscr{C}} \log(1 - P(y_{neg} \mid \{y^*\}_{<t}))$$

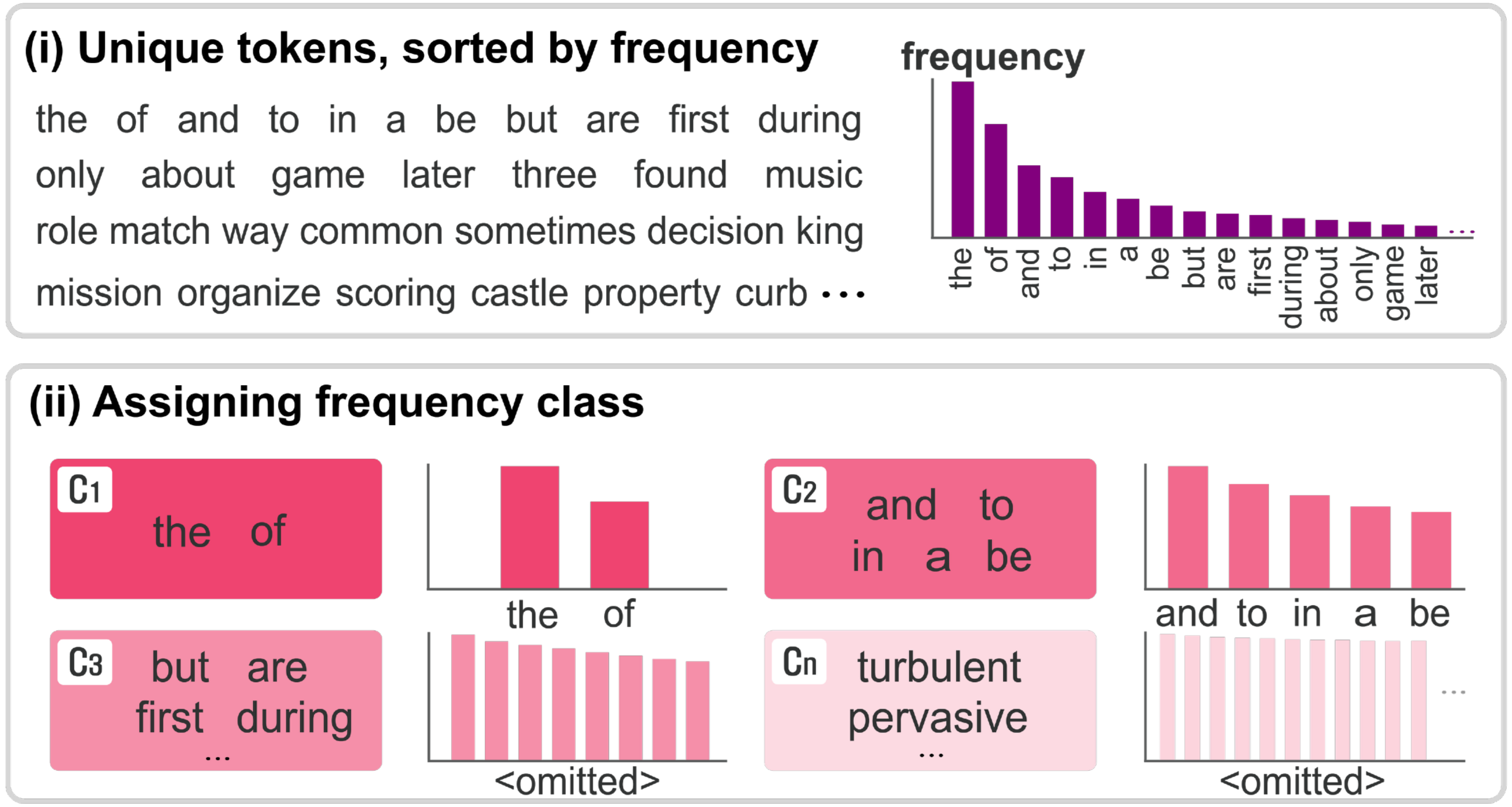
$$\mathscr{L}^t_{ULE} = \mathscr{L}^t_{MLE} + \alpha \mathscr{L}^t_{UL}$$

$$\mathscr{C} = \{y^*\}_{<t}$$

Welleck et. al., ICLR 2020

Alternatives: F^2 Softmax

- Avoid likelihood issues by factorizing the softmax
- Initialize C frequency classes
- Distribute vocabulary into classes so that token frequency uniformly distributed across **and** between classes



Choi et. al., EMNLP 2020

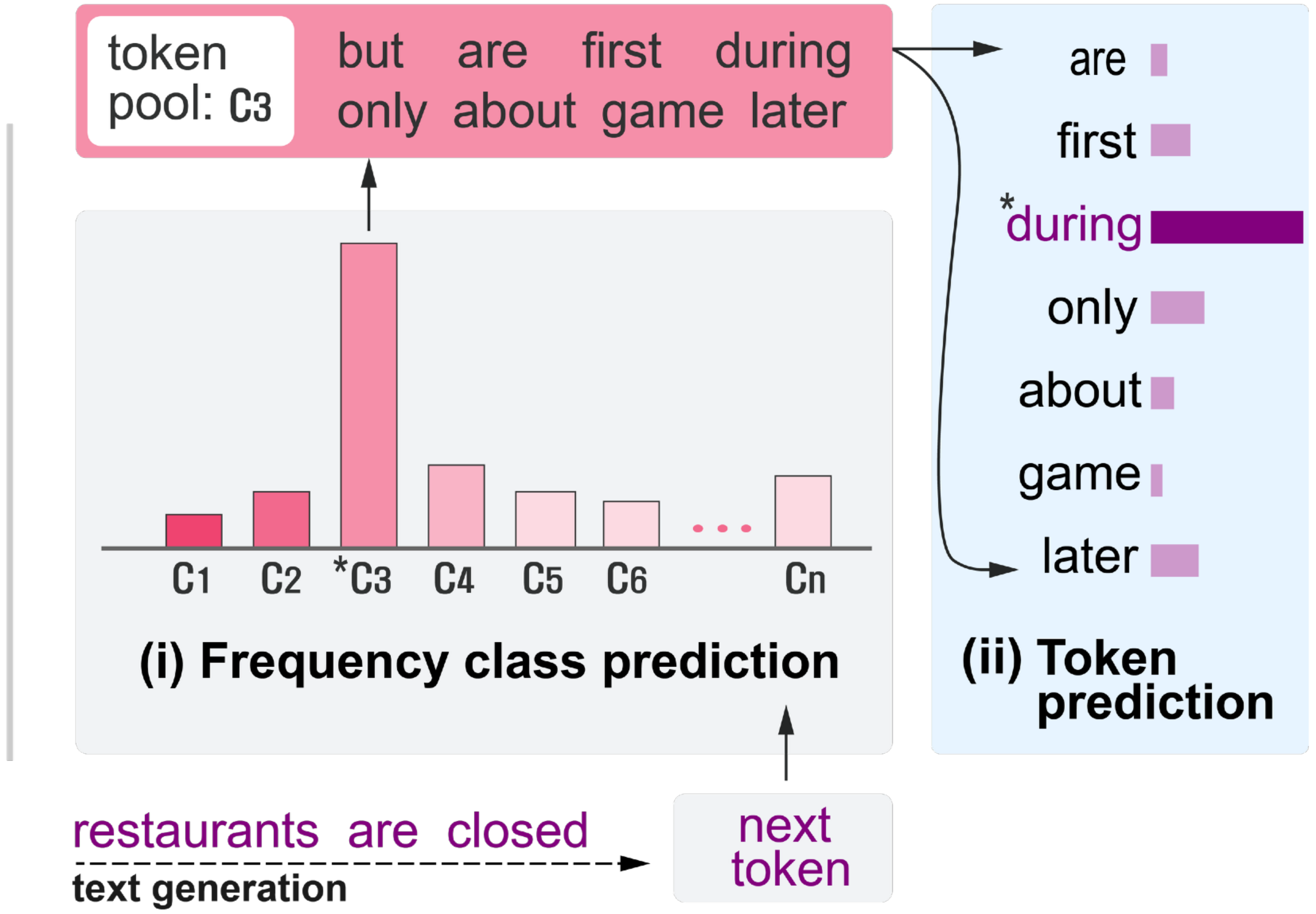
Alternatives: F^2 Softmax

- Learn to select both frequency class and vocabulary token during training

$$P(y_t = w_n \mid \{y\}_{<t}) = \frac{e^{U_n h}}{\sum_{m=1}^{M} e^{U_m h}}$$

$$\Downarrow$$

$$P(y_t = w_n \mid \{y\}_{<t}) = \left(\frac{e^{V_f h}}{\sum_{c=1}^{C} e^{V_c h}}\right)\left(\frac{e^{U_n h}}{\sum_{m=1}^{M_f} e^{U_m h}}\right)$$



Issue #2: Exposure Bias

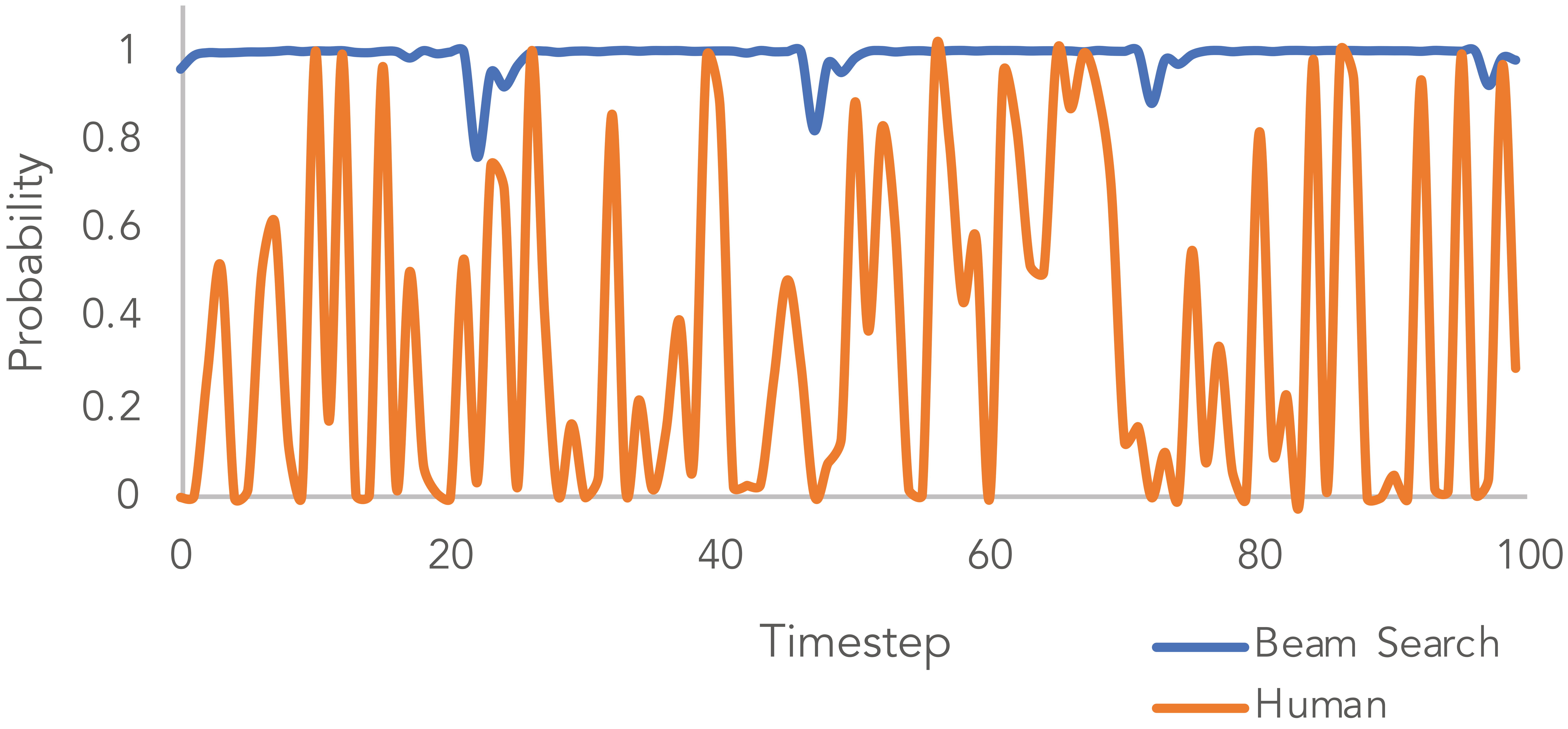
- During training, we condition on gold context tokens that are real human-generated text

$$\mathscr{L}_{MLE} = -\log P(y_t^* \mid \{y^*\}_{<t})$$

- During inference, we decode from distributions conditioned on previously generated tokens

$$\mathscr{L}_{dec} = -\log P(\hat{y}_t \mid \{\hat{y}\}_{<t})$$

Issue #2: Exposure Bias



Holtzman et. al., ICLR 2020

Issue #2: Exposure Bias

- During training, we condition on gold context tokens that are real human-generated text

$$\mathscr{L}_{MLE} = -\log P(y_t^* \mid \{y^*\}_{<t})$$

- During inference, we decode from distributions conditioned on previously generated tokens

$$\mathscr{L}_{dec} = -\log P(\hat{y}_t \mid \{\hat{y}\}_{<t})$$

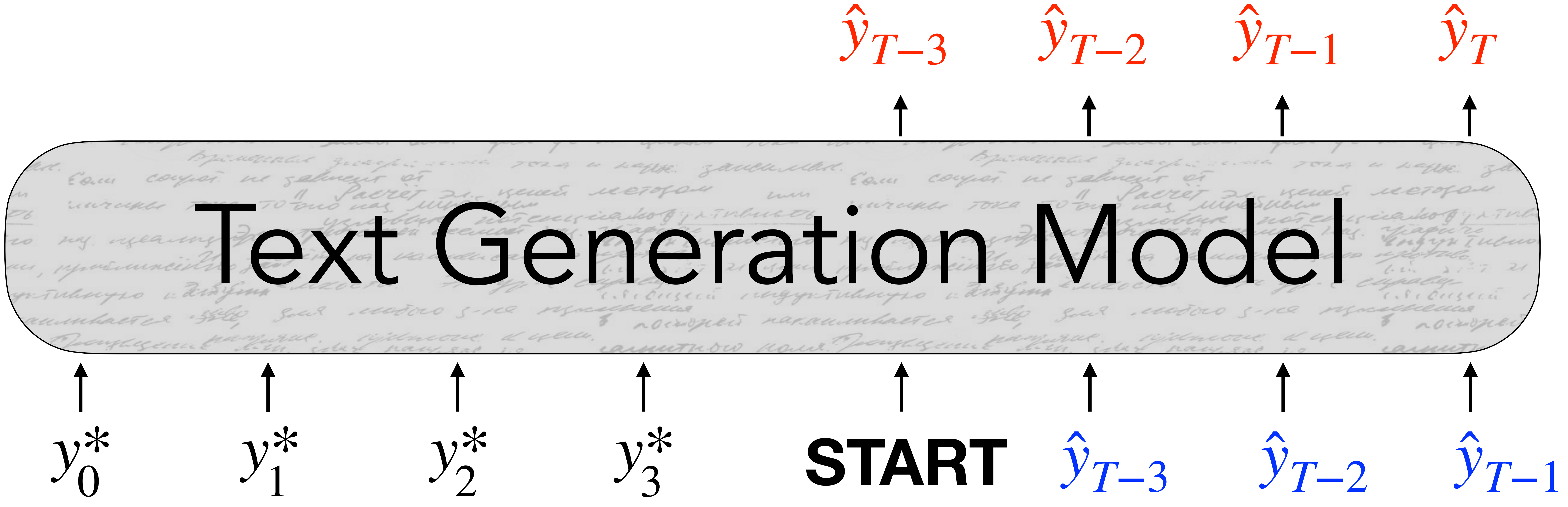
Reinforcement Learning

- Cast a text generation model as a MDP
  - State is denoted by a preceding context
  - Actions are the words you can generate
  - Policy is the text generation model
  - Rewards are provided by an external source

REINFORCE

- Trained to generate the next word given a set of preceding words

$$\mathscr{L}_{RL} = - \sum_{t=T-3}^{T} r(\hat{y}_t) \log P(\hat{y}_t \mid \{\hat{y}\}_{<t}; \{y^*\})$$



Reward Estimation

- How do we define a reward function?
  - BLEU (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)
  - ROUGE (summarization; Paulus et al., ICLR 2018; Celikyilmaz et al., NAACL 2018)
  - CIDEr (image captioning; Rennie et al., CVPR 2017)
  - SPIDEr (image captioning; Liu et al., ICCV 2017)

Reward Estimation

- How do we define a reward function?
  - BLEU (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)
  - ROUGE (summarization; Paulus et al., ICLR 2018; Celikyilmaz et al., NAACL 2020)
  - CIDEr (image captioning; Rennie et al., CVPR 2017)
  - SPIDEr (image captioning; Liu et al., ICCV 2017)

Optimizing for the task vs. Gaming the reward

What behaviors can we tie to rewards?

- Cross-modal consistency (Ren et al., CVPR 2017)
- Simplicity (Zhang and Lapata, EMNLP 2017)
- Temporal consistency (Bosselut et al., NAACL 2018)
- Politeness (Tan et al., TACL 2018)
- Paraphrasing (Li et al., EMNLP 2018)
- Sentiment (Gong et al., NAACL 2019)
- Formality (Gong et al., NAACL 2019)

Implementation Thoughts

- Credit Assignment

$$r(\hat{y}_t) \quad \text{vs.} \quad r(\hat{Y})$$

Implementation Thoughts

- Credit Assignment

$$r(\hat{y}_t) \quad \text{vs.} \quad r(\hat{Y})$$

- Set appropriate baseline

$$\mathcal{L}_{RL} = -\sum (r(\hat{y}_t) - b) \log P(\hat{y}_t \mid \{\hat{y}\}_{<t}; \{y^*\})$$

Implementation Thoughts

- Credit Assignment

$$r(\hat{y}_t) \quad \text{vs.} \quad r(\hat{Y})$$

- Set appropriate baseline

$$\mathcal{L}_{RL} = -\sum (r(\hat{y}_t) - b) \log P(\hat{y}_t \mid \{\hat{y}\}_{<t}; \{y^*\})$$

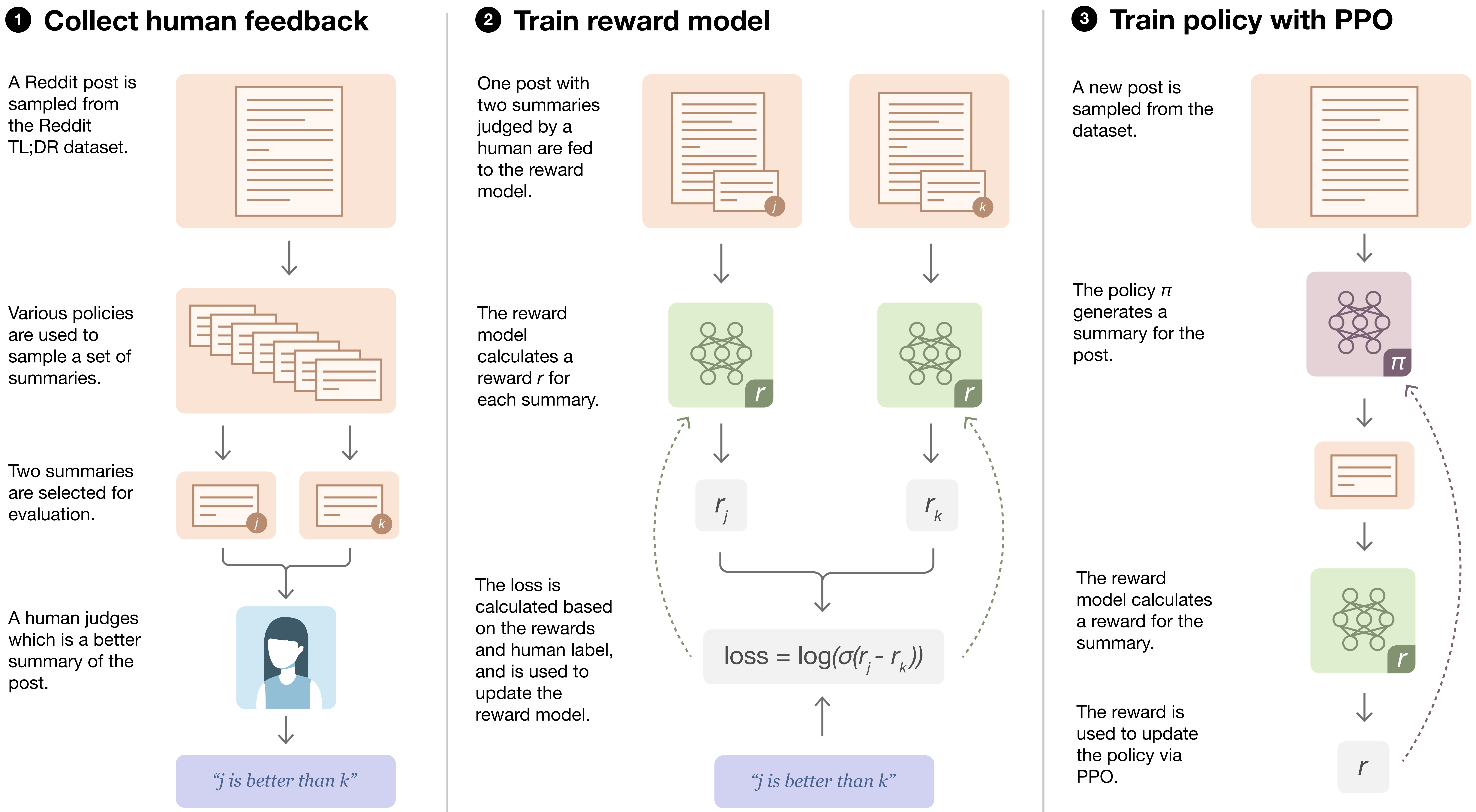
- Mix with MLE

$$\mathcal{L} = \mathcal{L}_{MLE} + \alpha \mathcal{L}_{RL}$$

What if you don’t know what to use as reward?

- **Adversarial Learning!**
- Use an adversarially-learned scoring function to provide rewards
- Still often uses REINFORCE
- Dialogue systems (Li et al., EMNLP 2017), Visual storytelling (Wang et al., ACL 2018)

Human-in-the-loop Learning



(Stiennon et al., NeurIPS 2020)

Takeaways

- Maximum likelihood estimation is still the premier algorithm for training text generation models
- Diversity is an issue with MLE, so new approaches focus on mitigating the effects of common words
- Exposure bias causes text generation models to lose coherence easily, so learning from its own samples is a promising way forward
- Reinforcement learning allows models to learn tough to quantify behaviors
- Much more!

EPFL

Decoding References

[1] Gulcehre et al., On Using Monolingual Corpora in Neural Machine Translation. arXiv 2015

[2] Wu et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arxiv 2016

[3] Venugopalan et al., Improving LSTM-based Video Description with Linguistic Knowledge Mined from Text. EMNLP 2016

[4] Li et al., A Diversity-Promoting Objective Function for Neural Conversation Models. EMNLP 2018

[5] Paulus et al., A Deep Reinforced Model for Abstractive Summarization. ICLR 2018

[6] Celikyilmaz et al., Deep Communicating Agents for Abstractive Summarization. NAACL 2018

[7] Holtzman et al., Learning to Write with Cooperative Discriminators. ACL 2018

[8] Fan et al., Hierarchical Neural Story Generation. ACL 2018

[9] Gabriel et al., Cooperative Generator-Discriminator Networks for Abstractive Summarization with Narrative Flow. arXiv 2019

[10] Dathathri et al., Plug and Play Language Models: A Simple Approach to Controlled Text Generation. ICLR 2020

[11] Holtzman et al., The Curious Case of Neural Text Degeneration. ICLR 2020

[12] Khandelwal et al., Generalization through Memorization: Nearest Neighbor Language Models. ICLR 2020

[13] Qin et al., Back to the Future: Unsupervised Backprop-based Decoding for Counterfactual and Abductive Commonsense Reasoning. EMNLP 2020

[14] Goyal and Durrett, Evaluating Factuality in Generation with Dependency-level Entailment. Findings of EMNLP 2020

[15] Lu et al., NeuroLogic Decoding: (Un)supervised Neural Text Generation with Predicate Logic Constraints. arXiv 2020

Training References

[1] Bengio et. al., Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. NeurIPS 2015

[2] Ranzato et al., Sequence Level Training with Recurrent Neural Networks. ICLR 2016

[3] Wu et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. 2016

[4] Ren et al., Deep Reinforcement Learning-based Image Captioning with Embedding Reward. CVPR 2017

[5] Rennie et al., Self-critical Sequence Training for Image Captioning. CVPR 2017

[6] Liu et al., Improved Image Captioning via Policy Gradient Optimization of SPIDEr. ICCV 2017

[7] Zhang and Lapata, Sentence Simplification with Deep Reinforcement Learning. EMNLP 2017

[8] Li et al., Adversarial Learning for Neural Dialogue Generation. EMNLP 2017

[9] Paulus et al., A Deep Reinforced Model for Abstractive Summarization. ICLR 2018

[10] Celikyilmaz et al., Deep Communicating Agents for Abstractive Summarization. NAACL 2018

[11] Bosselut et al., Discourse-Aware Neural Rewards for Coherent Text Generation. NAACL 2018

[12] Wang et al., No Metrics Are Perfect: Adversarial Reward Learning for Visual Storytelling. ACL 2018

[13] Tan and Bansal, Polite Dialogue Generation Without Parallel Data. TACL 2018

[14] Li et al., Paraphrase Generation with Deep Reinforcement Learning. EMNLP 2018

[15] Gong et al., Reinforcement Learning Based Text Style Transfer without Parallel Training Corpus. NAACL 2019

[16] Holtzman et. al., The Curious Case of Neural Text Degeneration. ICLR 2020

[17] Welleck et. al., Neural Text Generation With Unlikelihood Training. ICLR 2020

[18] Stiennon et al., Learning to summarize from human feedback. NeurIPS 2020