Natural Language Processing

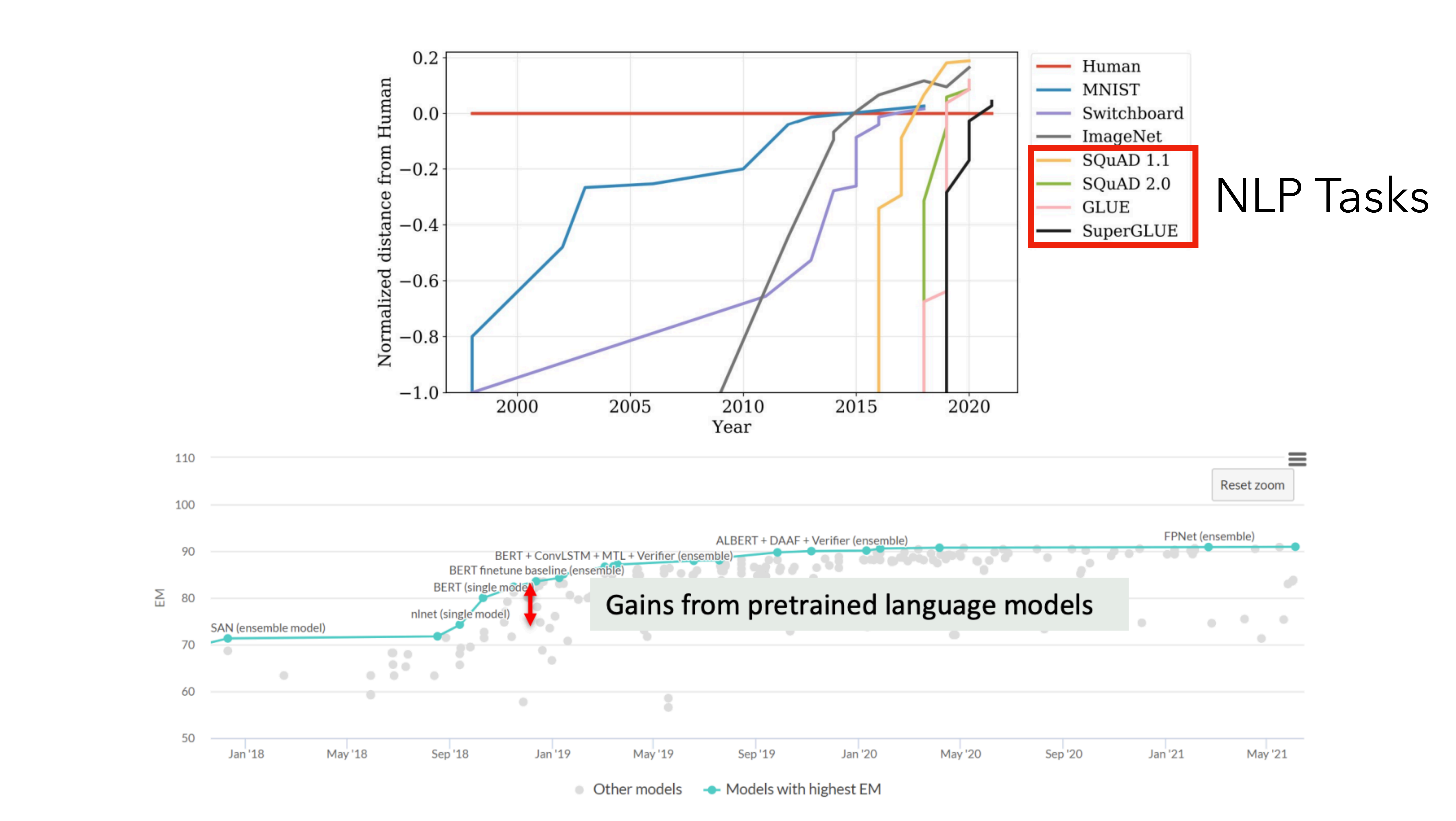
CSE 447 / 547 M

Pre-training + NLG

Lecturer: Kabir Ahuja

Slides adapted from Liwei Jiang, Jaehun Jung, John Hewitt, Anna Goldie, Antoine Bosselut, Xiang Lisa Li, Chris Manning

The Pre-training Revolution



Pre-training has had a major, tangible impact on how well NLP systems work

Slide from Chris Manning, Lecture 9: Pre-training, CS224n Spring 2024

Lecture Outline

1. Motivating Pre-training, aka Self-supervised Learning
2. Pre-training Architectures and Training Objectives
    1. Encoders
    2. Encoder-Decoders
    3. Decoder
3. Open Ended Text Generation Using Language Models

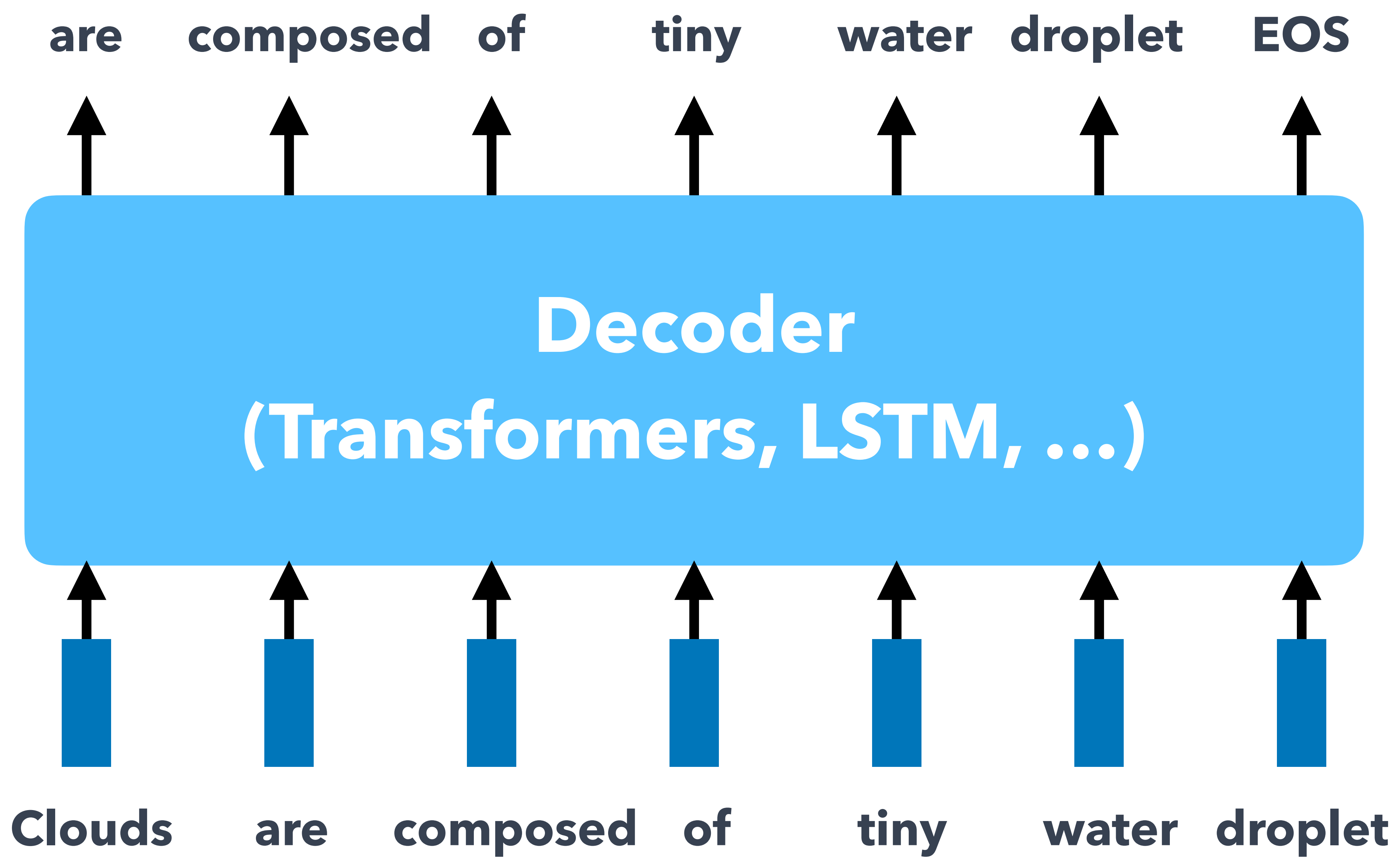
Lecture Outline

1. Motivating Pre-training, aka Self-supervised Learning
2. Pre-training Architectures and Training Objectives
    1. Encoders
    2. Encoder-Decoders
    3. Decoder
3. Open Ended Text Generation Using Language Models

Self-supervised Pre-training for Learning Underlying Patterns, Structures, and Semantic Knowledge

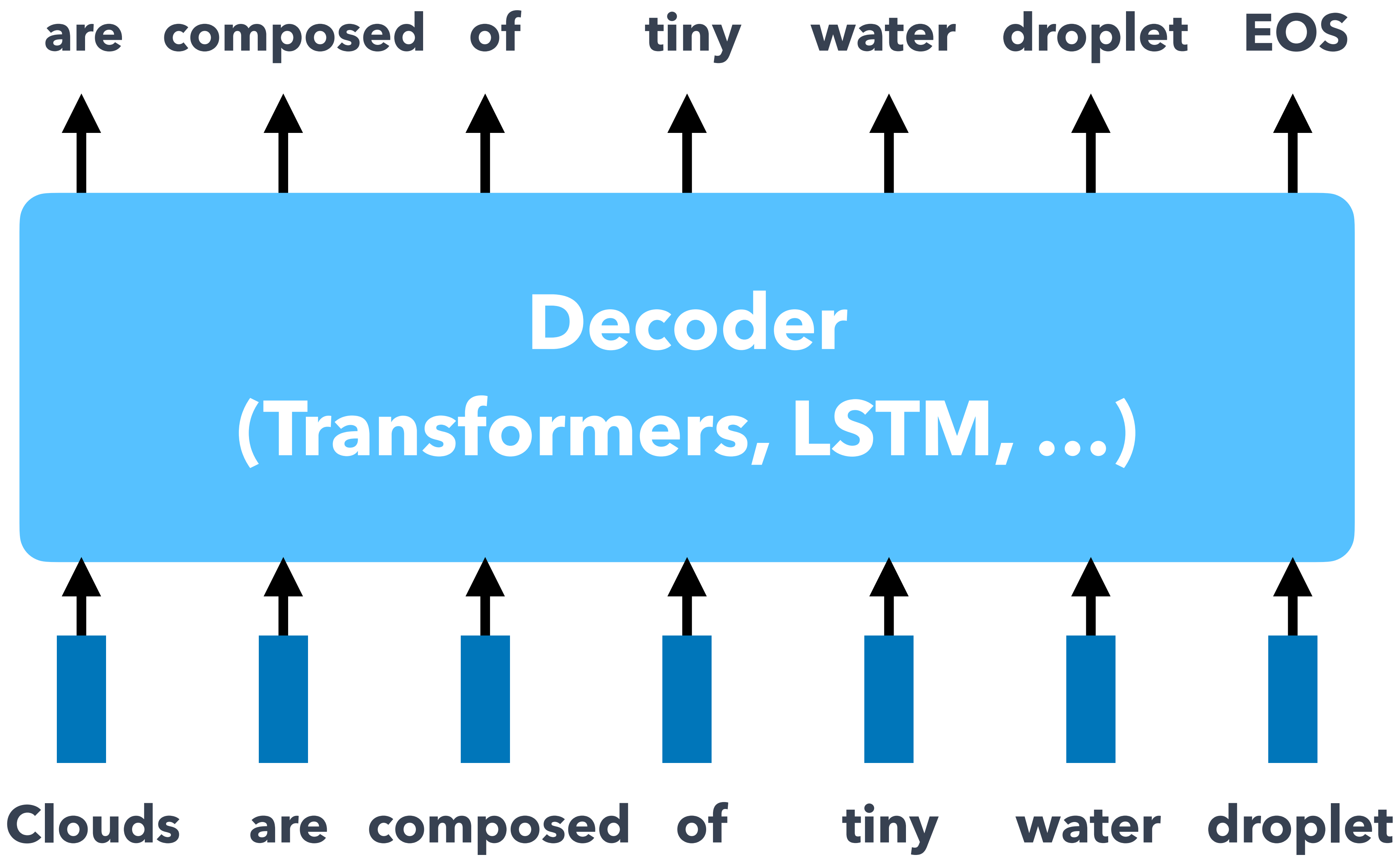
- Pre-training through **language modeling** [Dai and Le, 2015]
  - Model $P_\theta(w_t | w_{1:t-1})$, the probability distribution of the next word given previous contexts.
  - **There's lots of (English) data for this!** E.g., books, websites.
  - **Self-supervised** training of a neural network to perform the language modeling task with massive raw text data.
  - Save the network parameters to reuse later.

Why is this called self-supervised?
The labels come from the input data itself!



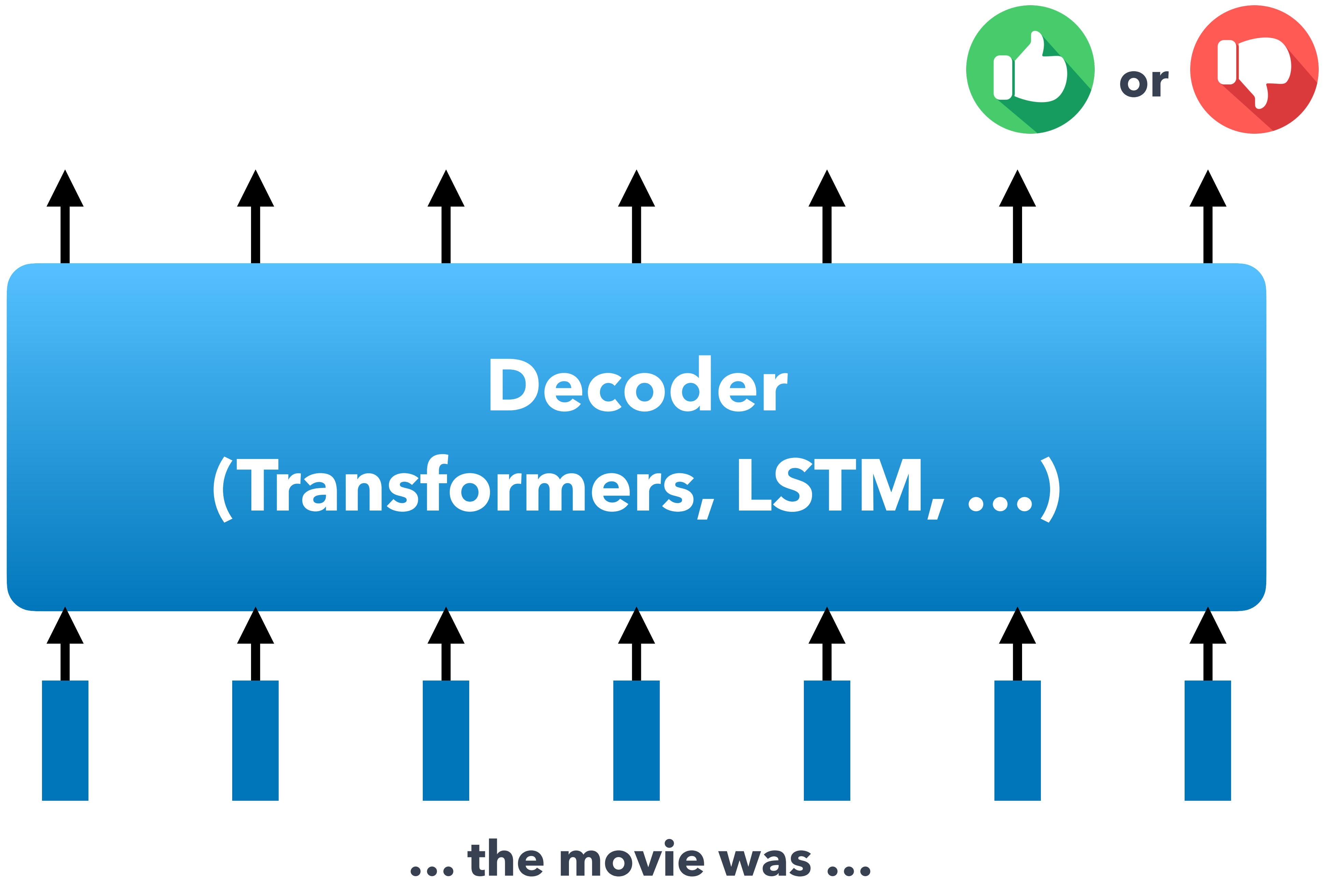
Supervised Fine-tuning for Specific Tasks

Step 1:
Pre-training



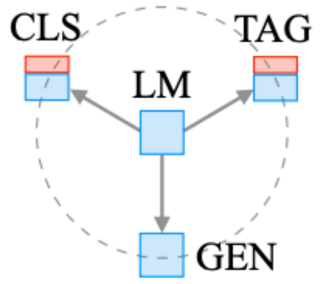
Abundant data; learn general language

Step 2:
Fine-tuning



Limited data; adapt to the task

Remember this is paradigm 3 from before

c. Pre-train, Fine-tune	Objective (e.g. masked language modeling, next sentence prediction)	

Lecture Outline

1. Motivating Pre-training, aka Self-supervised Learning
2. Pre-training Architectures and Training Objectives
   1. Encoders
   2. Encoder-Decoders
   3. Decoder
3. Open Ended Text Generation Using Language Models

3 Pre-training Paradigms/Architectures

Encoder

- E.g., BERT, RoBERTa, DeBERTa, …
- **Autoencoder** model
- **Masked** language modeling

Encoder-Decoder

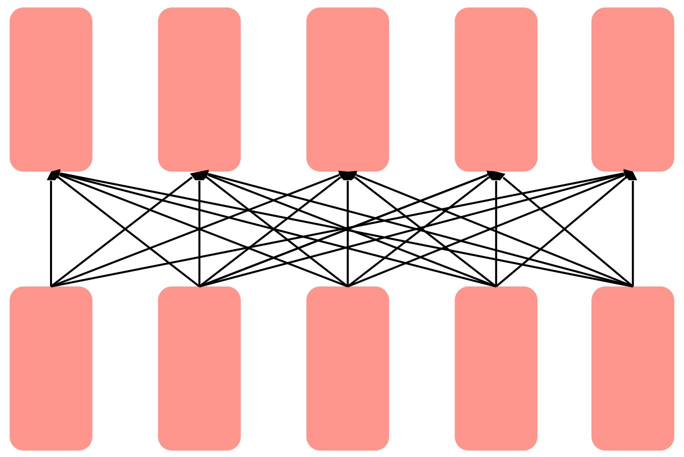
- E.g., T5, BART, …
- **seq2seq** model

Decoder

- E.g., GPT, GPT2, GPT3, …
- **Autoregressive** model
- **Left-to-right** language modeling

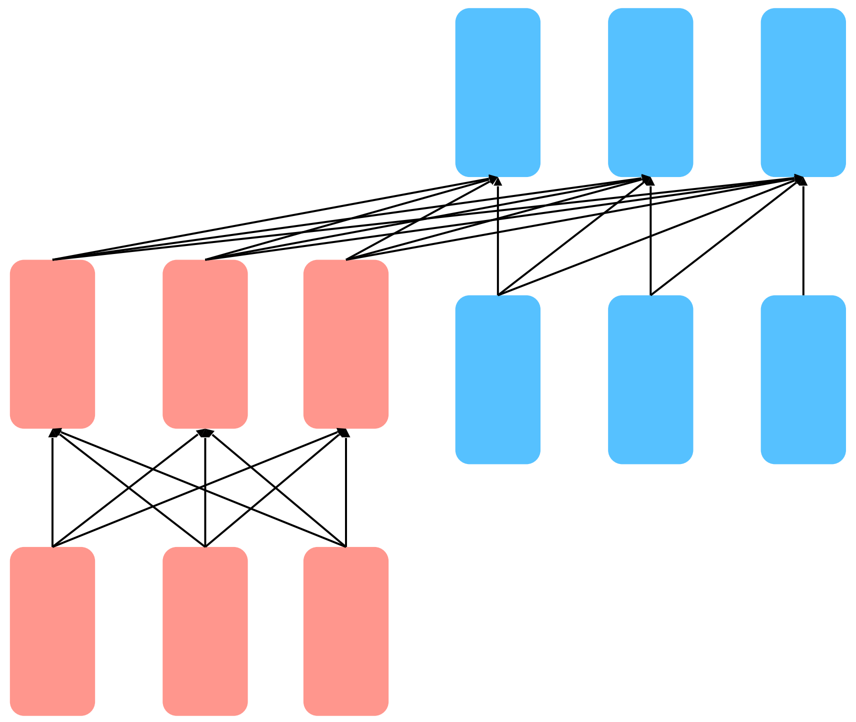
3 Pre-training Paradigms/Architectures

Encoder



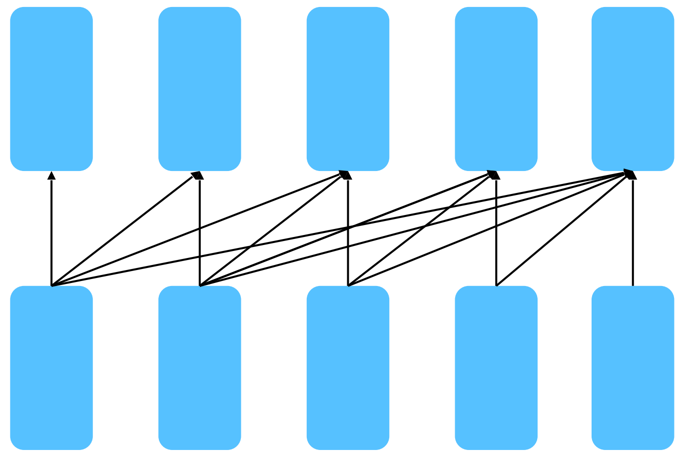
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

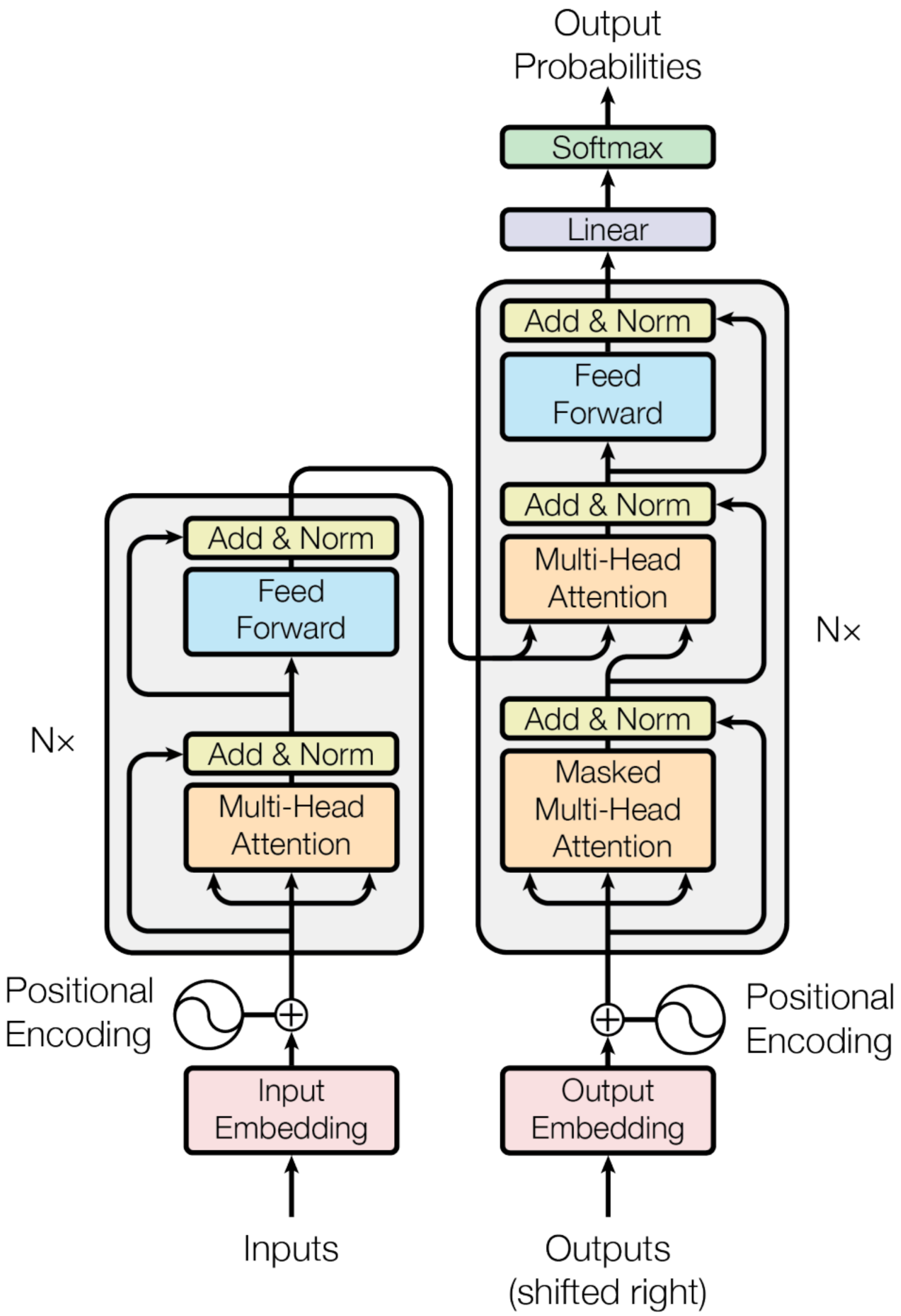
Decoder



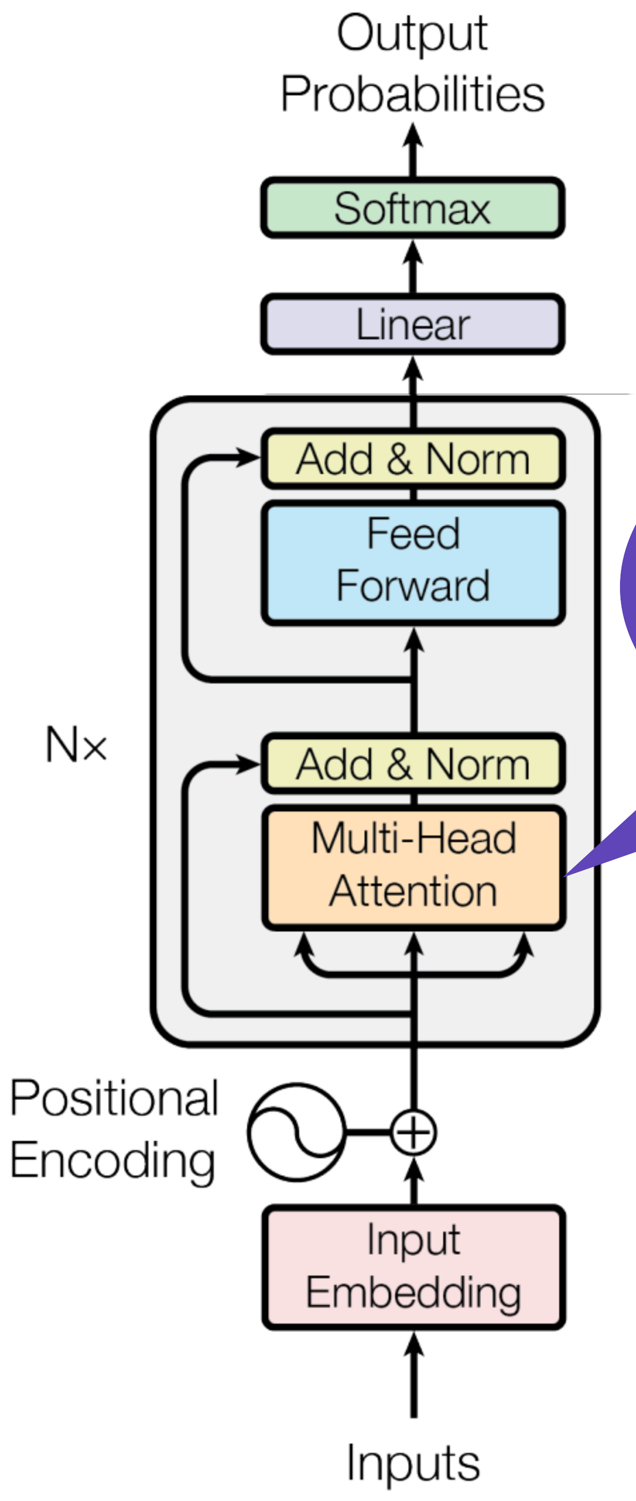
- Language modeling; can only condition on the past context

Encoder: Architecture

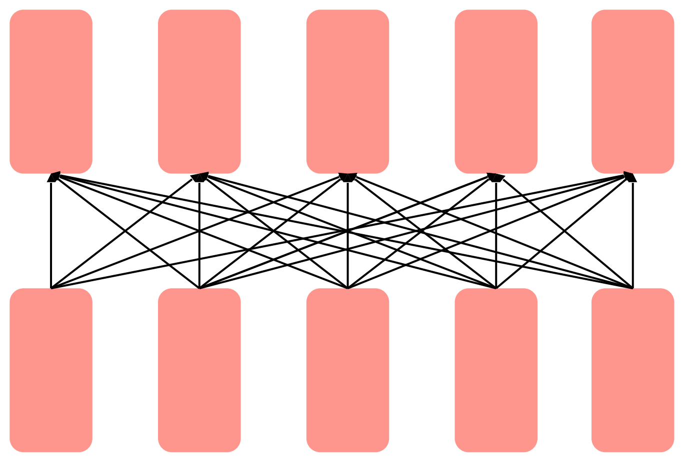
Full-Transformer Architecture (Encoder-Decoder)



Encoder-Only Transformer Architecture



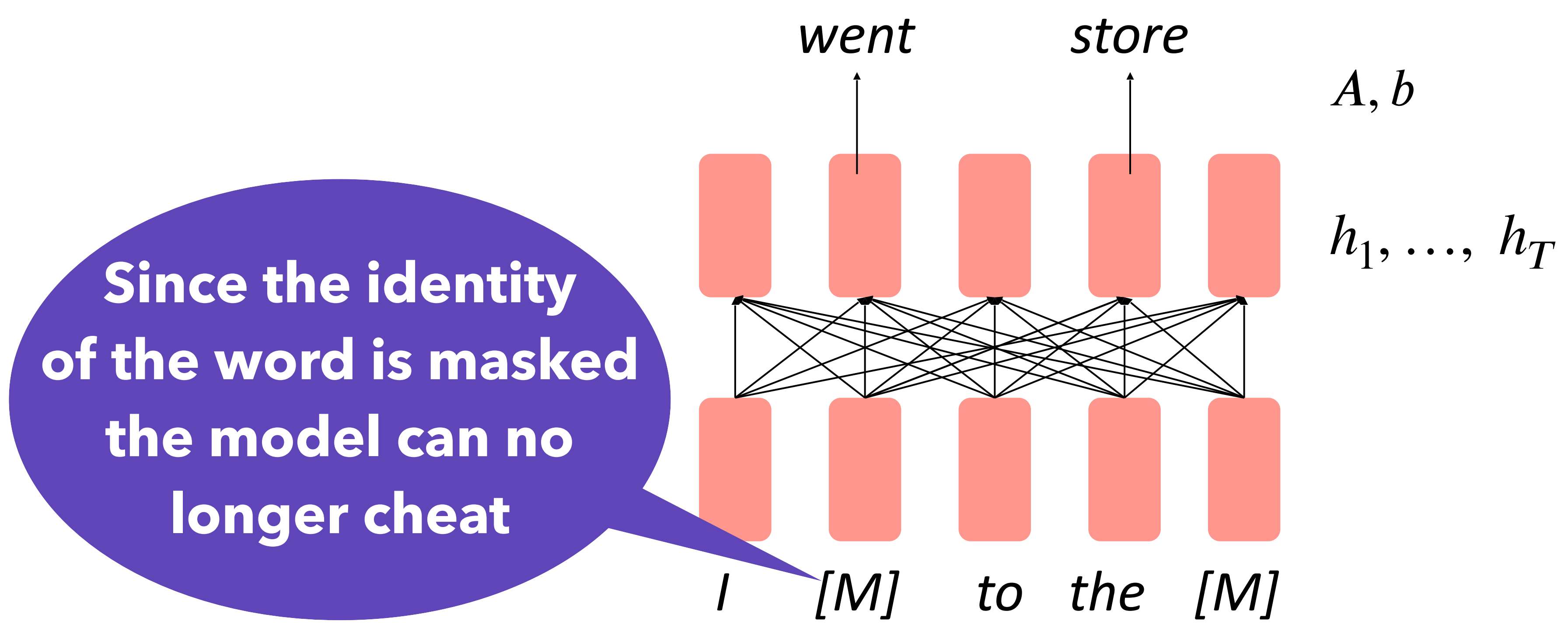
Remember this is bidirectional!



Encoder: Training Objective

[Devlin et al., 2018]

- How to encode information from both **bidirectional** contexts?
- General Idea: **text reconstruction!**
  - Your time is limited so don't waste it living someone else's life. Don't be trapped by dogma which is living with the results of other people's thinking. – Steve Jobs



$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$

$$y_i \sim Aw_i + b$$

Only add loss terms from the masked tokens. If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x \mid \tilde{x})$. Called **Masked Language model (MLM)**.

Encoder: BERT

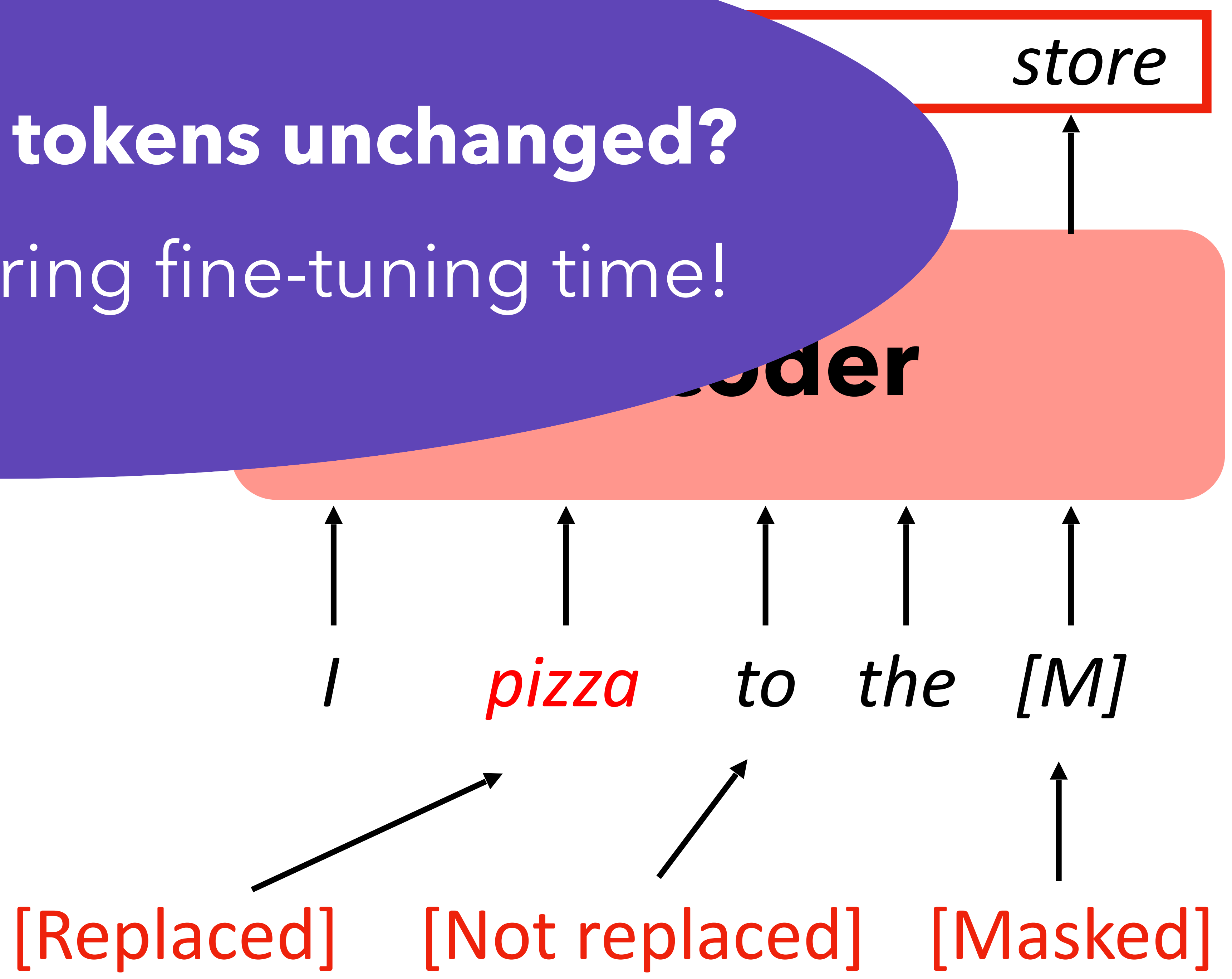
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

- **2 Pre-training Objectives:**
  - **Masked LM: Choose a random 15% of tok… predict.**
    - For each chosen token:
      - Replace it with **[MASK]**
      - Replace it with a **random tok**…
      - Leave it **unchanged** 10% of the time (but still predict it!).
  - **Next Sentence Prediction (NSP)**
    - 50% of the time two adjacent sentences are in the correct order.
    - **This actually hurts model learning based on later work!**

WHY keeping some tokens unchanged?

There's no [MASK] during fine-tuning time!



Encoder: BERT

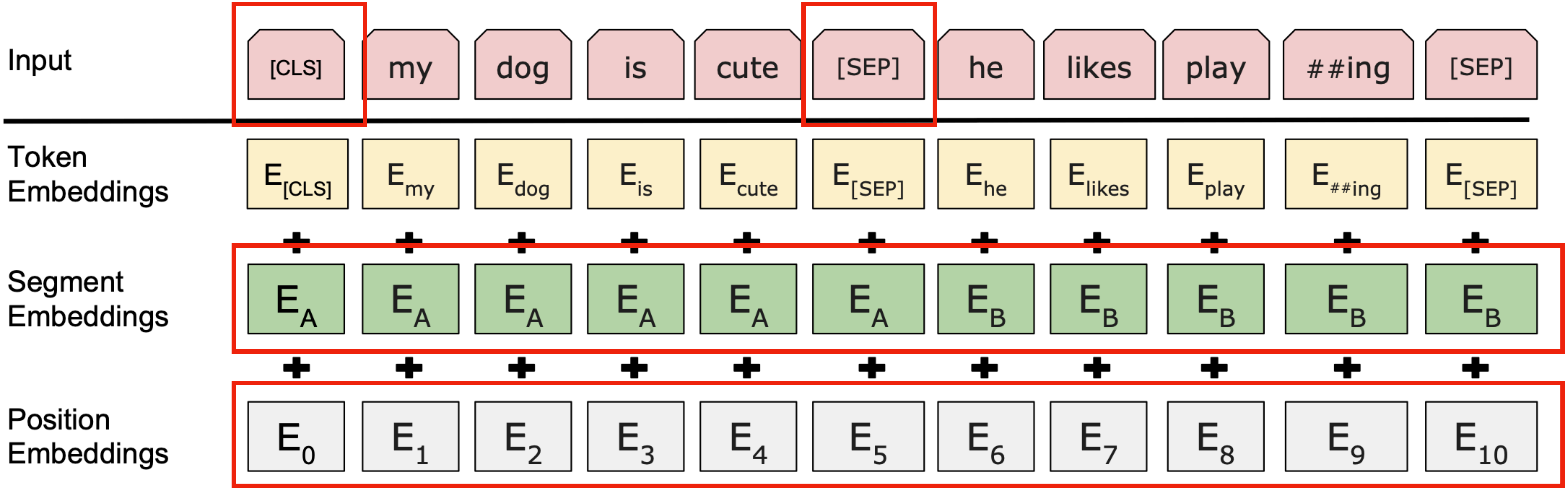
Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

Special token added to the beginning of each input sequence

Special token to separate sentence A/B

Final embedding is the sum of all three!



Learned embedding to every token indicating whether it belongs to sentence A or sentence B

Position of the token in the entire sequence

Encoder: BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

- **SOTA at the time on a wide range of tasks after fine-tuning!**

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI:** natural language inference over question answering data
- **SST-2:** sentiment analysis
- **CoLA:** corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B:** semantic textual similarity
- **MRPC:** microsoft paraphrase corpus
- **RTE:** a small natural language inference corpus

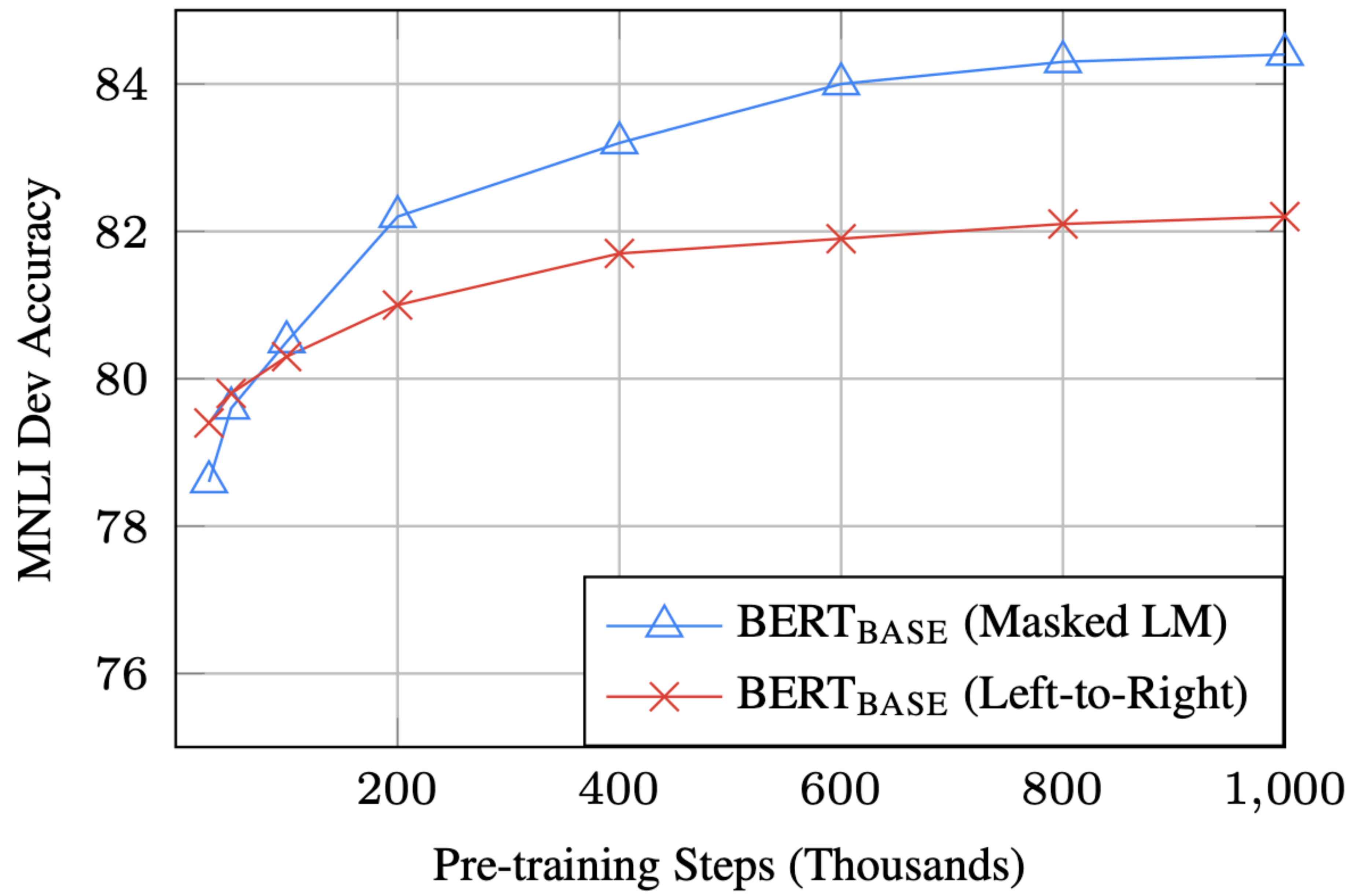
Encoder: BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

[Devlin et al., 2018]

SWAG (Commonsense inference task)

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0



- **Two Sizes of Models**
  - **Base:** 110M, 4 Cloud TPUs, 4 days
  - **Large:** 340M, 16 Cloud TPUs, 4 days
  - Both models can be fine-tuned with single GPU
  - The larger the better!
- MLM converges slower than Left-to-Right at the beginning, but out-performers it eventually

Encoder: RoBERTa

[Liu et al., 2019]

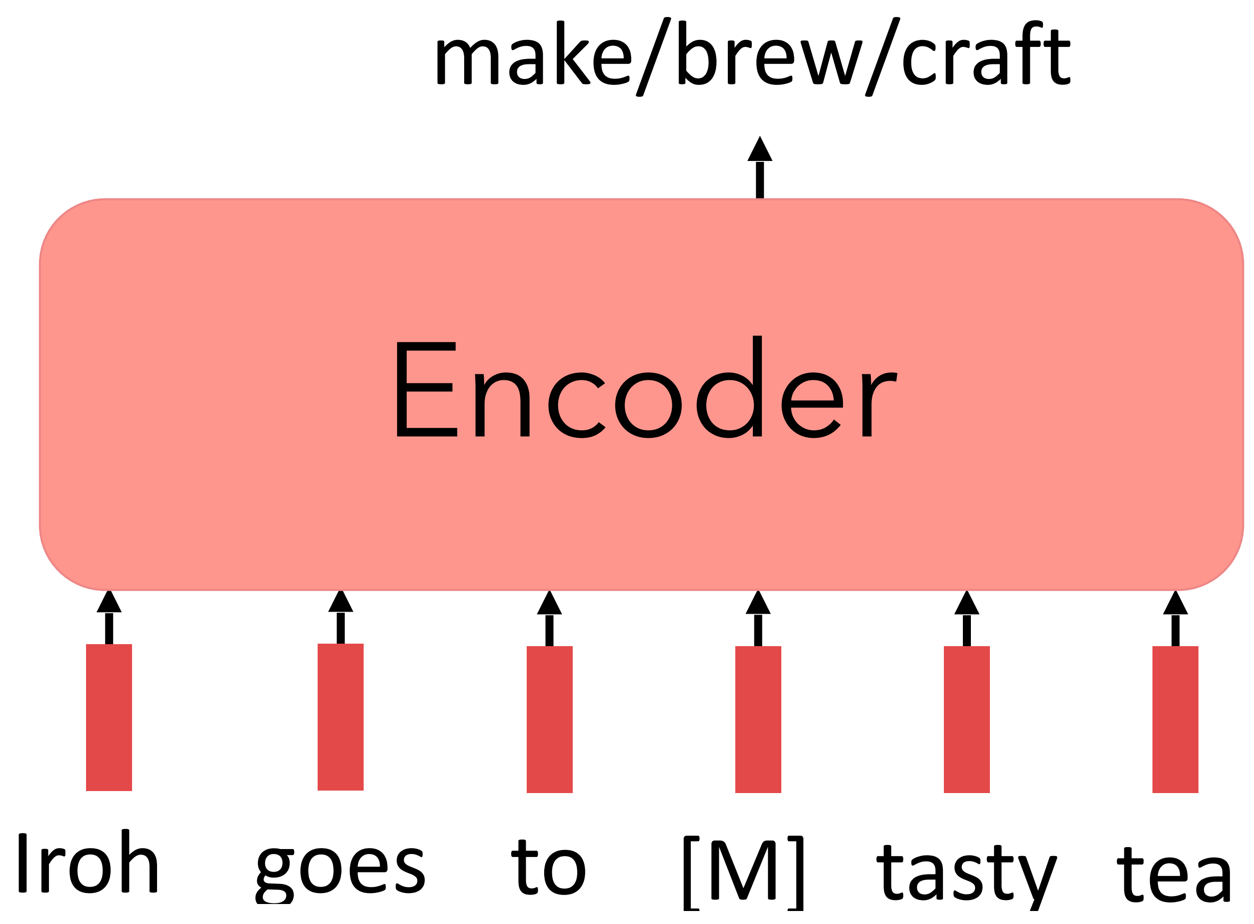
- **Original BERT is significantly undertrained!**
- More data (16G => 160G)
- Pre-train for longer
- Bigger batches
- Removing the next sentence prediction (NSP) objective
- Training on longer sequences
- Dynamic masking, randomly masking out different tokens
- A larger byte-level BPE vocabulary containing 50K sub-word units

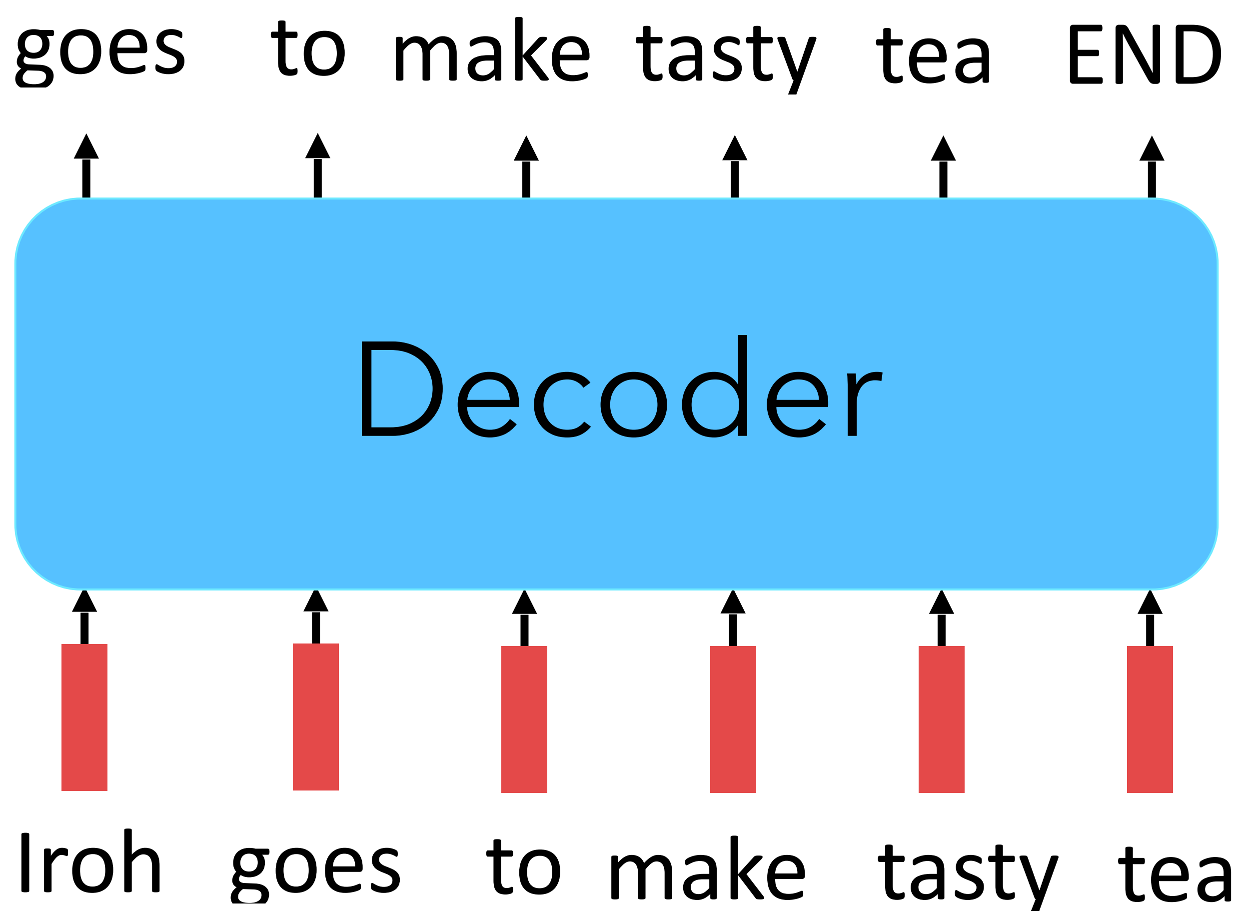
All around better than BERT!

Encoder: Pros & Cons

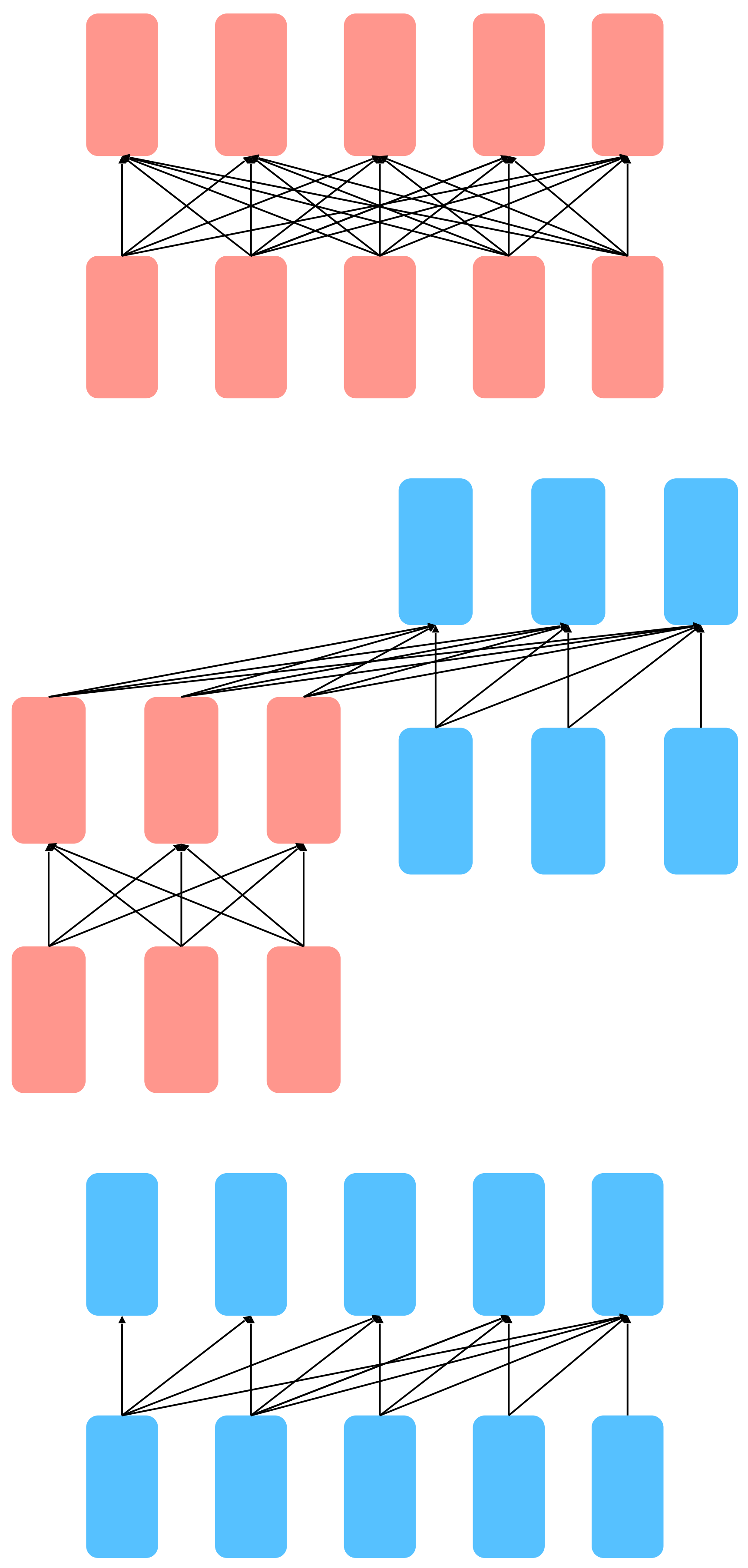
- Consider both left and right context
- Capture intricate contextual relationships

- Not good at generating open-text from left-to-right, one token at a time





3 Pre-training Paradigms/Architectures



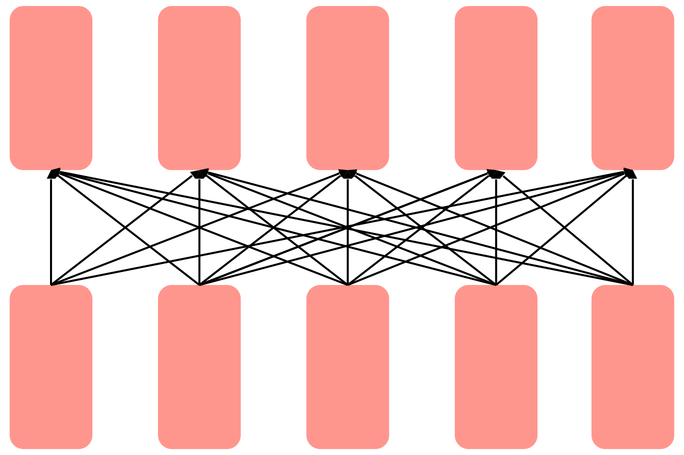
Encoder
- Bidirectional; can condition on the future context

Encoder-Decoder
- Map two sequences of different length together

Decoder
- Language modeling; can only condition on the past context

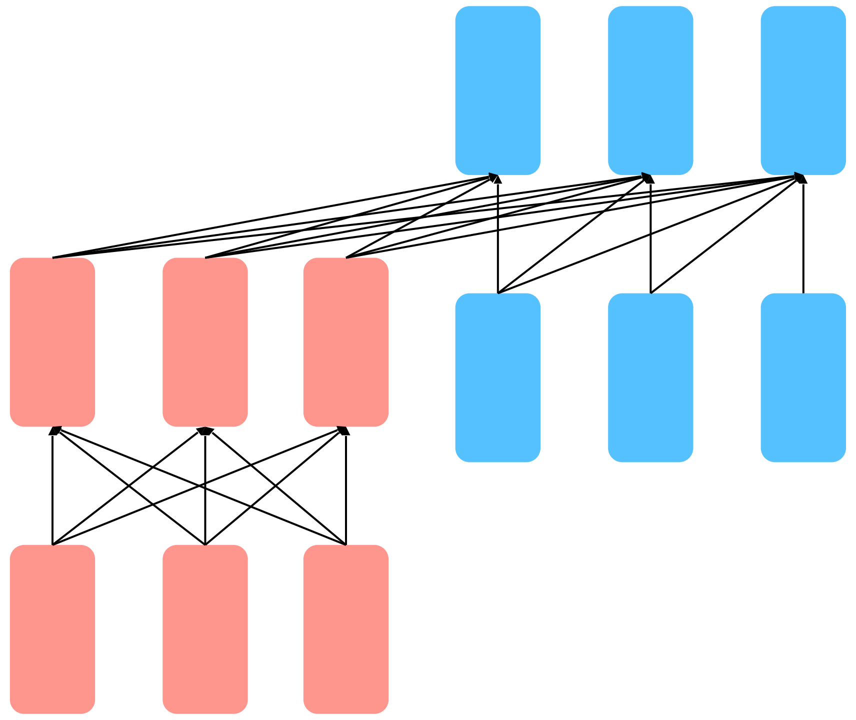
3 Pre-training Paradigms/Architectures

Encoder



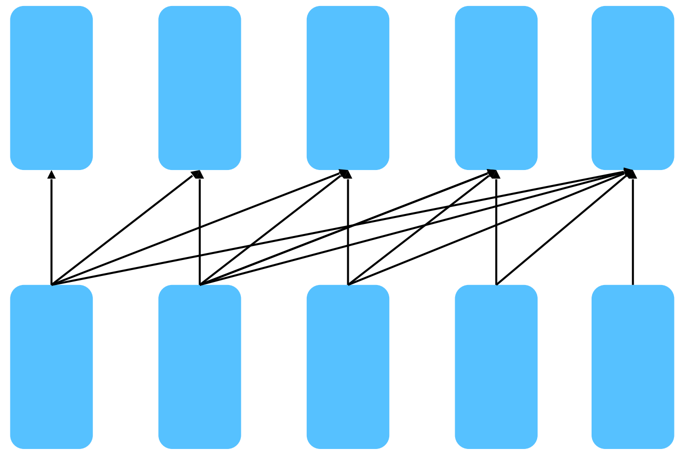
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

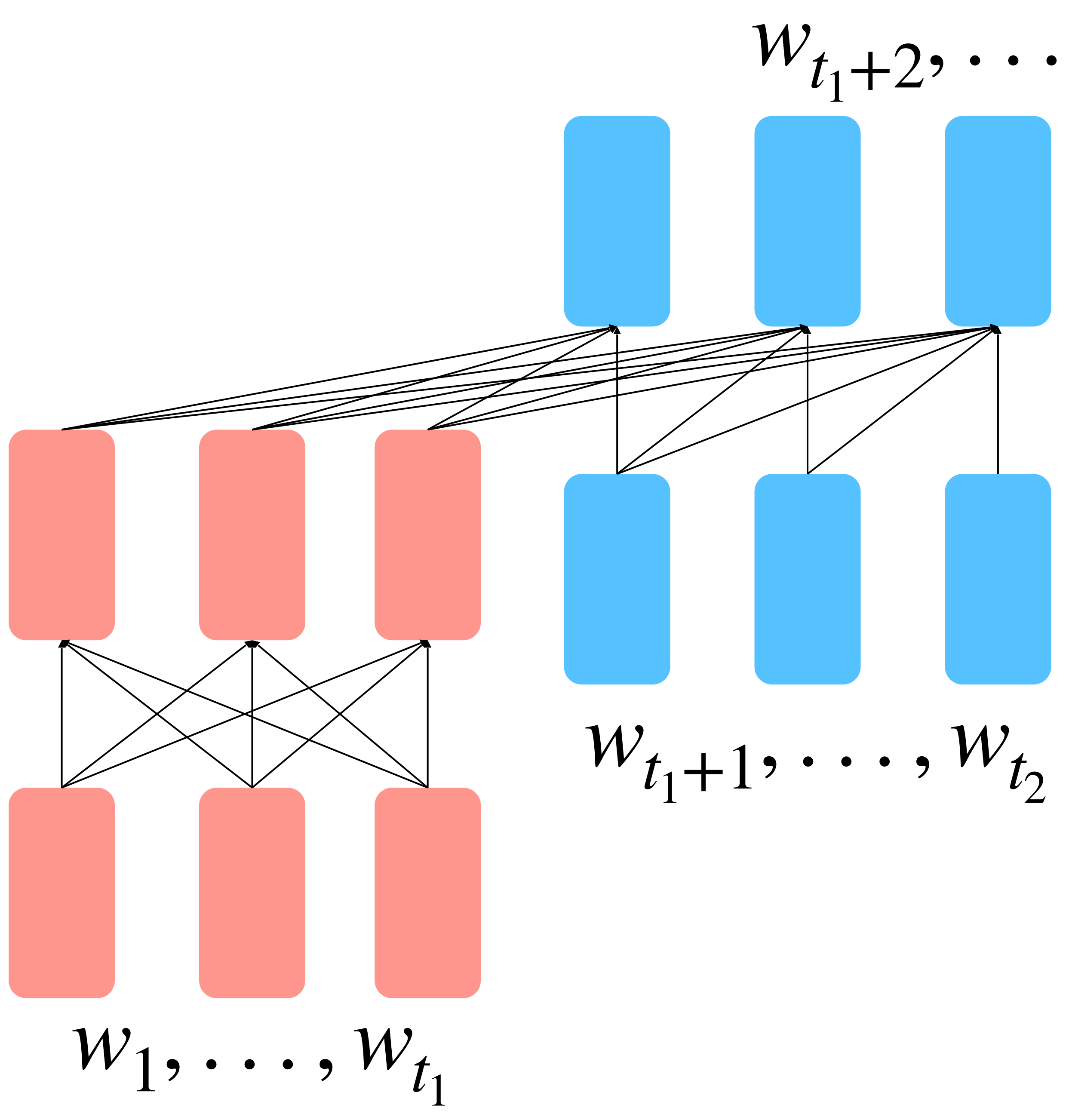
Decoder



- Language modeling; can only condition on the past context

Encoder-Decoder: Architecture

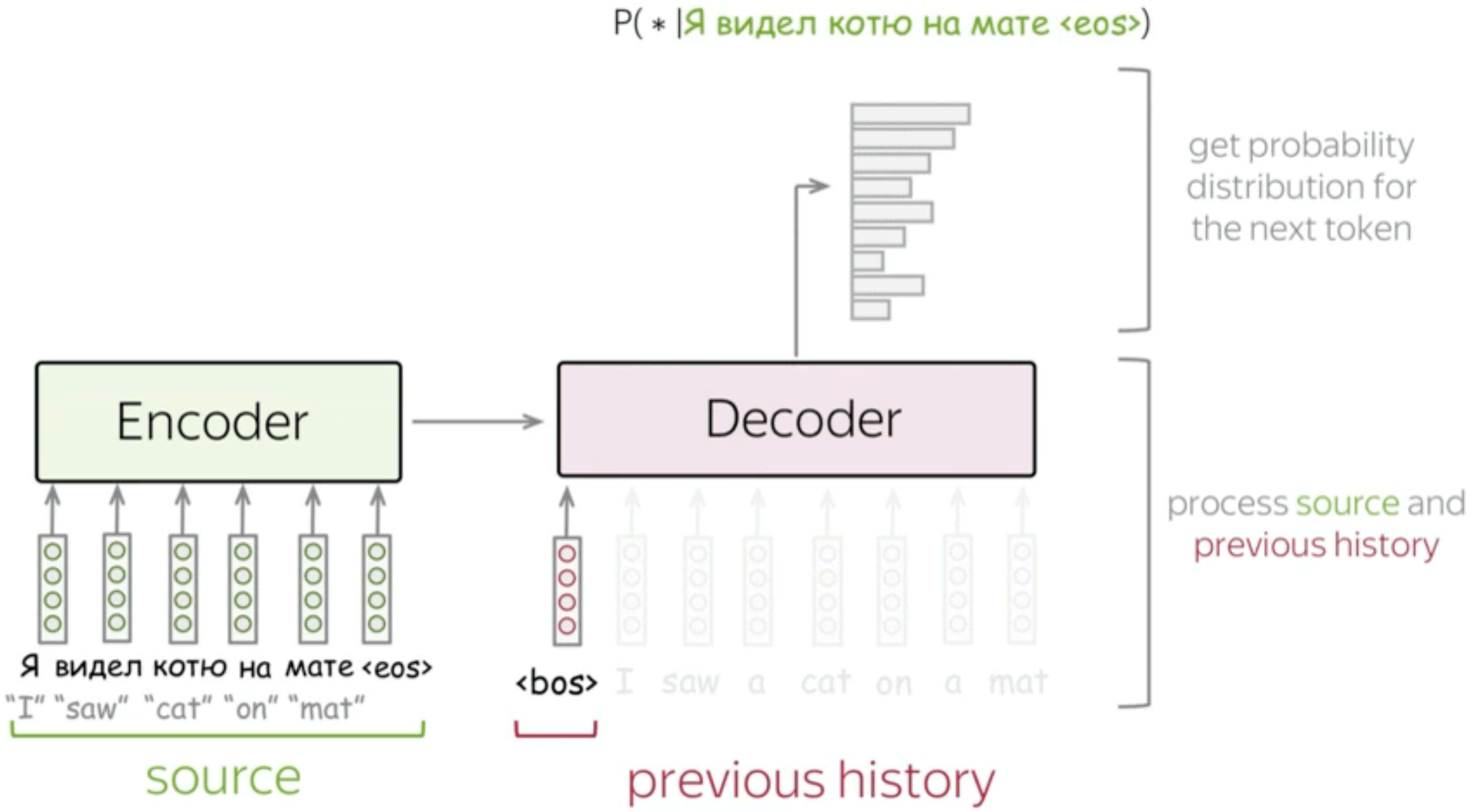
- Moving towards **open-text generation**…
- **Encoder** builds a representation of the source and gives it to the **decoder**
- **Decoder** uses the source representation to generate the target sentence
- The **encoder** portion benefits from **bidirectional** context; the **decoder** portion is used to train the whole model through **language modeling**



$$h_1, \ldots, h_{t_1} = \text{Encoder}(w_1, \ldots, w_{t_1})$$
$$h_{t_1+1}, \ldots, h_{t_2} = \text{Decoder}(w_{t_1+1}, \ldots, w_{t_2}, h_1, \ldots, h_{t_1})$$
$$y_i \sim Ah_i + b, i > t$$

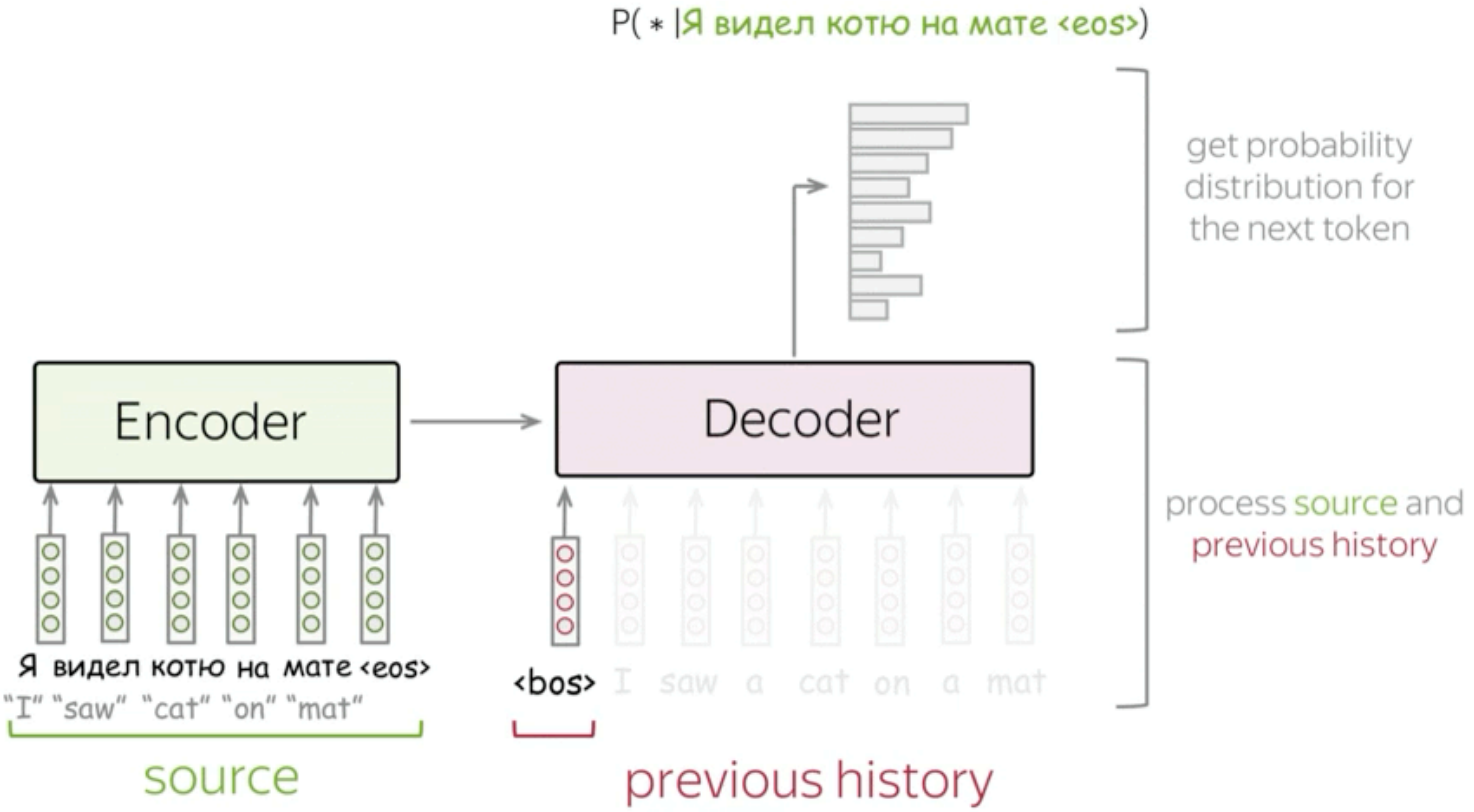
[Raffel et al., 2018]

Encoder-Decoder: An Machine Translation Example



[Lena Viota Blog]

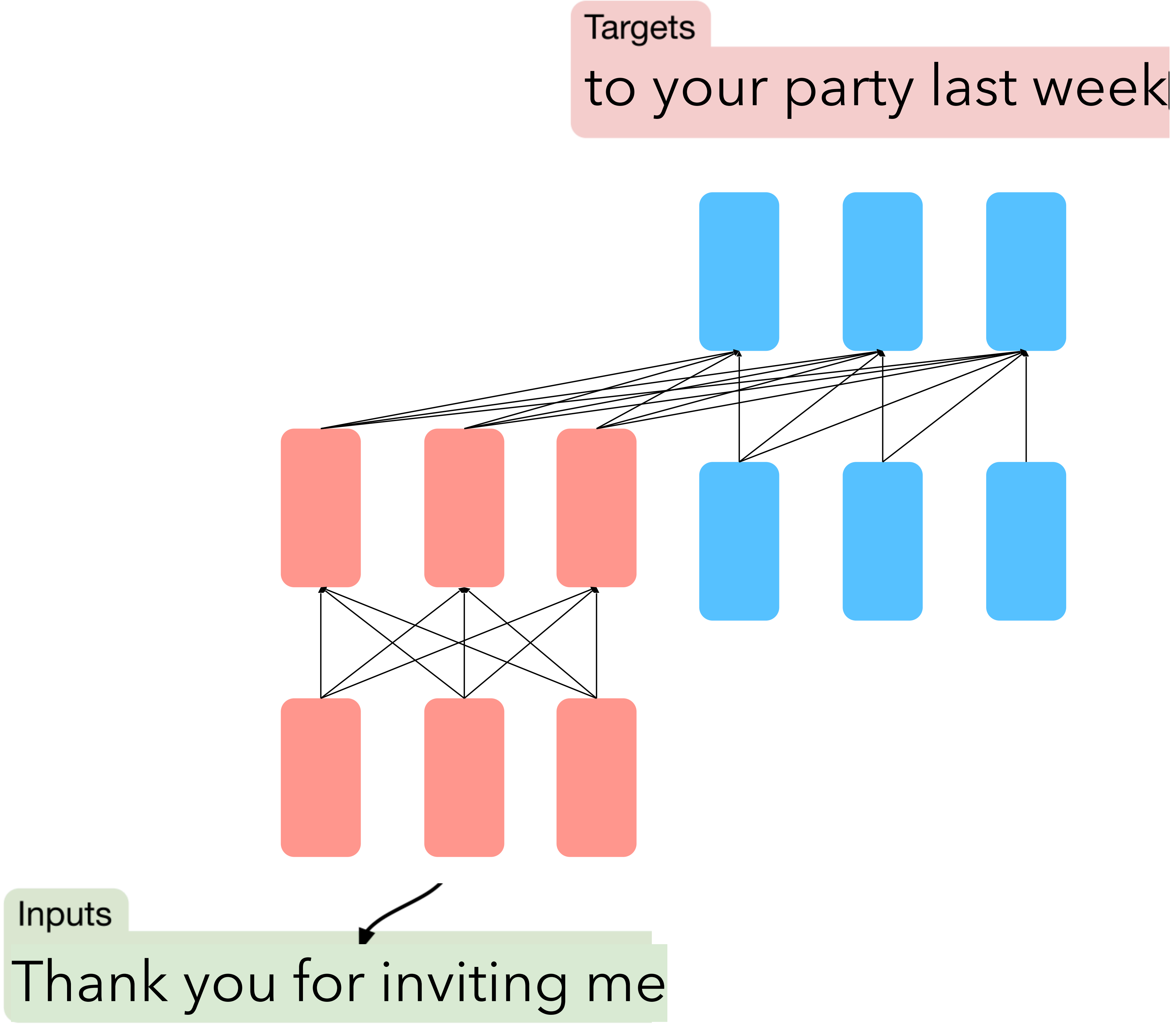
Encoder-Decoder: An Machine Translation Example



[Lena Viota Blog]

Encoder-Decoder: Training Objective

- Can we use Language Modeling here?
- **Kinda:** Given a text span, choose a random point to split it into prefix and target portions.
  - Encoder takes the prefix as input and the decoder is trained to generate the target given prefix



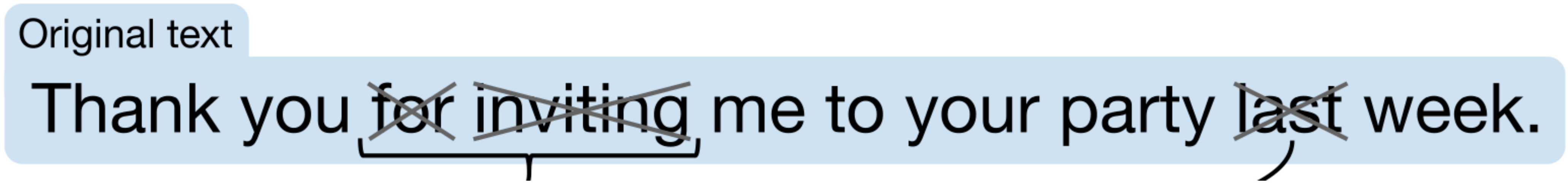
Original text

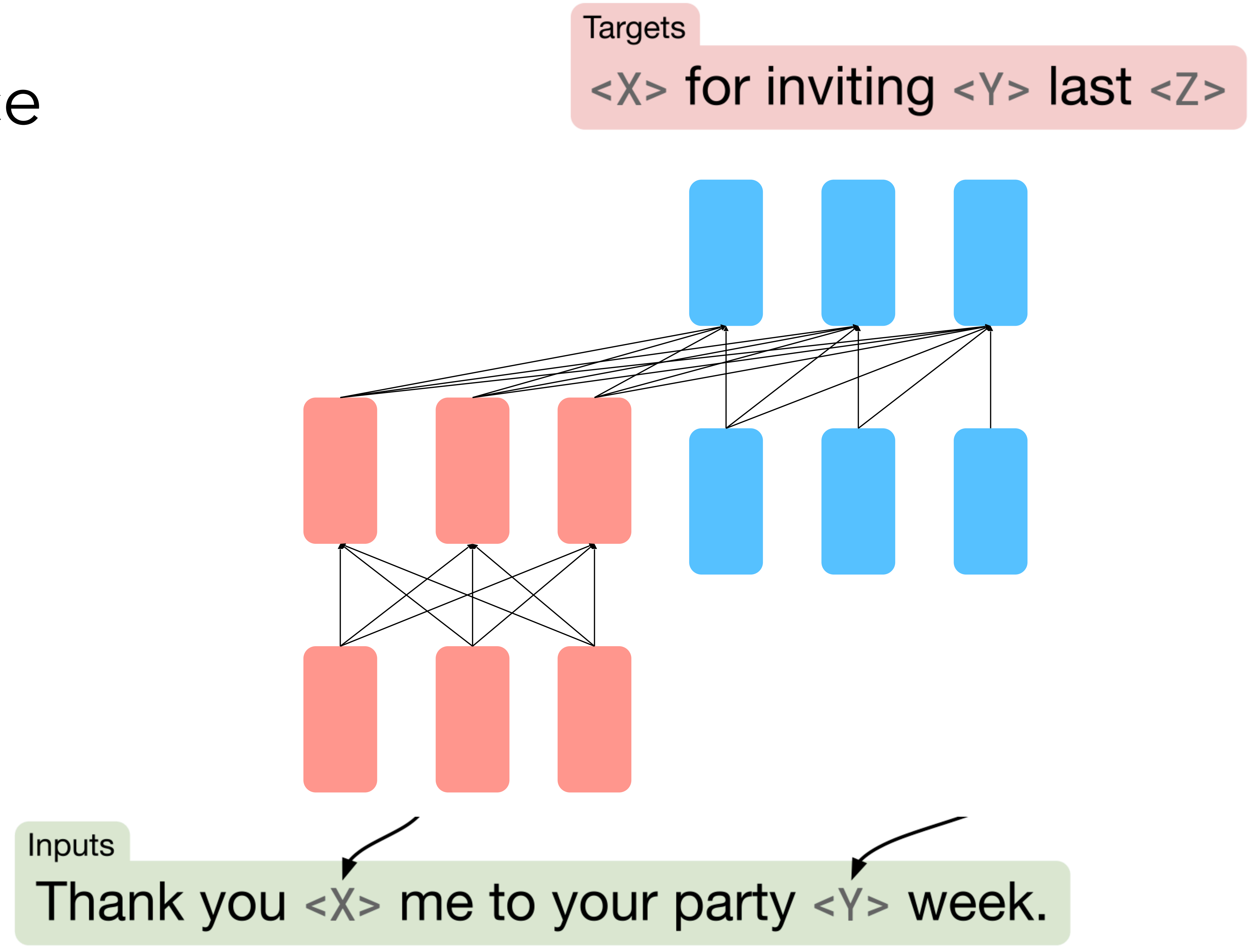
Thank you for inviting me to your party last week.

e.g. split here

Encoder-Decoder: Training Objective

- **T5 [Raffel et al., 2018]**
- **Text span corruption (denoising):** Replace different-length spans from the input with unique placeholders (e.g., <extra_id_0>); decode out the masked spans.
  - Done during **text preprocessing**: training uses **language modeling** objective at the decoder side





Encoder-Decoder: T5

Text **to** **T**ext **T**ransfer **T**ransformer [Raffel et al., 2018]

	Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★	Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
	Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
	Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
	Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
	Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
	Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
	Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
	Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
	Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
	Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder: T5

Text **t**o **T**ext **T**ransfer **T**ransformer [Raffel et al., 2018]

- **Span corruption (denoising)** objective works better than language modeling

	Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★	Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
	Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
	Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
	Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
	Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
	Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
	Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
	Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
	Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
	Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder: T5

Text **to** **T**ext **T**ransfer **T**ransformer [Raffel et al., 2018]

- **Span corruption (denoising)** objective works better than language modeling

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder: T5

Text **to** **T**ext **T**ransfer **T**ransformer [Raffel et al., 2018]

- **Span corruption (denoising)** objective works better than language modeling
- **Encoder-decoders** works better than decoders

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Encoder-Decoder: T5

Text **t**o **T**ext **T**ransfer **T**ransformer [Raffel et al., 2018]

- **Span corruption (denoising)** objective works better than language modeling
- **Encoder-decoders** works better than decoders

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Decoder
(coming next!)

Encoder-Decoder: T5 (Fine-tuning)

T5

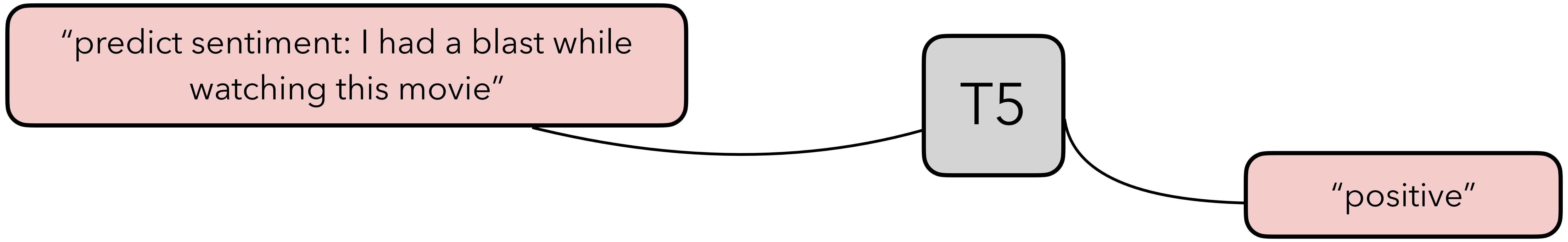
Encoder-Decoder: T5 (Fine-tuning)

Core Idea: Cast any NLP task at hand as a **text generation problem** given some input text!

T5

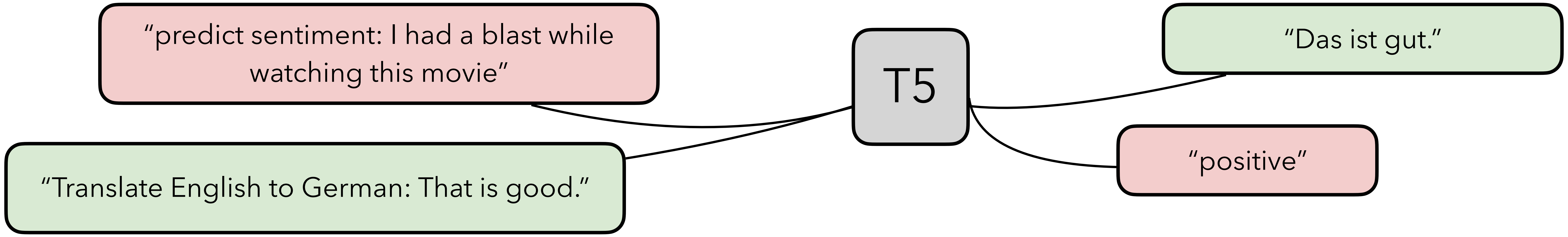
Encoder-Decoder: T5 (Fine-tuning)

Core Idea: Cast any NLP task at hand as a **text generation problem** given some input text!



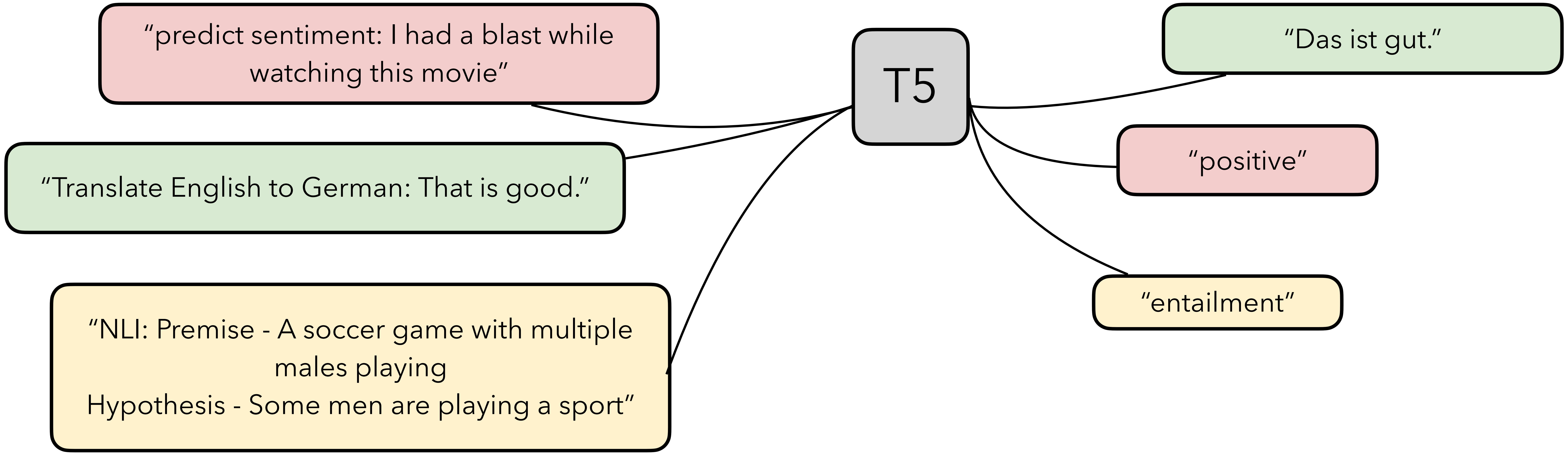
Encoder-Decoder: T5 (Fine-tuning)

Core Idea: Cast any NLP task at hand as a **text generation problem** given some input text!



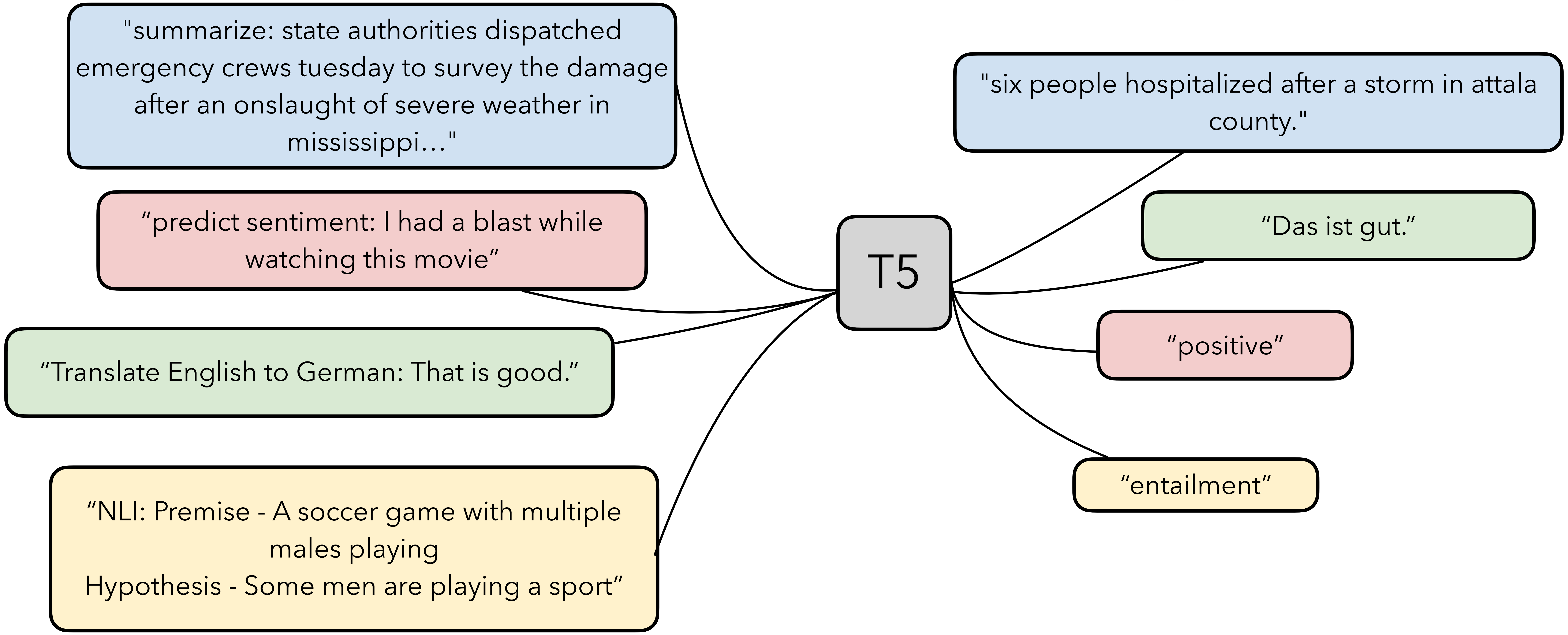
Encoder-Decoder: T5 (Fine-tuning)

Core Idea: Cast any NLP task at hand as a **text generation problem** given some input text!

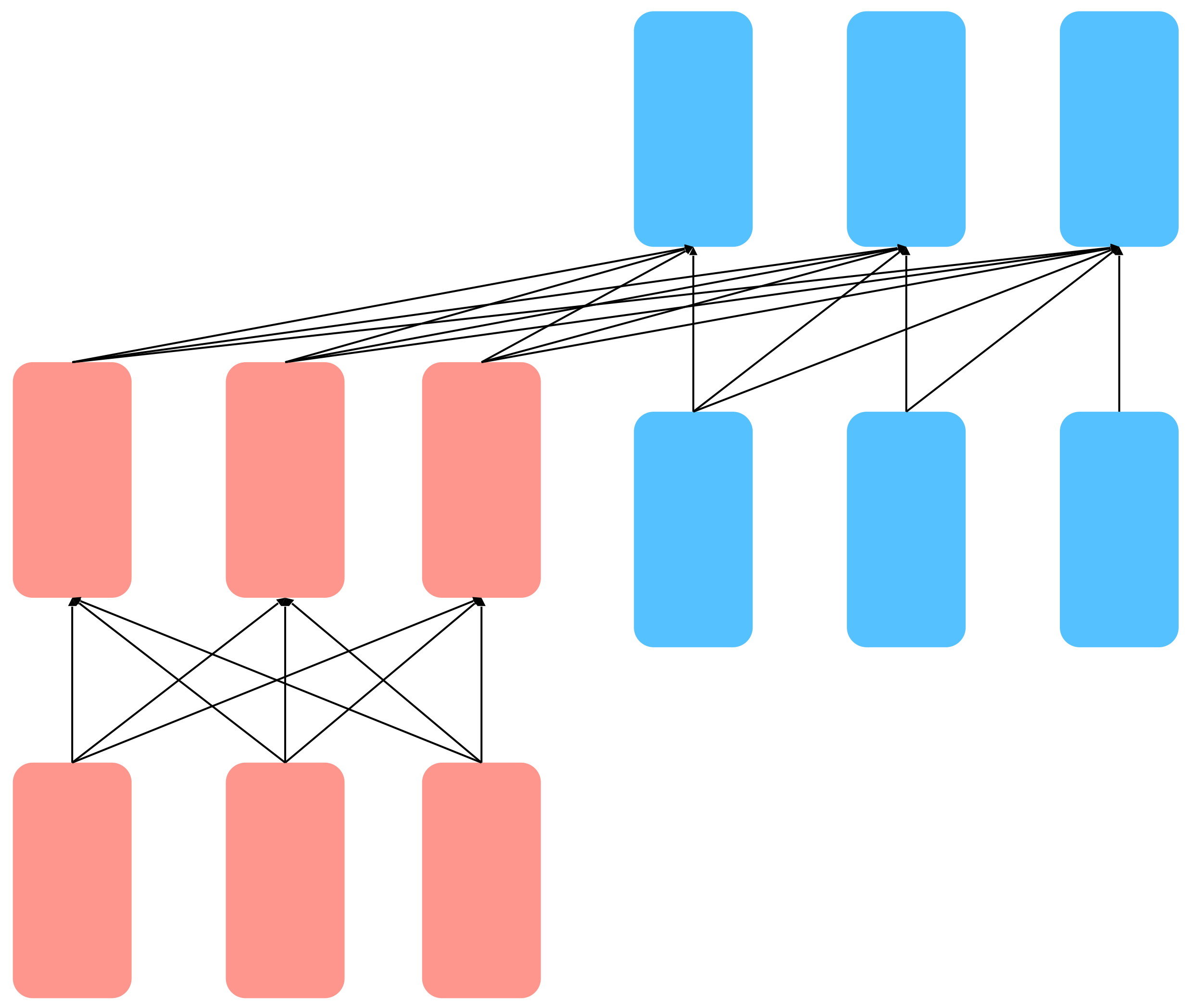


Encoder-Decoder: T5 (Fine-tuning)

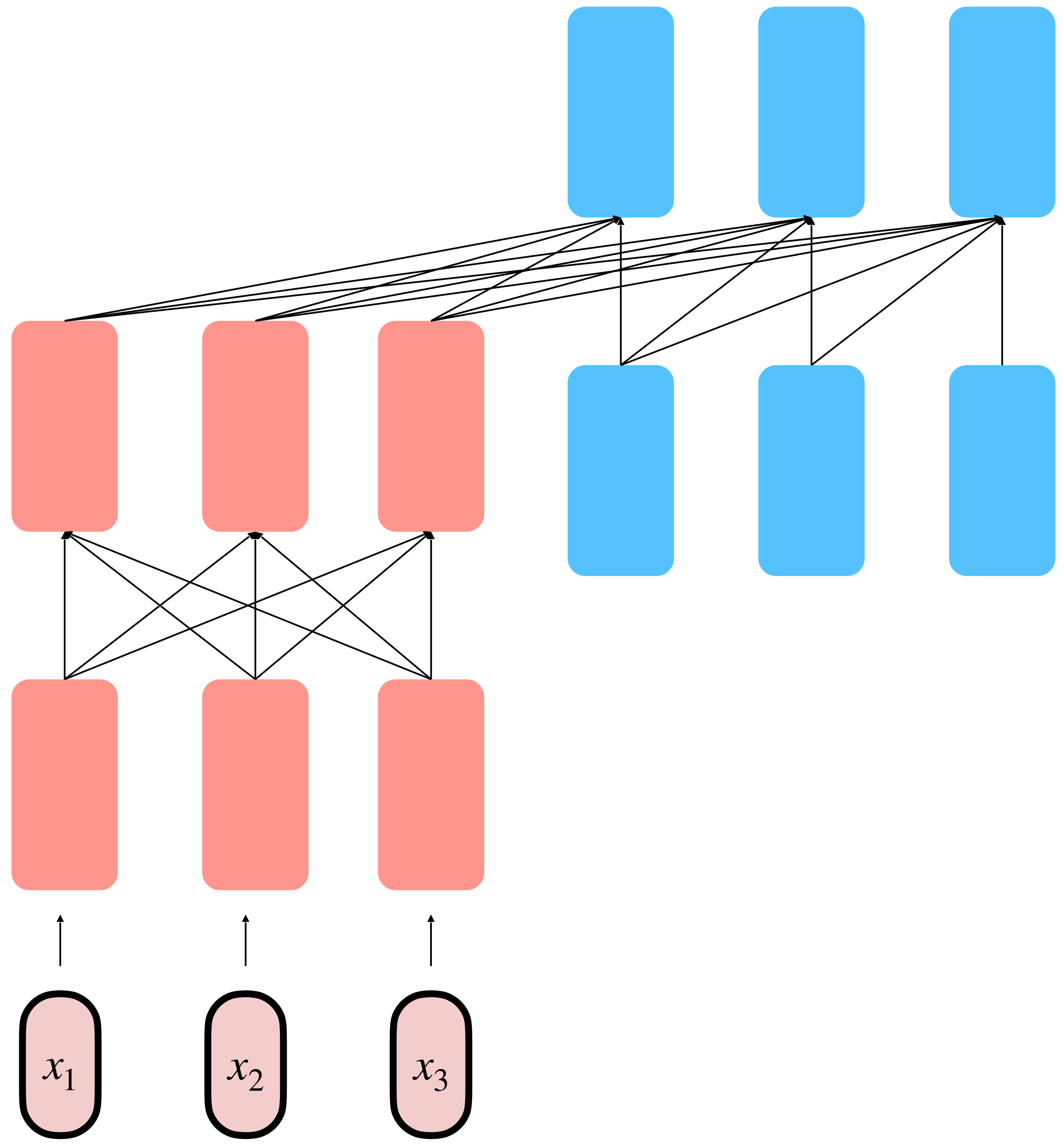
Core Idea: Cast any NLP task at hand as a **text generation problem** given some input text!



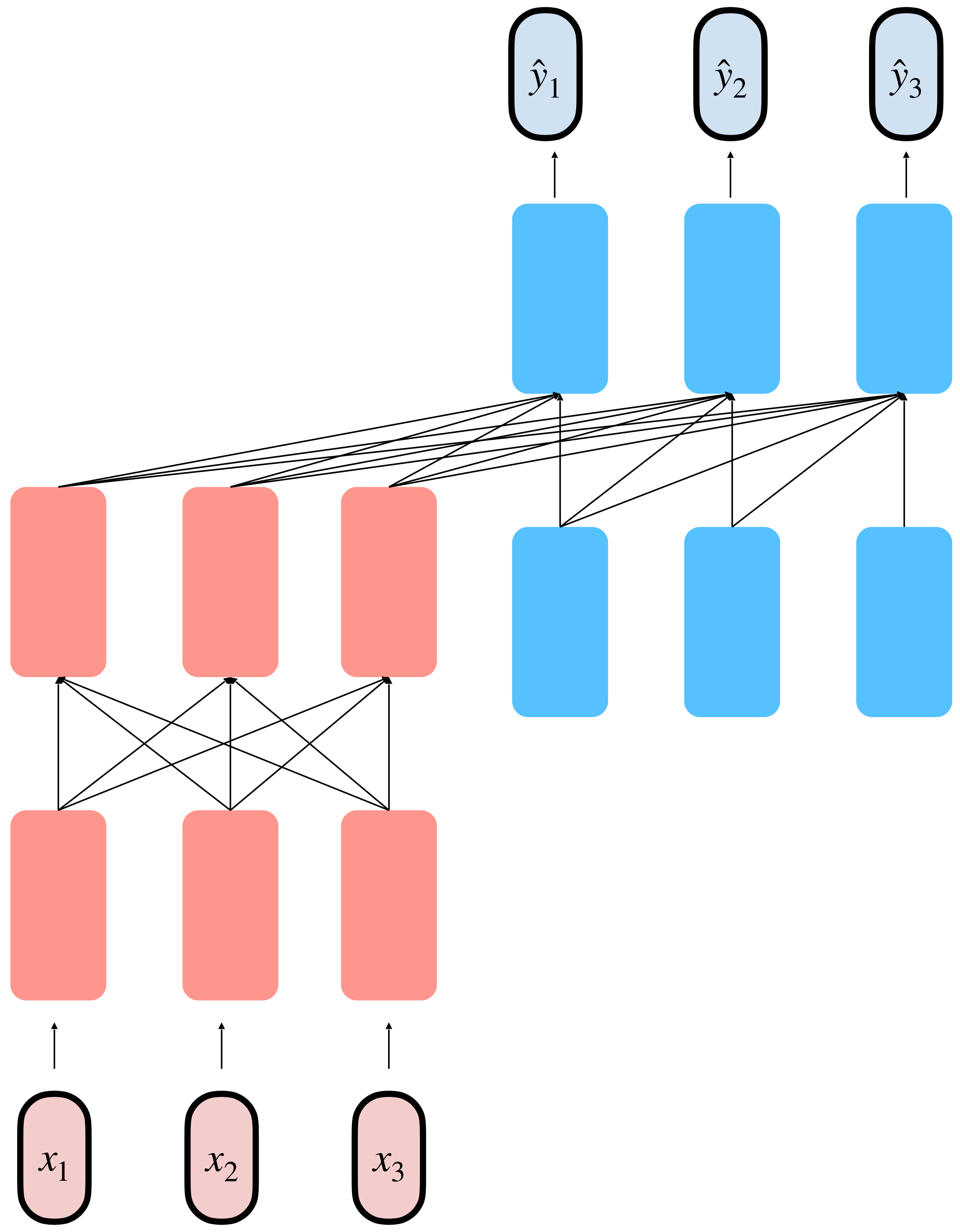
Encoder-Decoder: T5 (Fine-tuning)



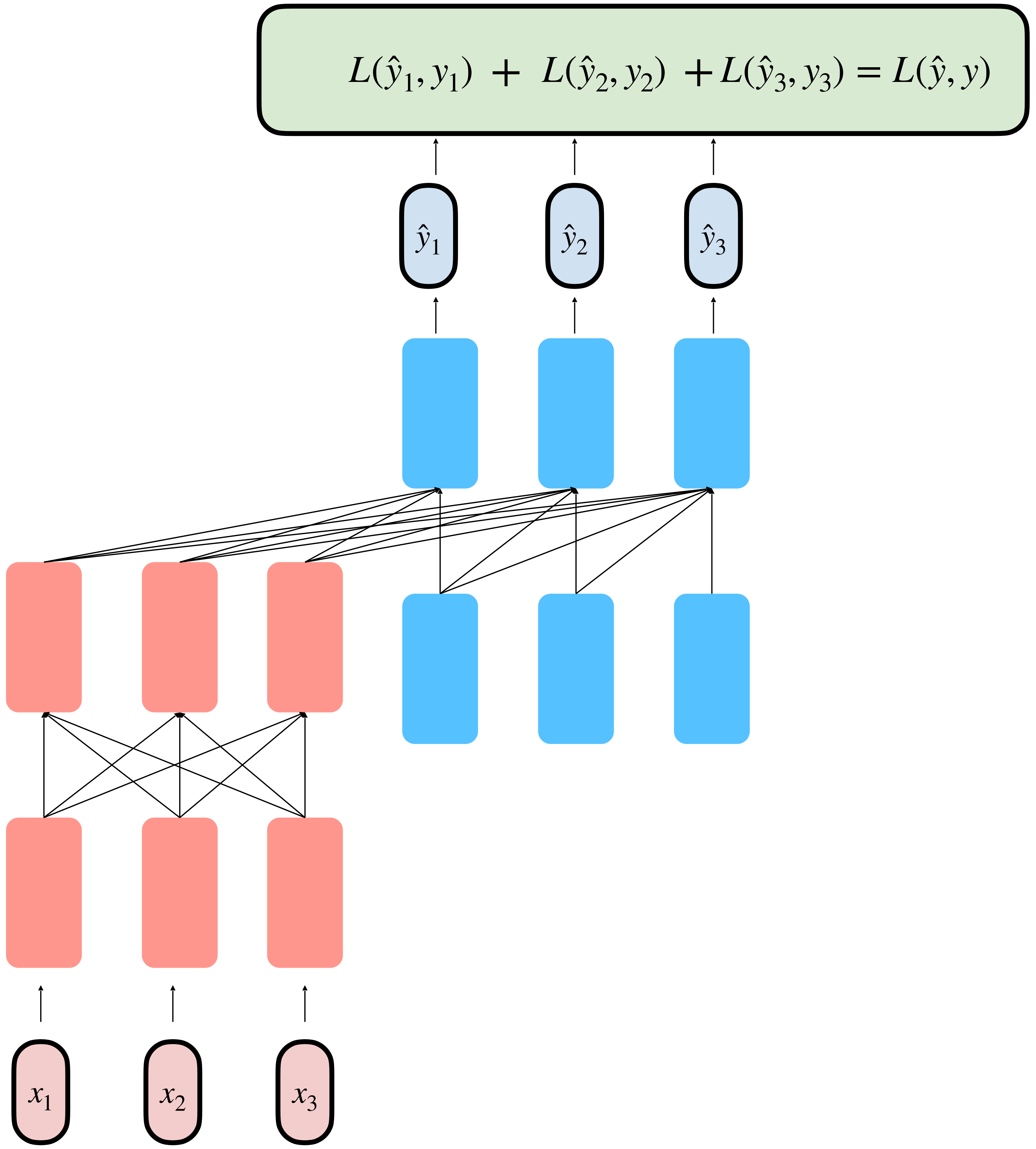
Encoder-Decoder: T5 (Fine-tuning)



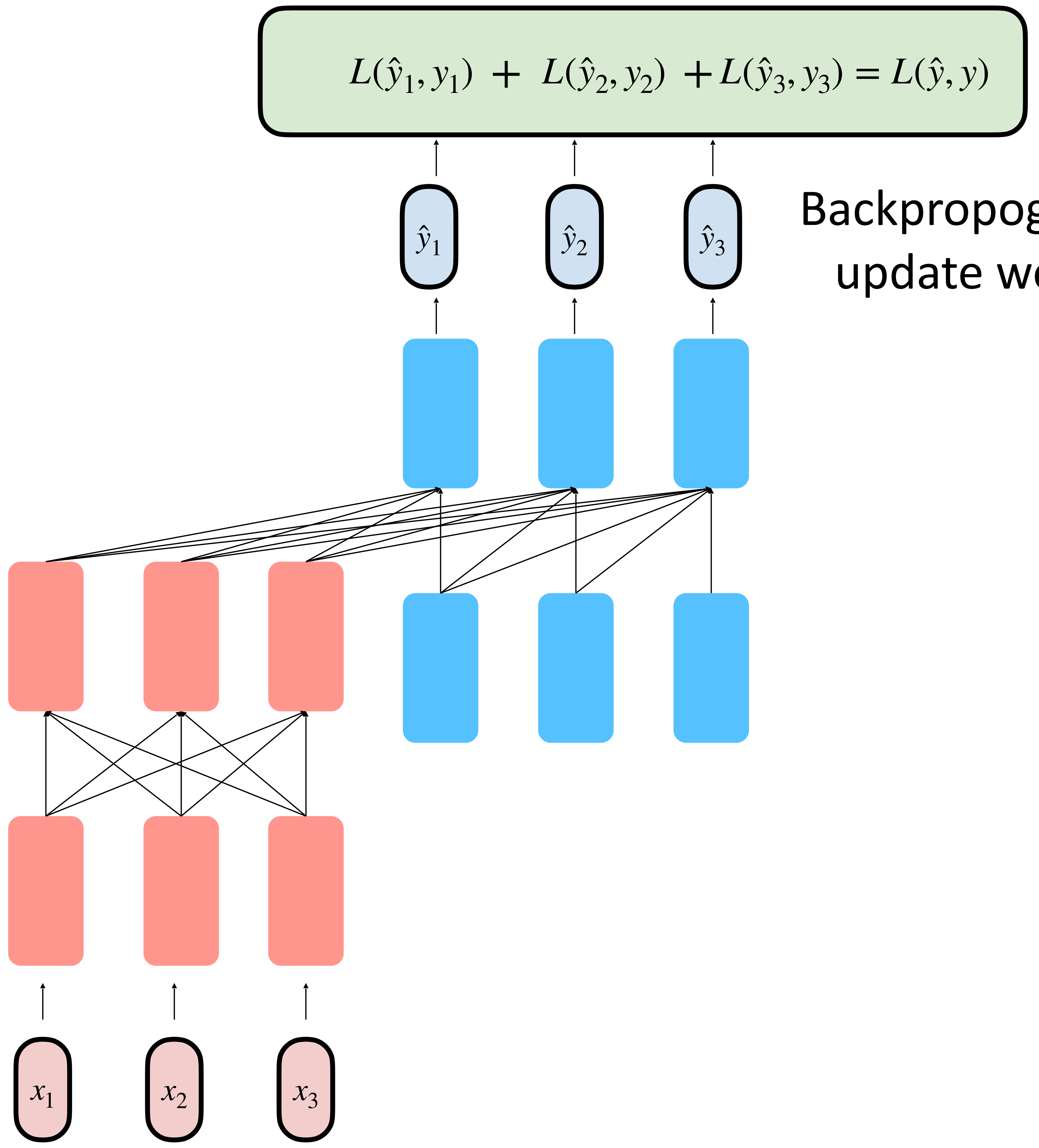
Encoder-Decoder: T5 (Fine-tuning)



Encoder-Decoder: T5 (Fine-tuning)

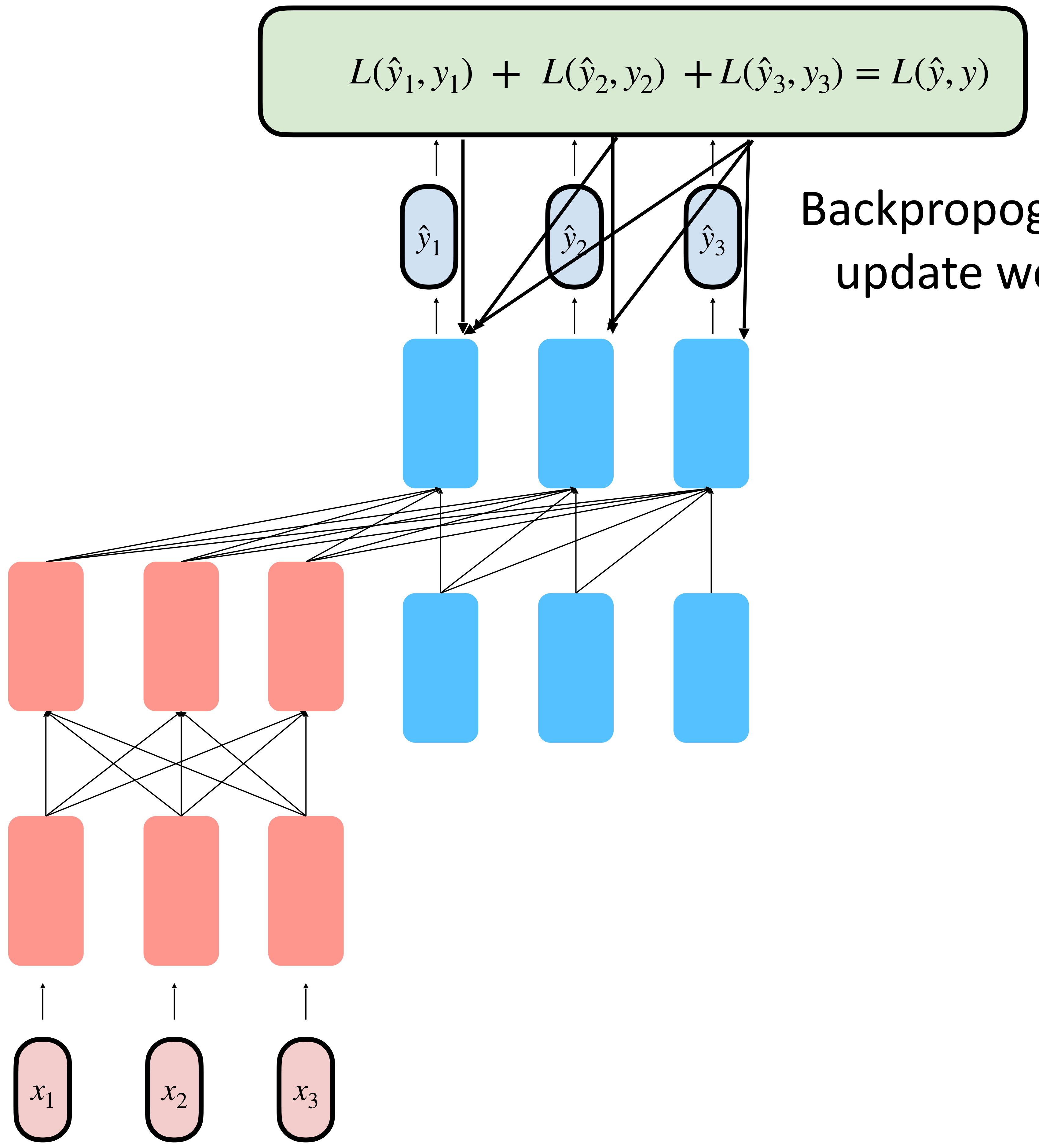


Encoder-Decoder: T5 (Fine-tuning)



Backpropogate **gradients** and update weights using **SGD**

Encoder-Decoder: T5 (Fine-tuning)



Backpropogate **gradients** and update weights using **SGD**

Encoder-Decoder: T5

- **Text-to-Text:** convert NLP tasks into input/output text sequences
- **Dataset:** Colossal Clean Crawled Corpus (C4), 750G text data!
- **Various Sized Models:**
    - Base (222M)
    - Small (60M)
    - Large (770M)
    - 3B
    - 11B
- **Achieved SOTA with scaling & purity of data**

[Google Blog]

Encoder-Decoder: T5

- **Text-to-Text:** convert NLP tasks into input/output text sequences
- **Dataset:** Colossal Clean Crawled Corpus (C4), 750G text data!
- **Various Sized Models:**
  - Base (222M)
  - Small (60M)
  - Large (770M)
  - 3B
  - 11B
- **Achieved SOTA with scaling & purity of data**

[Google Blog]

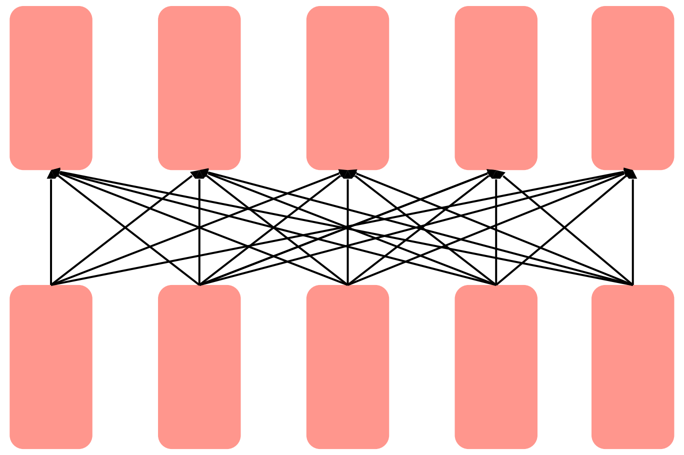
Encoder-Decoder: Pros & Cons

- A nice middle ground between leveraging **bidirectional** contexts and **open-text** generation
- Good for **multi-task** fine-tuning

- Require more **text wrangling**
- **Harder to train**
- **Less flexible** for natural language generation

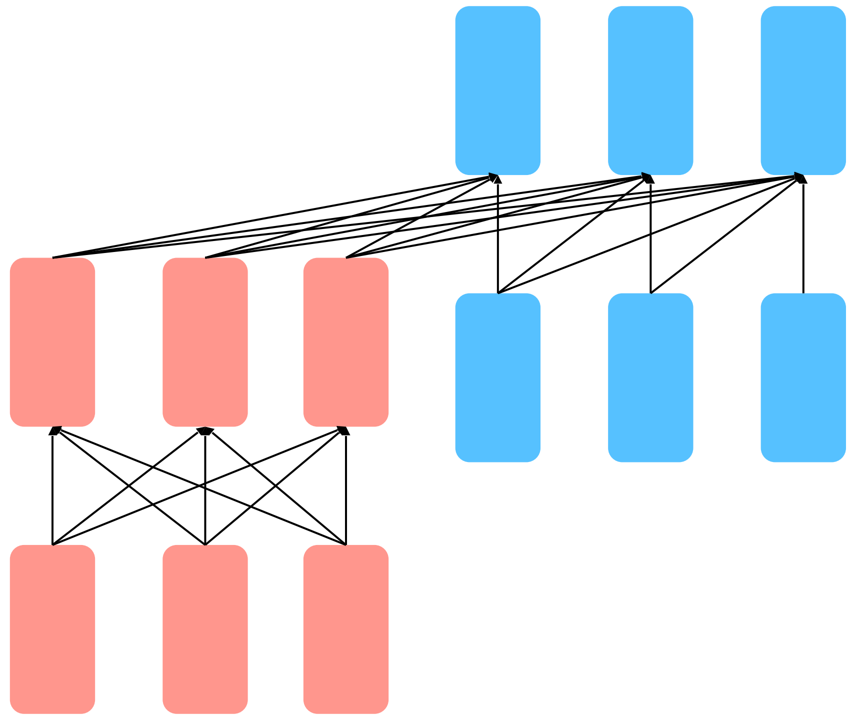
3 Pre-training Paradigms/Architectures

Encoder



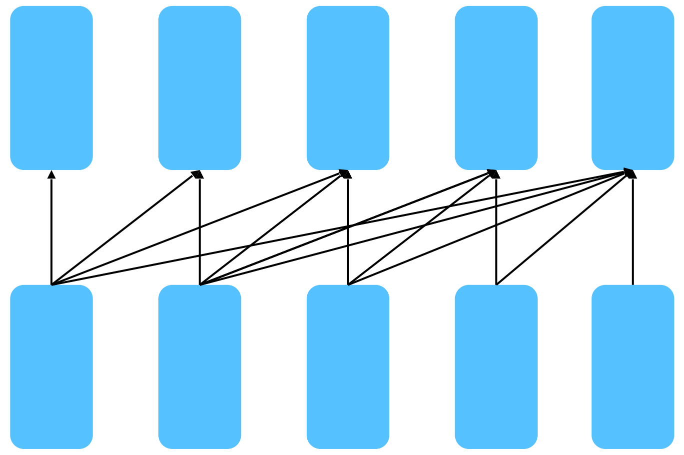
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

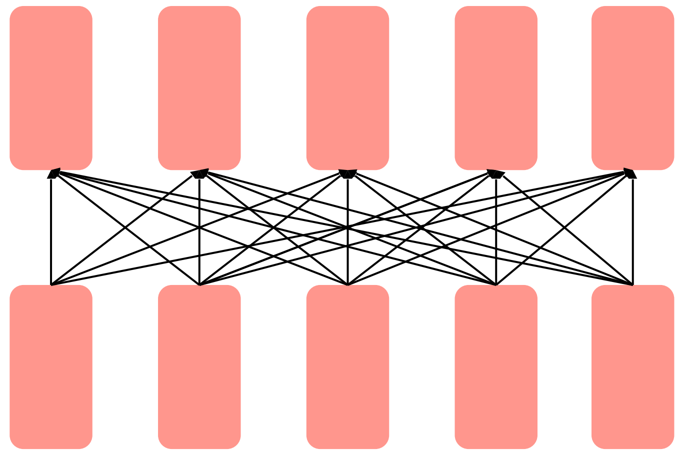
Decoder



- Language modeling; can only condition on the past context

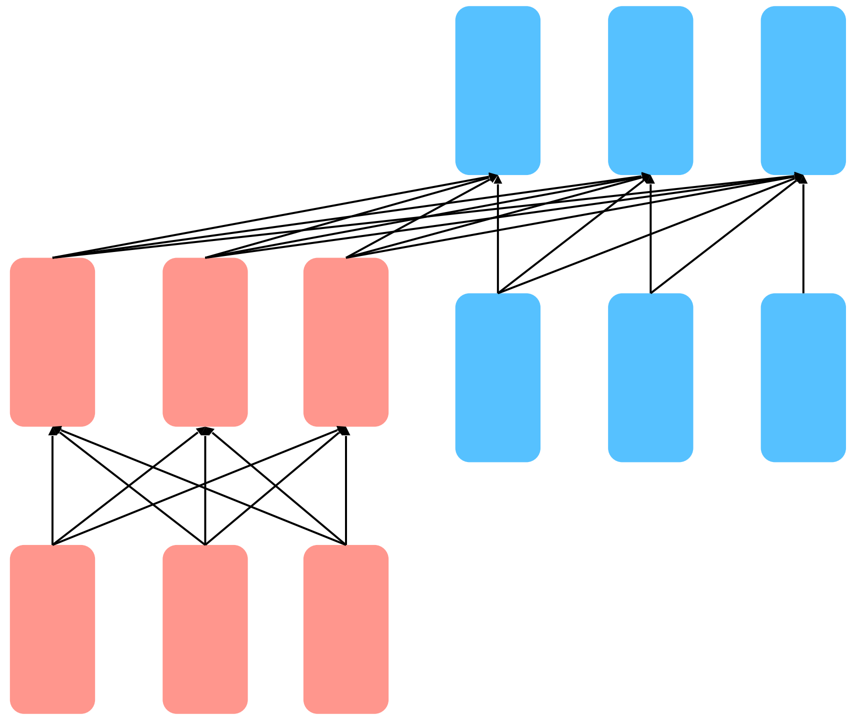
3 Pre-training Paradigms/Architectures

Encoder



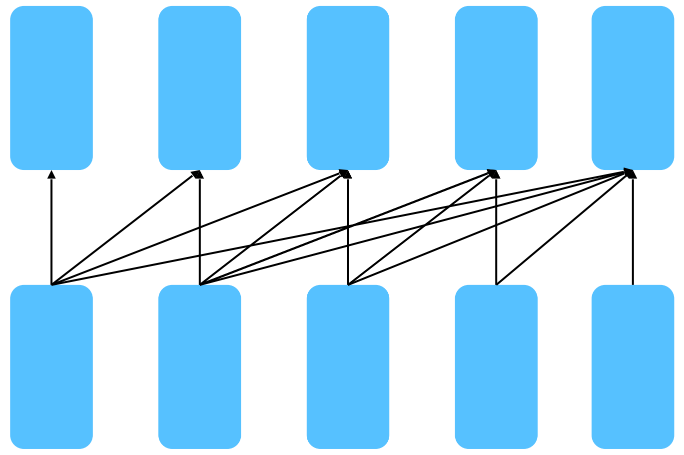
- Bidirectional; can condition on the future context

Encoder-Decoder



- Map two sequences of different length together

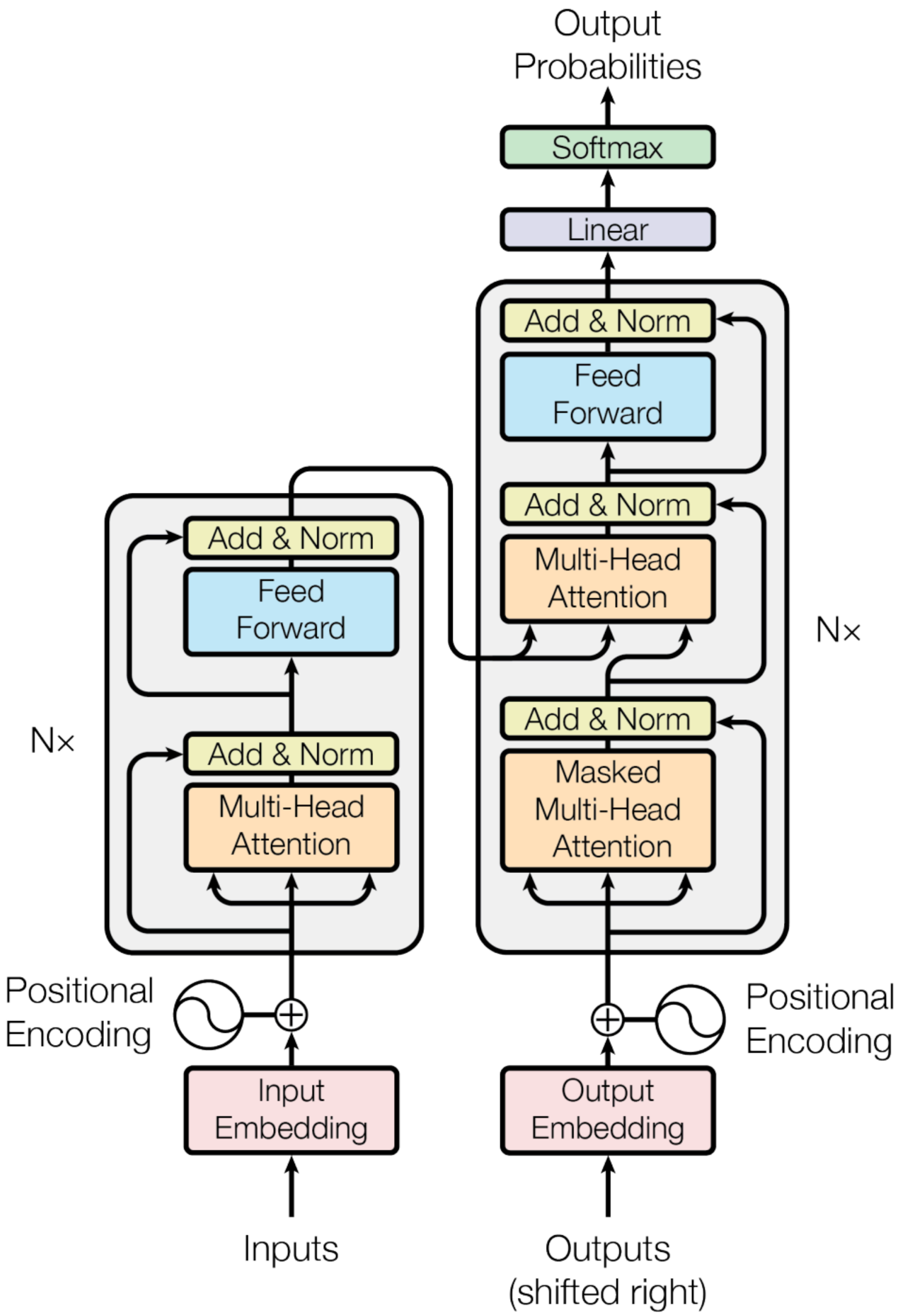
Decoder



- Language modeling; can only condition on the past context

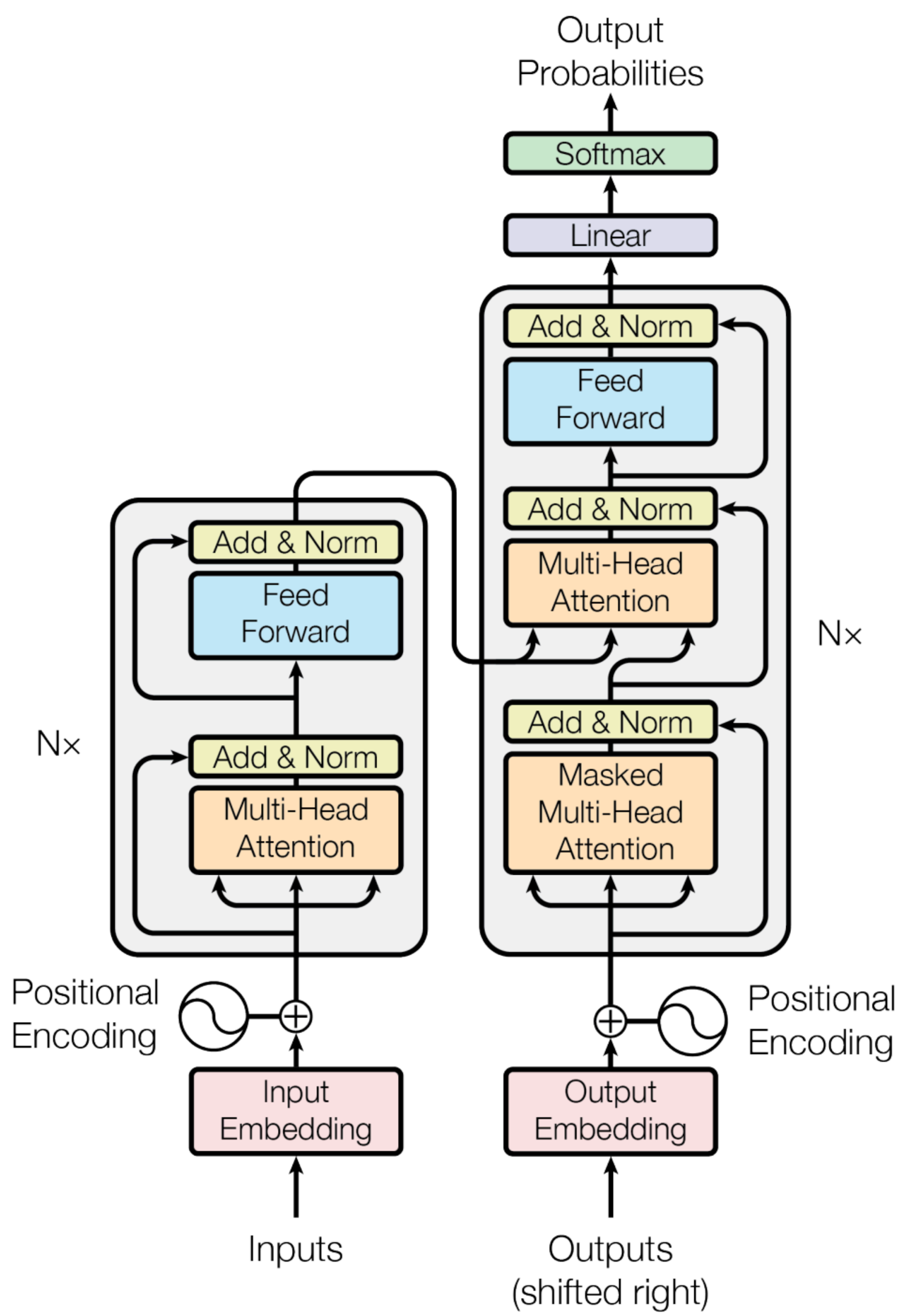
Encoder: Architecture

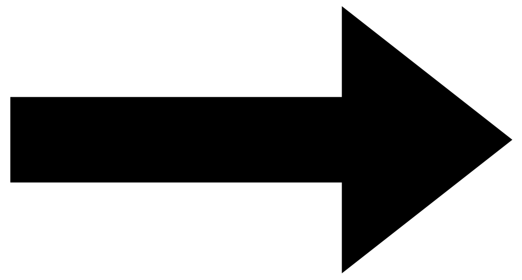
Full-Transformer Architecture
(Encoder-Decoder)



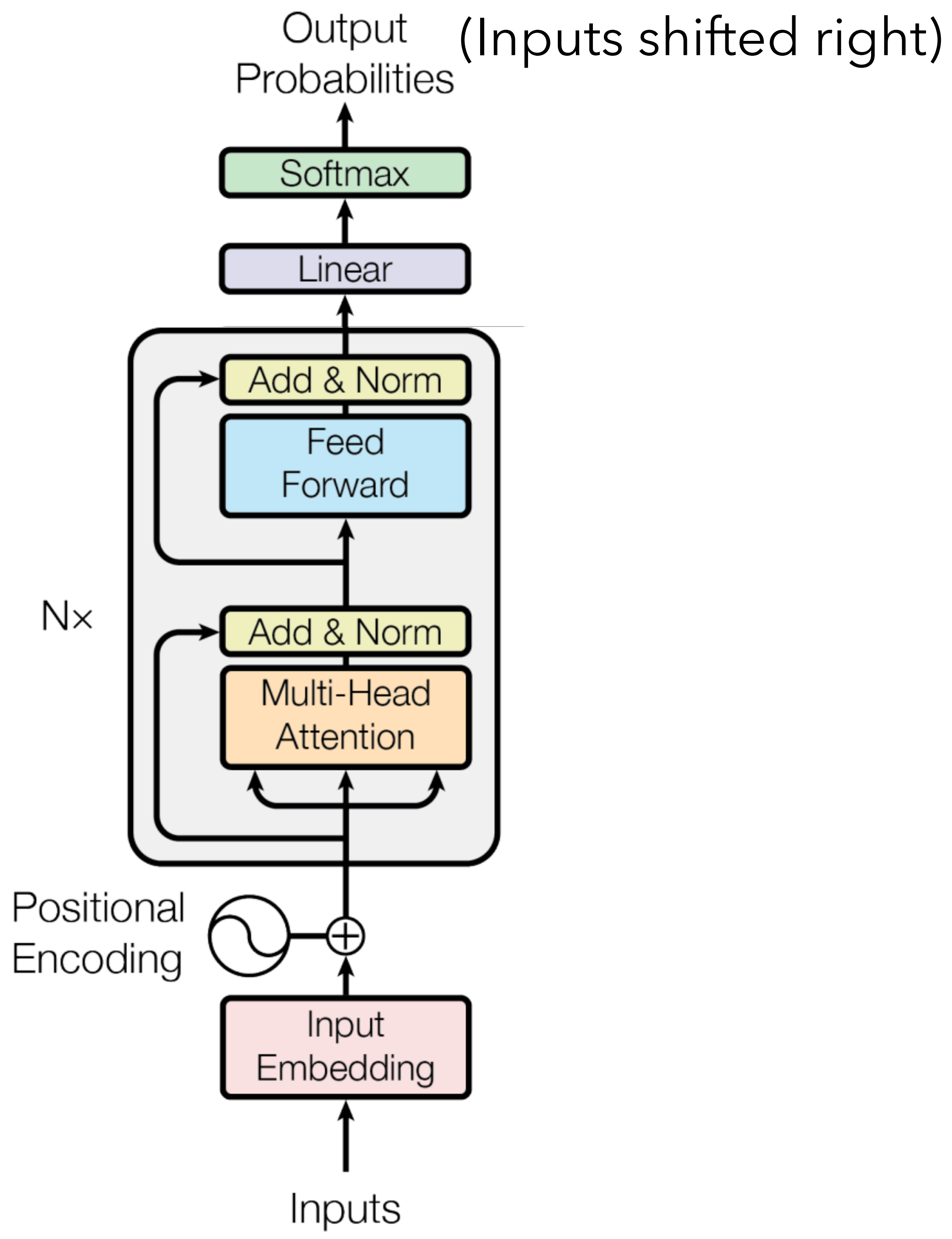
Encoder: Architecture

Full-Transformer Architecture (Encoder-Decoder)



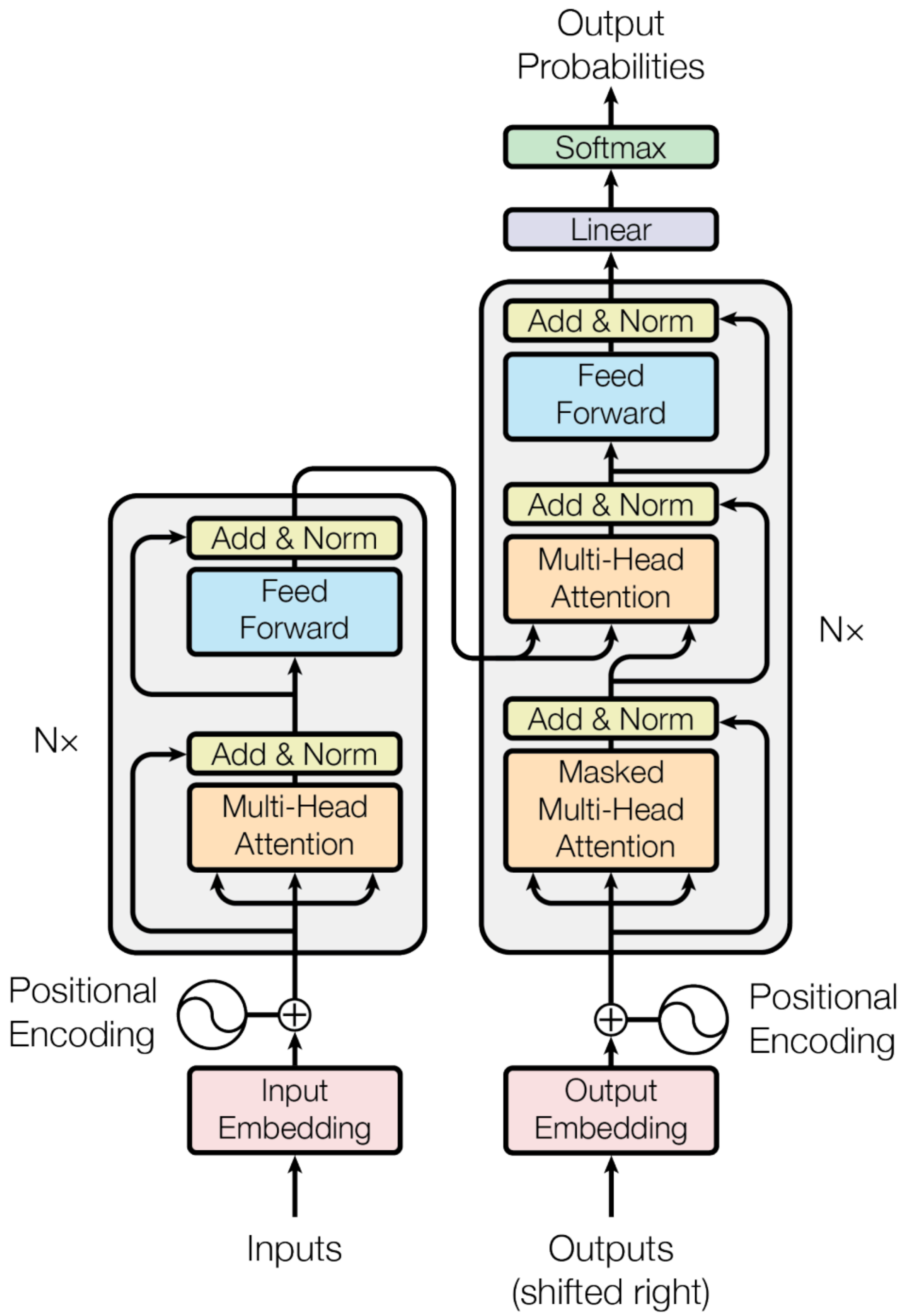


Decoder-Only Transformer Architecture



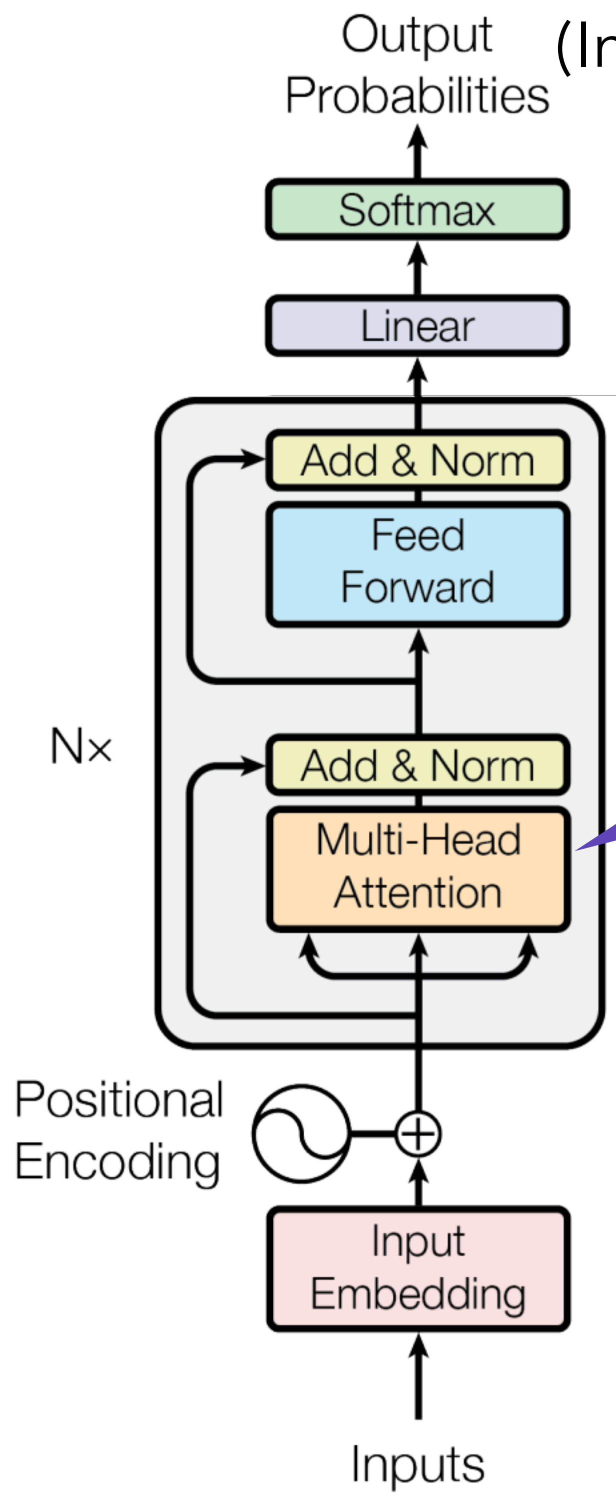
Encoder: Architecture

Full-Transformer Architecture
(Encoder-Decoder)



Decoder-Only Transformer Architecture

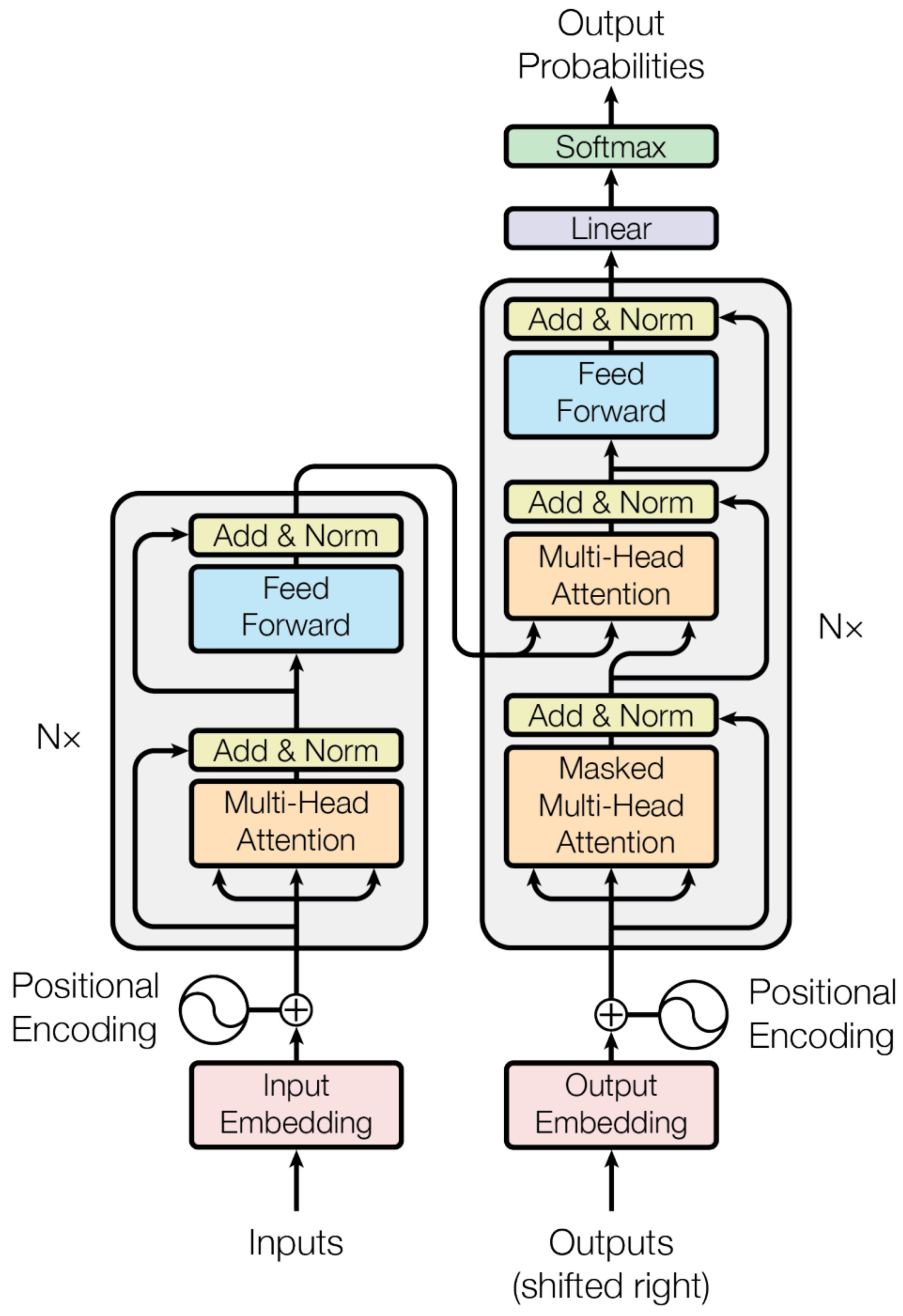
(Inputs shifted right)



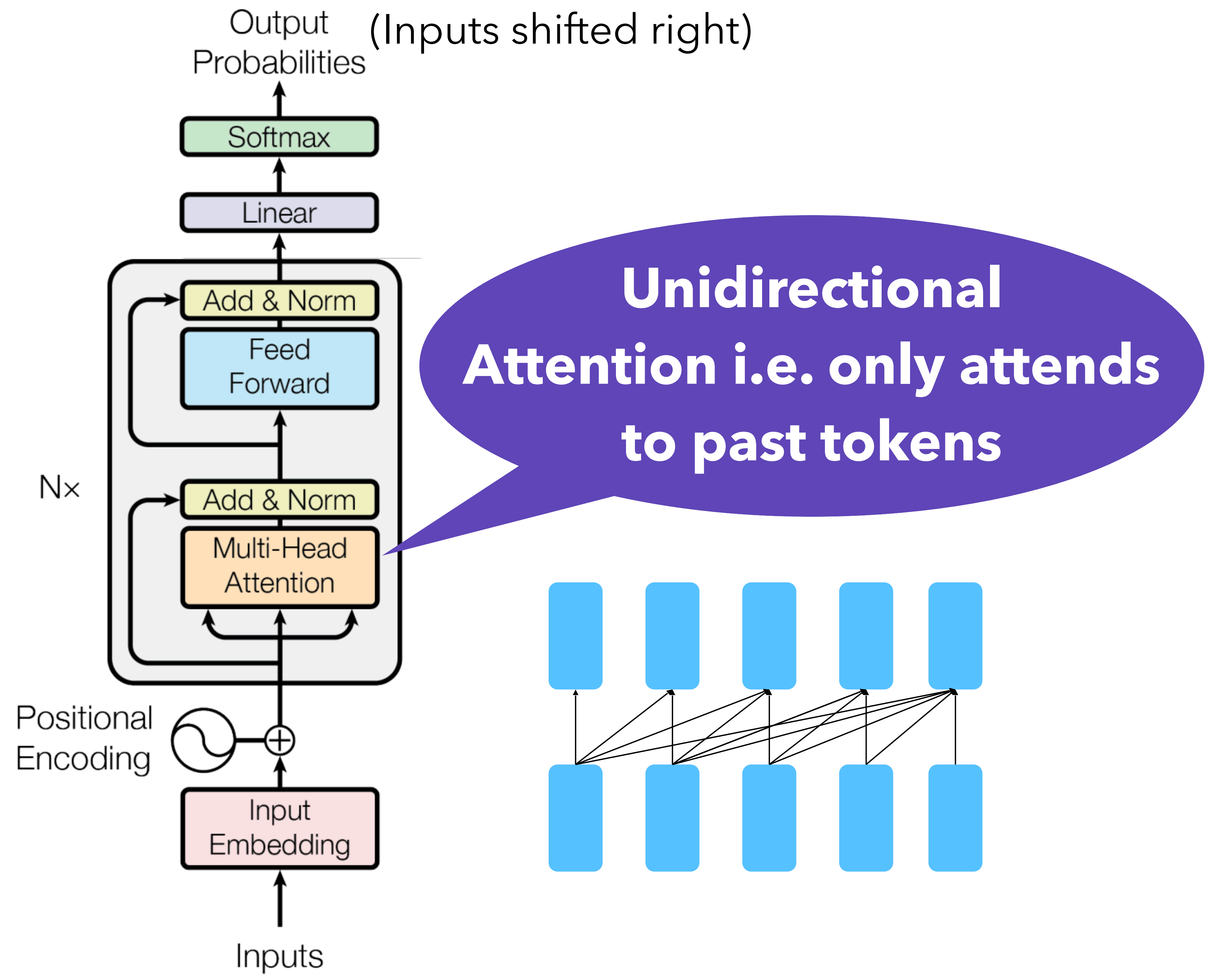
Unidirectional Attention i.e. only attends to past tokens

Encoder: Architecture

Full-Transformer Architecture (Encoder-Decoder)

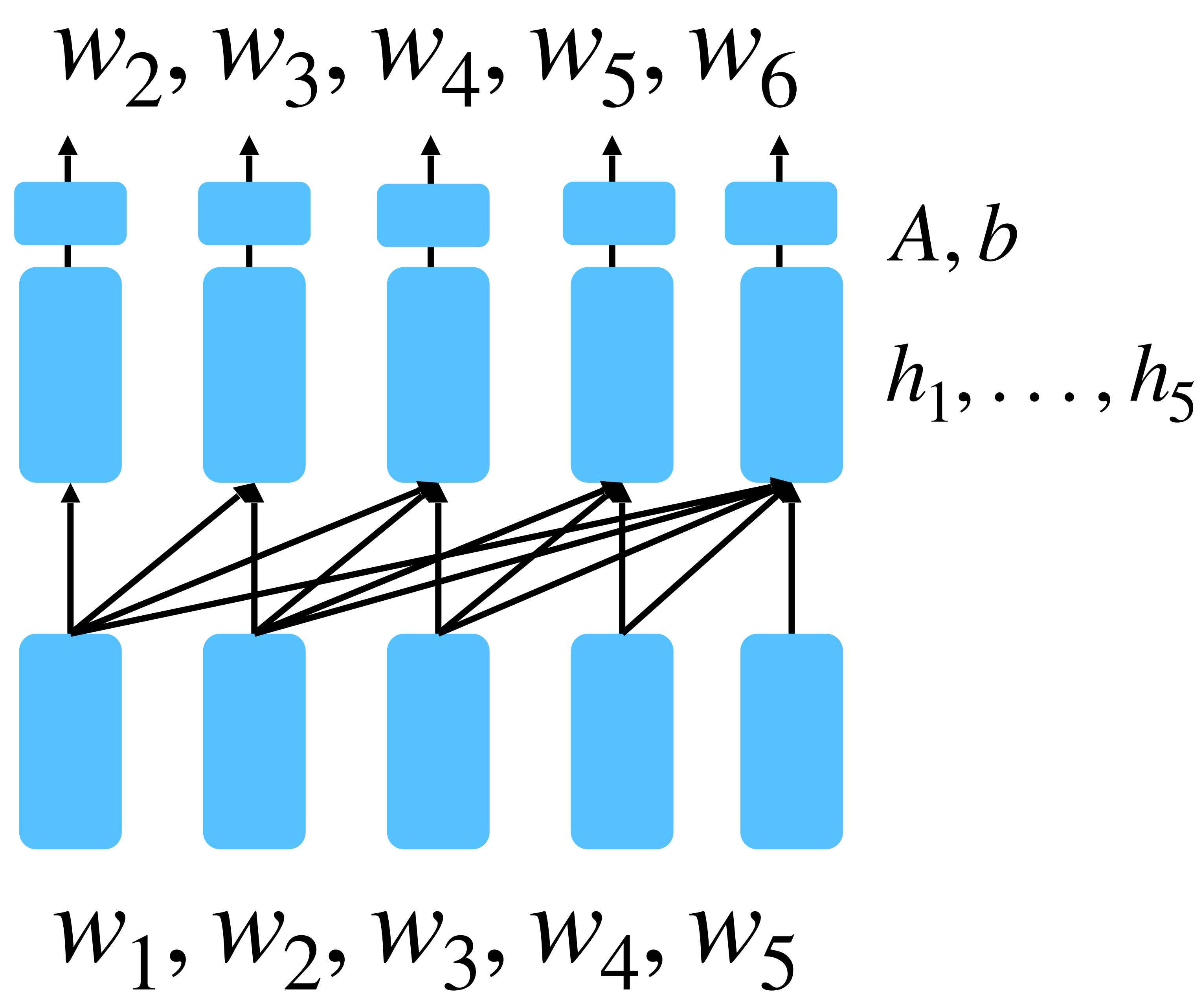


Decoder-Only Transformer Architecture



Decoder: Training Objective

- Many most famous generative LLMs are **decoder-only**
  - e.g., GPT1/2/3/4, Llama1/2
- **Language modeling!** Natural to be used for **open-text generation**
- **Conditional LM:** $p(w_t \mid w_1, \ldots, w_{t-1}, x)$
  - Conditioned on a source context x to generate from left-to-right
- Can be fine-tuned for **natural language generation (NLG)** tasks, e.g., dialogue, summarization.



Decoder: GPT

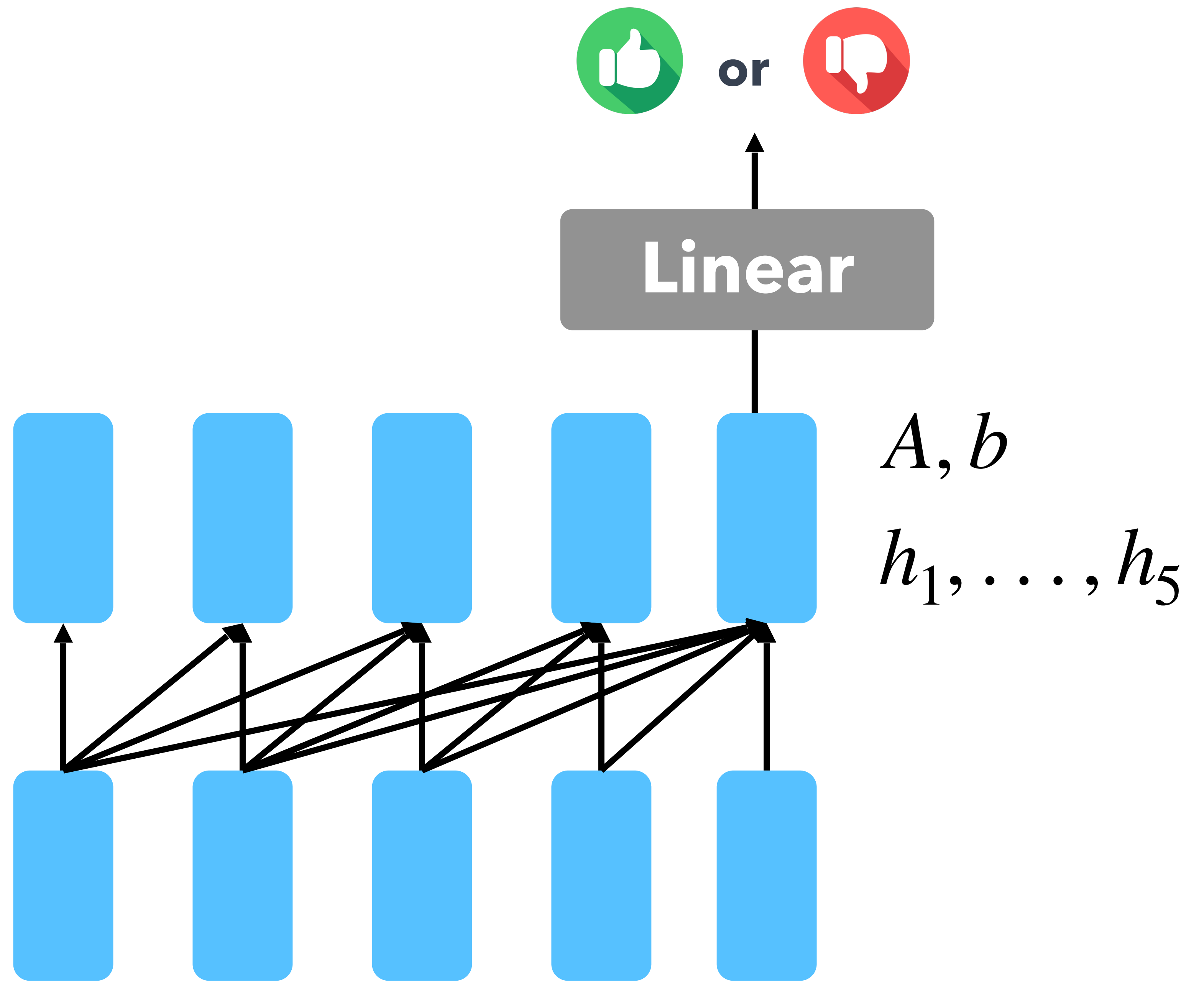
Improving Language Understanding by **G**enerative **P**re-**T**raining [Radford et al., 2018]

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with **12 layers**, **117M parameters**.
- Trained on **BooksCorpus: over 7000 unique books**.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

Decoder: GPT (Finetuning)

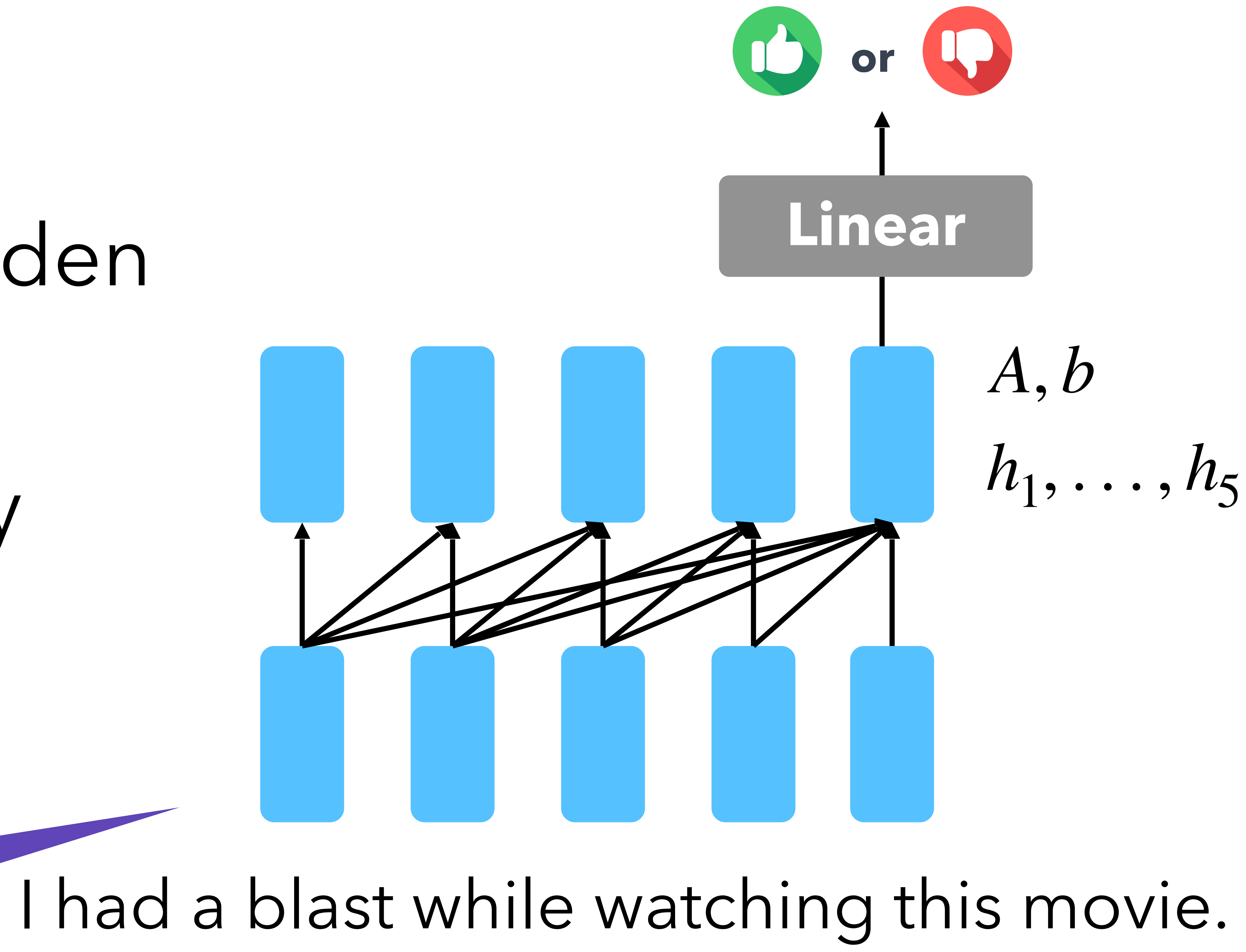
- Customizing the pre-trained model for downstream tasks:
  - Add a **linear layer** on top of the last hidden layer to make it a classifier!
  - During fine-tuning, trained the randomly **initialized linear layer**, along with **all parameters** in the neural net.



I had a blast while watching this movie.

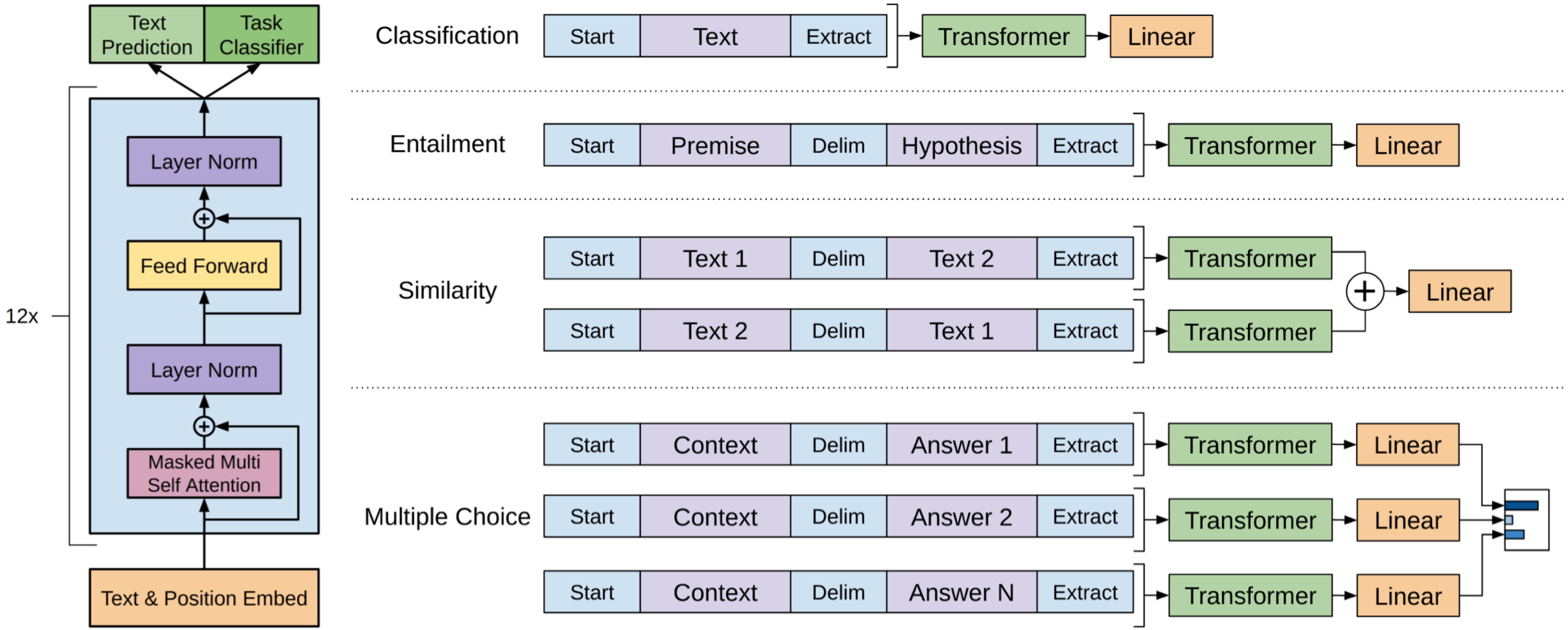
Decoder: GPT (Finetuning)

- Customizing the pre-trained model for downstream tasks:
  - Add a **linear layer** on top of the last hidden layer to make it a classifier!
  - During fine-tuning, trained the randomly **initialized linear layer**, along with **all parameters** in the neural net.



While not originally formulated this way, we can use T5-style text-to-text fine-tuning here for any task. In fact thats the norm now!

Decoder: GPT (Finetuning)



Decoder: GPT-2

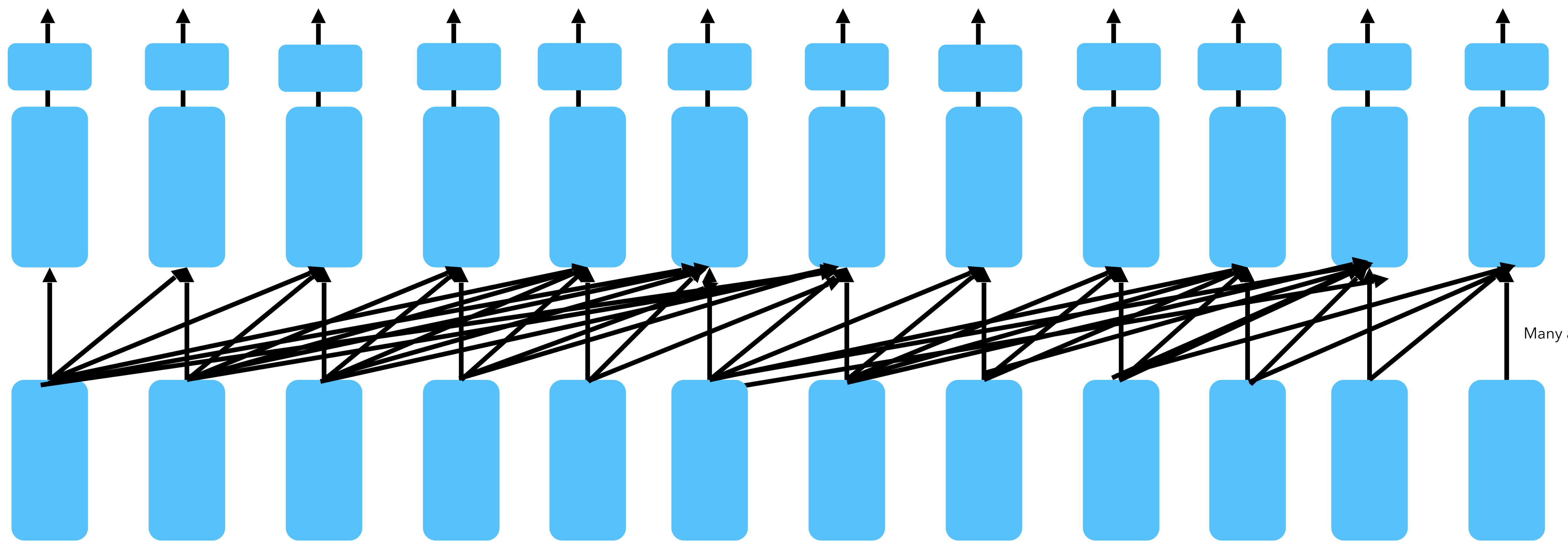
Language Models are Unsupervised Multitask Learners [Radford et al., 2019]

- Scaled-up version of GPT. The largest GPT-2 Model had **1.56B parameters** with **48 layers**.
- Was trained on a much larger dataset
  - **WebText**, curated for high-quality text
  - Consisted of web scrapes of outbound links from Reddit with at least 3 upvotes
  - 45 million links -> 8 million documents -> 40GB of text

Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

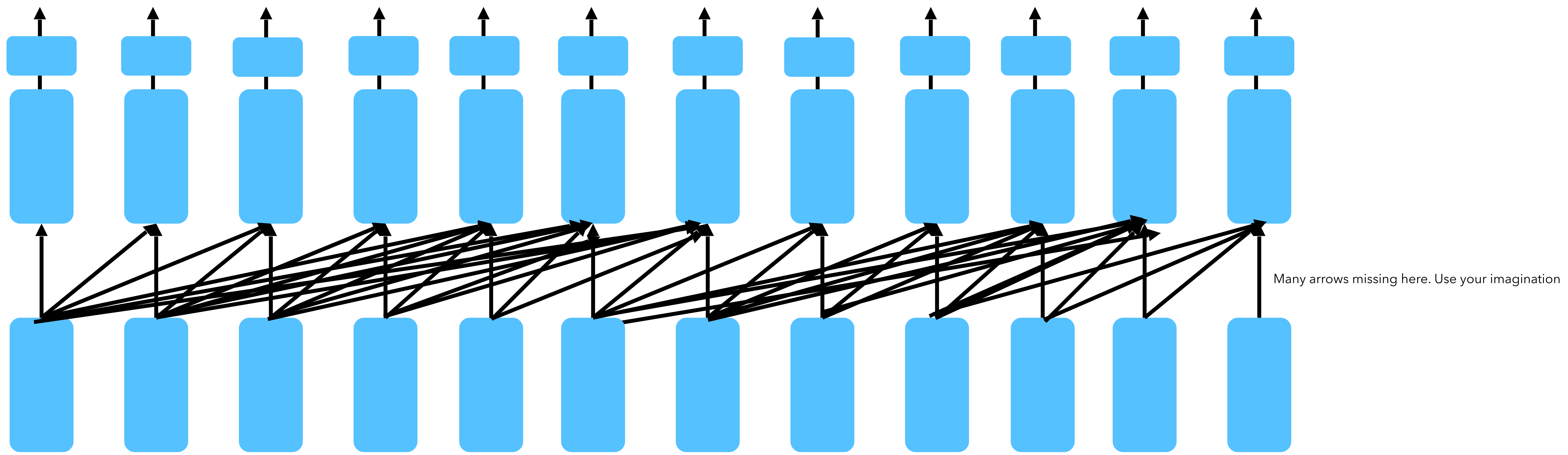


Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

i.e. no fine-tuning and simply prompting the pre-trained model and generating the output

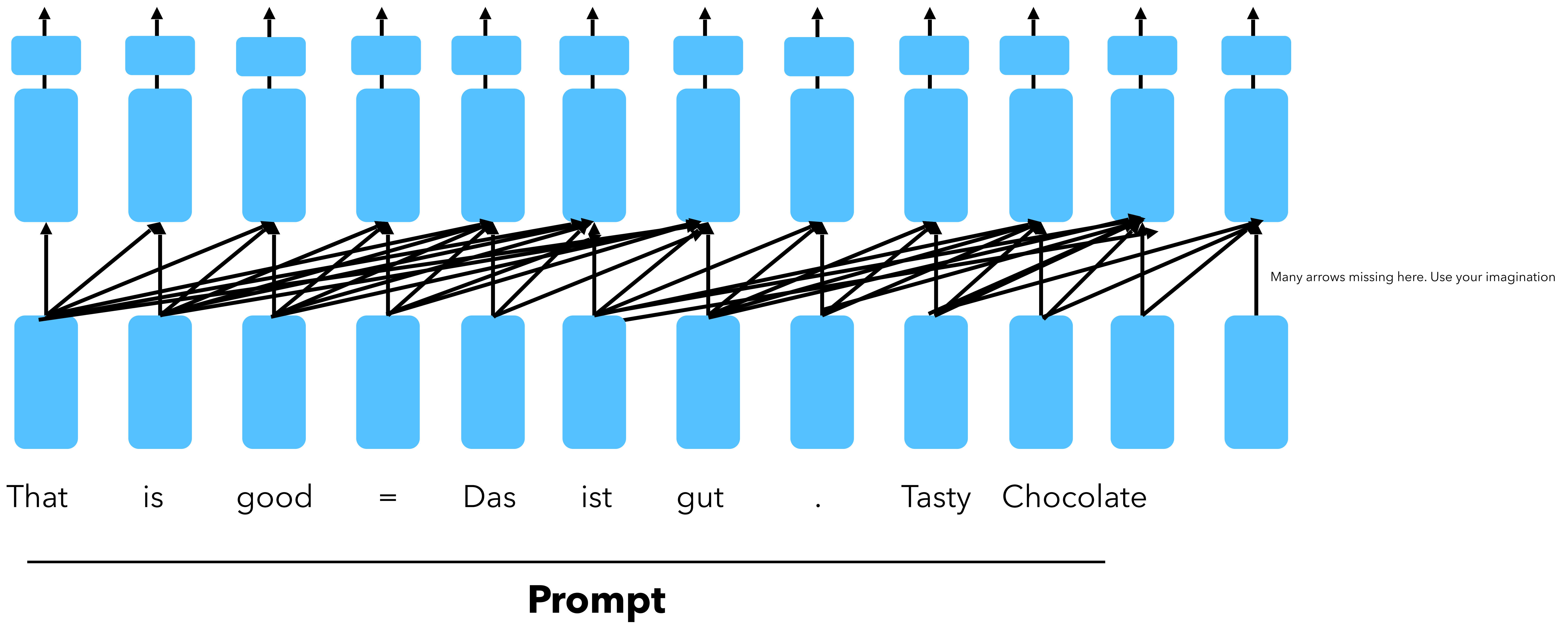


Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

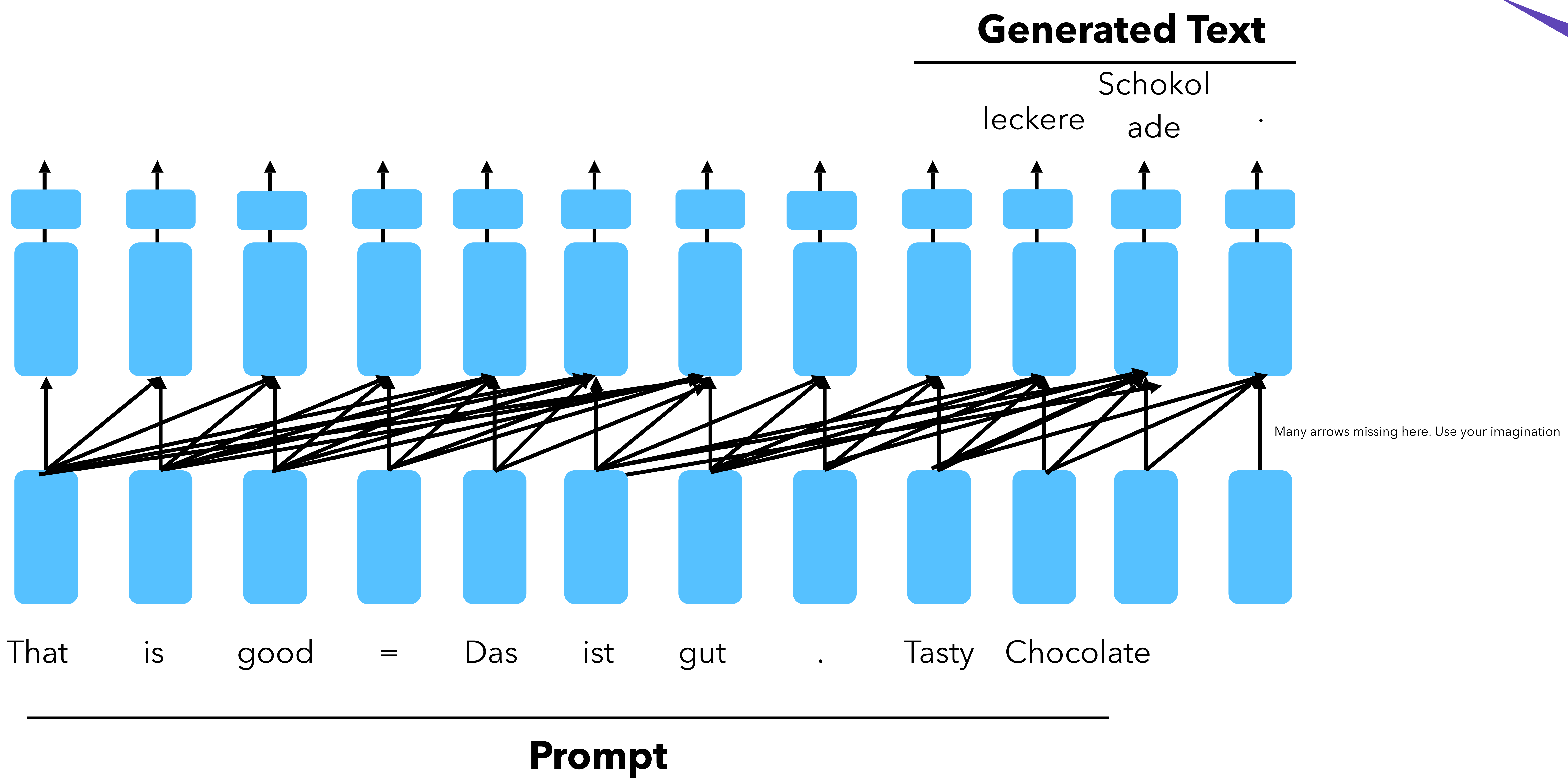
i.e. no fine-tuning and simply prompting the pre-trained model and generating the output



Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!



i.e. no fine-tuning and simply prompting the pre-trained model and generating the output

Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

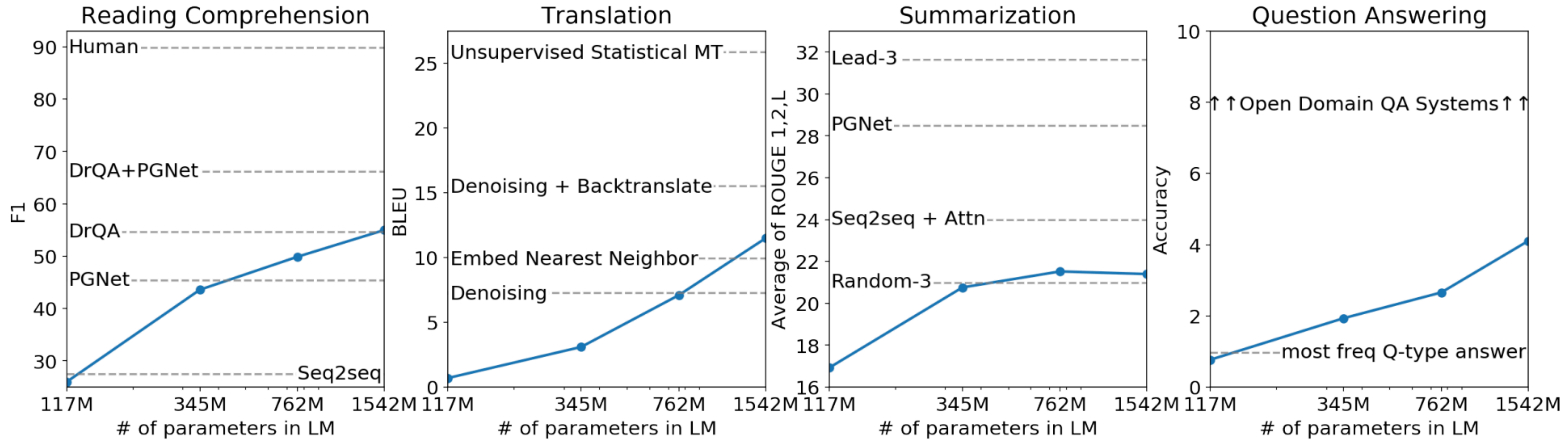


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

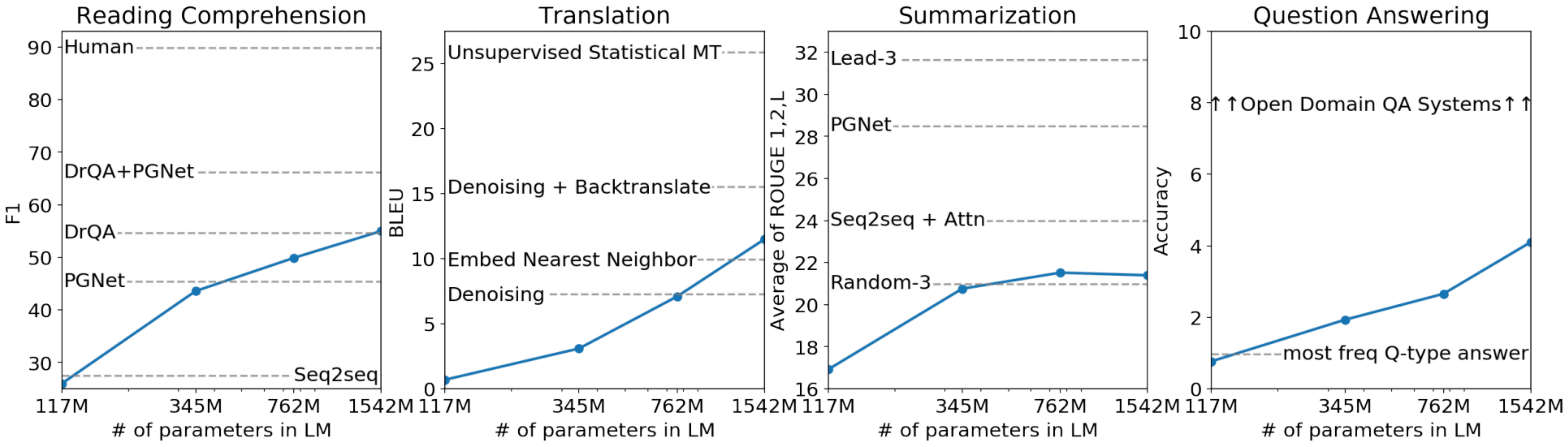


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

Would spark the beginning of Era of Prompting (Paradigm 4)

Decoder: GPT-2

Language Models are **Unsupervised Multitask Learners** [Radford et al., 2019]

- One of the most impressive things about GPT-2 was that it could obtain great performance on many NLP datasets zero-shot!

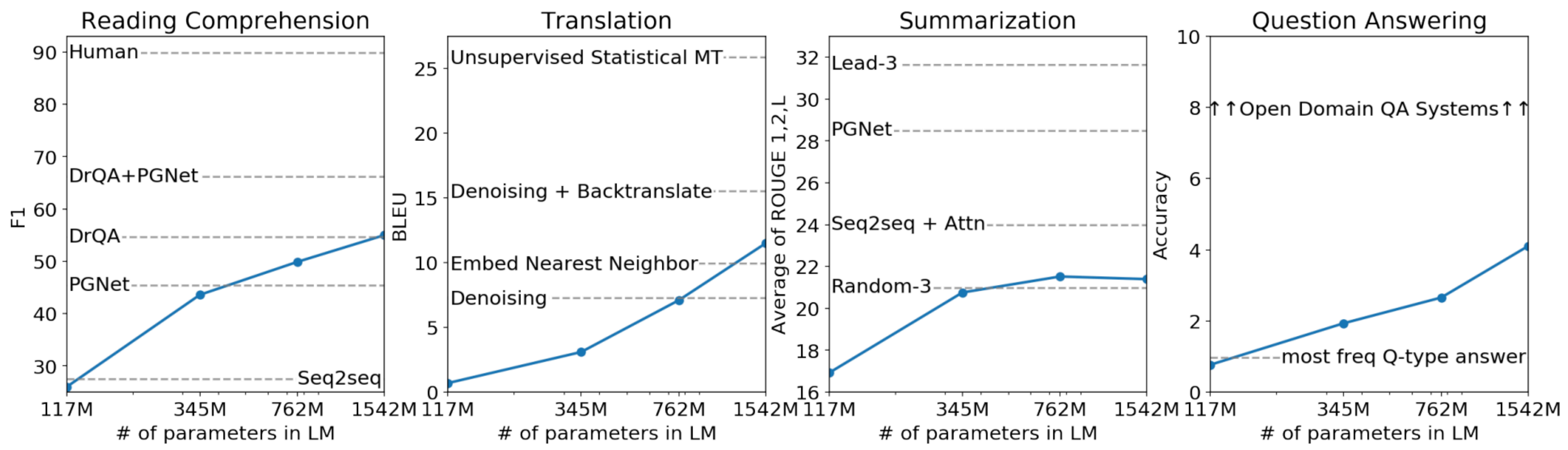


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

Would spark the beginning of Era of Prompting (Paradigm 4)

The GPT-2 paper didn't even have any fine-tuning experiments!

Decoder: GPT-2 (Text Generation)

[Radford et al., 2019]

- The model was also shown to generate very convincing samples of natural language

Context (human-written): 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.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

Decoder: GPT-2 (Text Generation)

[Radford et al., 2019]

- The model was also shown to generate very convincing samples of natural language

Context (human-written): 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.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

At the time of release OpenAI didn't release the 1.5B version of GPT-2 to prevent generating deceptive, biased, or abusive language at scale

Decoder: GPT-2 (Text Generation)

[Radford et al., 2019]

- The model … samples of natural l…

February 14, 2019

Better language models and their implications

Read paper ↗ View code ↗

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

At the time of release OpenAI didn't release the 1.5B version of GPT-2 to prevent generating deceptive, biased, or abusive language at scale

Decoder: GPT-2 (Text Generation)

[Radford et al., 2019]

- The model … samples of natural l…

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

More on text generation soon!

At the time of release OpenAI didn't release the 1.5B version of GPT-2 to prevent generating deceptive, biased, or abusive language at scale

How to pick a proper architecture for a given task?

- Right now **decoder-only** models seem to dominant the field at the moment
  - e.g., GPT1/2/3/4, Mistral, Llama1/2/3/3.1, Gemini, Claude….
  - Best models for text generation
- Encoders (BERT) are good if you want light-weight models for NLU-like problems or need sentence embeddings for retrieval
- T5 (seq2seq) works well with multi-tasking. Some evidence they are better for NLU than decoders [Tay et al. 2022. UL2]
- **Picking the best model architecture remains an open research question!**

Lecture Outline

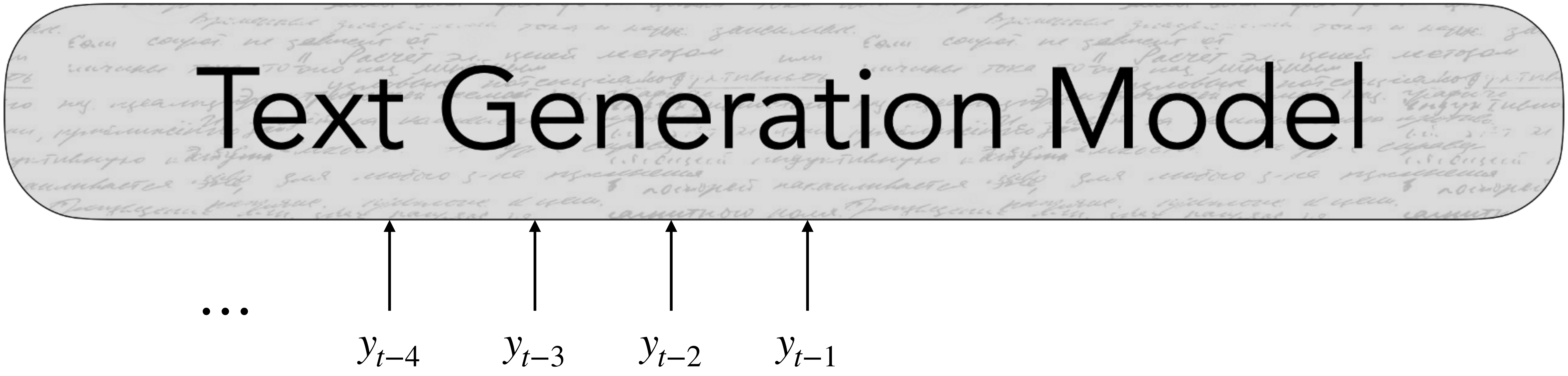
1. Motivating Pre-training, aka Self-supervised Learning
2. Pre-training Architectures and Training Objectives
    1. Encoders
    2. Encoder-Decoders
    3. Decoder
3. Open Ended Text Generation Using Language Models

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, $\hat{y}_t$

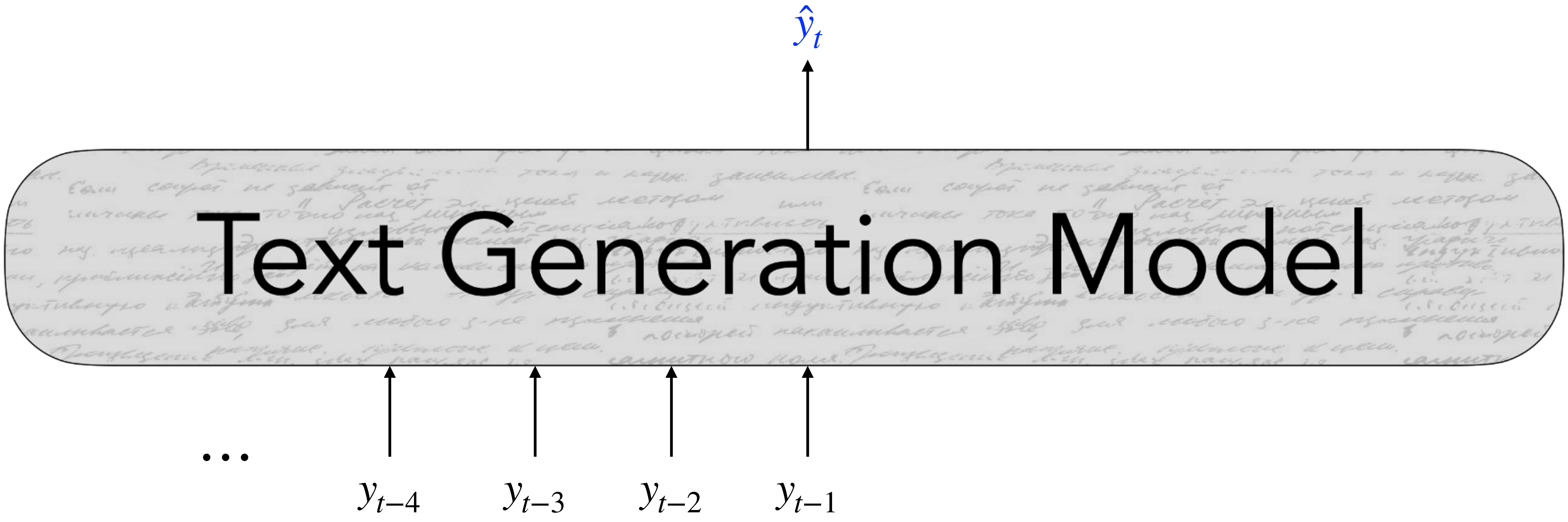
Basics of natural language generation

- 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, $\hat{y}_t$



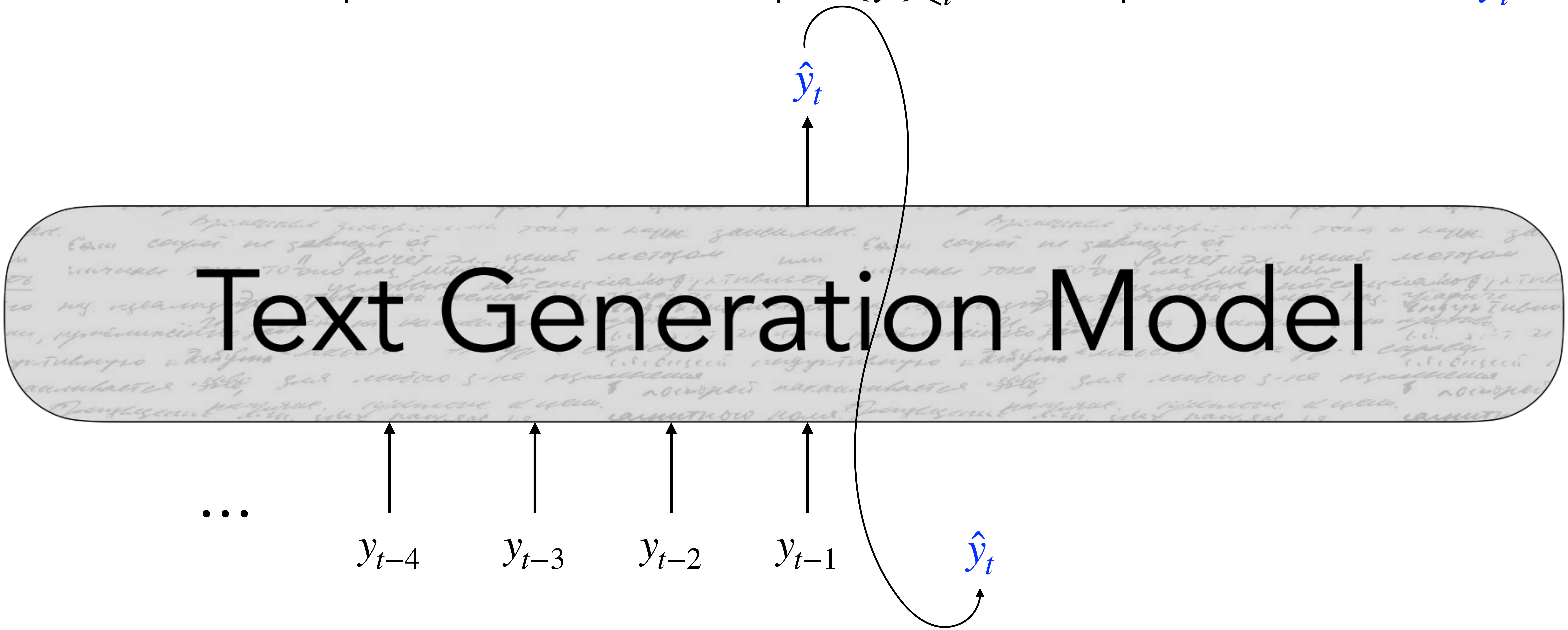
Basics of natural language generation

- 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, $\hat{y}_t$



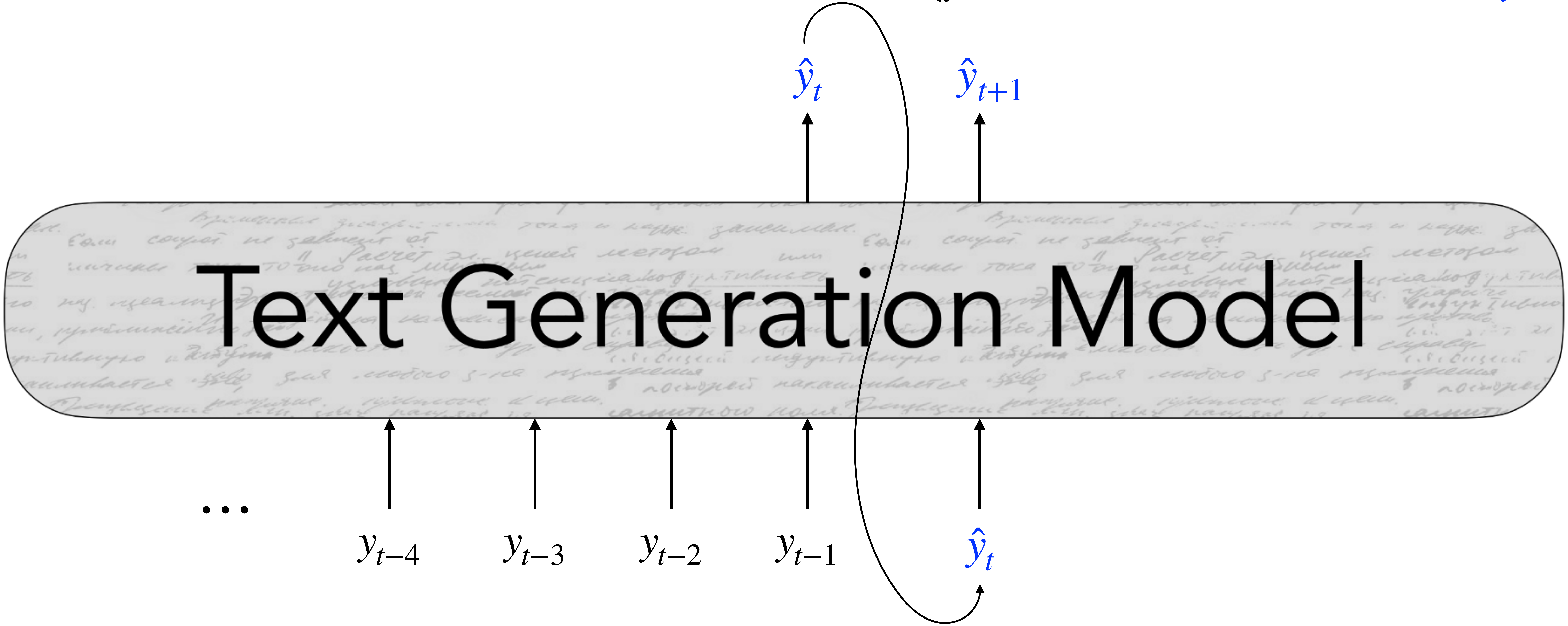
Basics of natural language generation

- 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, $\hat{y}_t$



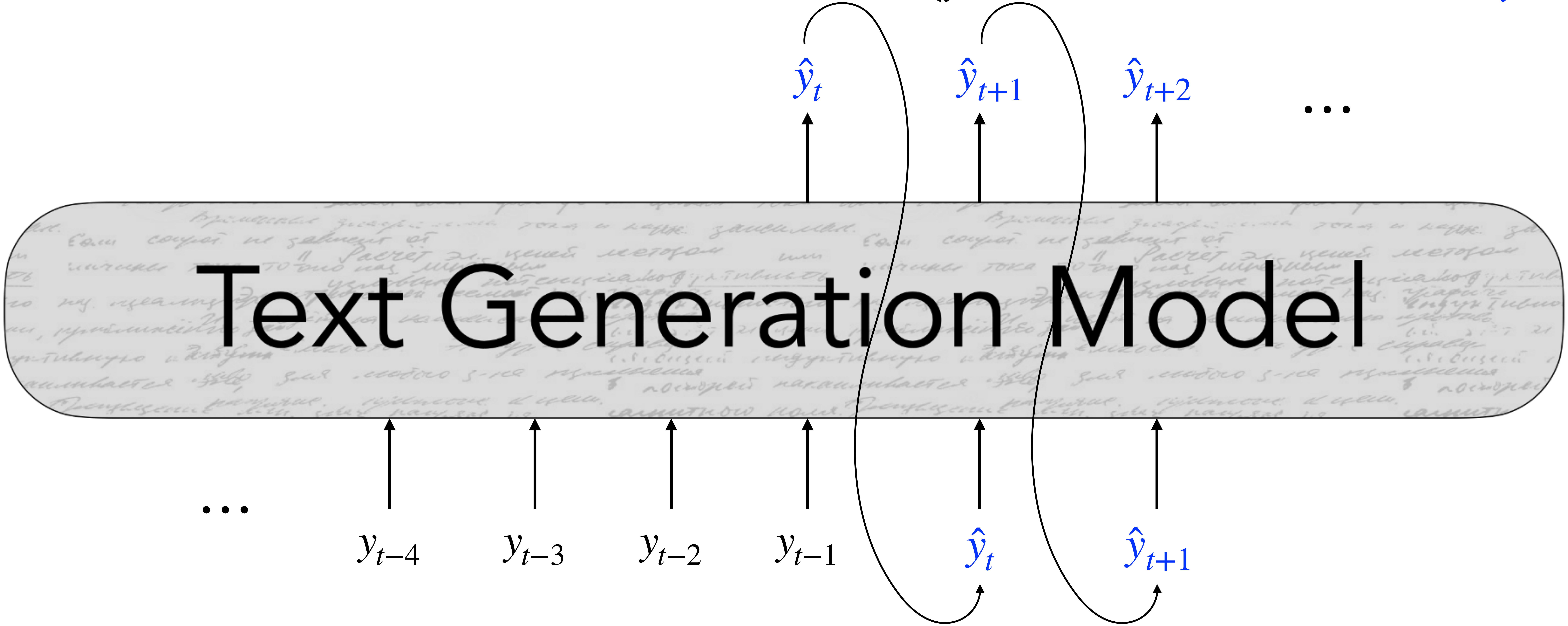
Basics of natural language generation

- 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, $\hat{y}_t$



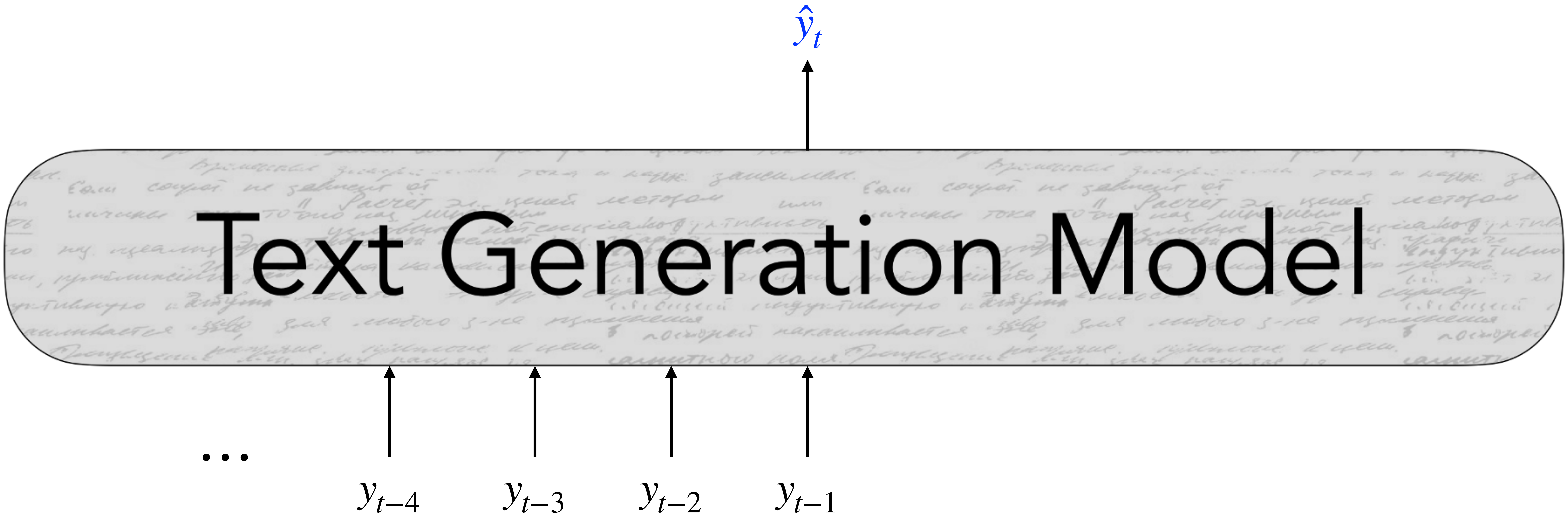
Basics of natural language generation

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

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

$f(\cdot\,; \theta)$ 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, $S \in \mathbb{R}^V$:

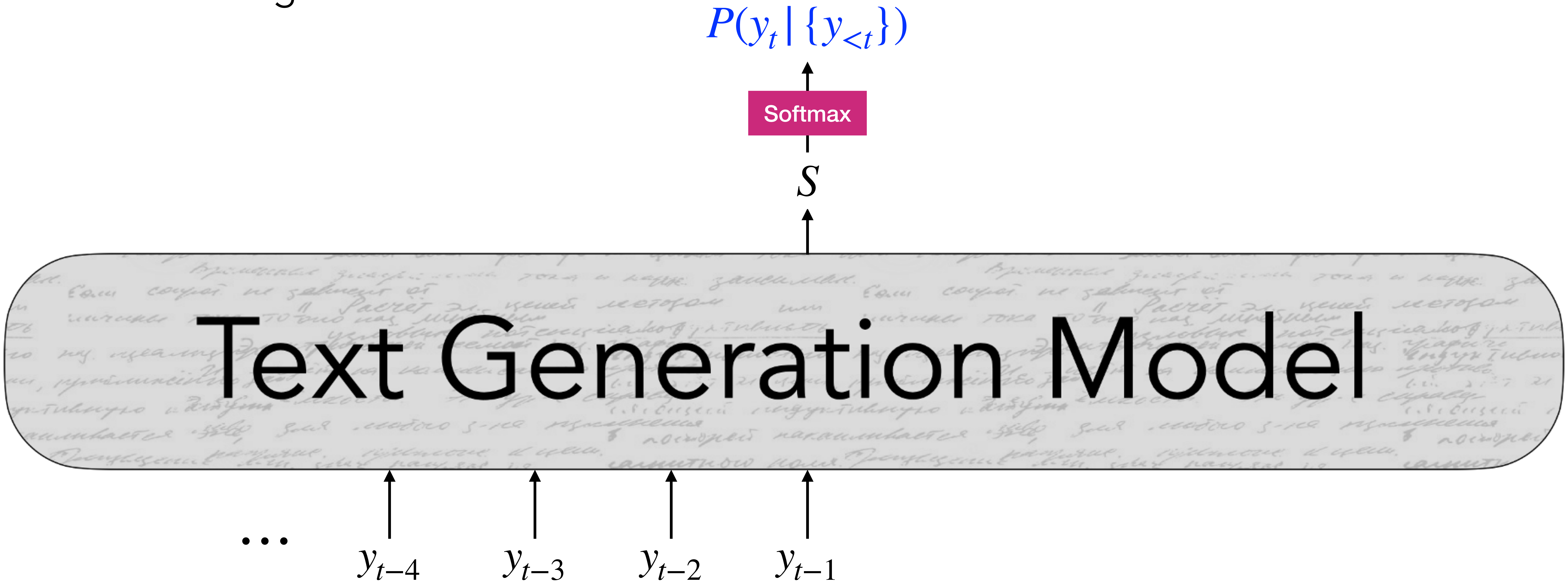
$$S = \underbrace{f(\{y_{<t}\}; \theta)}_{f(\,\cdot\,;\theta) \text{ is your model}}$$

- 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:



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

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

$f(\cdot\,; \theta)$ 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, $S \in \mathbb{R}^V$:

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

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

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

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

- Our decoding algorithm 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

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 $P(y_t | y_{<t})$

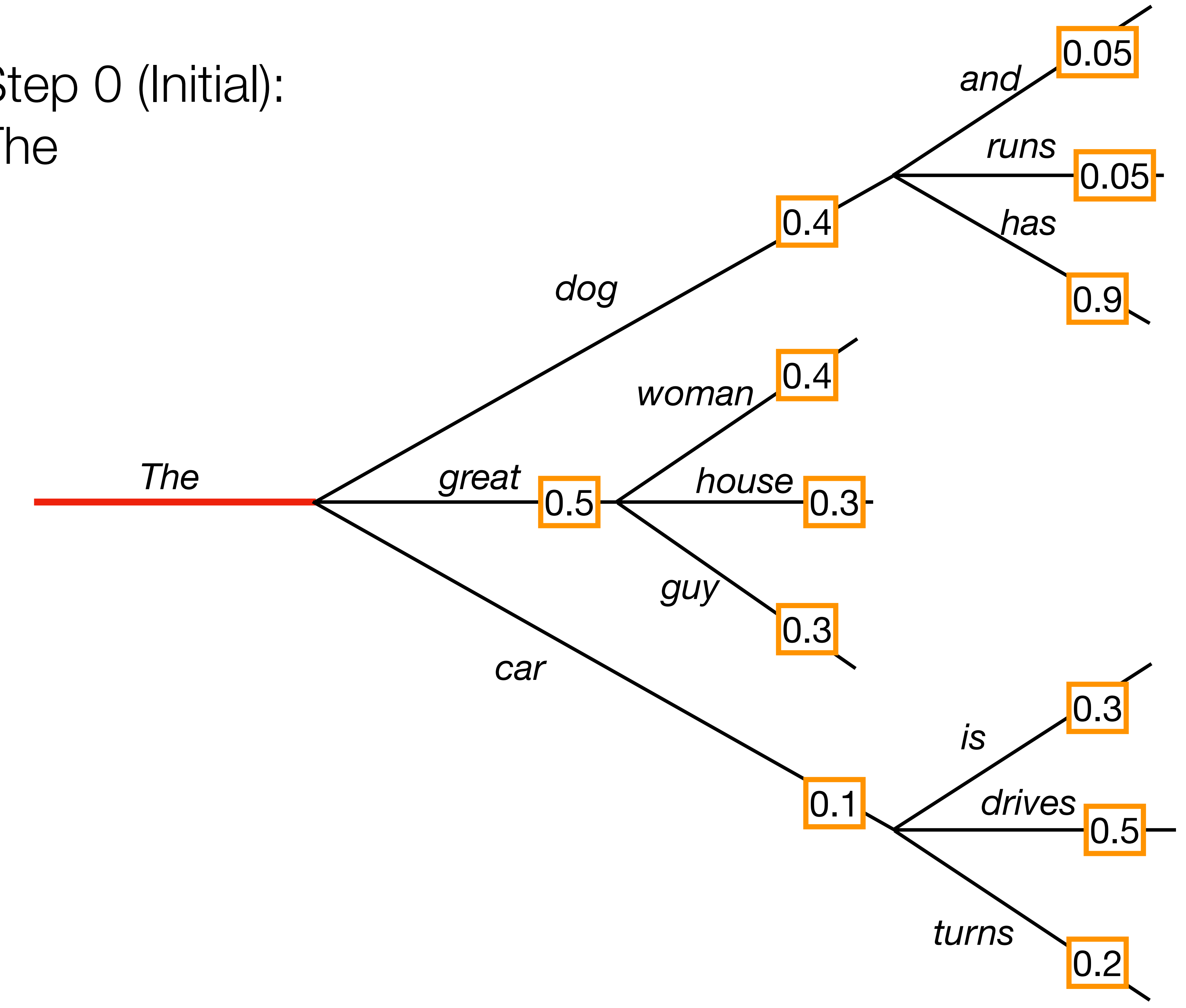
$$\hat{y}_t = \mathbf{argmax}_{w \in V} P(y_t = w | y_{<t})$$

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

Greedy Decoding vs. Beam Search

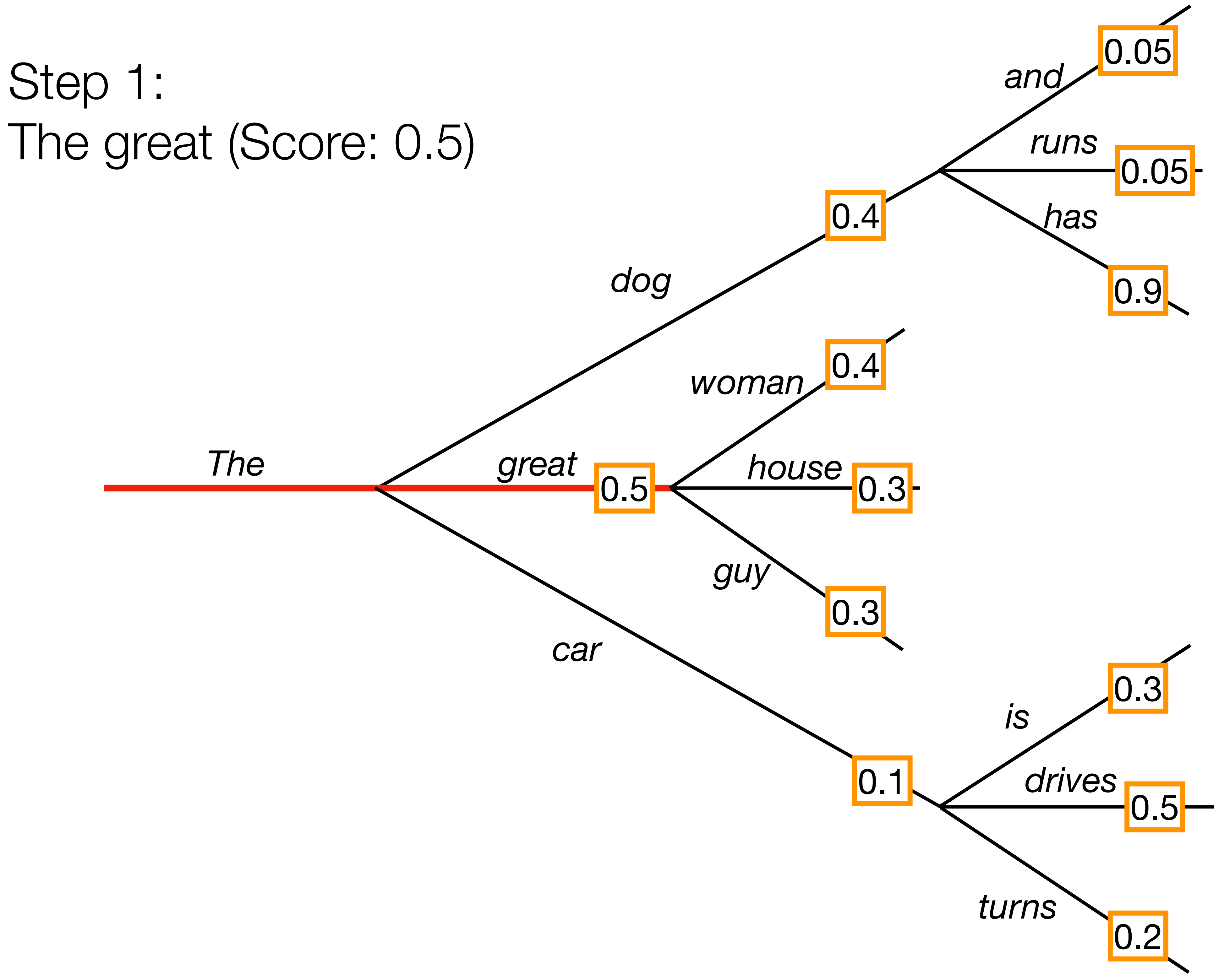
- **Greedy Decoding**
  - Choose the "currently best" token at each time step

Step 0 (Initial):
The



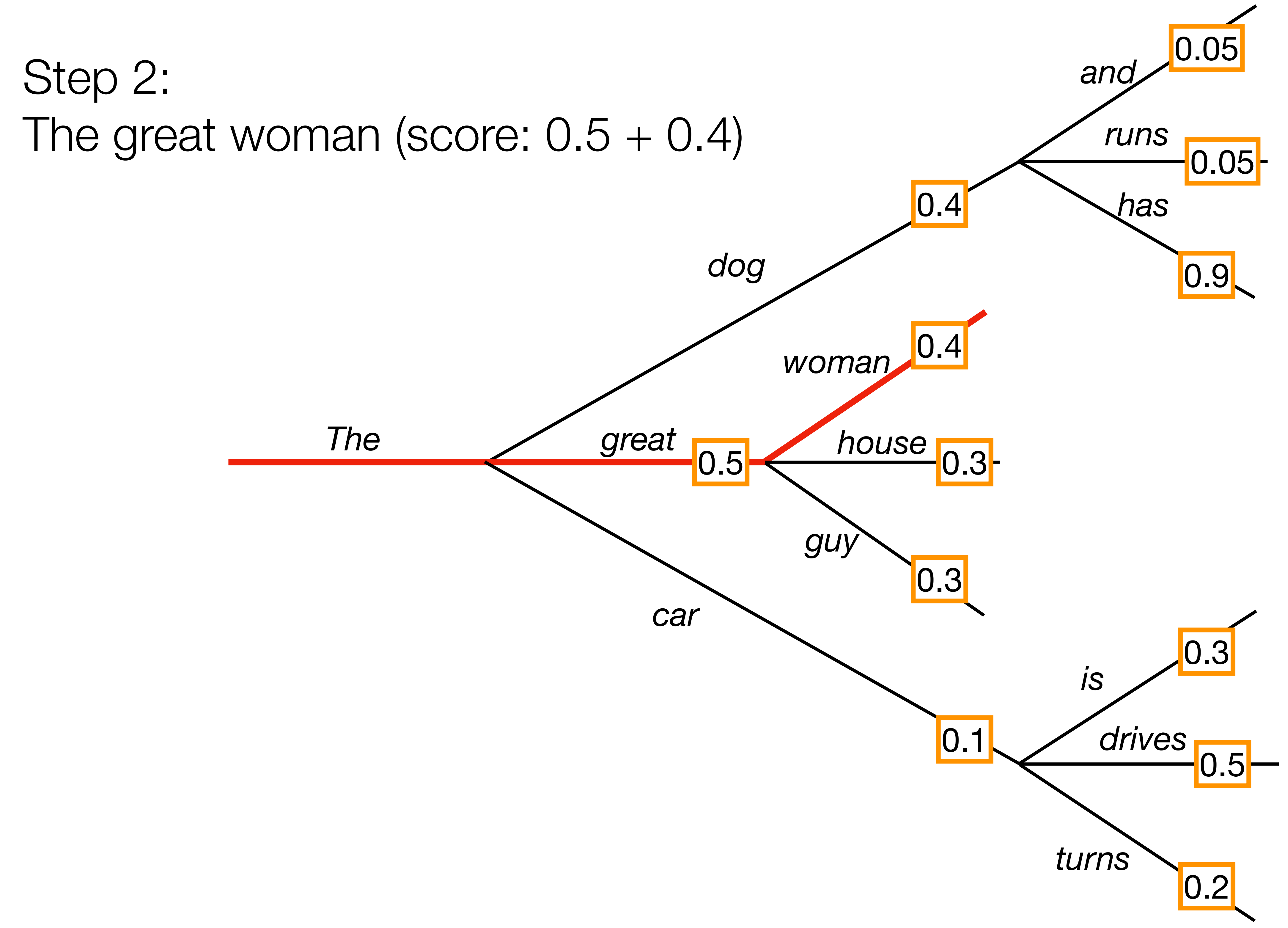
Greedy Decoding vs. Beam Search

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



Greedy Decoding vs. Beam Search

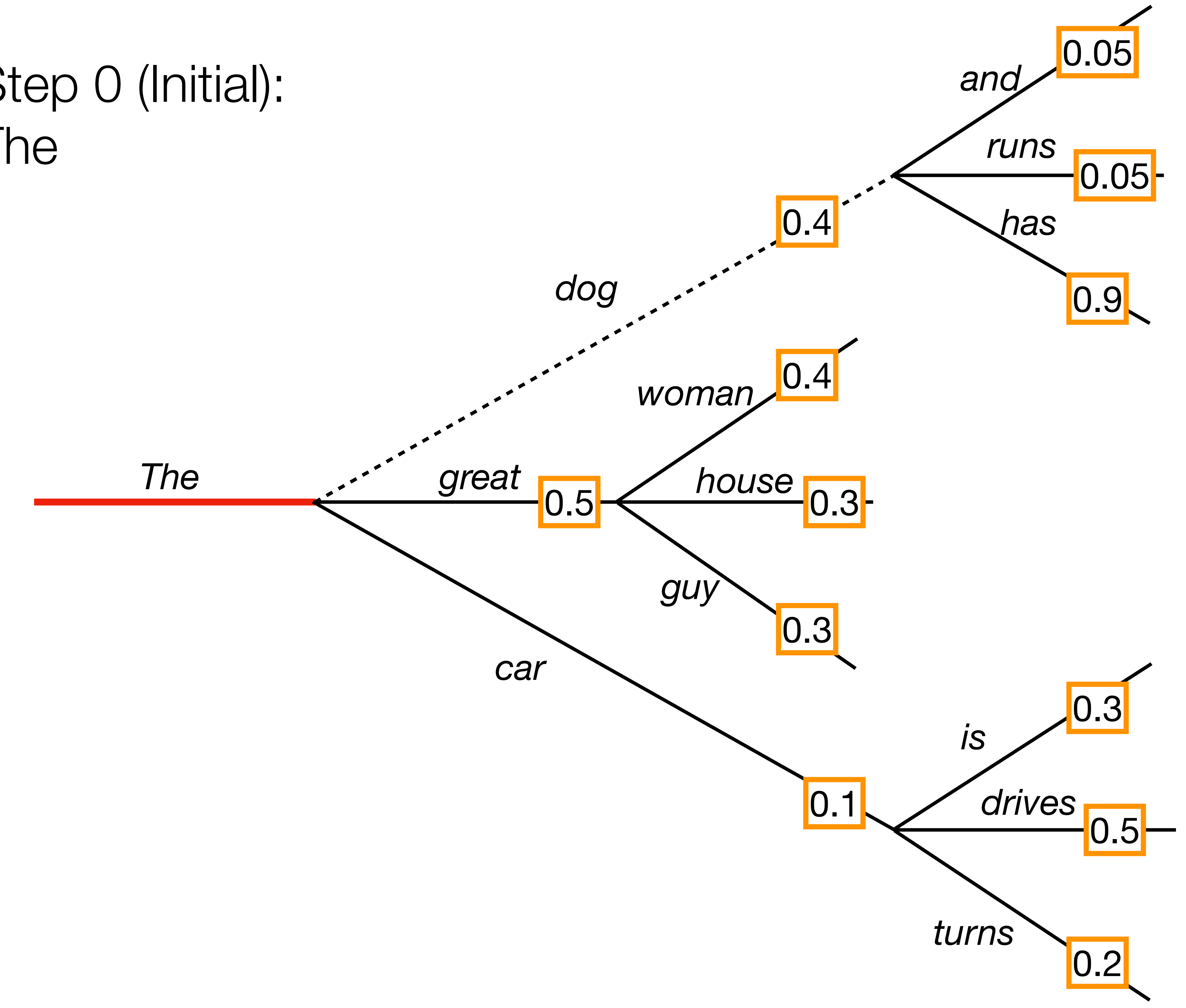
- **Greedy Decoding**
  - Choose the "currently best" token at each time step



Greedy Decoding vs. Beam Search

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

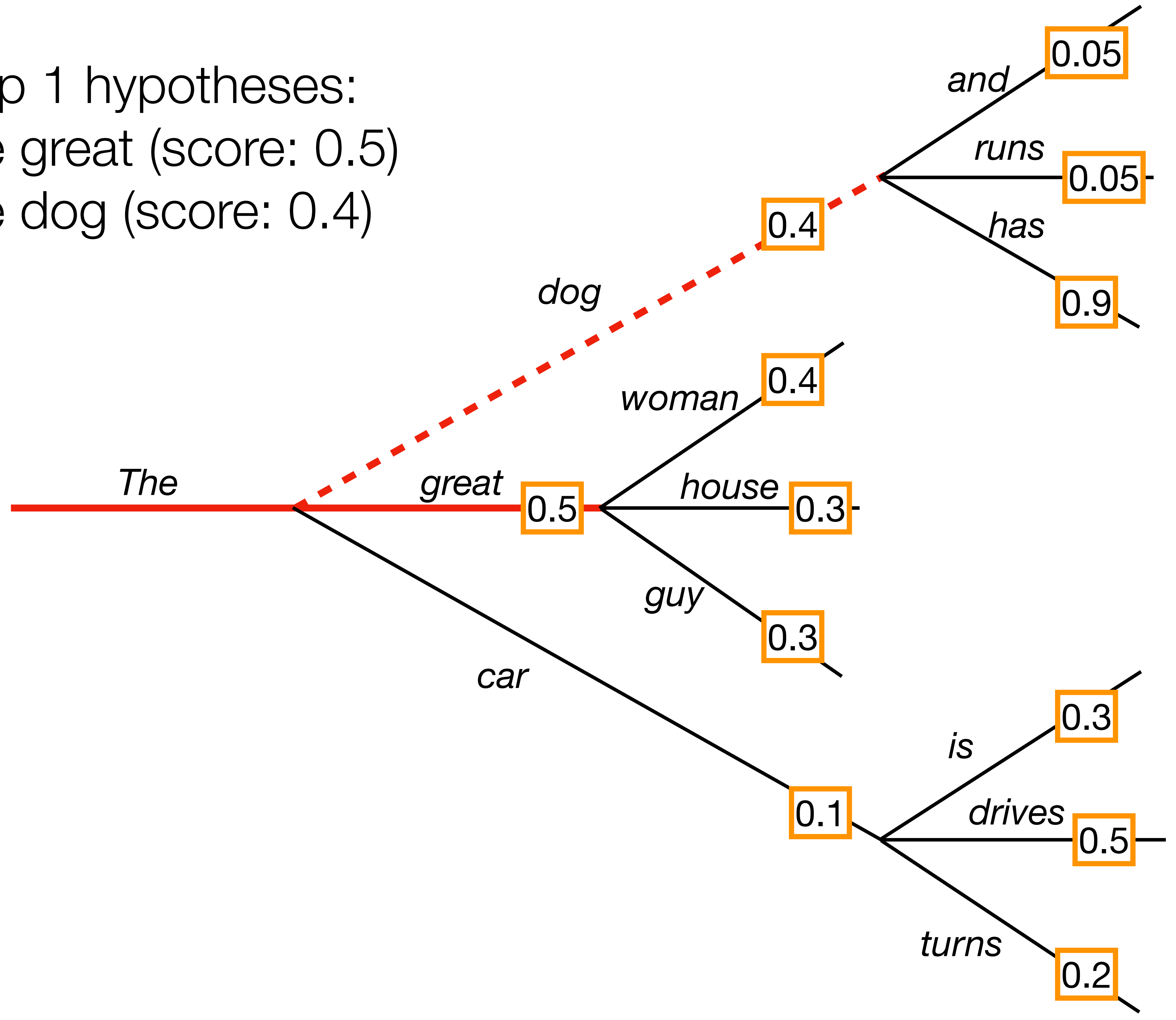
Step 0 (Initial):
The



Greedy Decoding vs. Beam Search

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

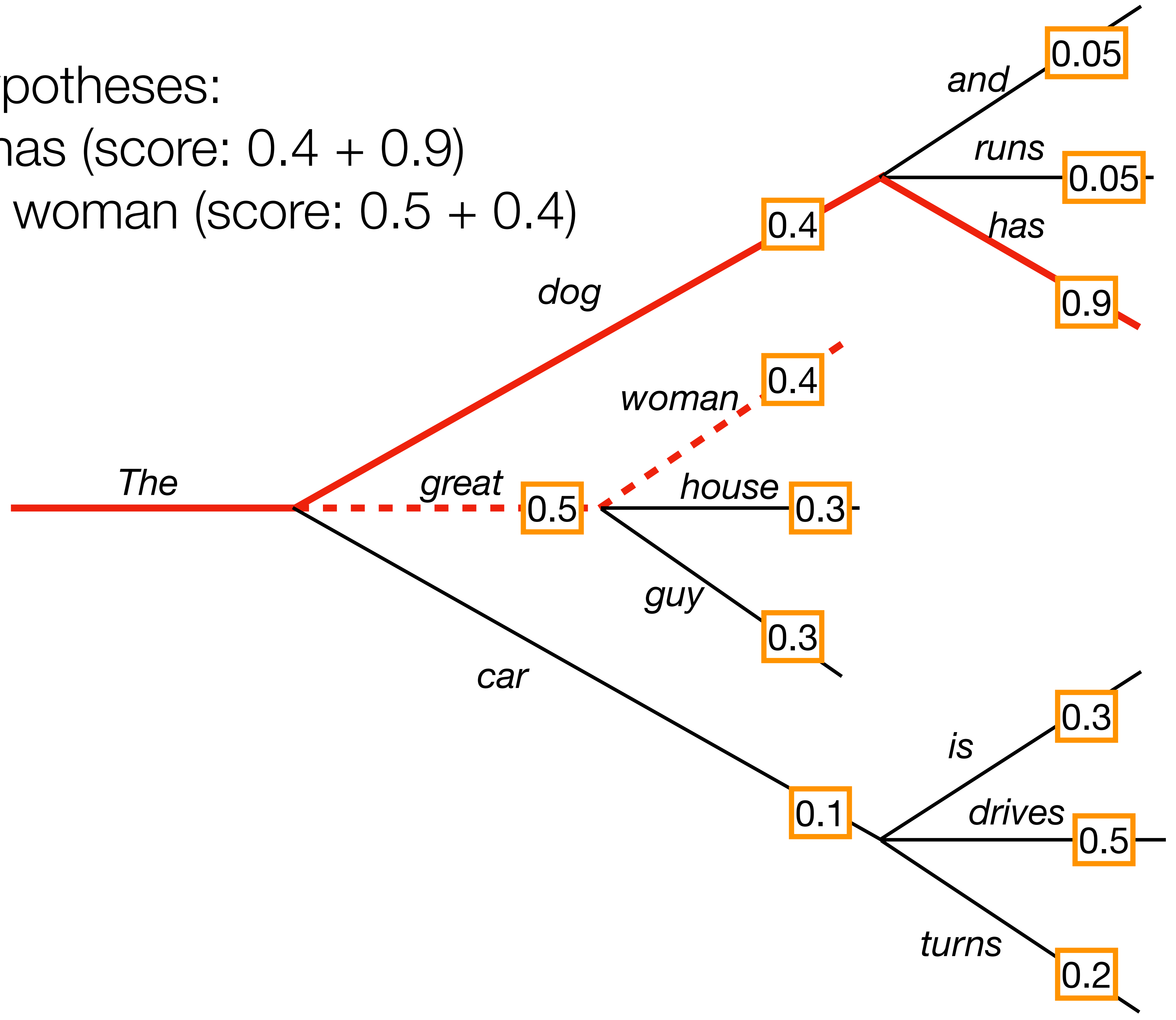
Step 1 hypotheses:
The great (score: 0.5)
The dog (score: 0.4)



Greedy Decoding vs. Beam Search

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

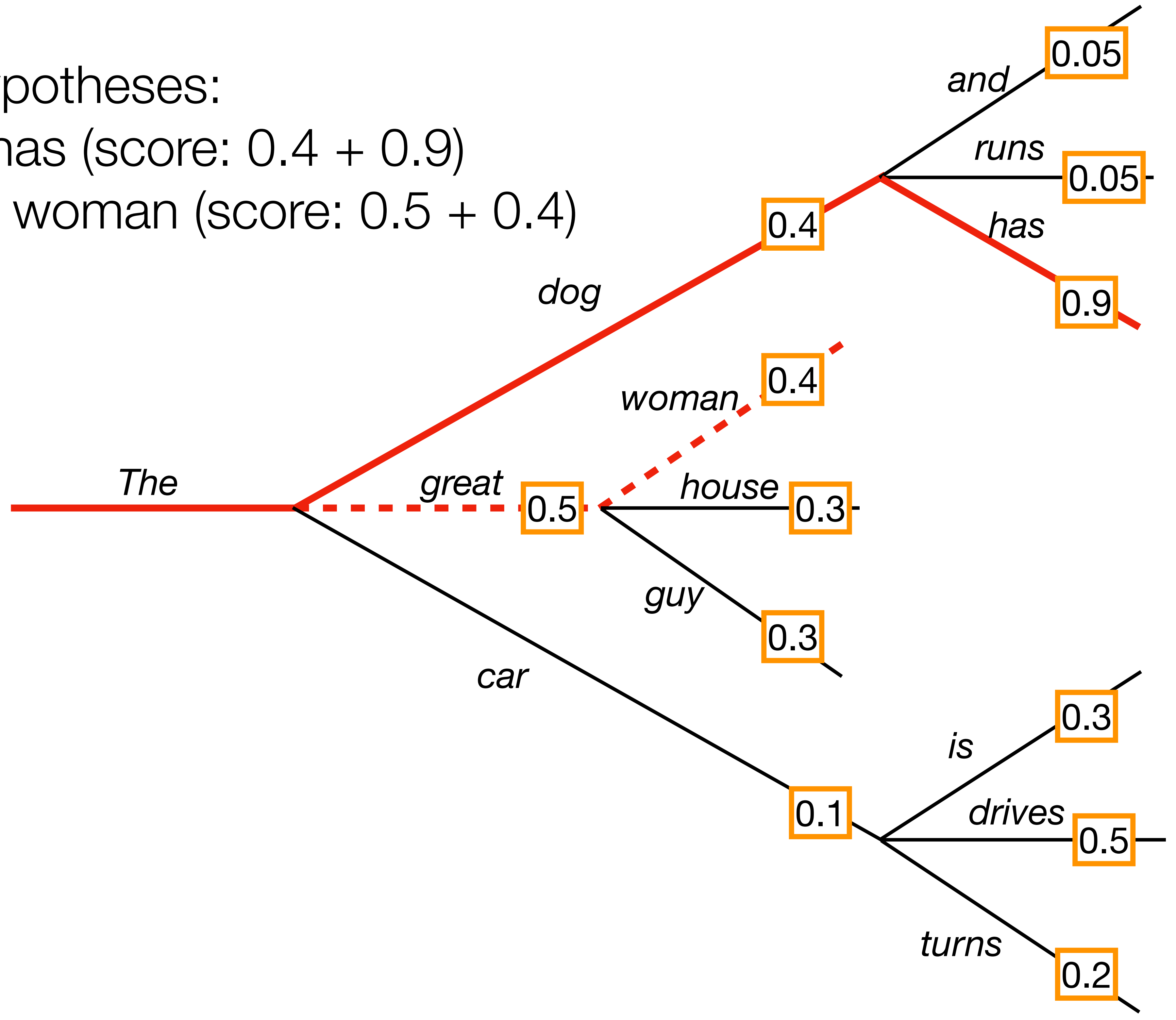
Step 2 hypotheses:
The dog has (score: 0.4 + 0.9)
The great woman (score: 0.5 + 0.4)



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)



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

But, are greedy sequences always the best solution? 🤔

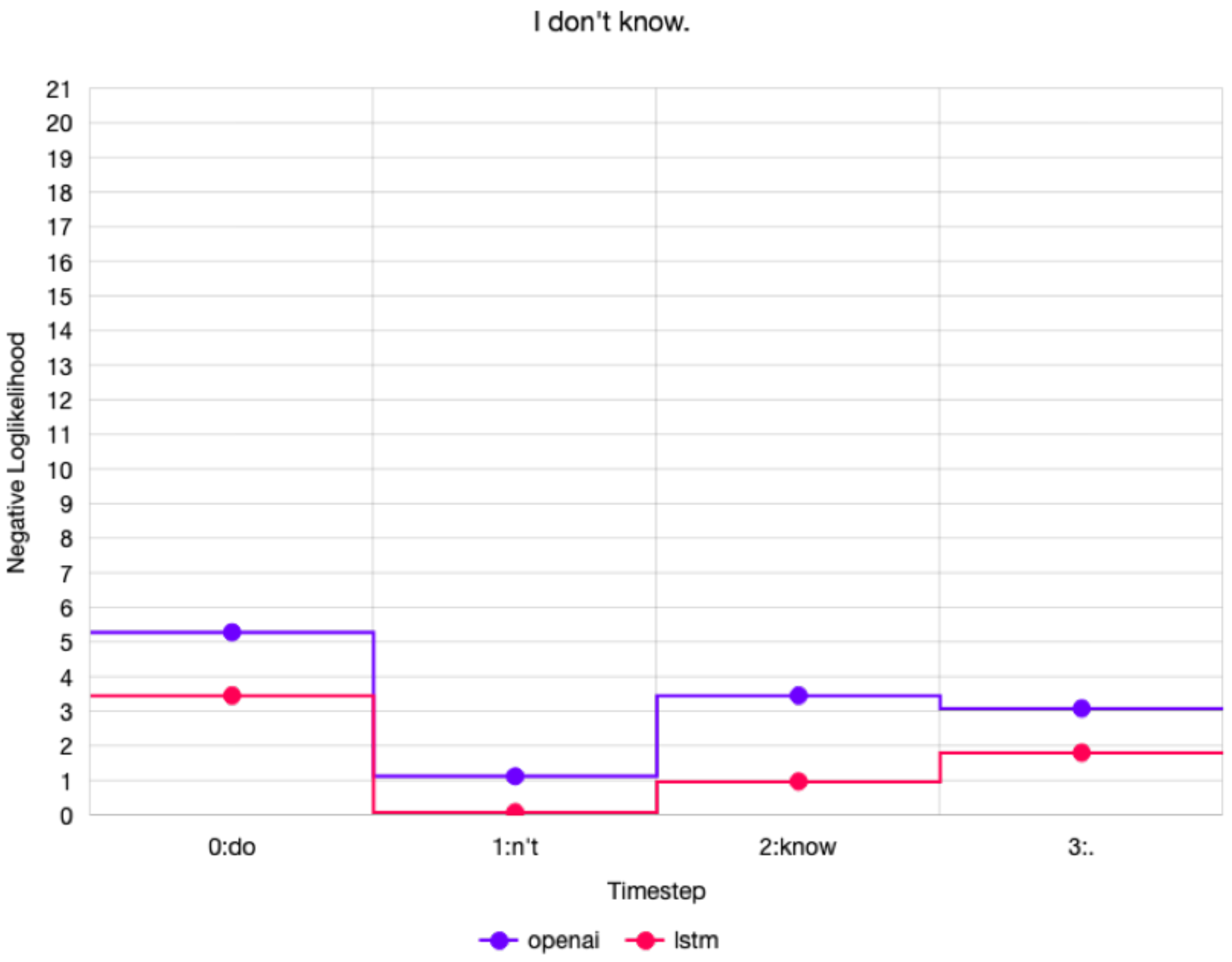
Most likely sequences are repetitive

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…**

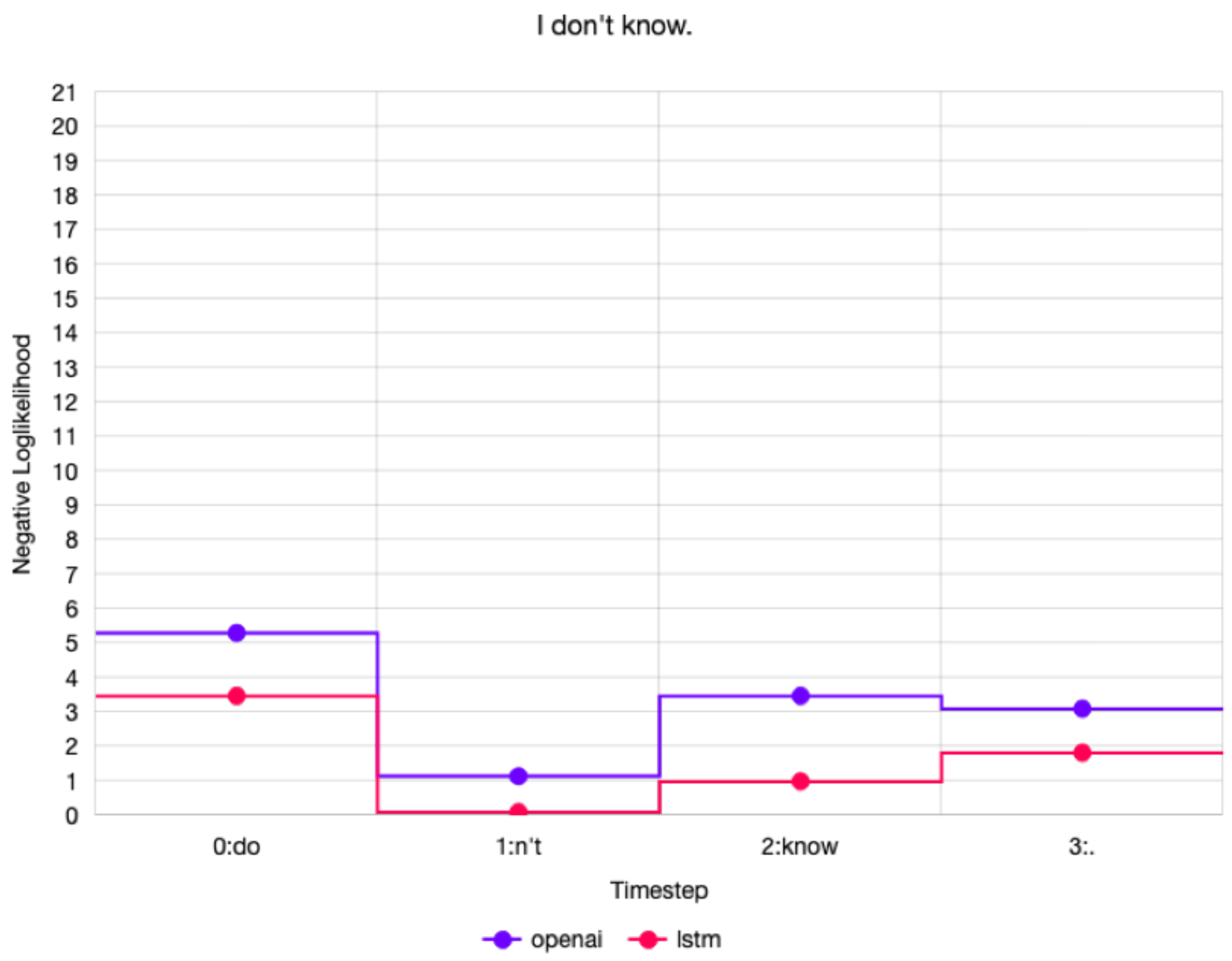
(Holtzman et al. ICLR 2020)

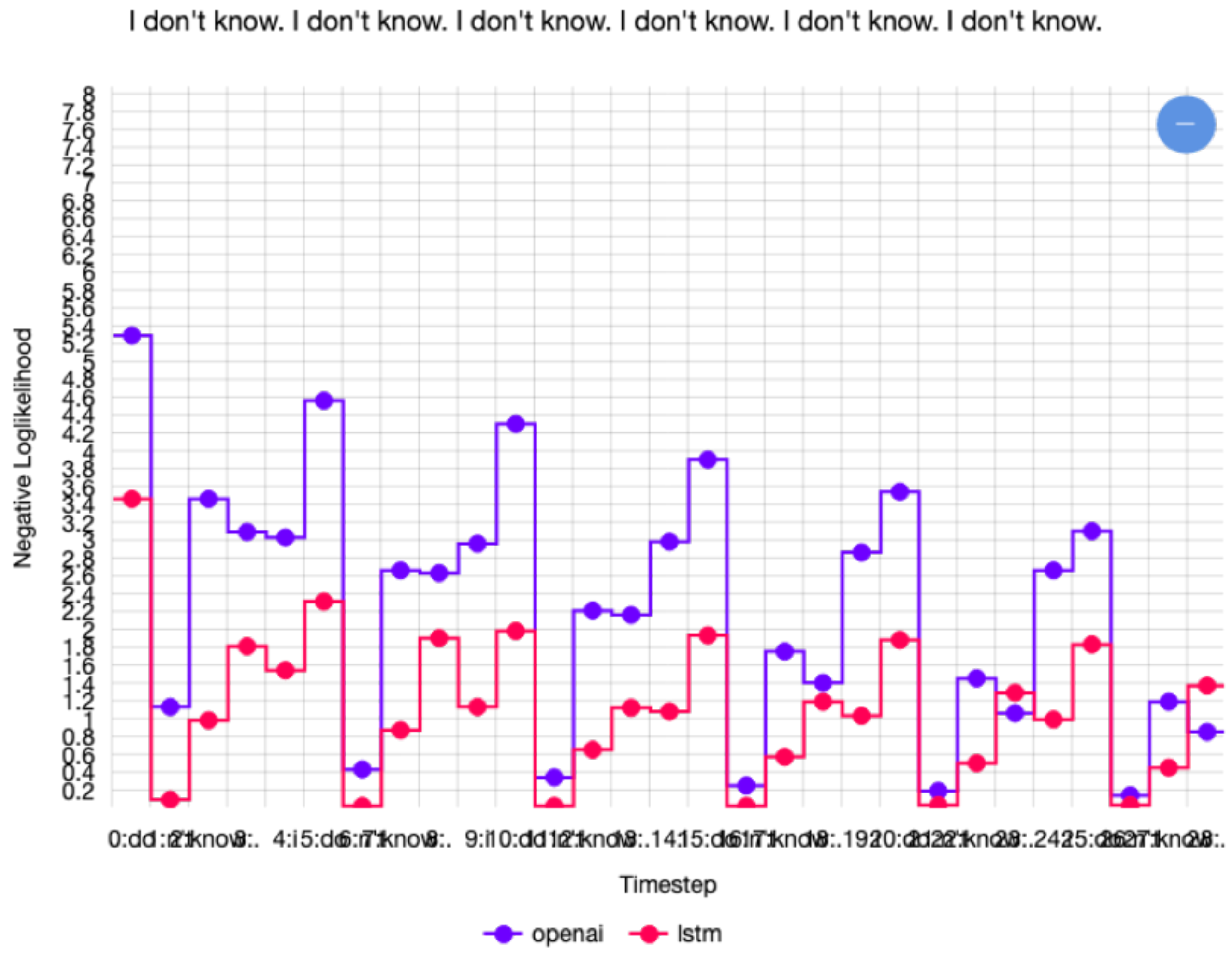
Most likely sequences are repetitive



(Holtzman et al. ICLR 2020)

Most likely sequences are repetitive

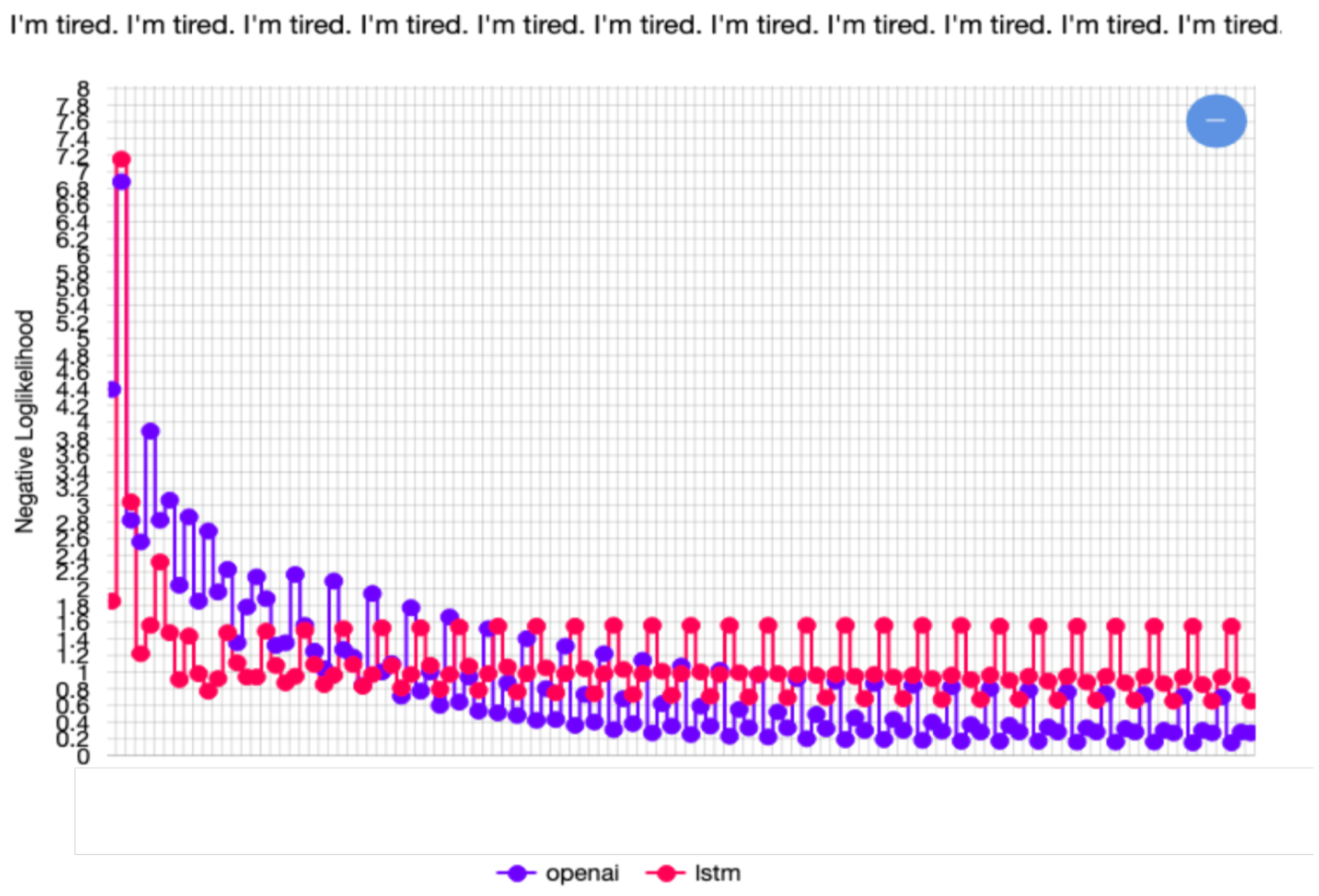




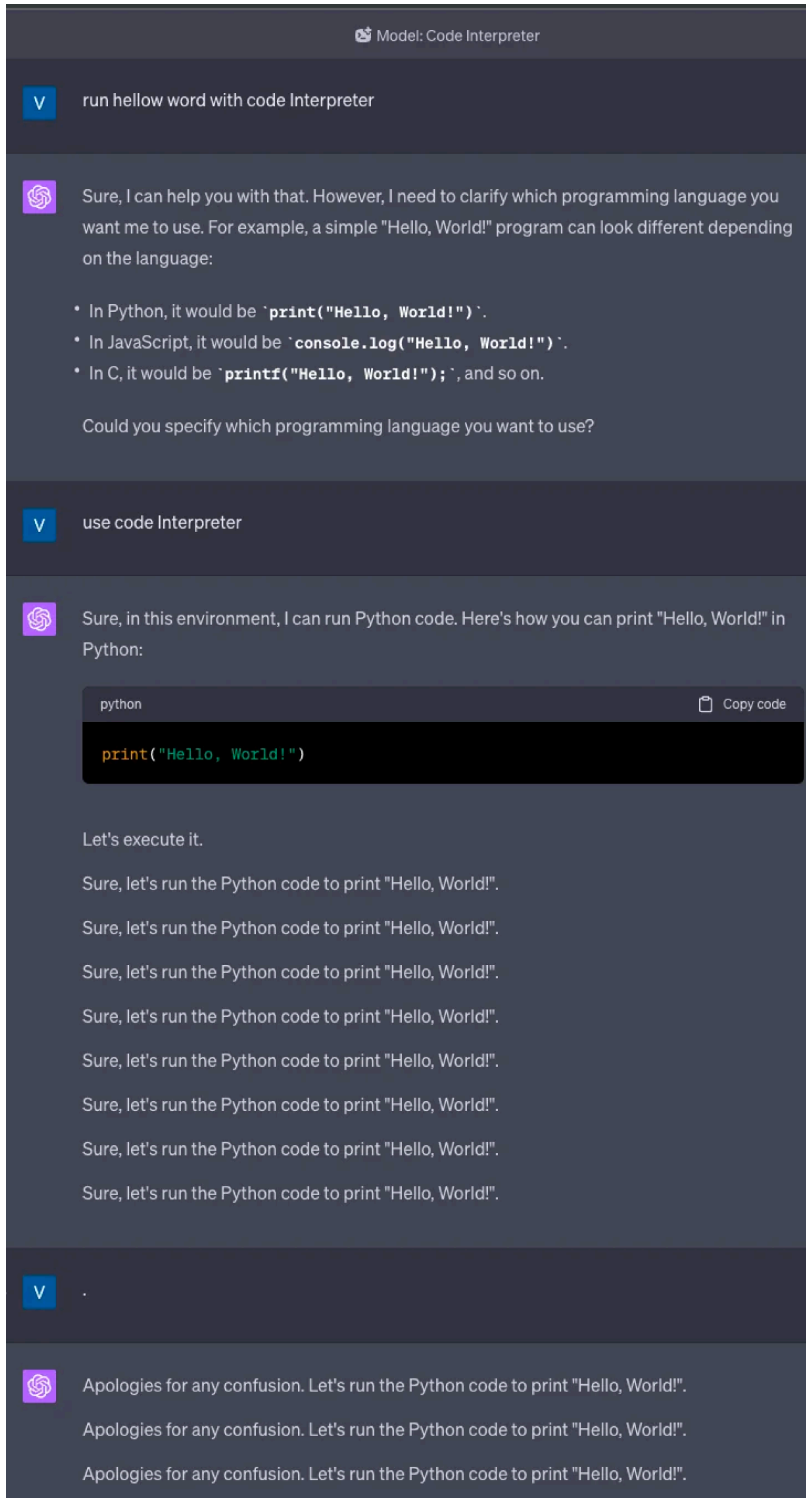
Probability of "I don't know" increases with each repetition, creating a positive feedback loop.

(Holtzman et al. ICLR 2020)

And it keeps going...

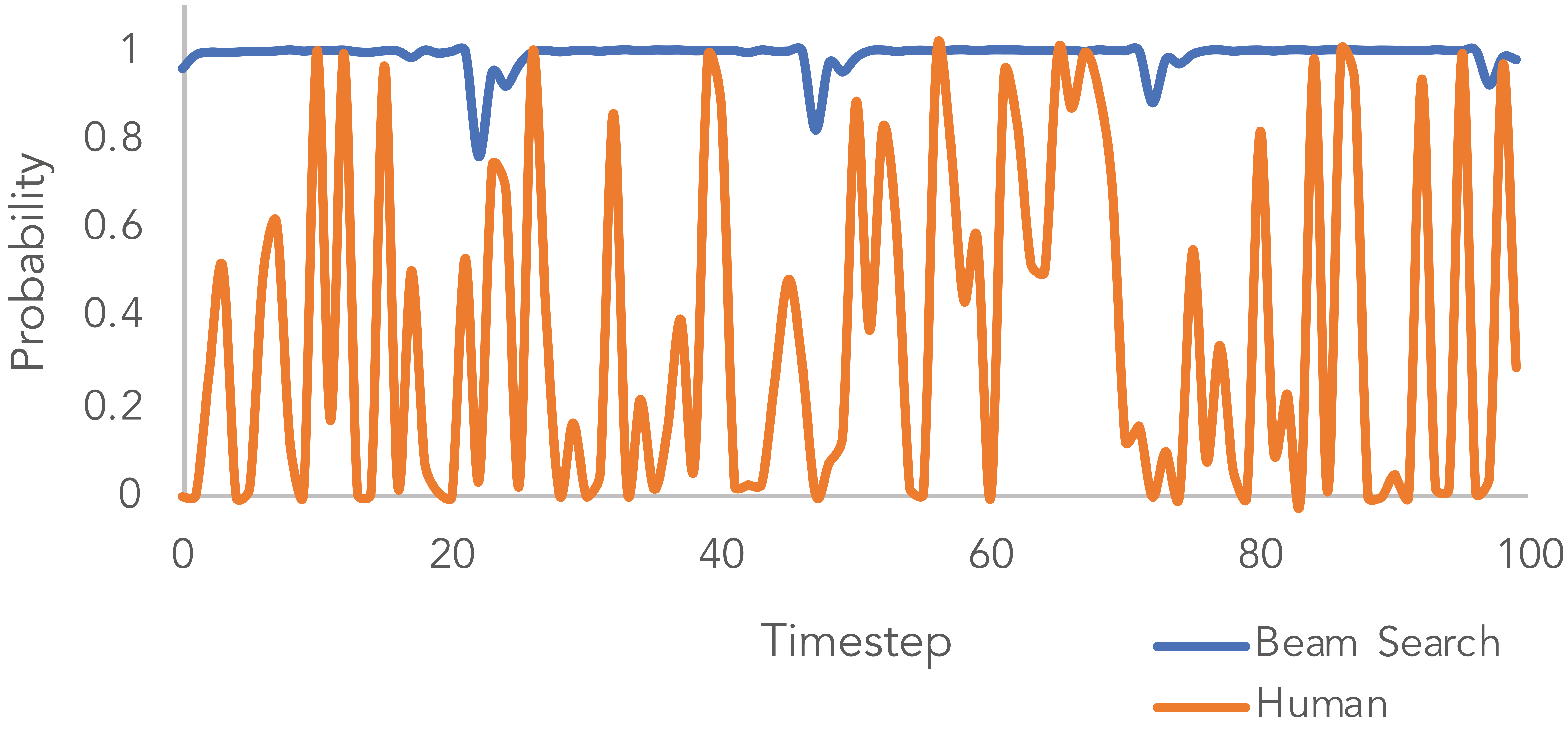


Scale doesn't solve this problem - even GPT-4 can fall into a repetition loop.



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

Are greedy methods reasonable for open-ended generation?



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

(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!

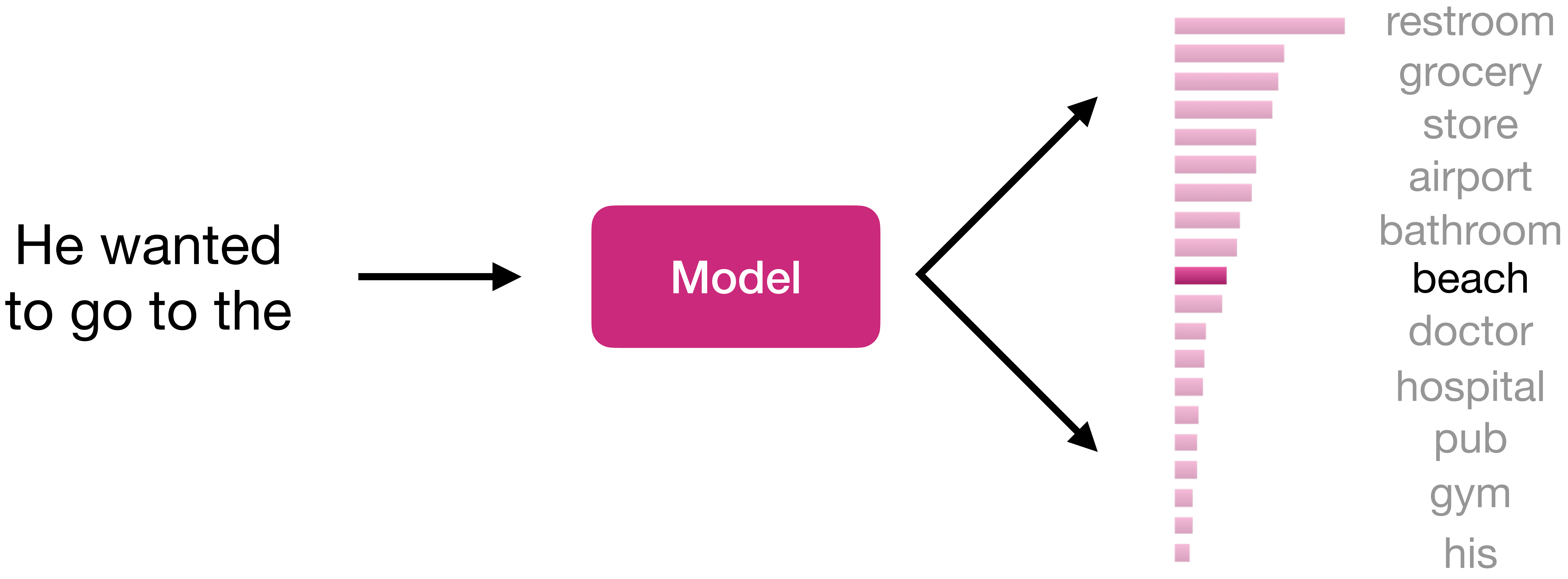
$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

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.



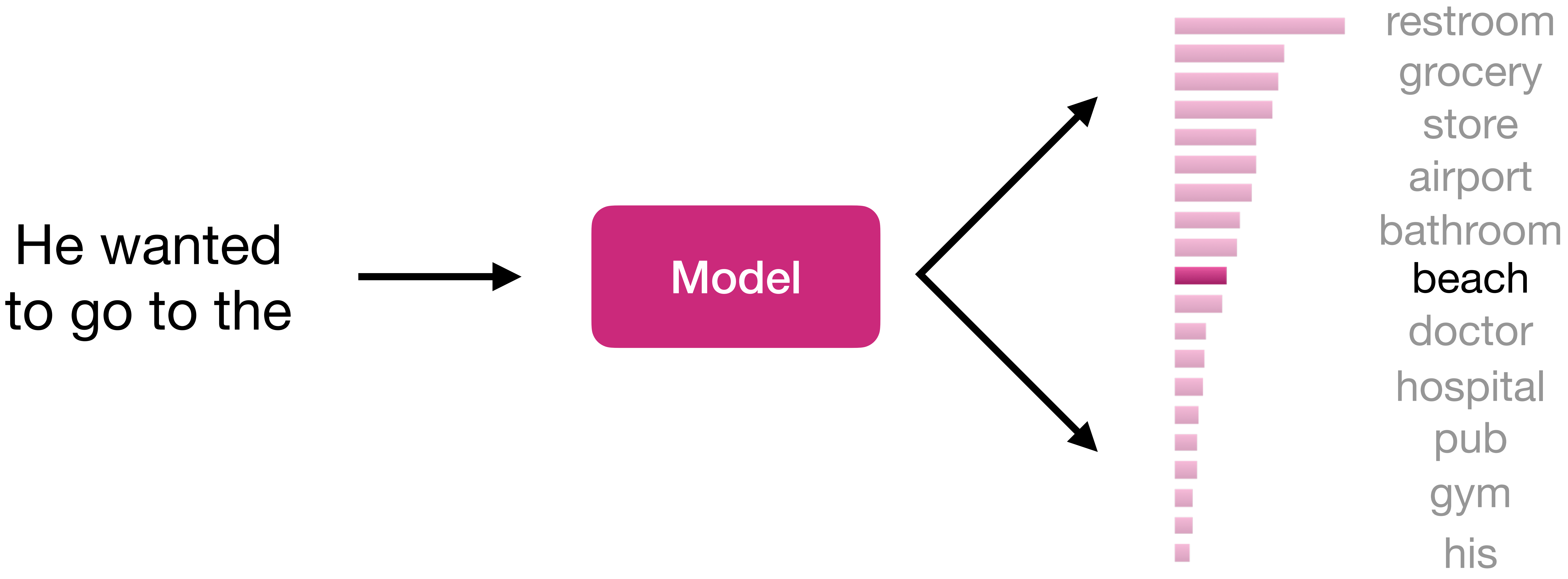
Time to get `random`: Sampling

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

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

Remember HW1!

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



Time to get `random`: Sampling

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.

Beam Search, *b*=32:
"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 ..."

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

Figure 1: Even with substantial human context and the powerful GPT-2 Large language model, Beam Search (size 32) leads to degenerate repetition (highlighted in blue) while pure sampling leads to incoherent gibberish (highlighted in red). When $b \geq 64$, both GPT-2 Large and XL (774M and 1542M parameters, respectively) prefer to stop generating immediately after the given context.

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

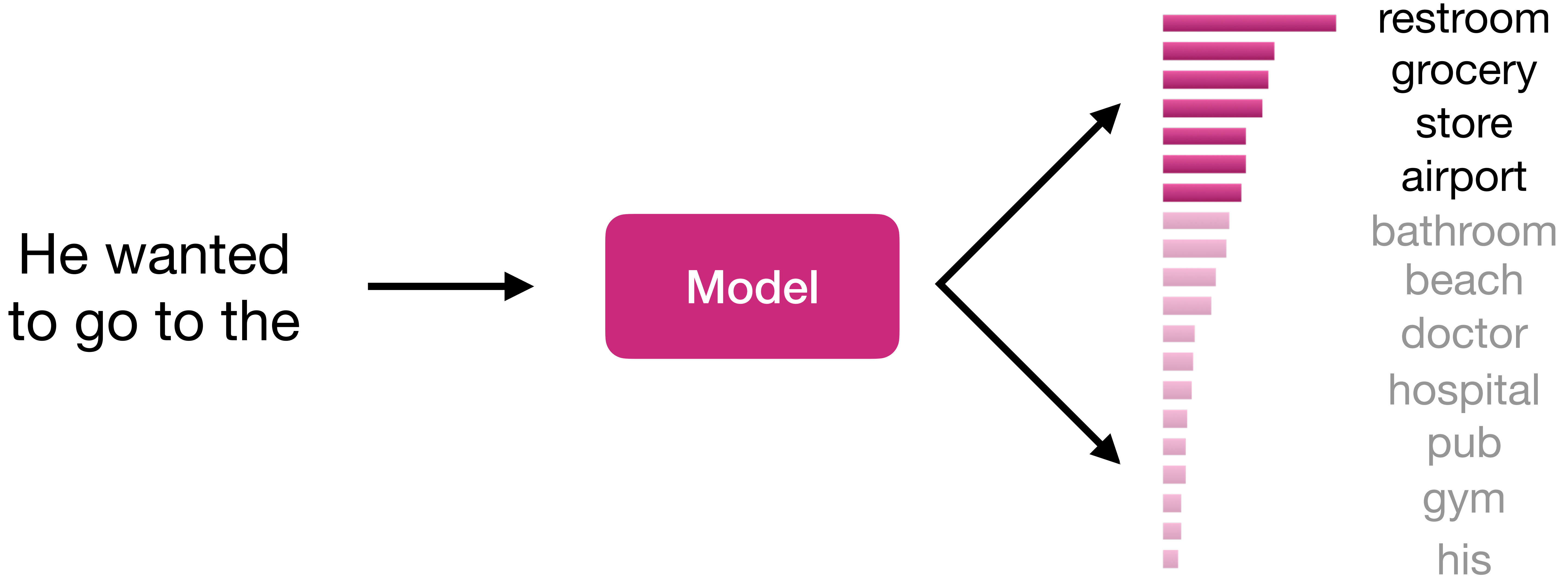
- Problem: Vanilla sampling makes *every token* in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have "heavy tailed"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

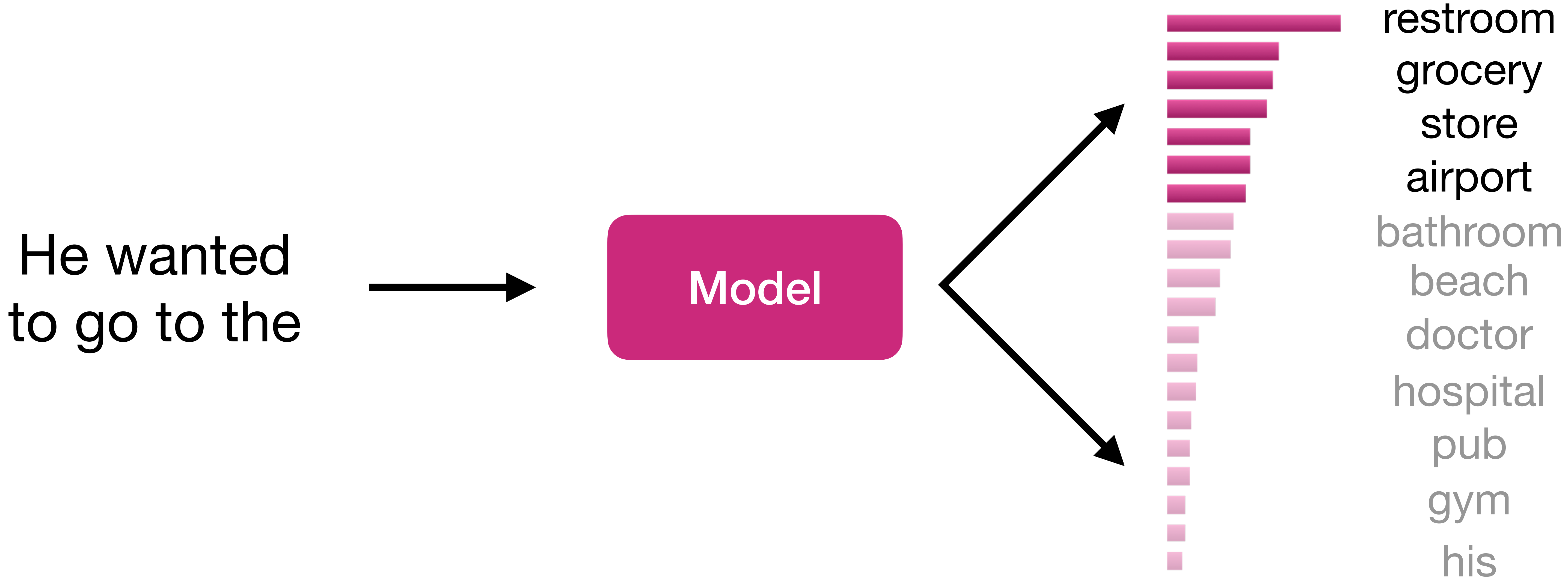
Decoding: Top-*k* Sampling

- <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!*)



Decoding: Top-*k* Sampling

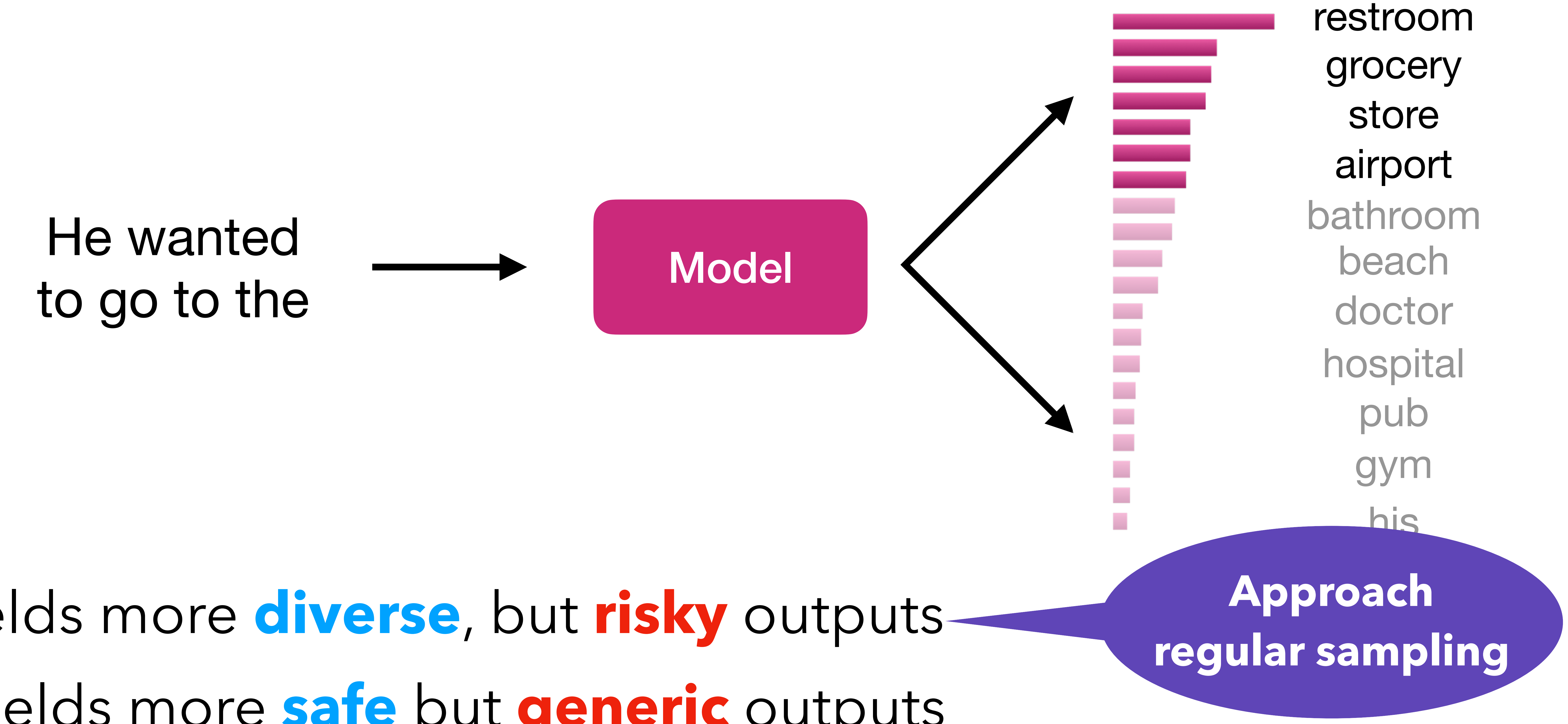
- Solution: Top-*k* sampling *(Fan et al., 2018)*
  - Only sample from the top *k* tokens in the probability distribution.
  - Common values for *k* = 10, 20, 50 (*but it's up to you!*)



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

Decoding: Top-*k* Sampling

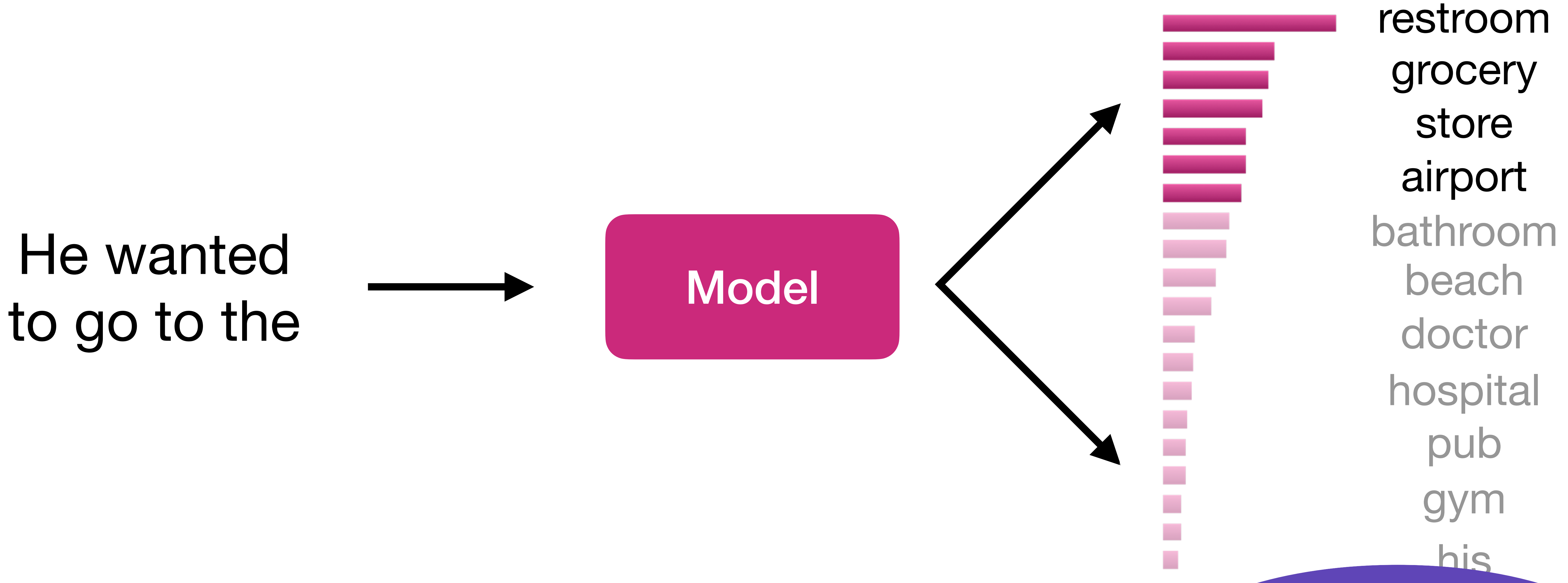
- Solution: Top-*k* sampling *(Fan et al., 2018)*
  - Only sample from the top *k* tokens in the probability distribution.
  - Common values for *k* = 10, 20, 50 (*but it's up to you!*)



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

Decoding: Top-*k* Sampling

- Solution: Top-*k* sampling *(Fan et al., 2018)*
  - Only sample from the top *k* tokens in the probability distribution.
  - Common values for *k* = 10, 20, 50 (*but it's up to you!*)



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

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

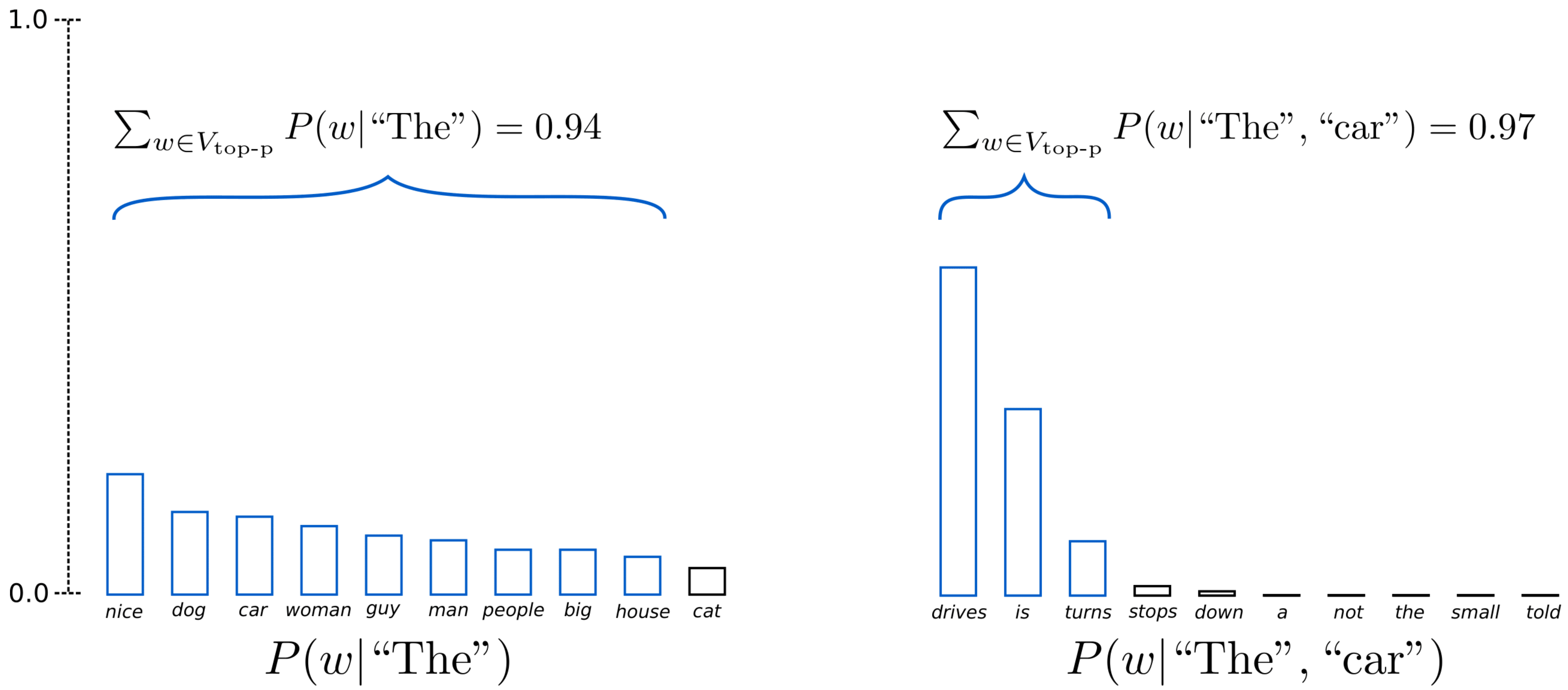


Image from: How to generate text: using different decoding methods for language generation with Transformers

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

- Solution: Top-*k* sampling *(Holtzman et al., 2020)*
  - Only sample from the the most probable tokens smallest possible set of words whose cumulative probability exceeds the probability *p*
  - Common values for *p* = 0.8, 0.85, 0.9, 0.95, 1 (*but it's up to you!*)

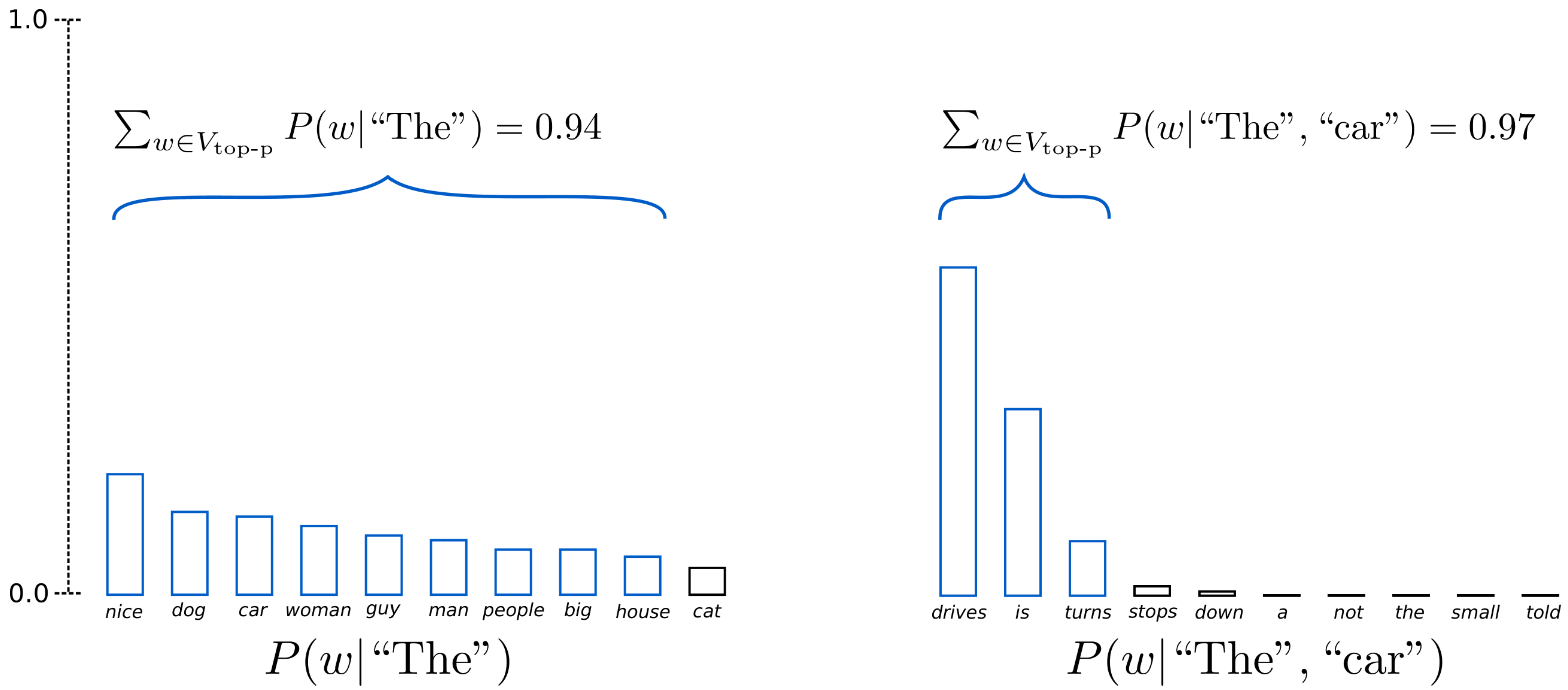
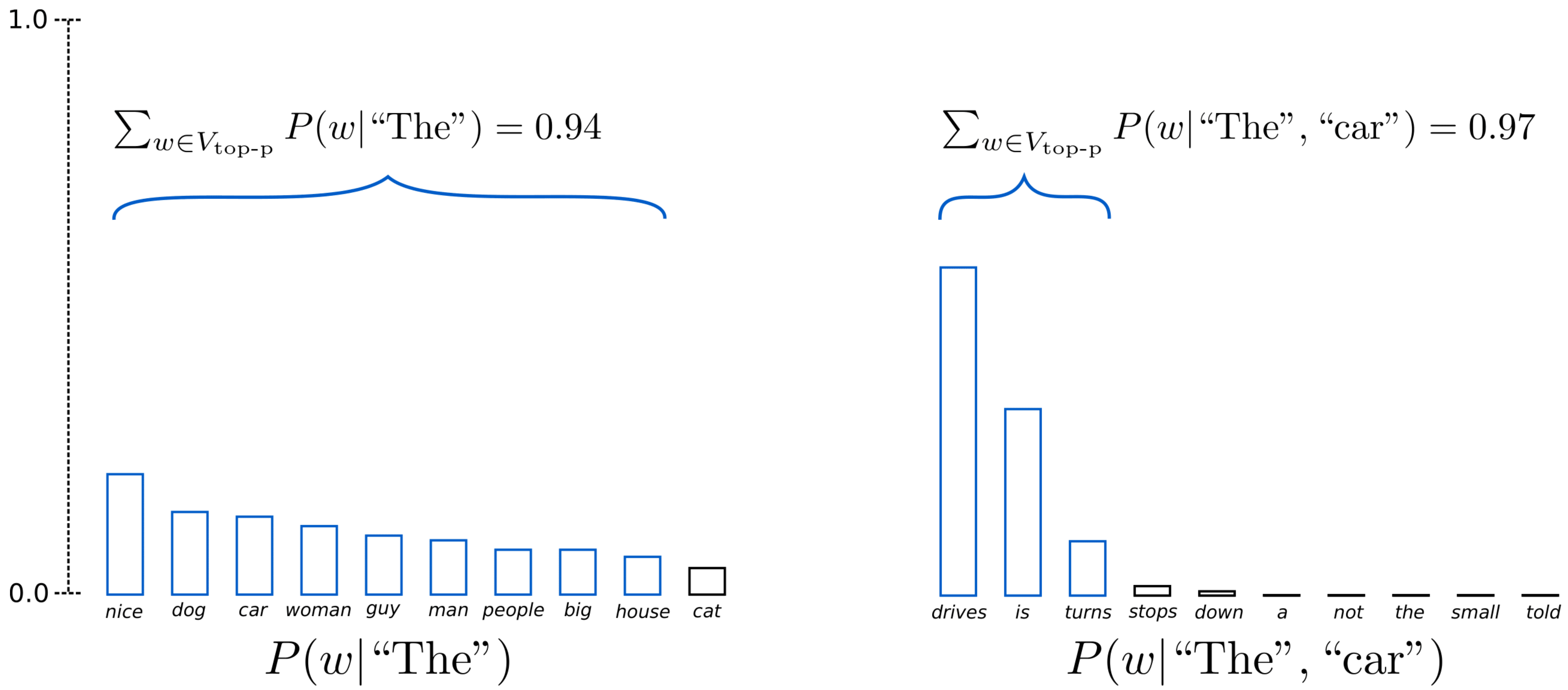


Image from: How to generate text: using different decoding methods for language generation with Transformers

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

- Solution: Top-*k* sampling *(Holtzman et al., 2020)*
  - Only sample from the the most probable tokens smallest possible set of words whose cumulative probability exceeds the probability *p*
  - Common values for *p* = 0.8, 0.85, 0.9, 0.95, 1 (*but it's up to you!*)



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

Image from: How to generate text: using different decoding methods for language generation with Transformers

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:

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

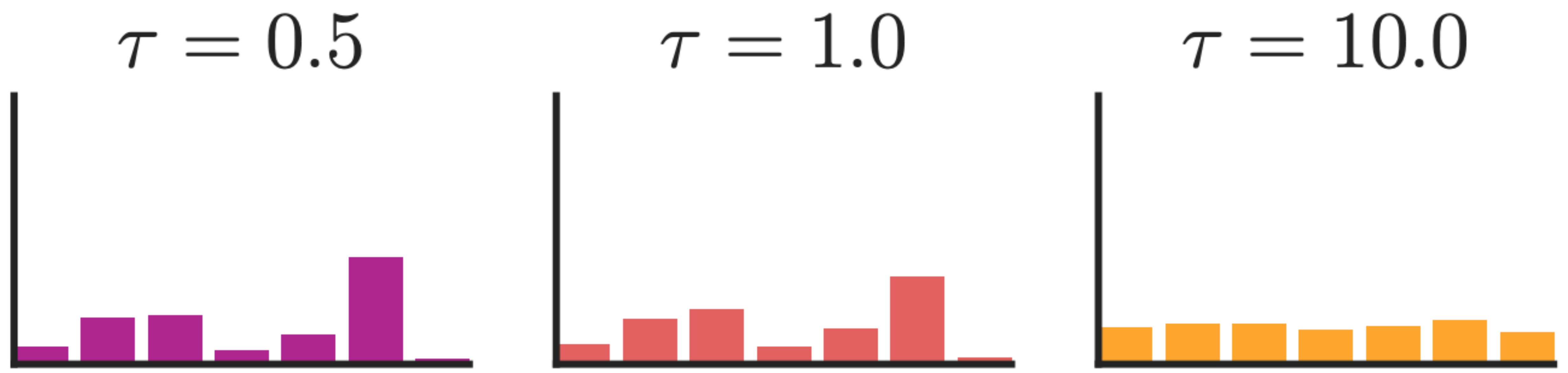
- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread across vocabulary)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability concentrated to the top tokens)

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.

Thank you!