

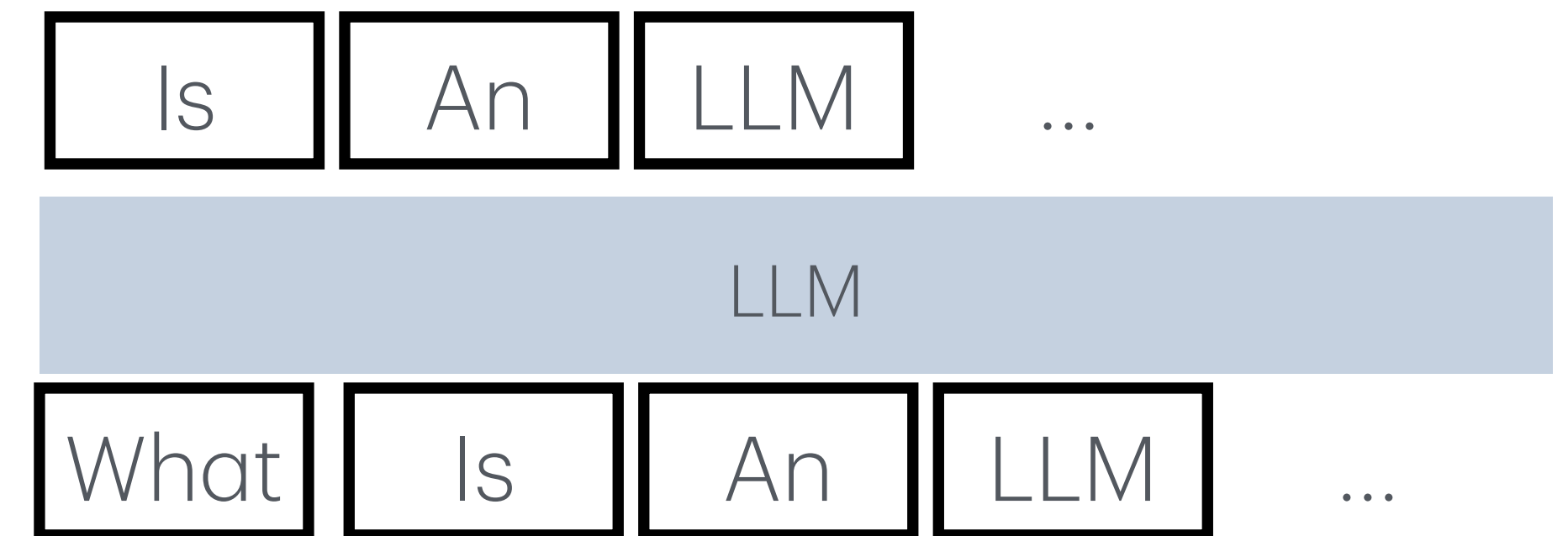
# Large Language Models I

# What is a large language model?

- Demo  
ollama run qwen3

# What is an LLM technically?

- A probability distribution of (written) language
- In past: Many different architectures
- Today: A really big auto-regressive model



$$P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$$

# LLMs

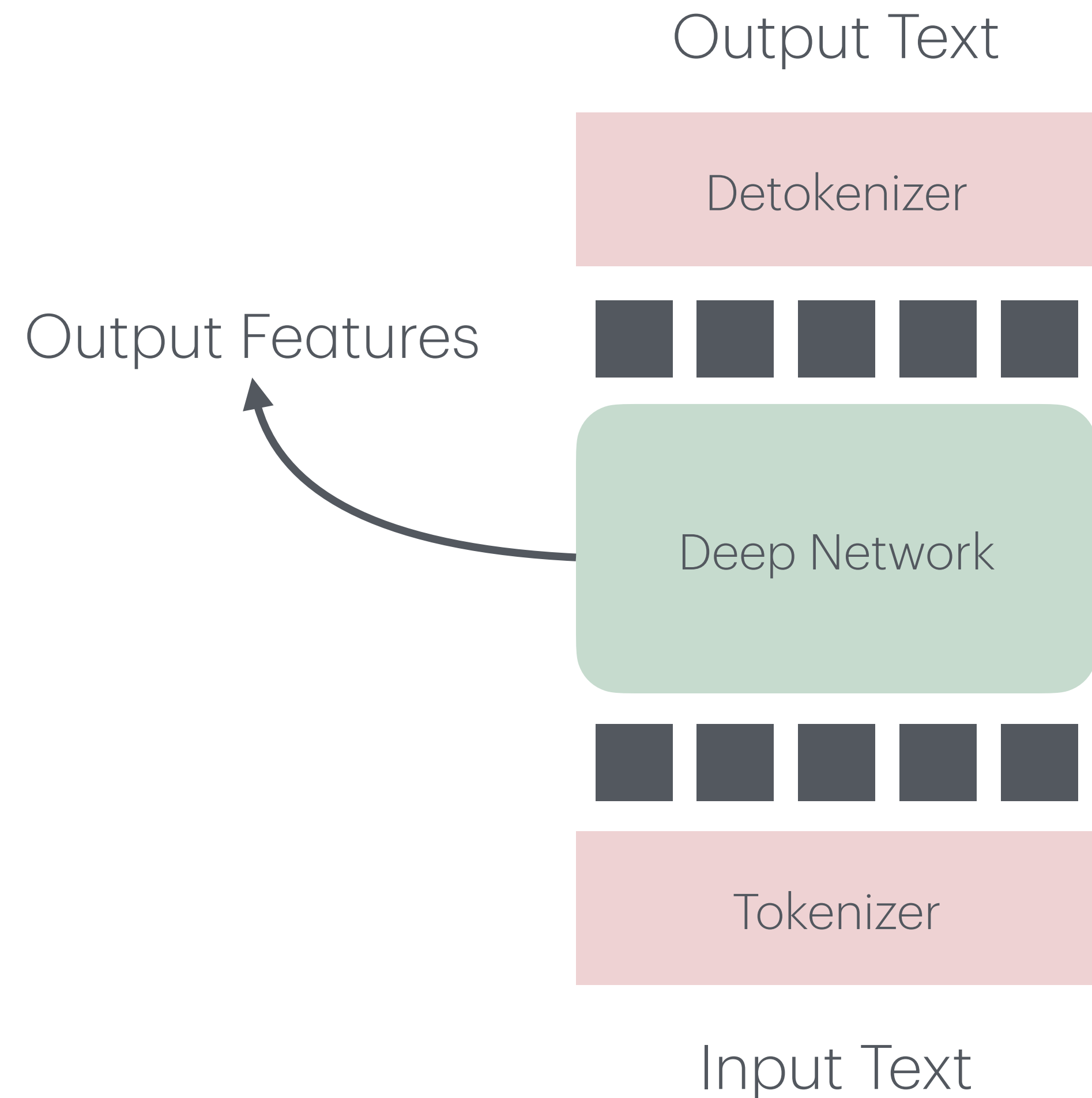
- Architectures
- Generation, Instruction Tuning, RLHF, DPO, Tasks and Datasets
- Tool use and Structured Outputs
- Long Context and RAGs
- Structured Dialogues, Reflection
- Limitations of LLMs

# Architectures

Philipp Krähenbühl, UT Austin

# LLM - Basics

- Tokenizer
- Deep Network
  - Encoder-Decoder (original transformer)
  - Encoder-only
  - Decoder-only
- Sequence Models



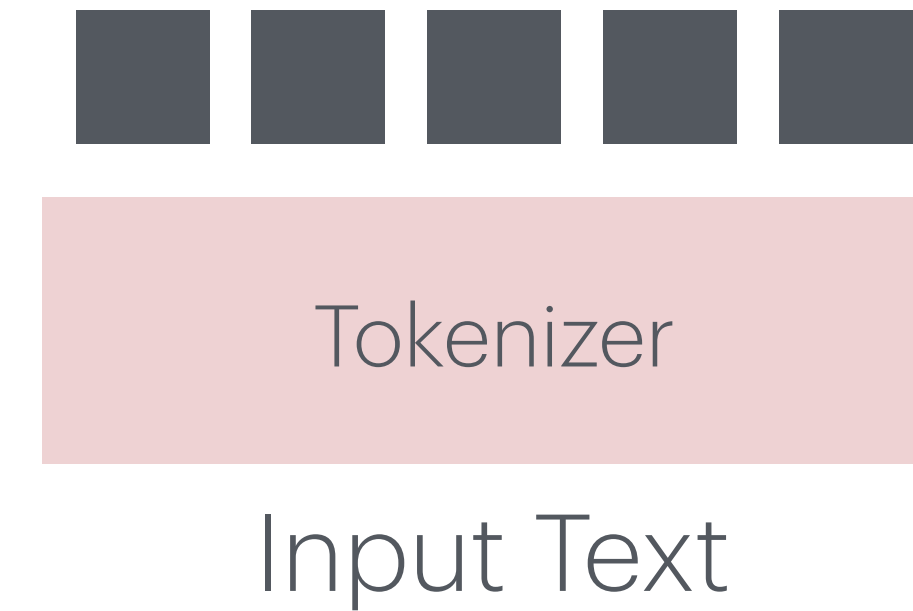
# Tokenization

- Convert text into inputs / outputs of network
- What inputs a network understand?
  - Continuous vectors
- What outputs a network understand?
  - Continuous vectors
  - Distributions over categories



# What is text?

- A sequence of characters  
 $c_1 \dots c_N \in \{0 \dots 255\}$
- English language
  - 1 symbol = 1 character
- Other languages (UTF-8)
  - 1 symbol = 1-4 characters





# Tokenization

- Demo

<https://tiktokenizer.vercel.app/?model=meta-llama%2FMeta-Llama-3-70B>

Token count

33

A cat was climbing a tree or Tree.

Tree was climbed by a cat.

A tree-climbing cat...

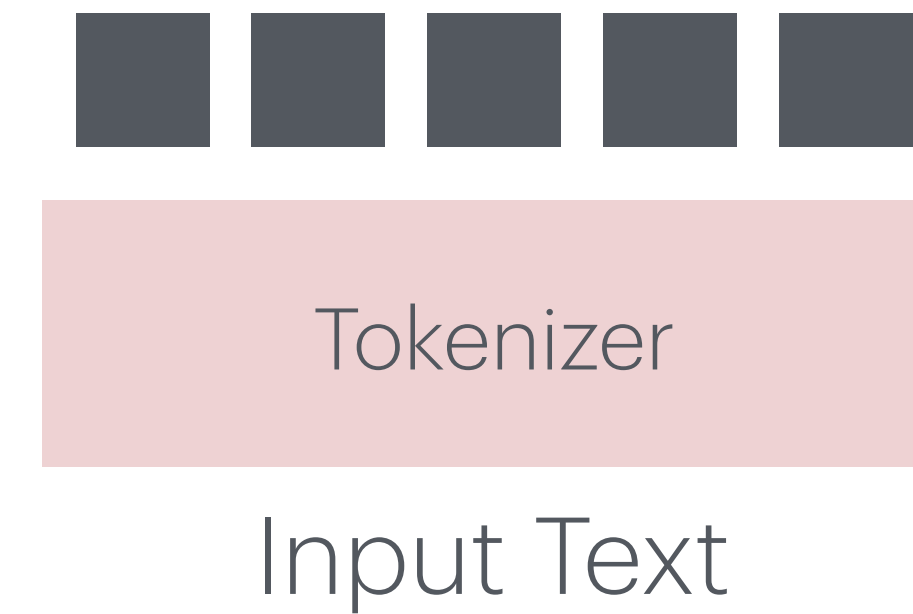
一隻貓正在爬樹。

32, 8415, 574, 30608, 264, 5021, 477, 9119, 627, 6670,  
574, 45519, 555, 264, 8415, 627, 32, 5021, 31717, 318,  
7278, 8415, 9522, 15120, 37795, 119, 80631, 241, 9765  
5, 76207, 105, 111398, 1811

# Tokenization

## Character-level tokenizer V0

- Convert input characters  $c_1 \dots c_N$  to
  - integers  $\text{ord}(c_i)$
- Bad idea
  - Integers do not have an order
  - Math on ordinal values does not make sense
$$\text{ord}('a') + \text{ord}('b') \neq \text{ord}('c')$$



# Tokenization

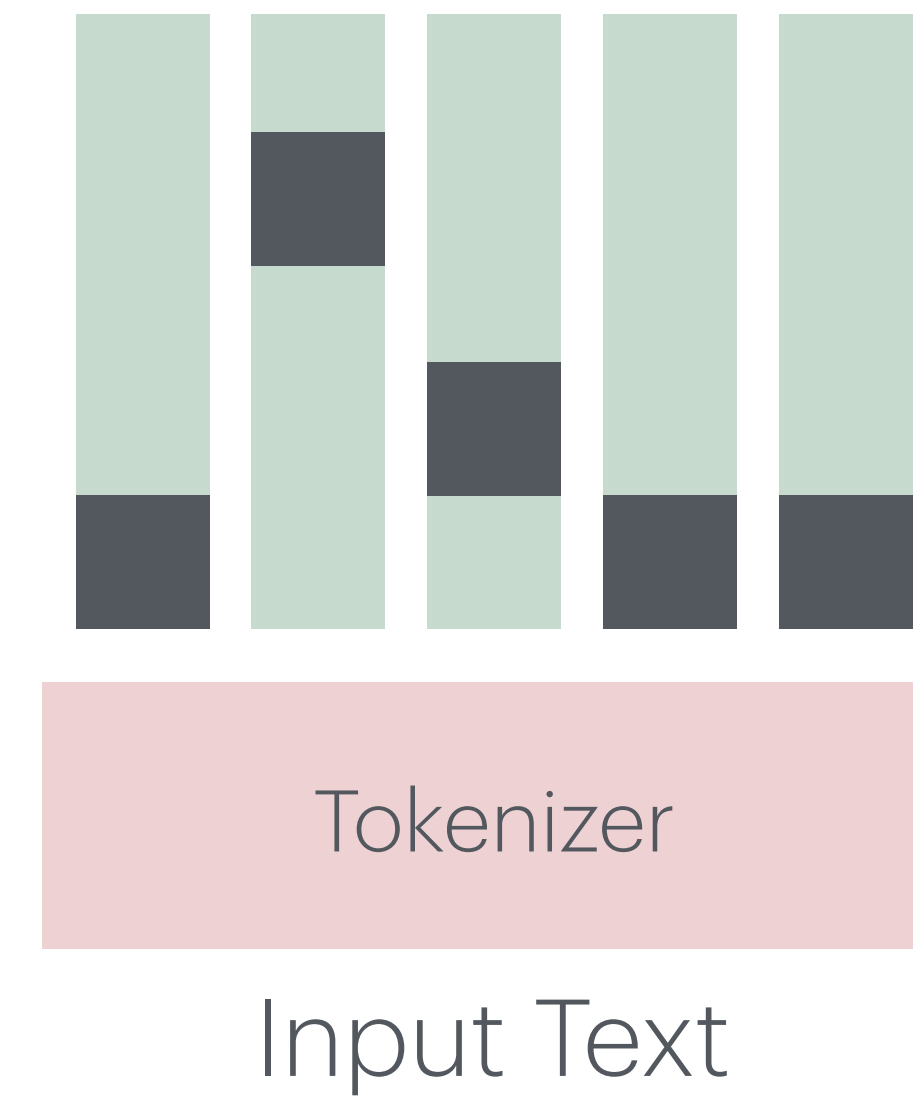
## Character-level tokenizer V0.1

- Convert input characters  $c_1 \dots c_N$  to

- Use one-hot encodings

$$\begin{bmatrix} \text{ord}(c_i) = 1 \\ \text{ord}(c_i) = 2 \\ \text{ord}(c_i) = 3 \\ \dots \end{bmatrix}$$

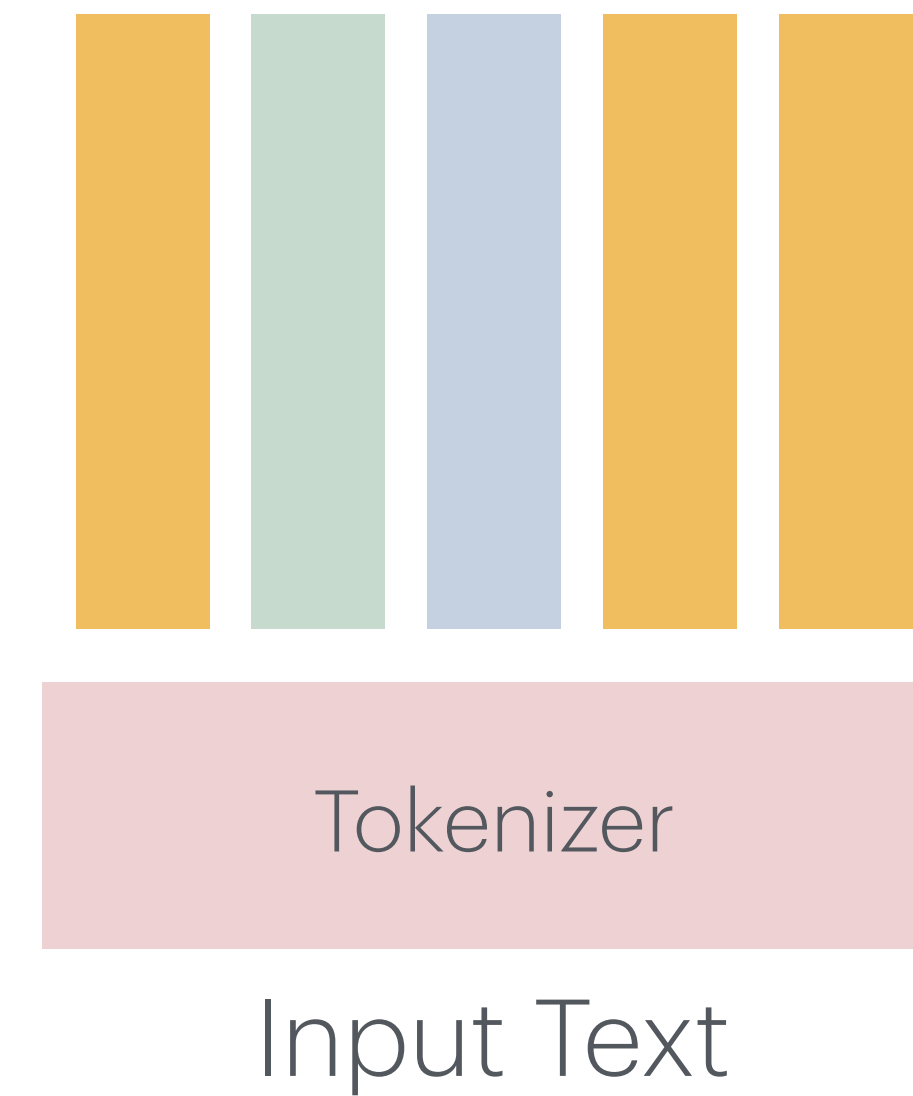
- Improvement
  - Combine with first layer in LLM



# Tokenization

## Character-level tokenizer

- Convert input characters  $c_1 \dots c_N$  to
  - Learned character-level embeddings  $e_1, \dots, e_{256}$
  - Tokenize  $e_{\text{ord}(c_i)}$

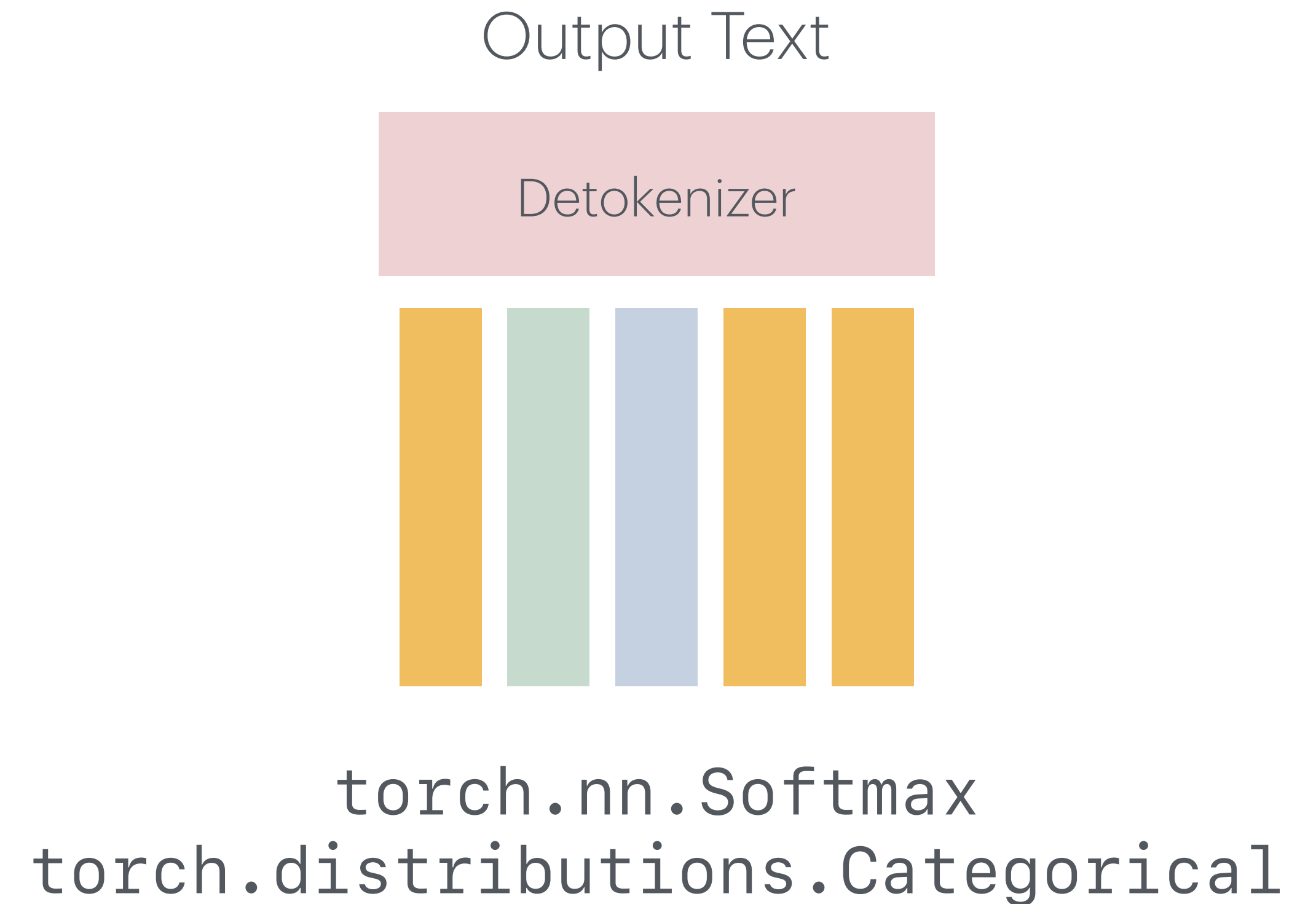


`torch.nn.Embedding`

# Detokenization

## Character-level tokenizer

- Convert network outputs to
  - Characters  $c_1 \dots c_N$
- Predict distribution over characters
  - Sample and convert to UTF-8
- Any issues?
  - Not every sequence of characters is valid UTF-8



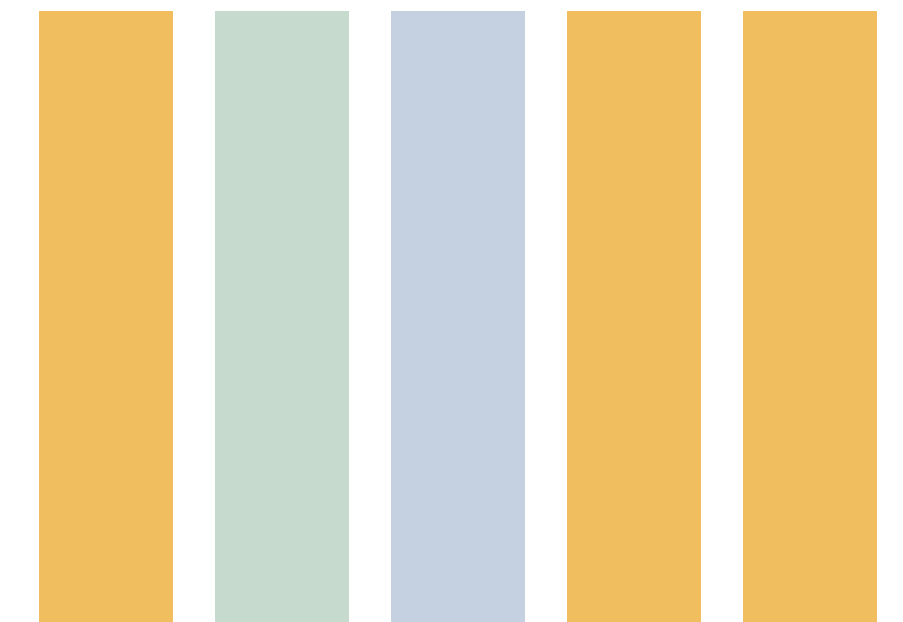
# Tokenization

## Character-level tokenizer

- Discussion
  - Symmetric
    - Optionally share input embeddings + output classifier weights
- Few tokens
- Very long sequences

Output Text

Detokenizer



Tokenizer

Input Text

# Tokenization

## Byte-pair encoding

- Learned Text Compression Algorithm
  - Training set  $D$
- Vocabulary  $V$  (string  $\rightarrow$  int)
- Merge most frequent consecutive tokens
- Trained on large text corpus
  - Vocabulary size 40k - 256k

```
def BPE(D: bytes, N: int):  
    # Initialize single character vocab  
    V = {chr(i): i for i in range(256)}  
  
    # Tokenize  
    d = [V[c] for c in D]  
  
    while len(V) < N:  
        # Find most frequent pair of tokens  
        cnt = Counter([(d[i], d[i+1])  
                        for i in range(len(d)-1)])  
        a,b = cnt.most_common(1)[0][0]  
  
        # Add a new token  
        V[(a,b)] = len(V)  
  
        # Update the dataset  
        d_update = []  
        for x in d:  
            if len(d_update) and d_update[-1] == a\  
                and x == b:  
                d_update[-1] = V[(a,b)]  
            else:  
                d_update.append(x)  
        D = d_update
```

# Tokenization

The cat in the hat sat in a chair

Byte-pair encoding training

- In practice
  - **Split** words / numbers / special strings **with regex** in entire dataset
  - Attach white space to next word / token
- Run BPE over entire dataset
  - Converts dataset into sequence of tokens

The

•Cat

•in

•the

•Hat

•sat

•in

•a

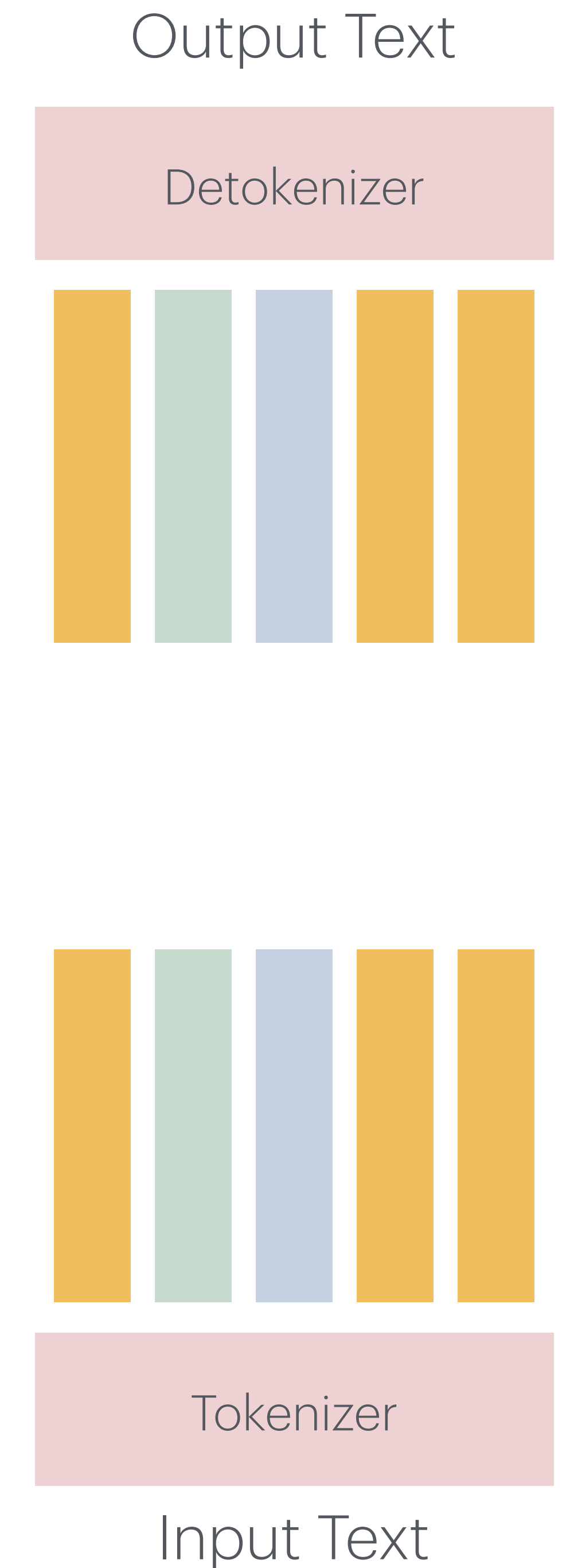
•chair



# Tokenization

## Byte-pair encoding

- Tokenization
  - Merge elements of vocabulary one by one
- Detokenization
  - Replace token with vocabulary text
- Embed and predict compressed tokens



# Tokenization

## Byte-pair encoding

- Demo

<https://tiktokenizer.vercel.app/?model=meta-llama%2FMeta-Llama-3-70B>

Token count

33

A cat was climbing a tree or Tree.

Tree was climbed by a cat.

A tree-climbing cat...

一隻貓正在爬樹。

32, 8415, 574, 30608, 264, 5021, 477, 9119, 627, 6670,  
574, 45519, 555, 264, 8415, 627, 32, 5021, 31717, 318,  
7278, 8415, 9522, 15120, 37795, 119, 80631, 241, 9765  
5, 76207, 105, 111398, 1811

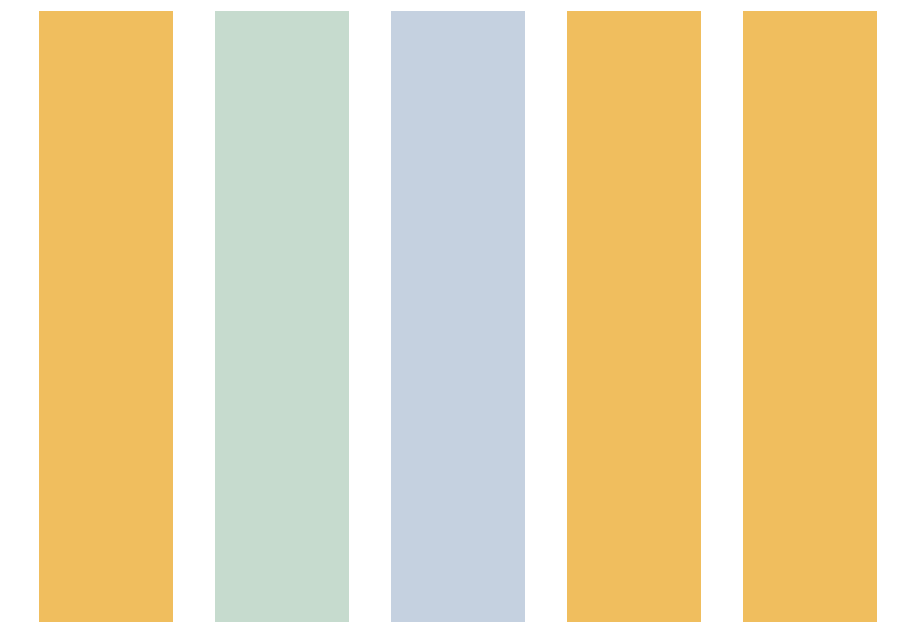
# Tokenization

## Byte-pair encoding

- Special tokens
  - Beginning of Sequence [BOS]
  - End of Sequence [EOS]
  - Classification [CLS]
  - Separator [SEP]
- ...

Output Text

Detokenizer



Tokenizer

Input Text

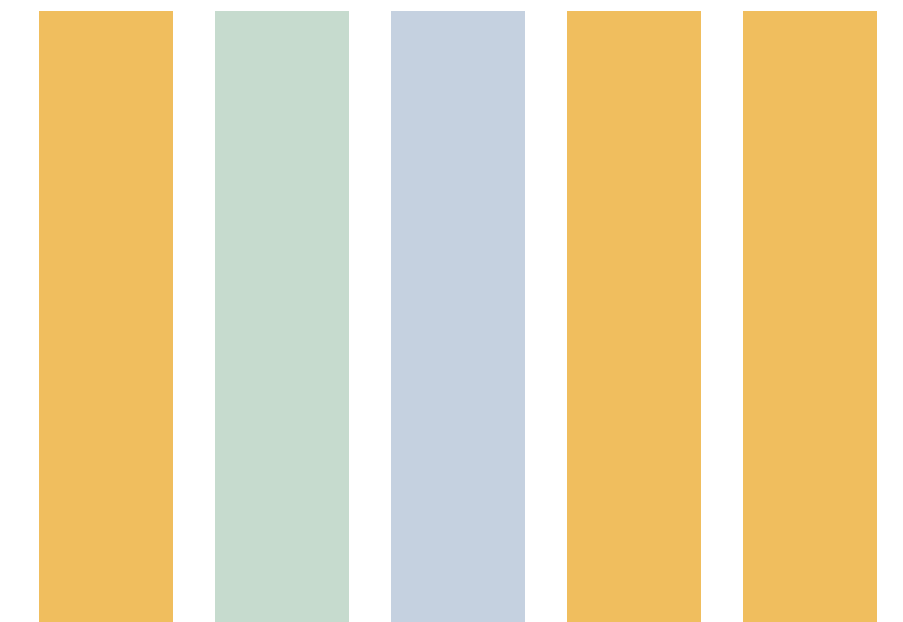
# Tokenization

## Byte-pair encoding

- Alternative: WordPiece
- Discussion
  - Many tokens (large vocabulary)
  - Shorter sequences
- One of the main sources of issues for LLMs

Output Text

Detokenizer



Tokenizer

Input Text

# Tokenization - Issues

- Demo

<https://tiktokenizer.vercel.app/?model=meta-llama%2FMeta-Llama-3-70B>

# Tokenization - Issues

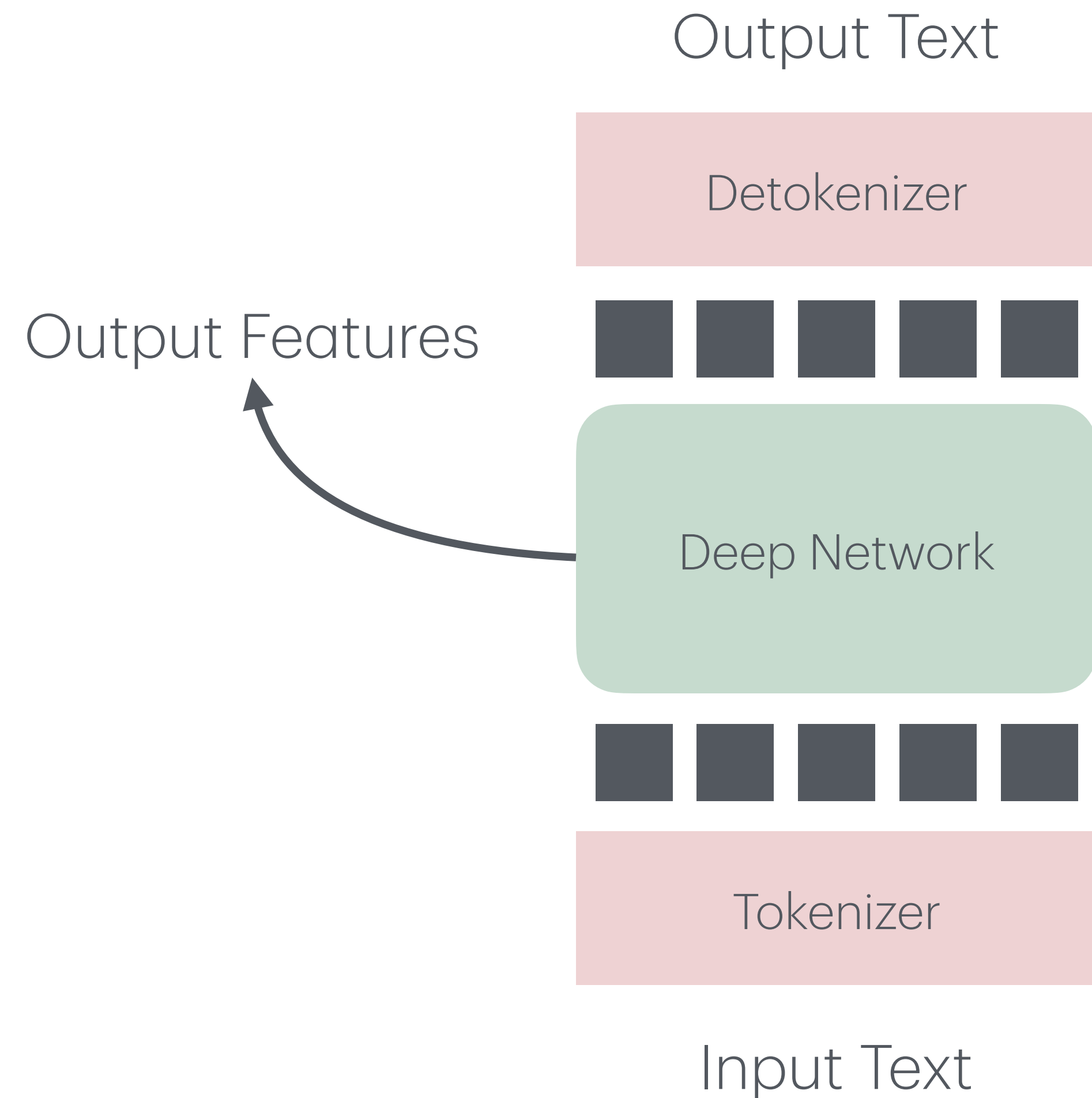
- Not a one-to-one mapping
  - Many token-streams detokenize to same text
- Counting letters / word manipulations / string processing are hard
- LLMs are bad at math
- Potential exploits around tokens with few training examples

# Tokenization - Issues

- Demo  
ollama run llama3.1

# LLM - Basics

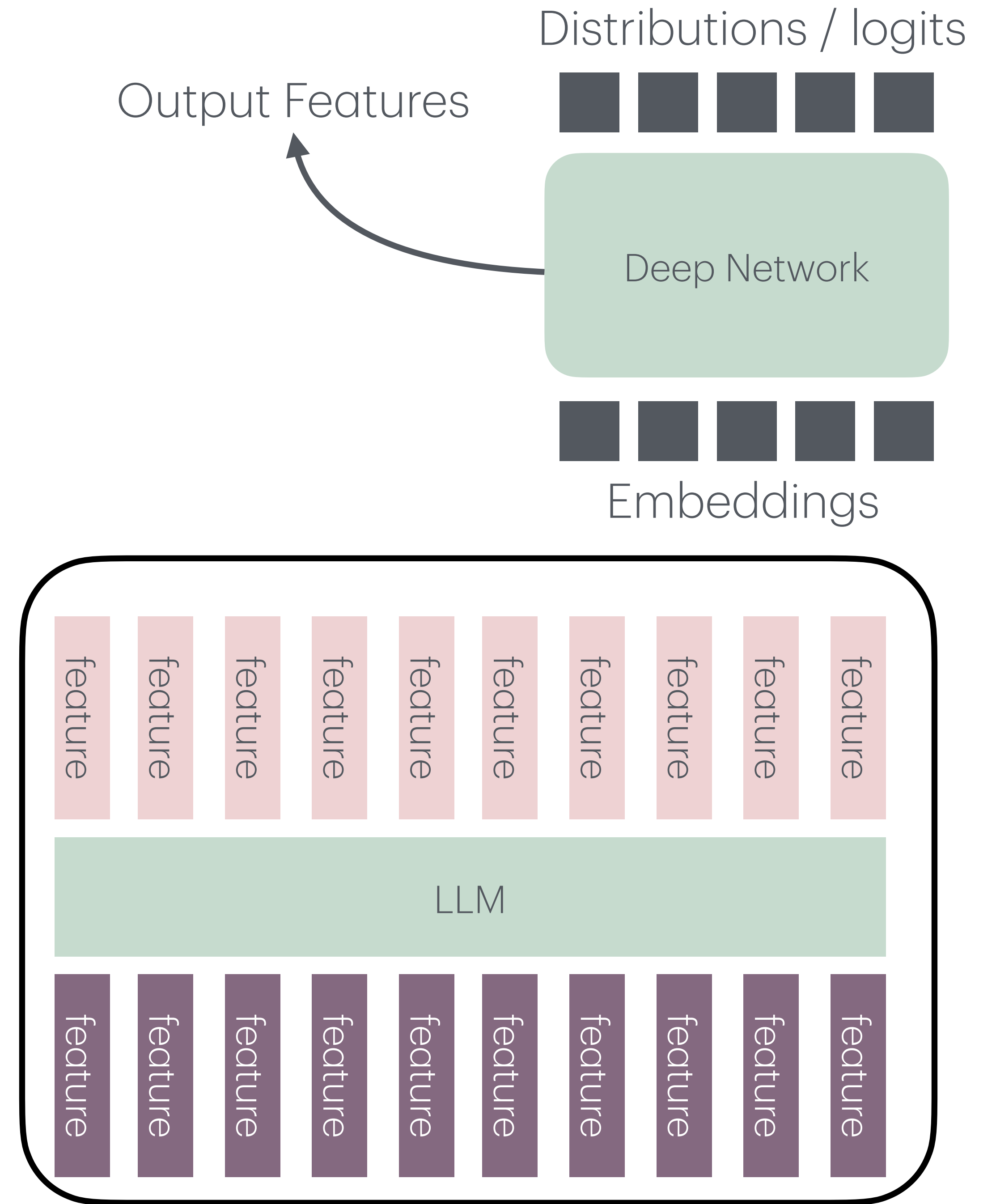
- Tokenizer
- Deep Network
  - Encoder-Decoder (original transformer)
  - Encoder-only
  - Decoder-only
- Sequence Models





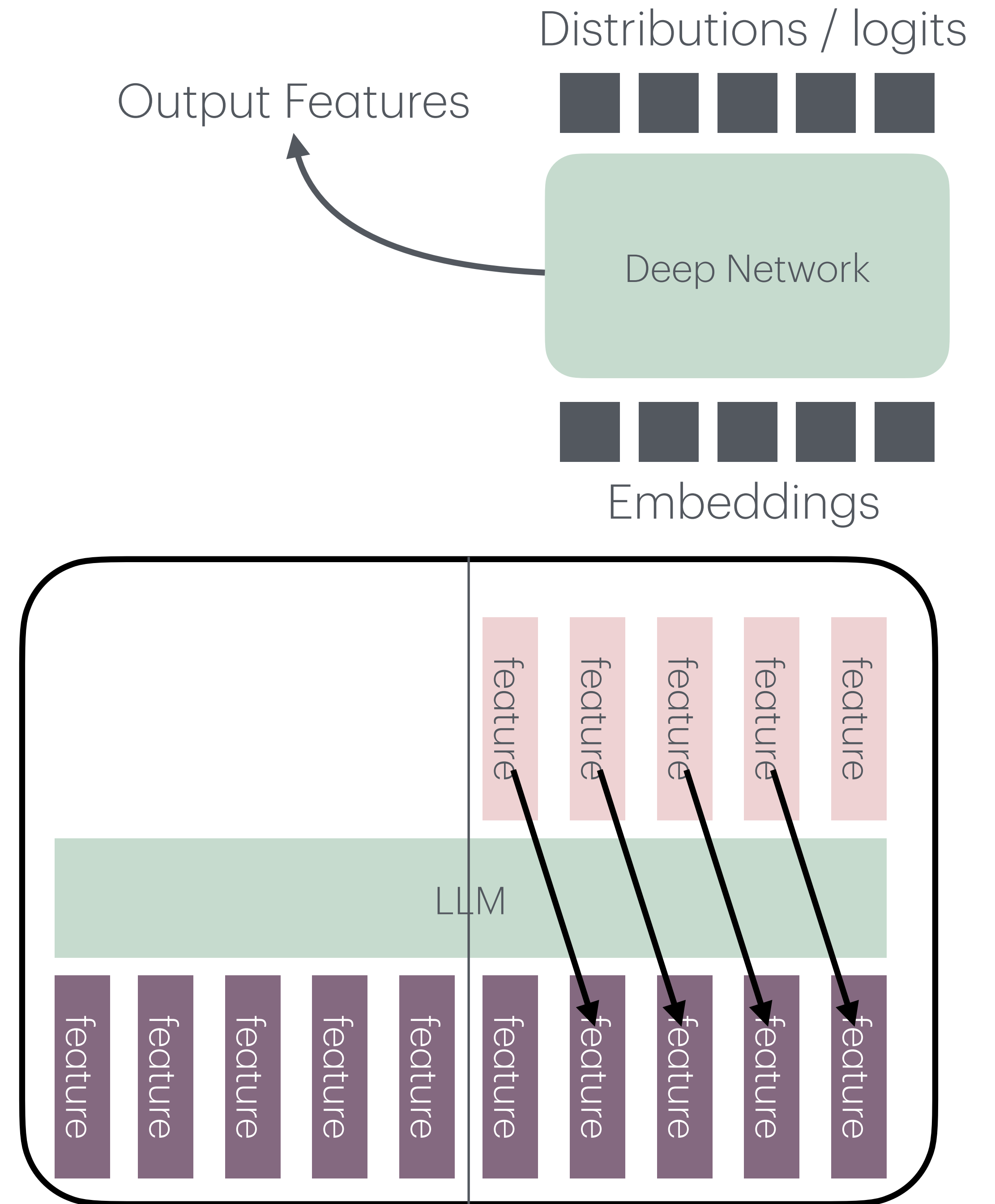
# LLM - Architectures

- **Encoder-Decoder (original transformer)**
- Encoder-only
- Decoder-only
- Sequence Models



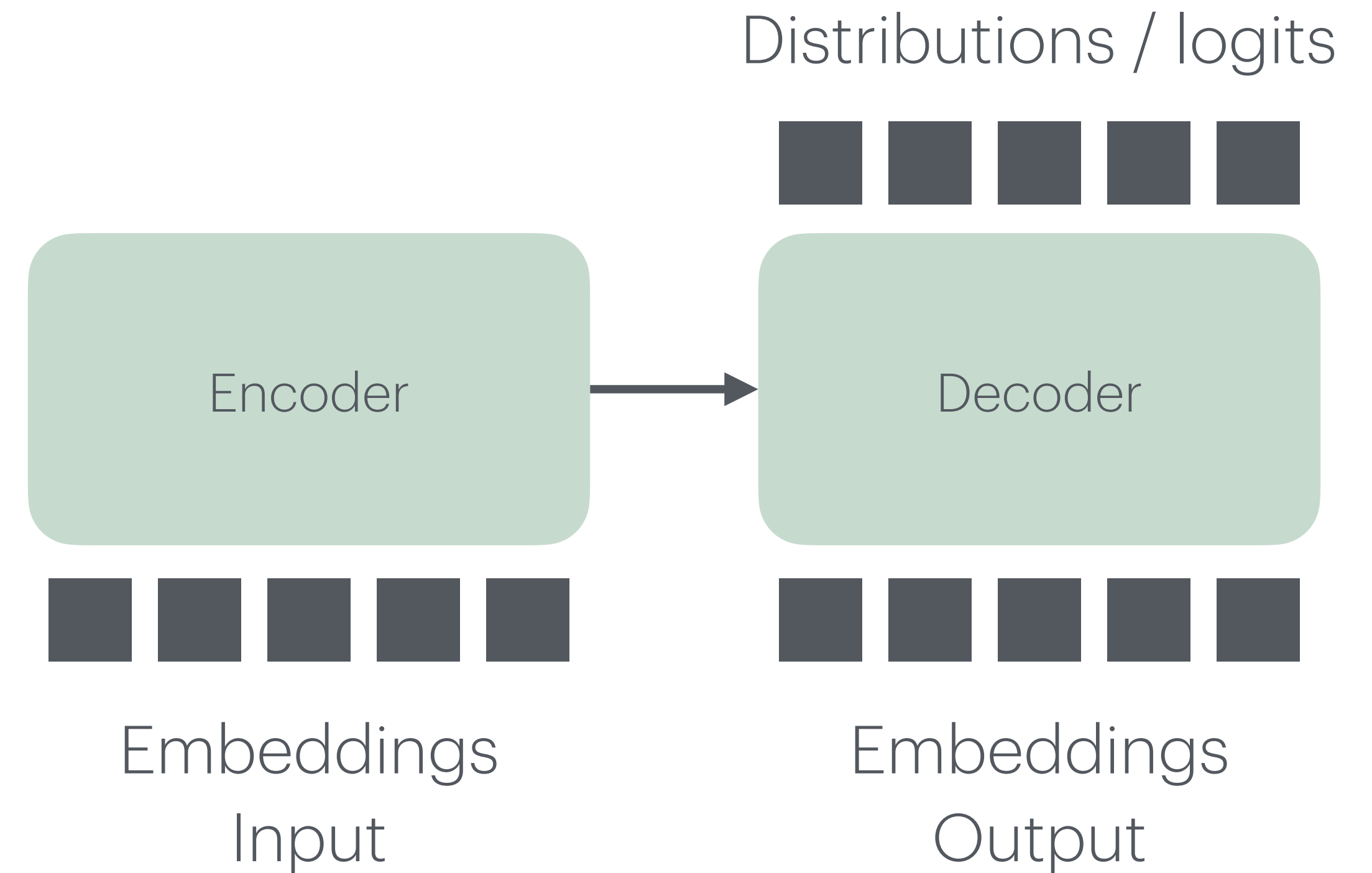
# LLM - Architectures

- **Encoder-Decoder (original transformer)**
- Encoder-only
- Decoder-only
- Sequence Models



# The transformer

- Input:
  - Text: Question / prompt
- Output:
  - Autoregressive probability over text
  - Token-by-token

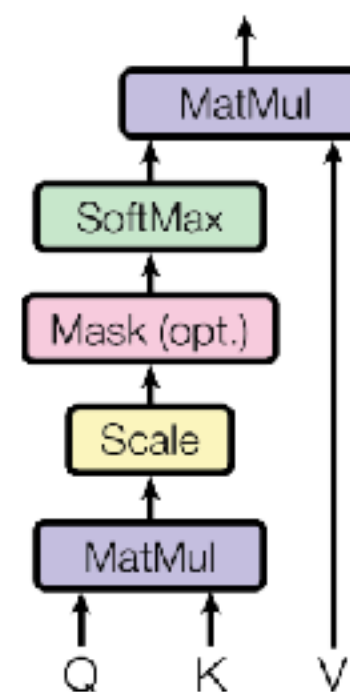


# The transformer

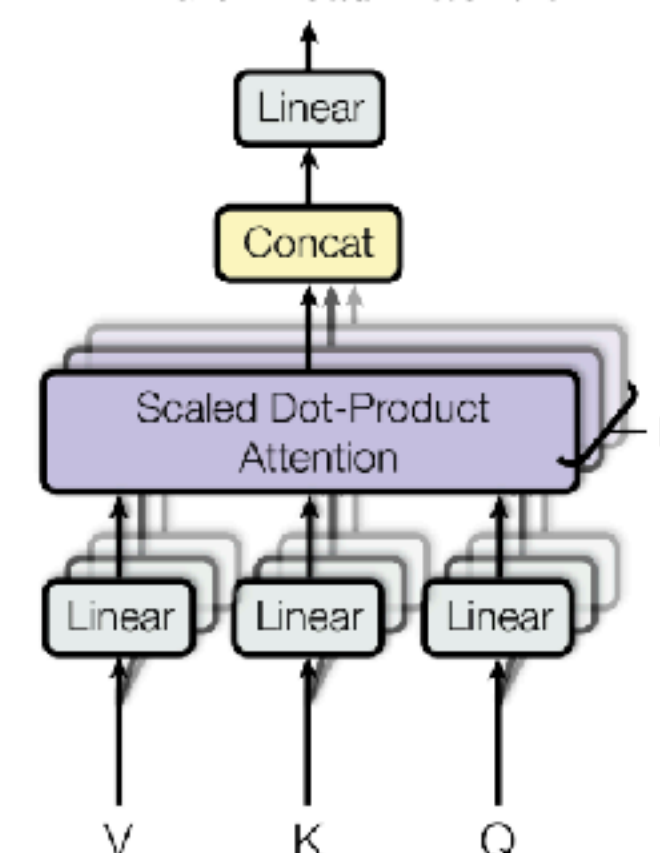
## Architecture

- Encoder
  - Self-attention
  - MLP
- Decoder
  - Causal self-attention
  - MLP

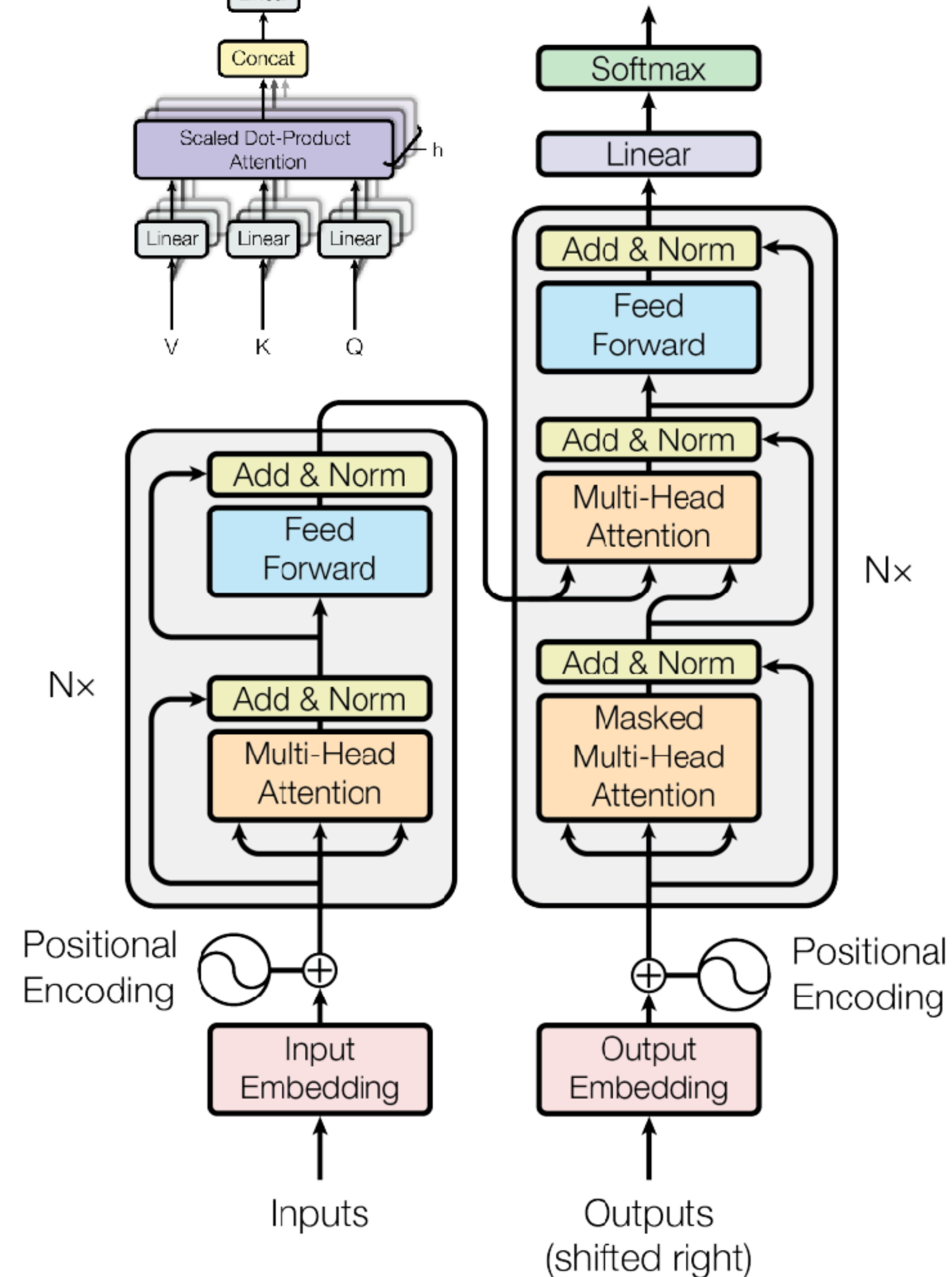
Scaled Dot-Product Attention



Multi-Head Attention



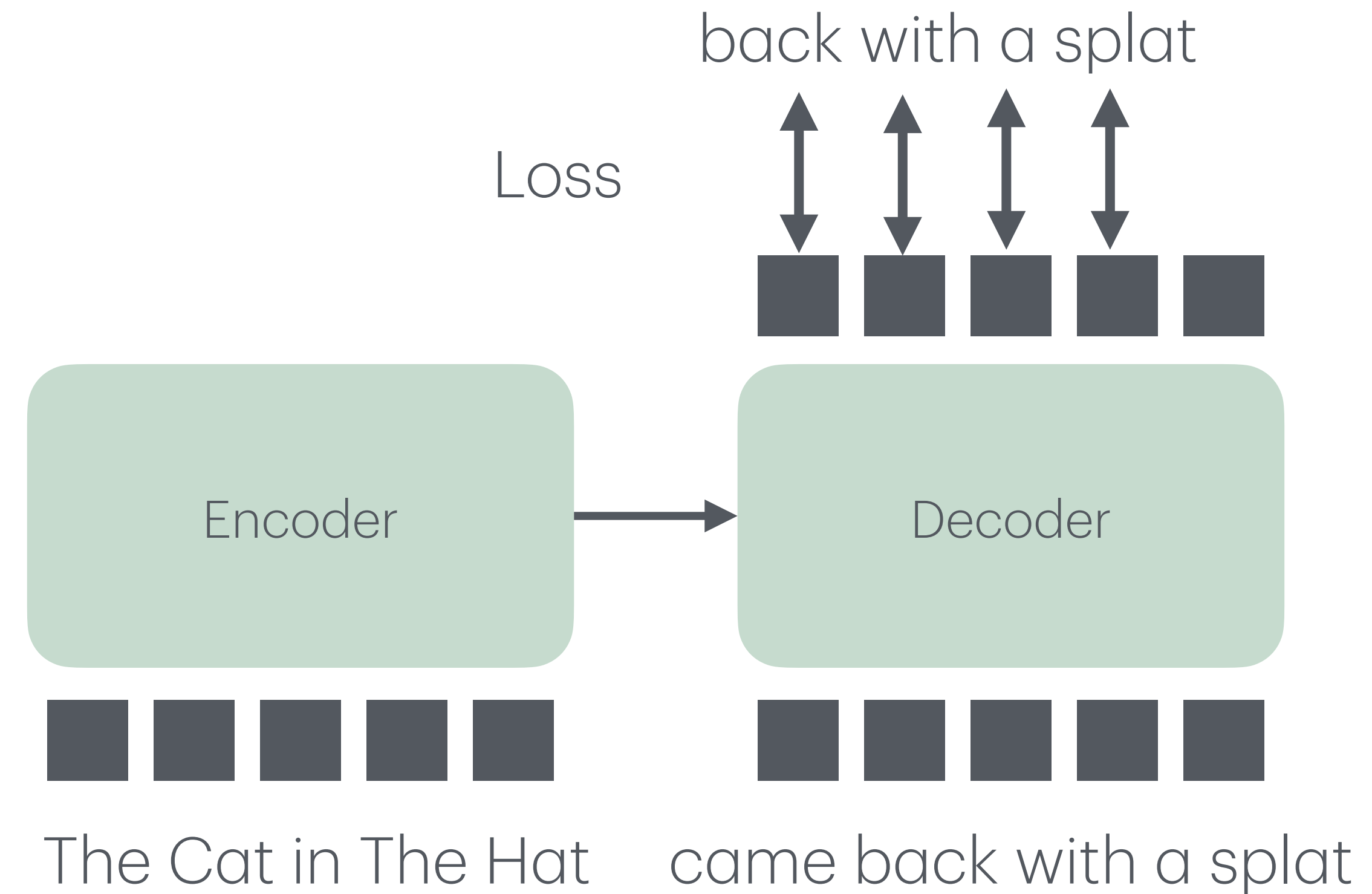
Output Probabilities



# The transformer

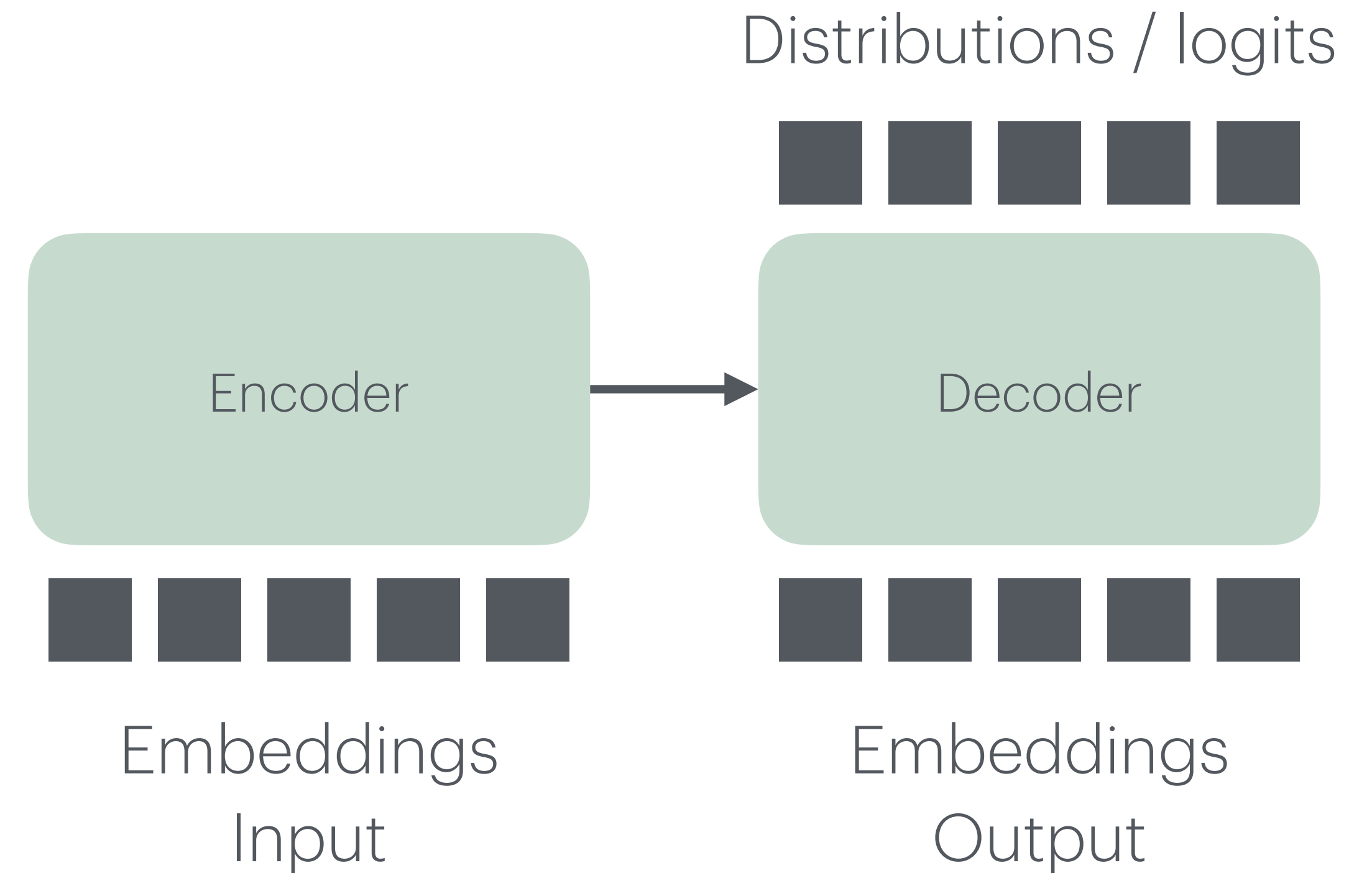
## Pre-training

- Encoder
  - Training set text input
  - No loss
- Decoder
  - Training set text input
  - Loss: CrossEntropy shifted training text



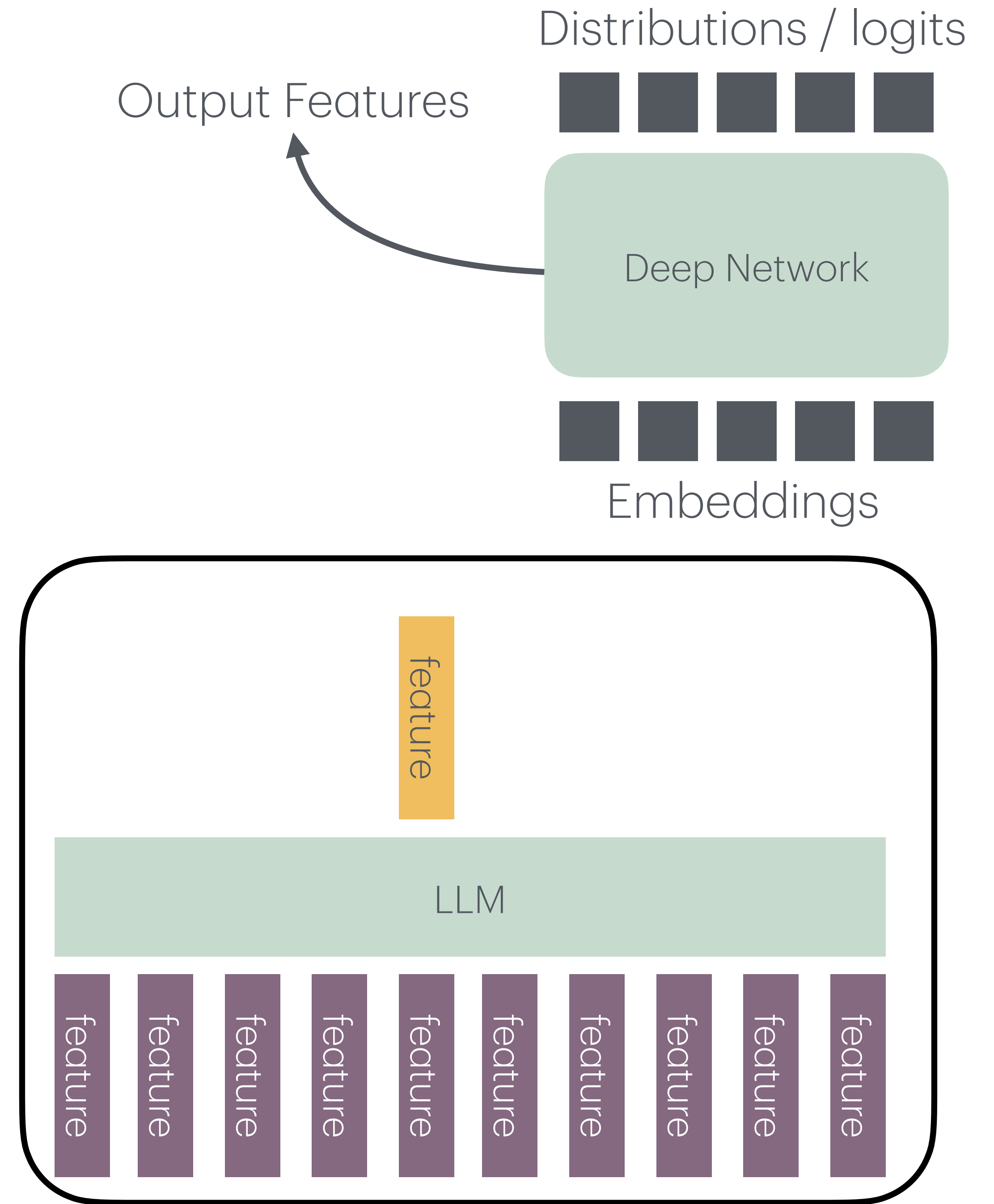
# Encoder-Decoder models

- Original transformer
- Not used much anymore
- Now decoder-only models more popular



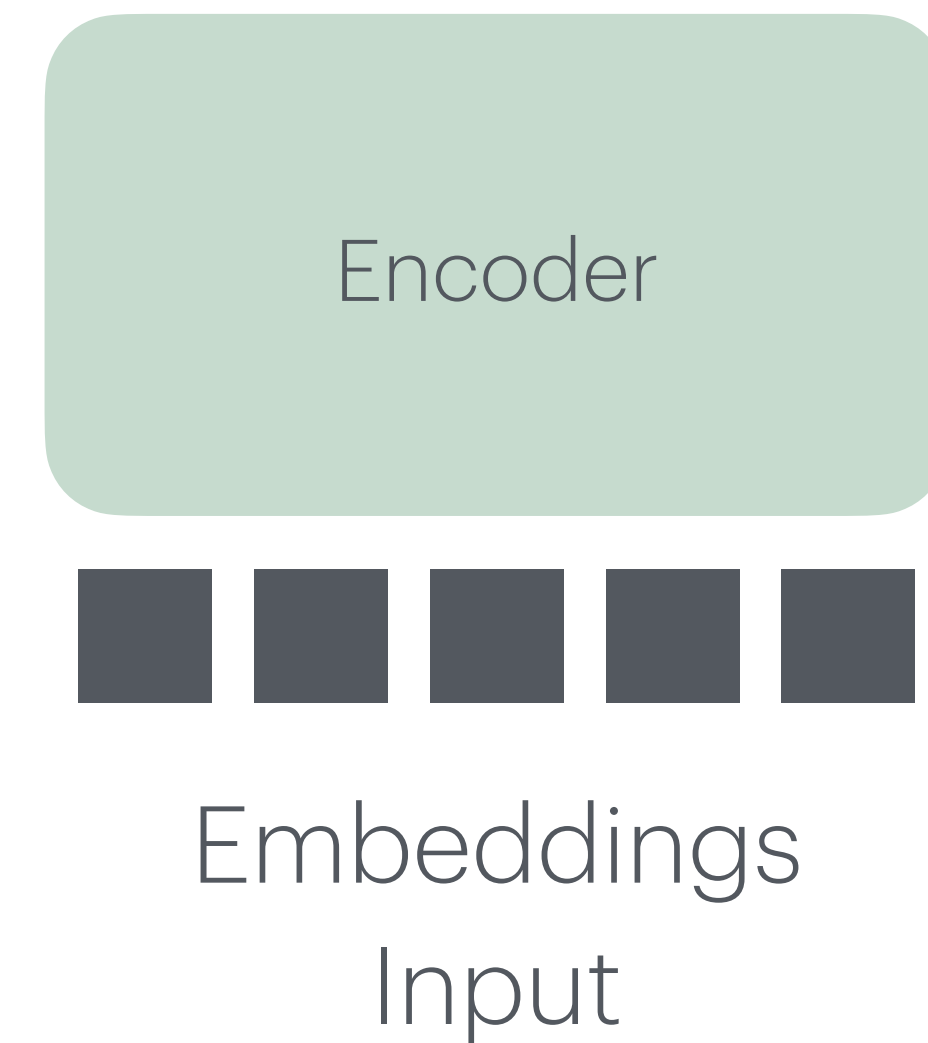
# LLM - Architectures

- Encoder-Decoder (original transformer)
- **Encoder-only**
- Decoder-only
- Sequence Models



# Encoder-only

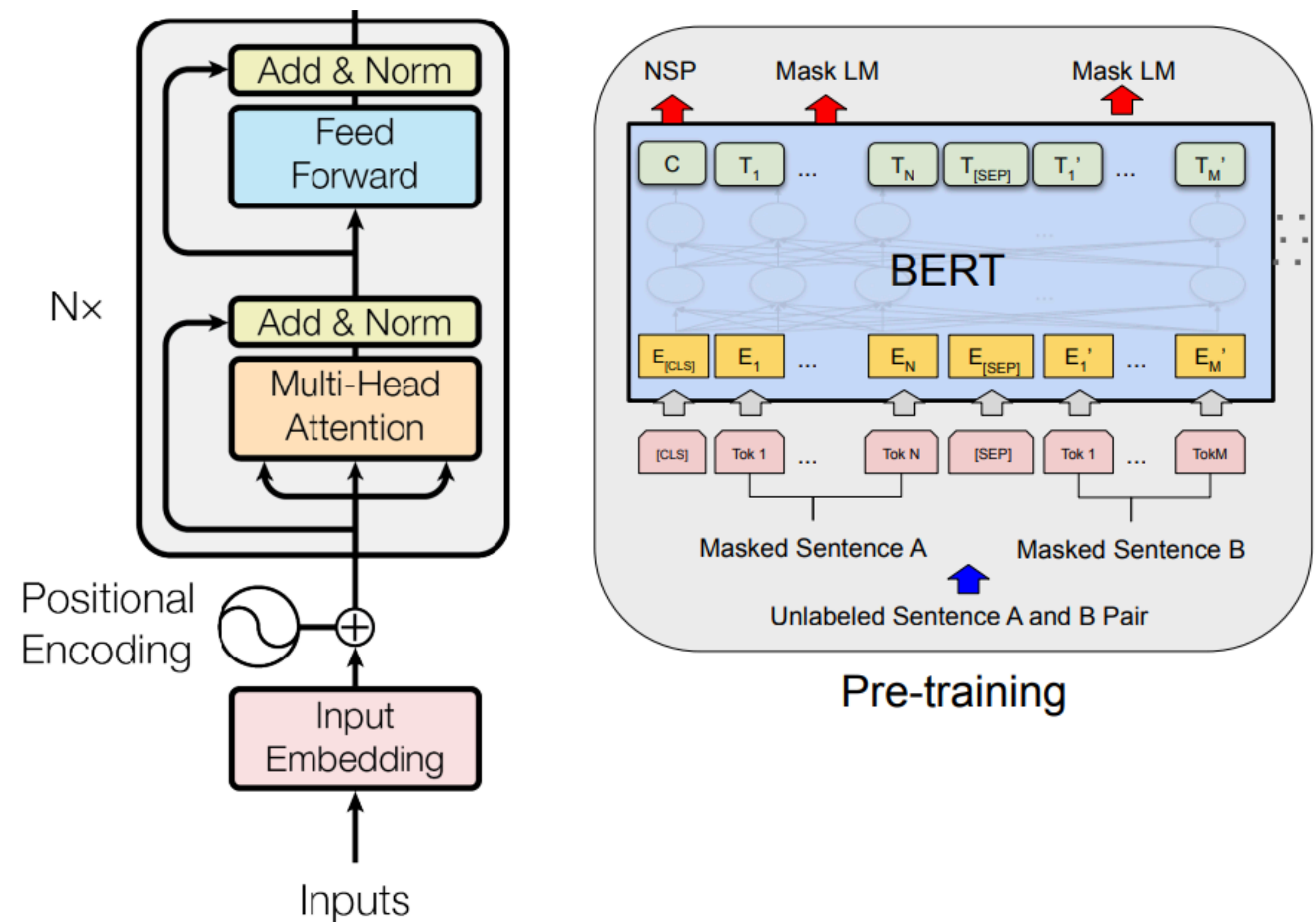
- Input:
  - Text: Question / prompt
- Output:
  - A (sequence of) embeddings





# Encoder-only Architecture

- Transformer encoder (i.e. BERT)
  - Bidirectional attention
- Multiple sizes (BERT)
  - L : #layers 12, 24
  - H: Hidden size 768, 1024
  - A: # Attention heads 12, 16

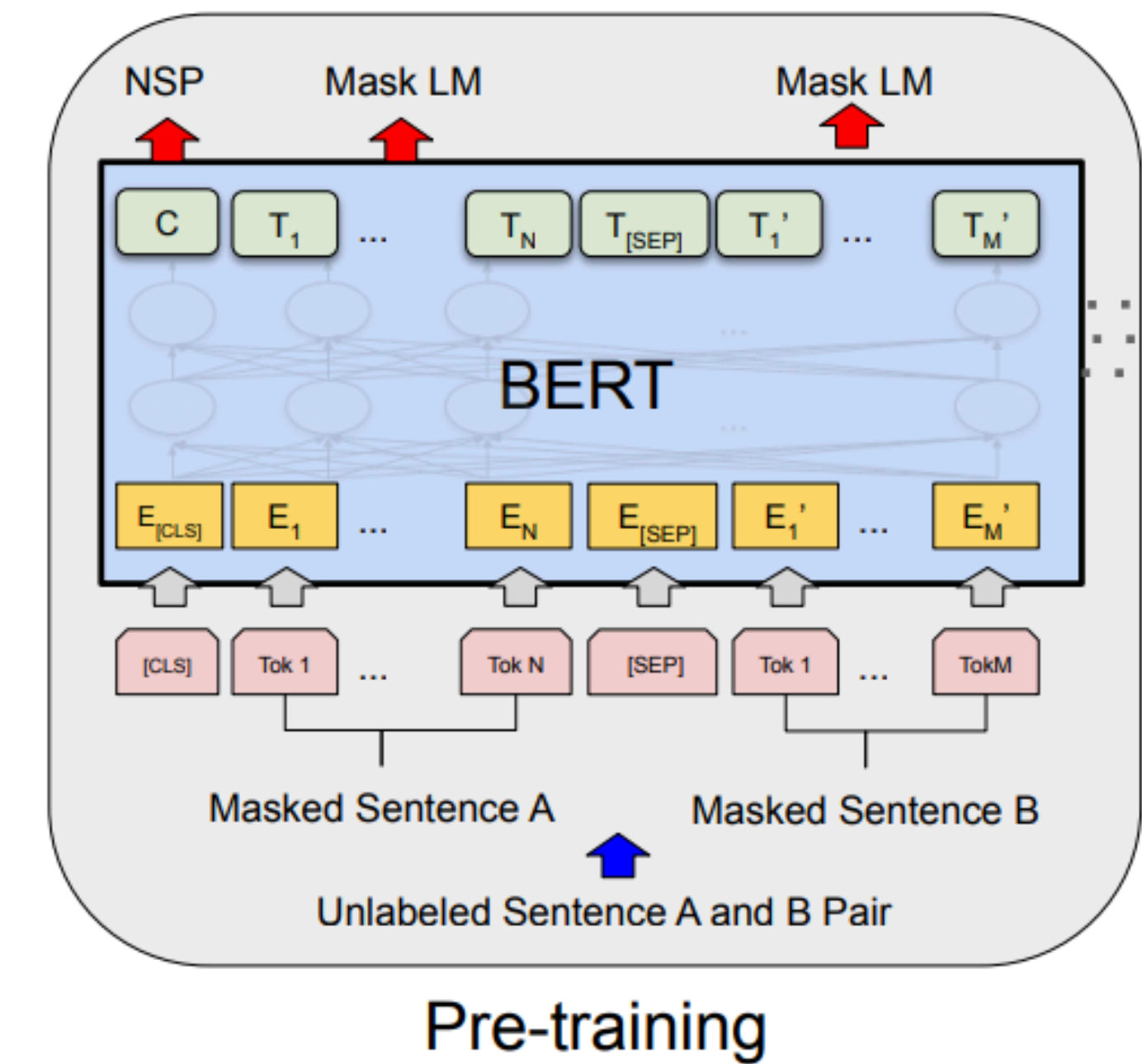


Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{my}$	$E_{dog}$	$E_{is}$	$E_{cute}$	$E_{[SEP]}$	$E_{he}$	$E_{likes}$	$E_{play}$	$E_{\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_B$	$E_B$	$E_B$	$E_B$	$E_B$
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

# Encoder-only

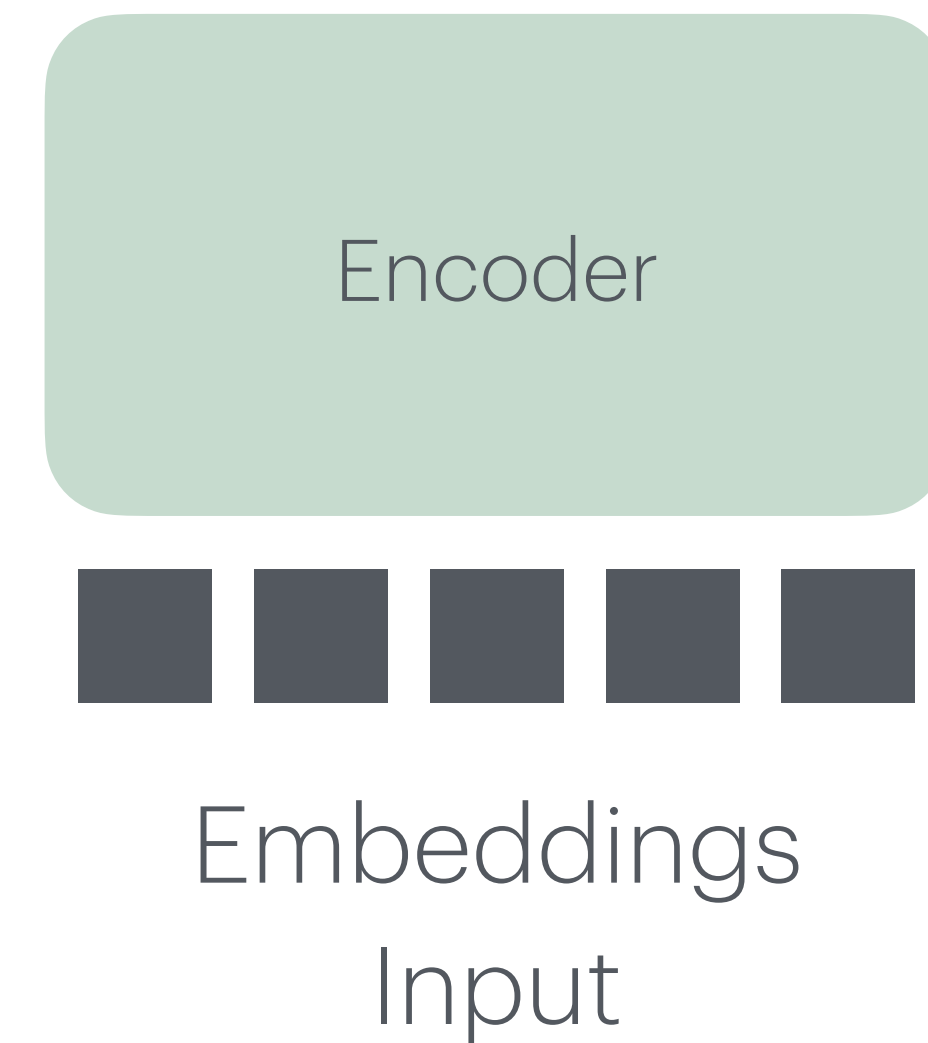
## Pre-training

- Masked language modeling
  - Mask out tokens in input
  - Reconstruct masked out tokens
- Next sentence prediction
  - Binary tasks
  - Uses [CLS] token



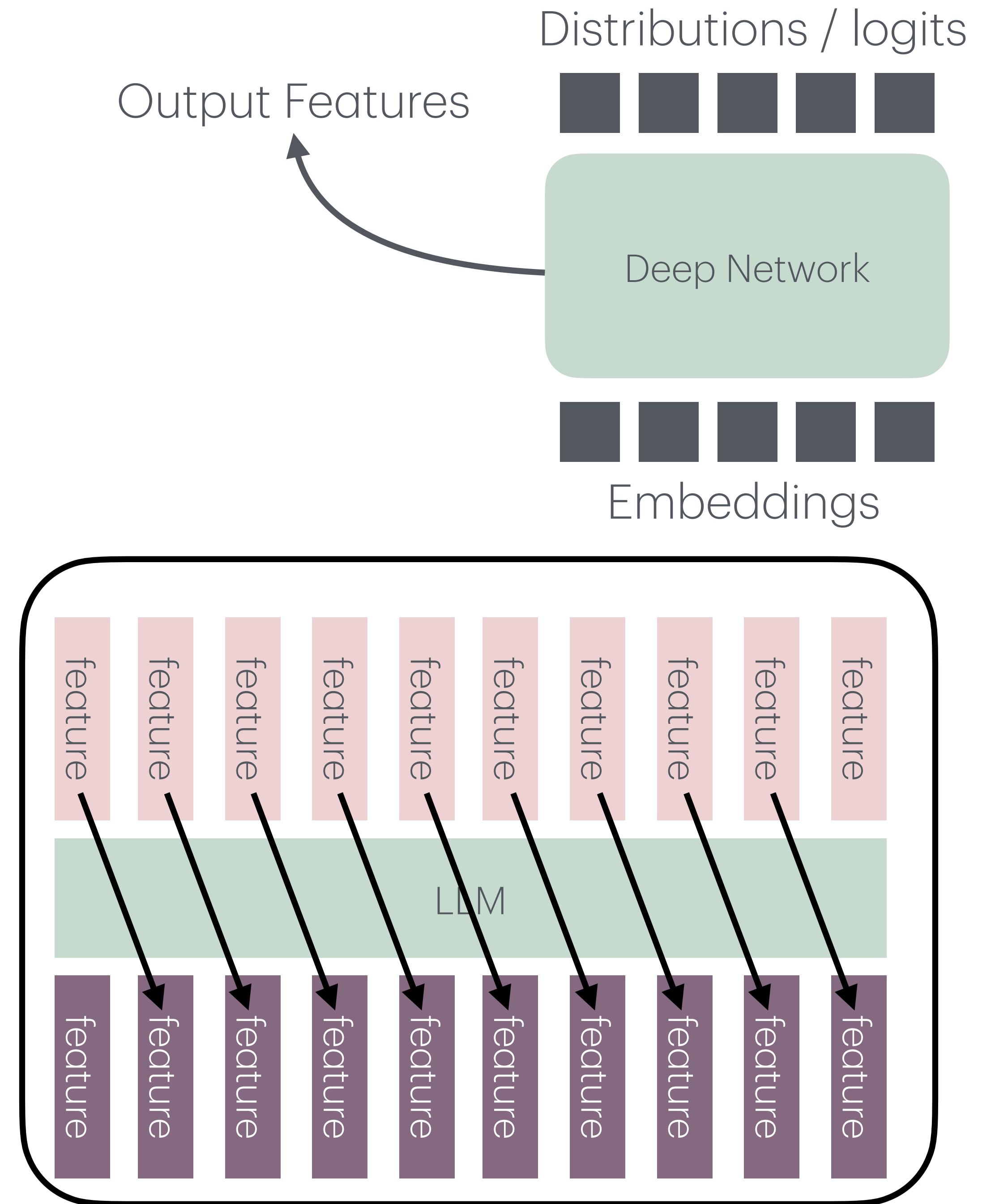
# Encoder-only model

- Very popular 5 years ago
- Good at embedding / classifying text
- Used less today
  - Not good at generation
  - Scale worse with data



# LLM - Architectures

- Encoder-Decoder (original transformer)
- Encoder-only
- **Decoder-only**
- Sequence Models

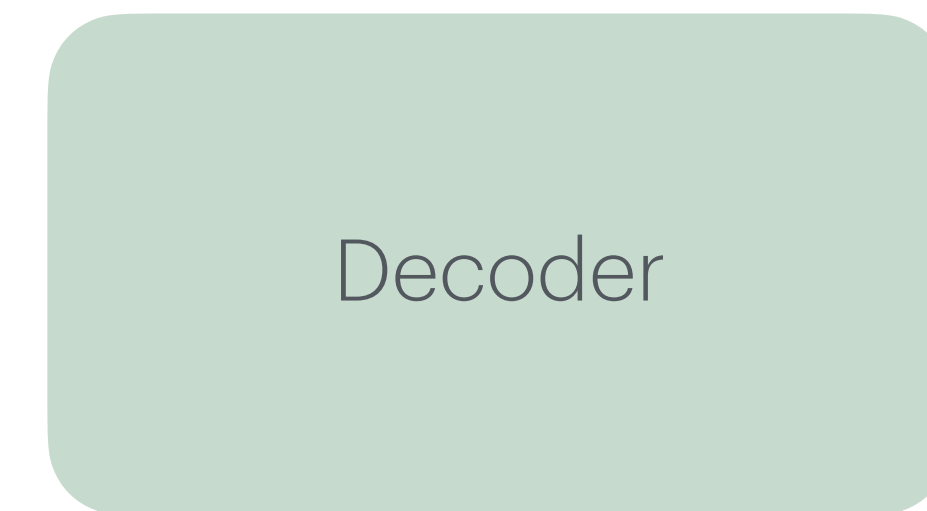


# Decoder-only models

## GPT and friends

- Input:
  - Text
- Output:
  - Autoregressive probability over text
  - Token-by-token

Distributions / logits



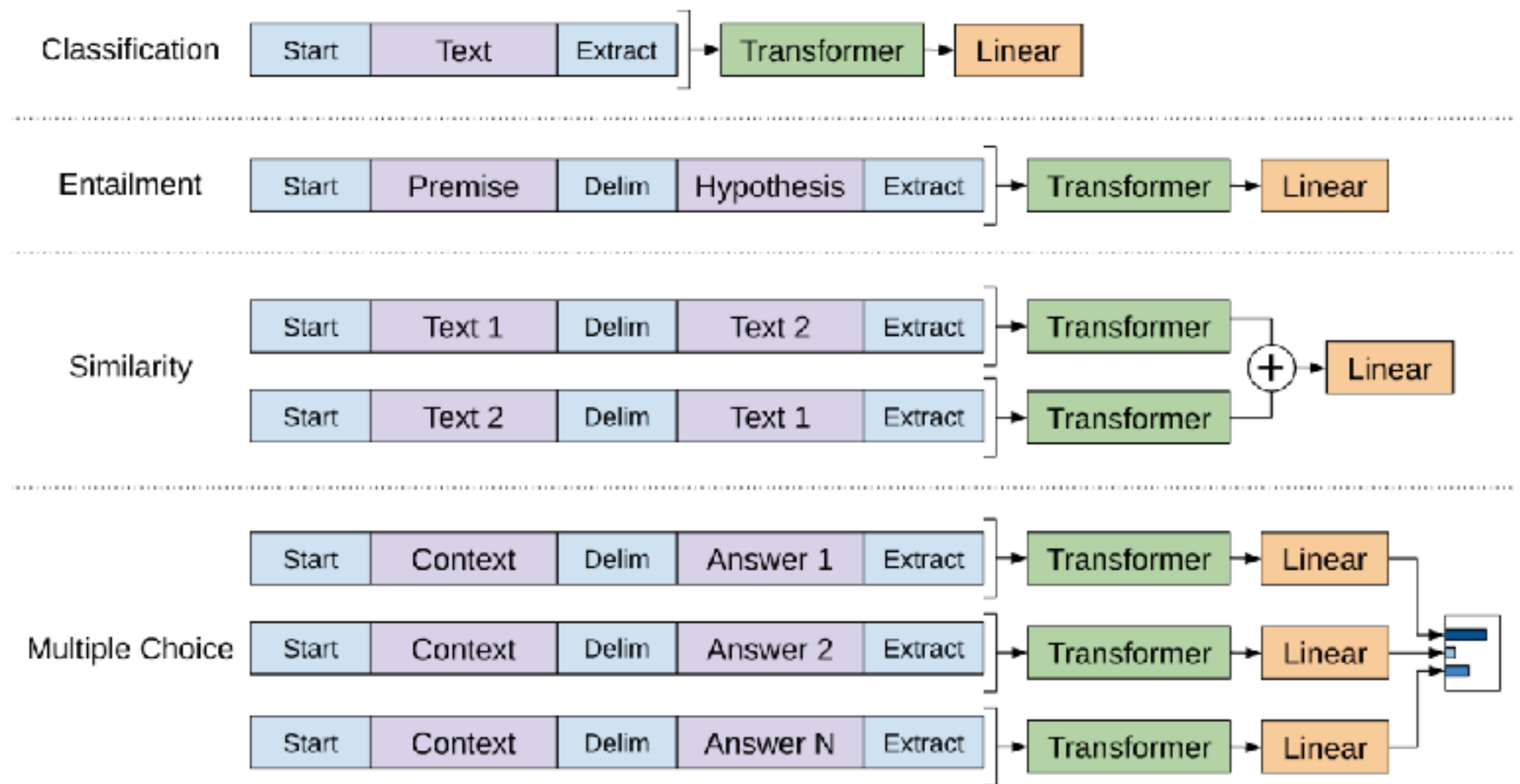
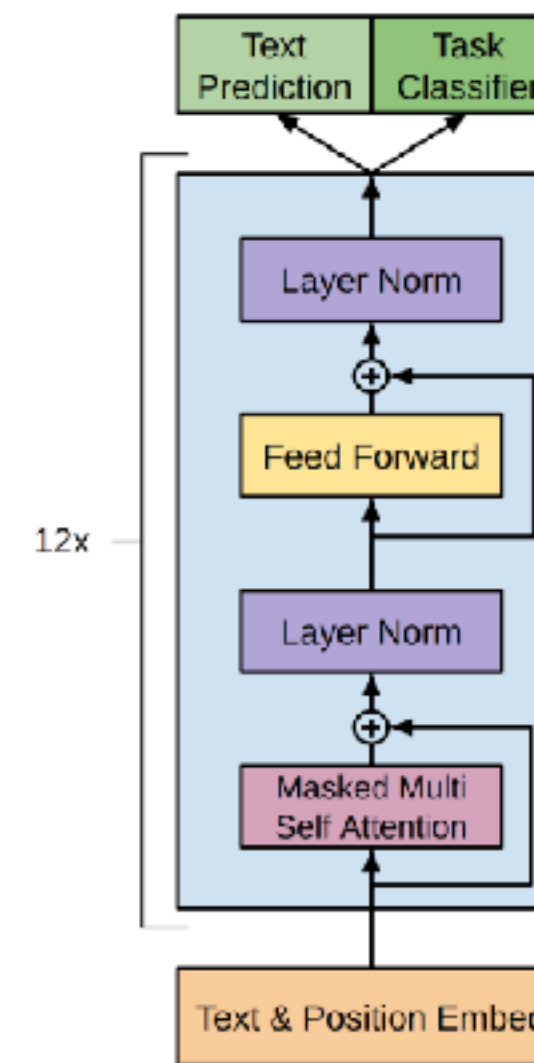
Embeddings

Output

# Decoder-only models

## Architecture

- Decoder-only (GPT-1)
- Causal self-attention
- MLP
- Special [extract] token for classification tasks

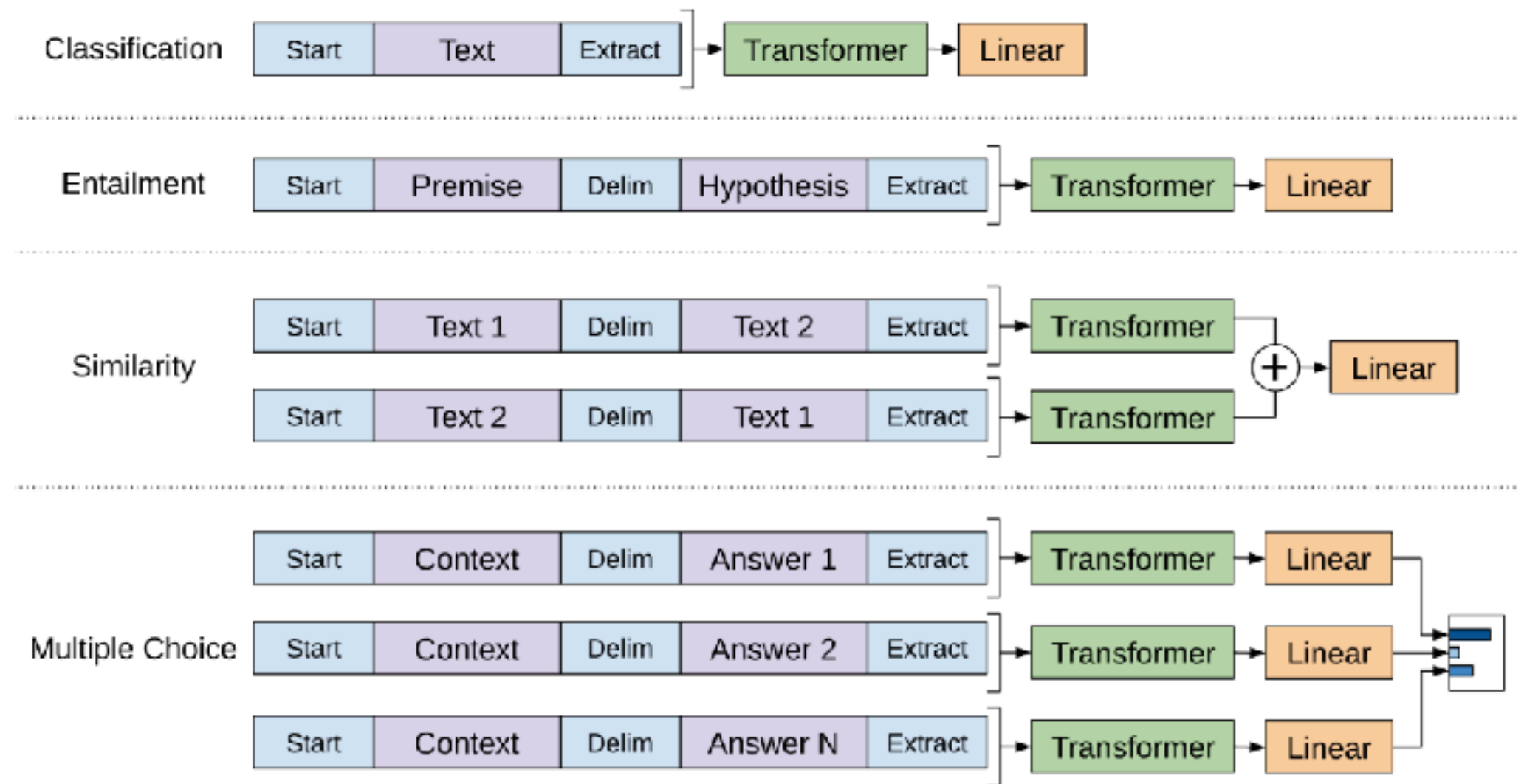
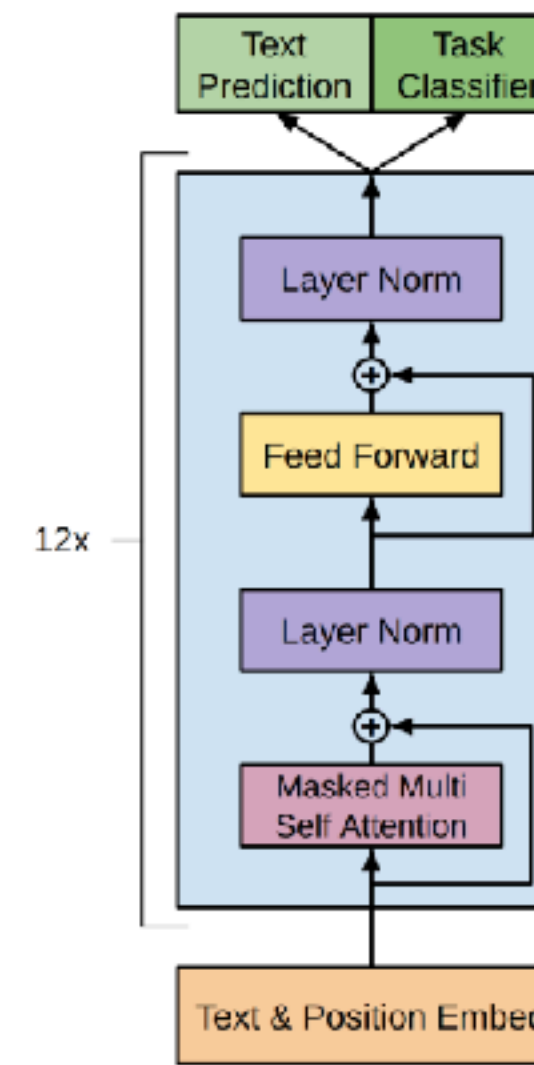




# Decoder-only models

## Architecture

- GPT-1: 12 layers, 768 hidden size, MLP 3072 dim, 12 attention heads; **0.1B** params
- GPT-2: 48 layers, 1600 hidden size, larger vocab (50k); **1.5B** params
- GPT-3: 96 layers, 12288 hidden size, 96 attention heads; **175.0B** params
- GPT-4: ??



Improving Language Understanding by Generative Pre-Training. Radford et al. 2018.

Language Models are Unsupervised Multitask Learners. Radford et al. 2019.

Language Models are Few-Shot Learners. Brown et al. 2020.

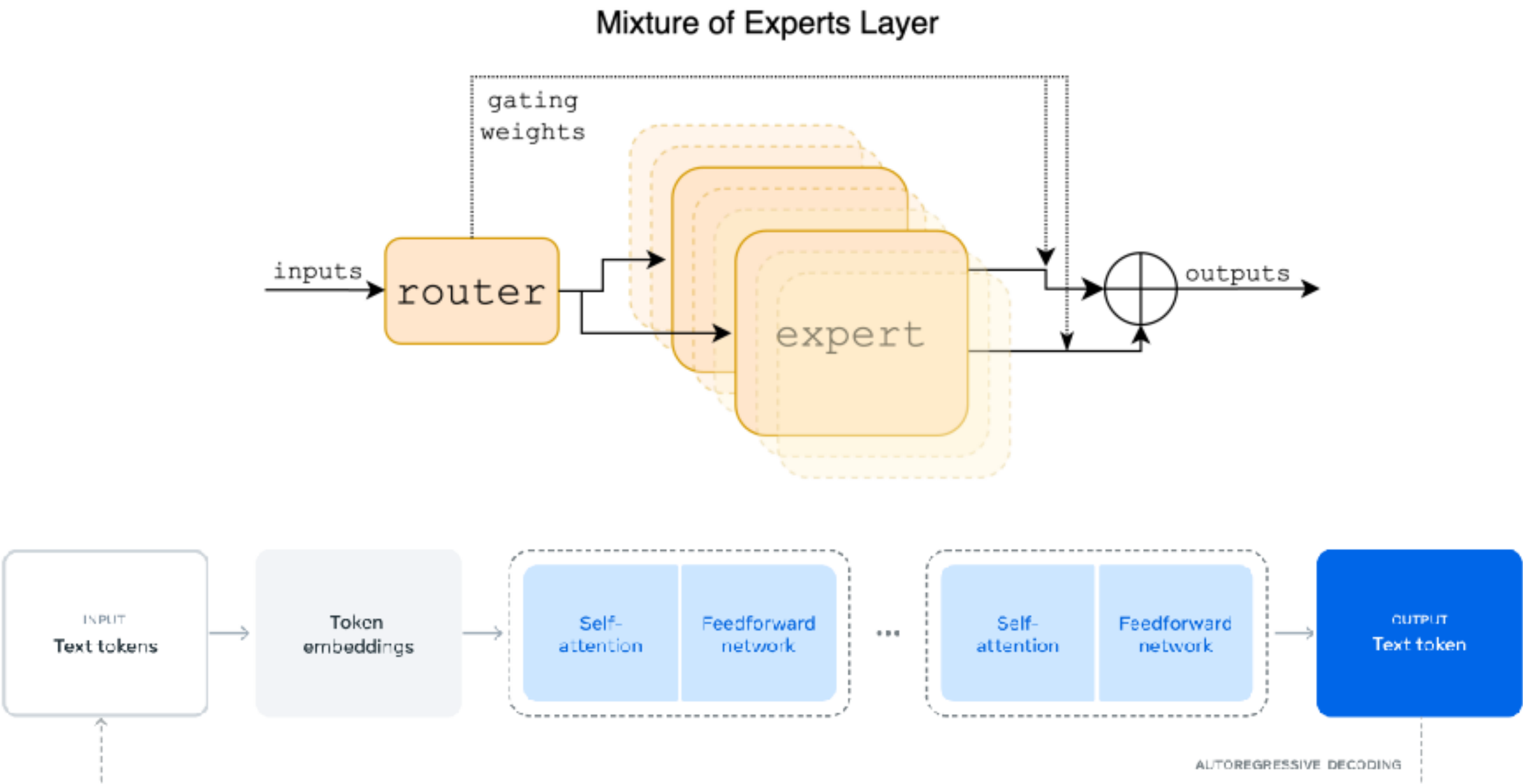
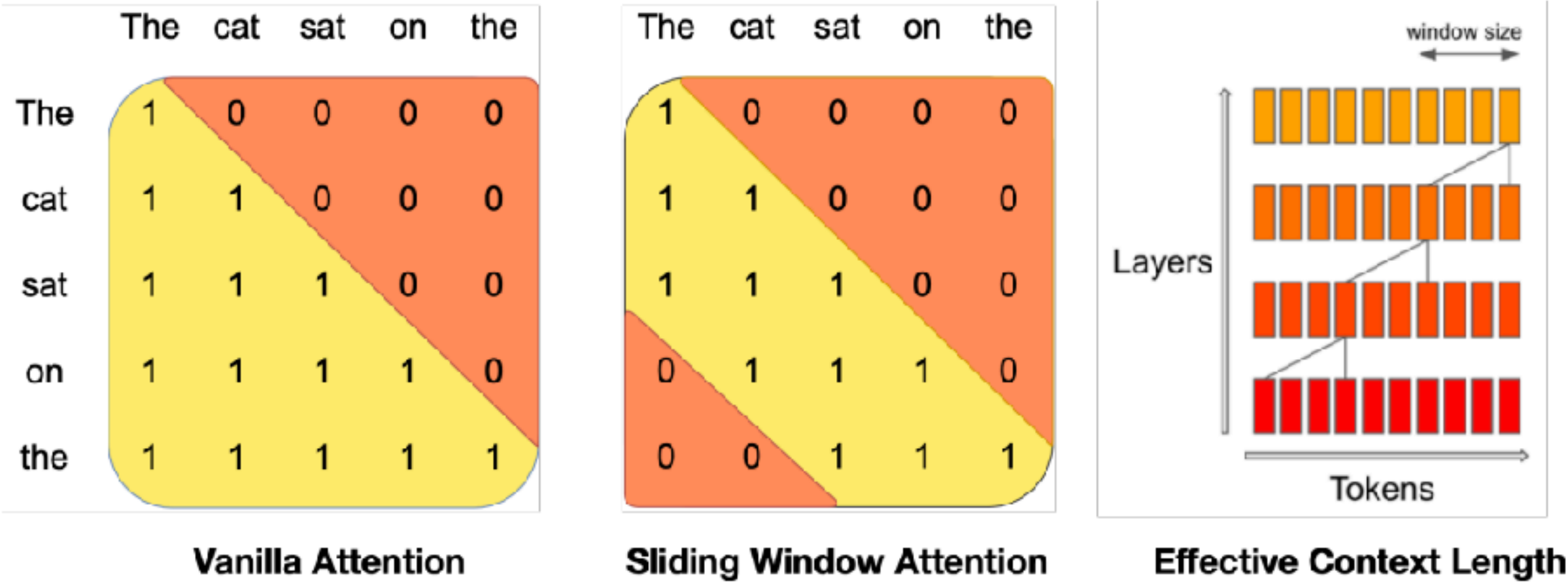
# Decoder-only models

## Architecture

- Mistral 7b: 32 layers, 4096 hidden size, MLP 14336 dim, 32 attention heads; window attention, Group Query Attention
- Mixtral 8x7b: Mistral 7b + Mixture of experts per MLP layer

- Llama 3.x:

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	$3 \times 10^{-4}$	$1.5 \times 10^{-4}$	$8 \times 10^{-5}$
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ( $\theta = 500,000$ )		



Mistral 7B. Jiang et al. 2023.

Mixtral of Experts. Jiang et al. 2023.

Llama 3 Herd of Models. Dubey et al. 2024.

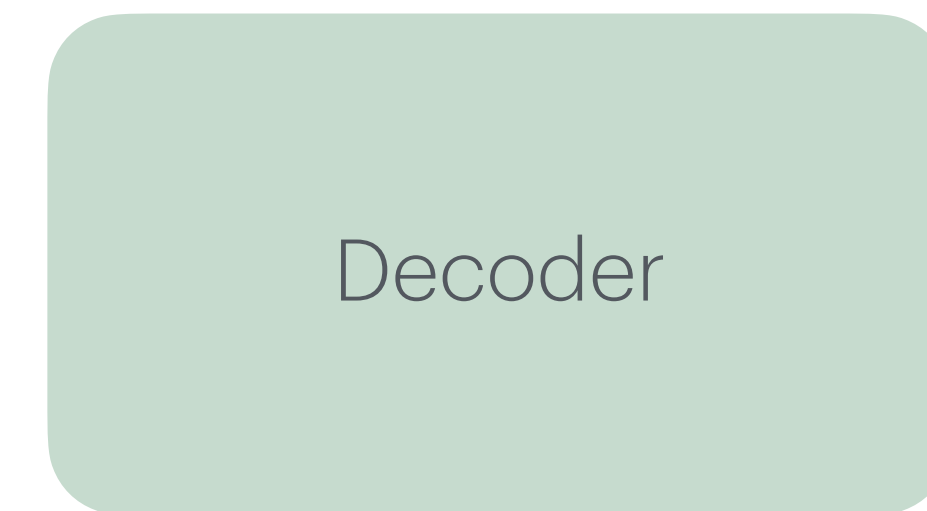


# Decoder-only models

## Architecture

- A big transformer
  - Input: Sequence
  - Output: Probability over next token

Distributions / logits



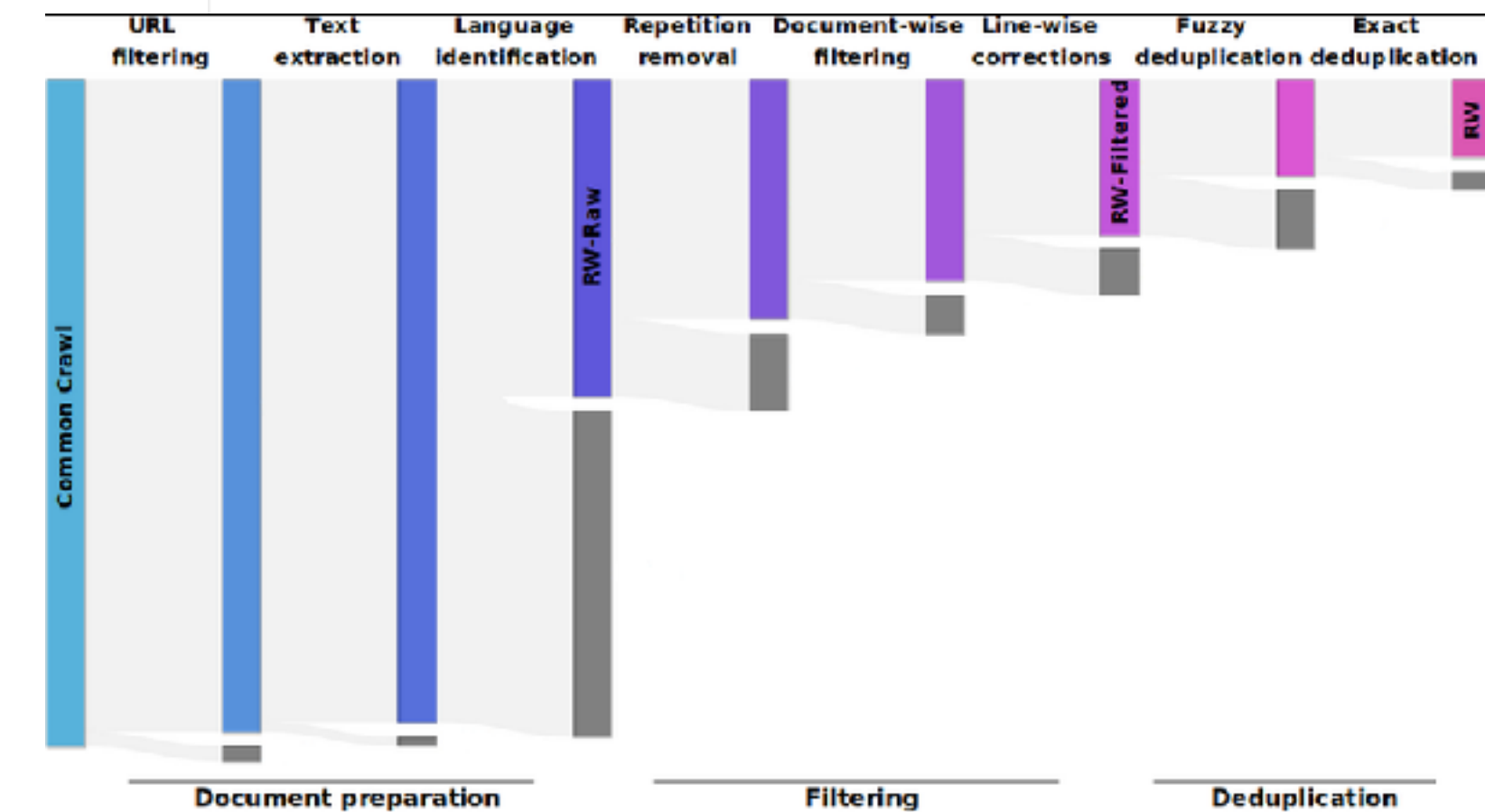
Embeddings

Output

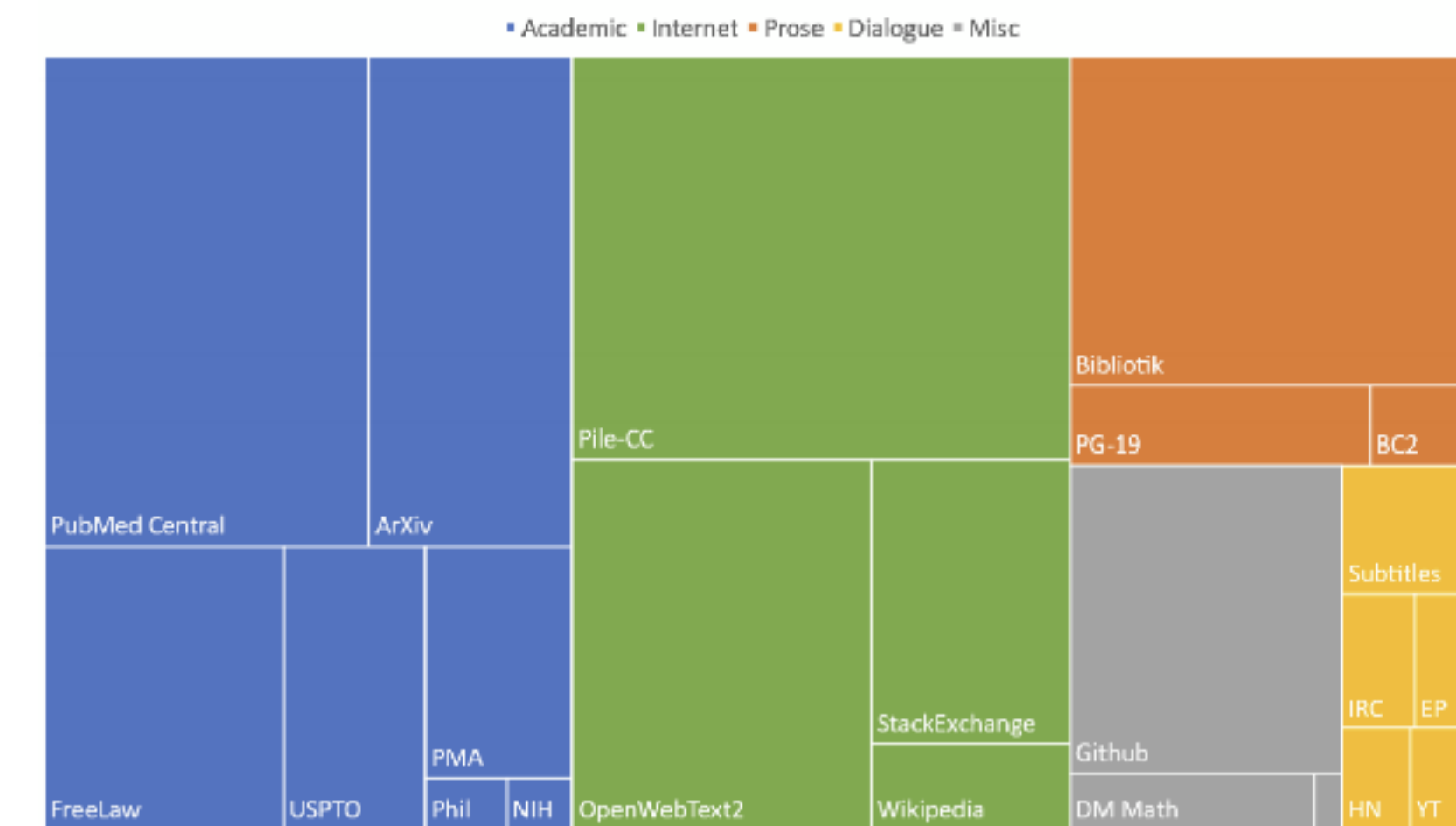
# Decoder-only models

## Pre-training

- Next token prediction
- On all written and digitized text available
- Filtered and cleaned up
- Industry: Lots of mined (and now licensed) internal data



Composition of the Pile by Category



Common Crawl. Common Crawl Foundation. 2008-present.

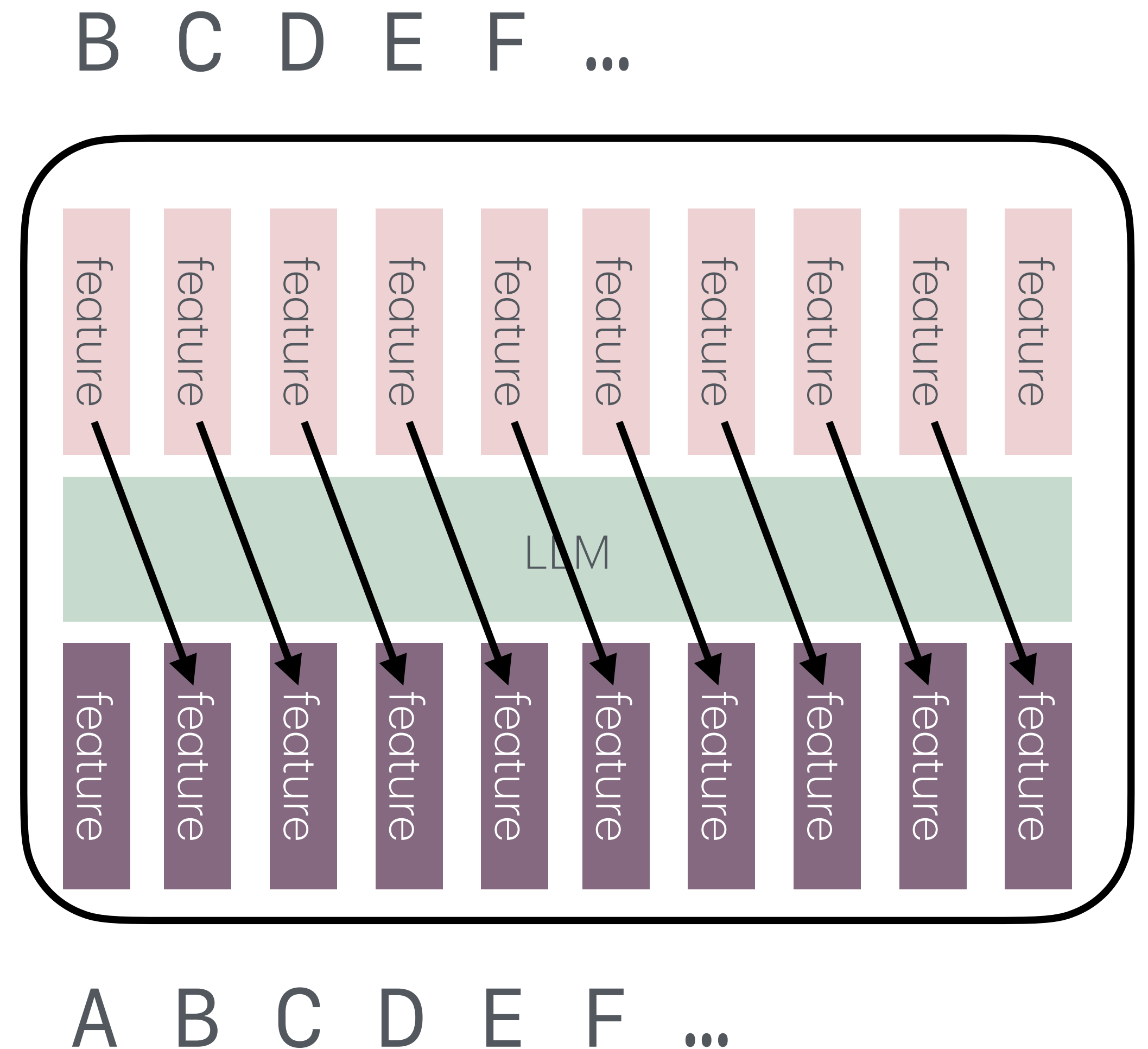
The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data. Penedo et al. 2023.

The Pile: An 800GB Dataset of Diverse Text for Language Modeling. Gao et al. 2020.

# Decoder-only models

## Pre-training

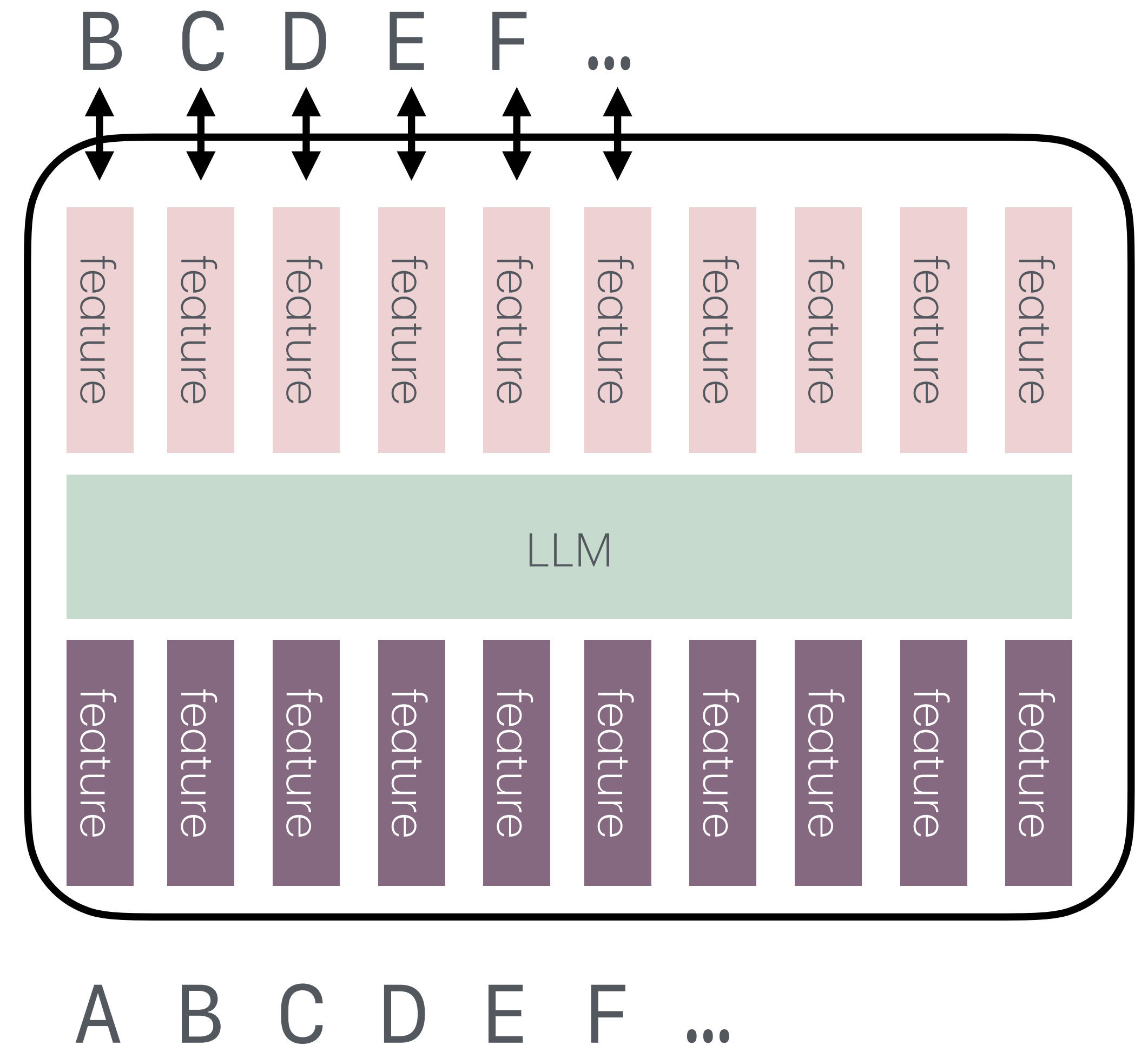
- Teacher forcing
  - Feed in ground truth tokens
  - Supervised shifted ground truth



# Decoder-only models

## Pre-training

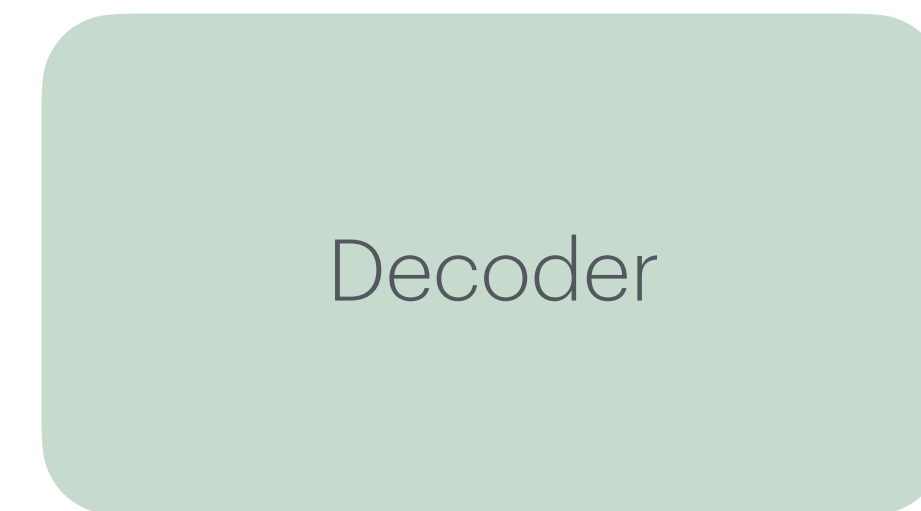
- Teacher forcing
  - Feed in ground truth tokens
  - Supervised shifted ground truth



# Decoder-only models

- Demo  
ollama run llama3.1:8b-text-q4\_0

Distributions / logits



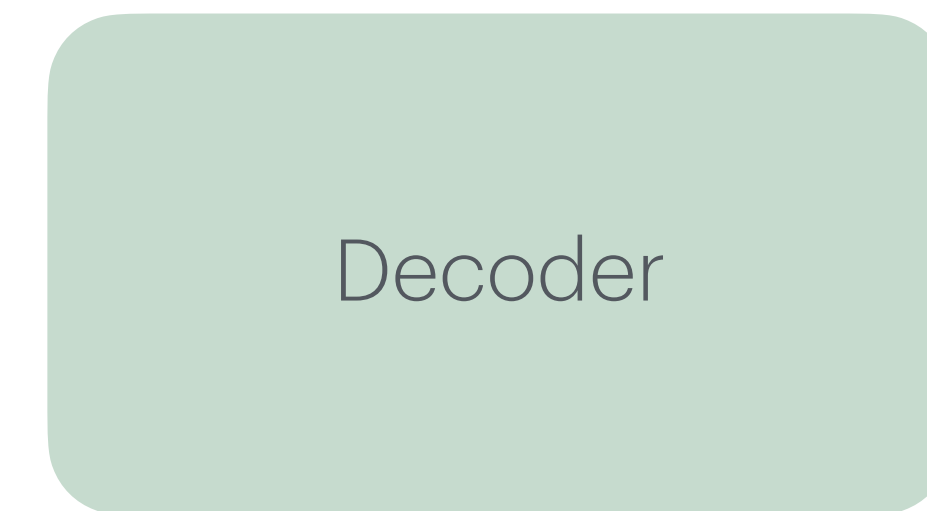
Embeddings

Output

# Decoder-only model

- Decoder-only models = LLM nowadays
- Best generation models
- Good embeddings
- Able to memorize training data well
  - Up to 2 bits per parameter [1]

Distributions / logits

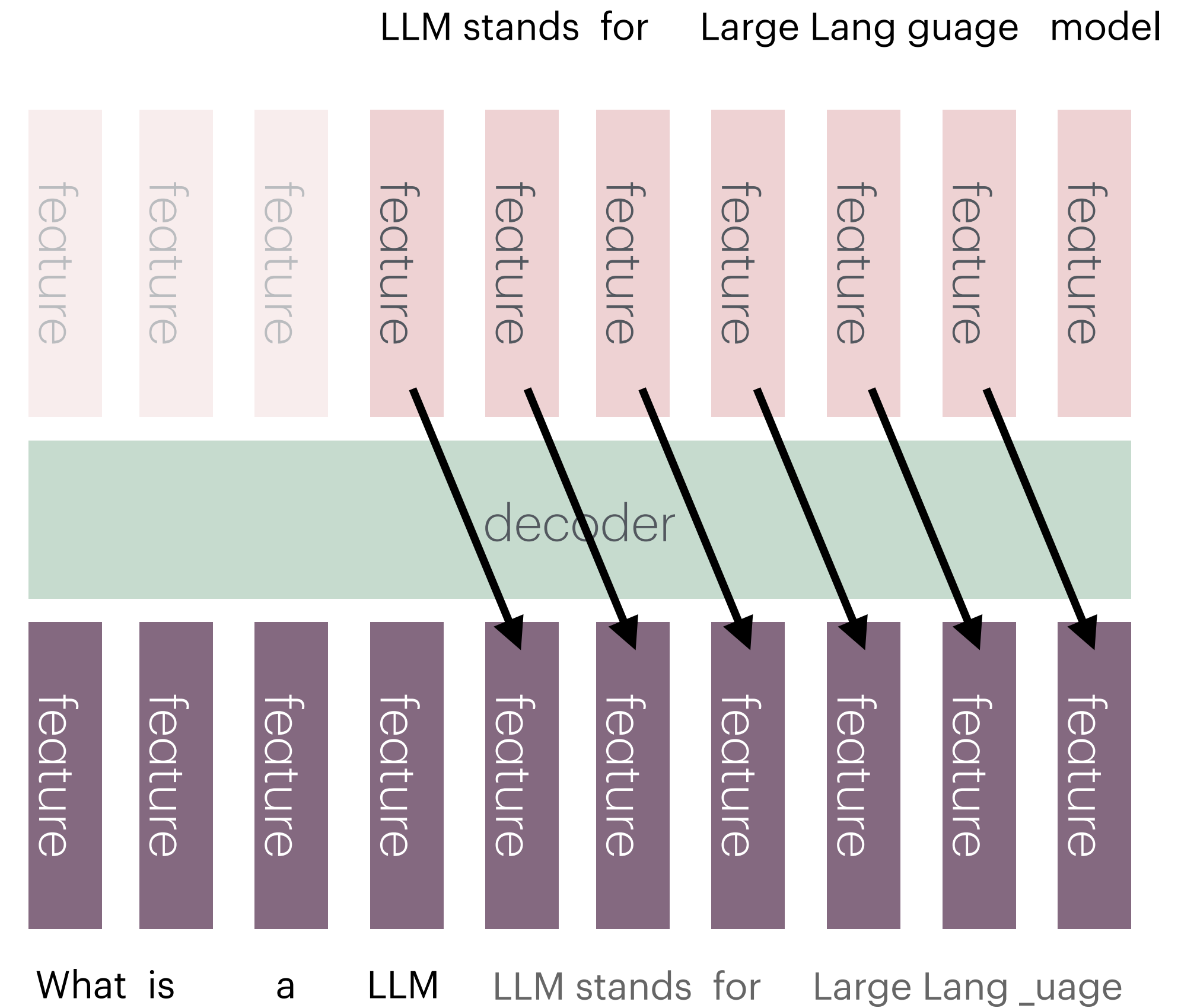


Embeddings

Output

# Decoder-only model

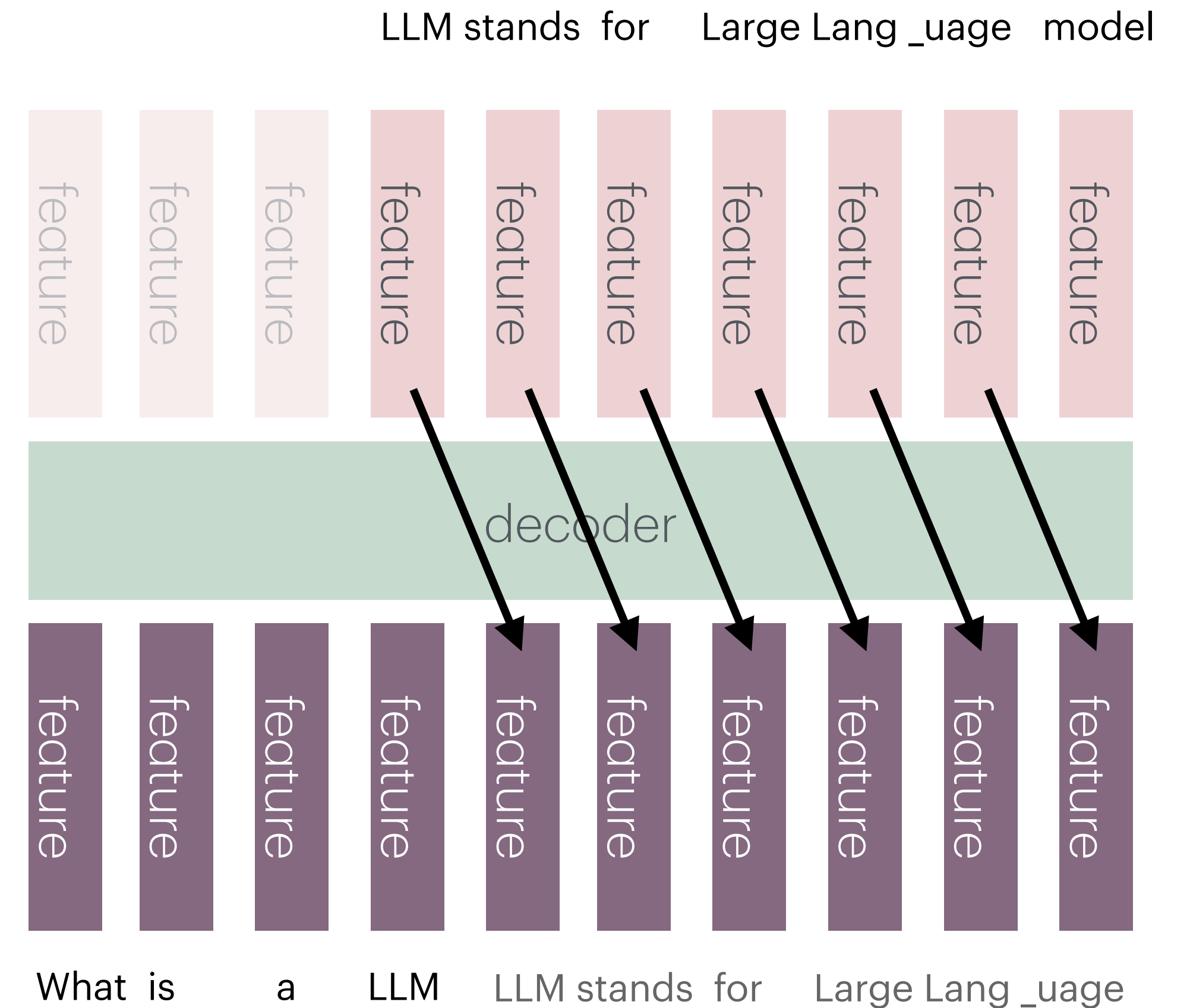
- Decoder-only models can mimic encoder-decoder models



# Decoder-only model

Where do models store data?

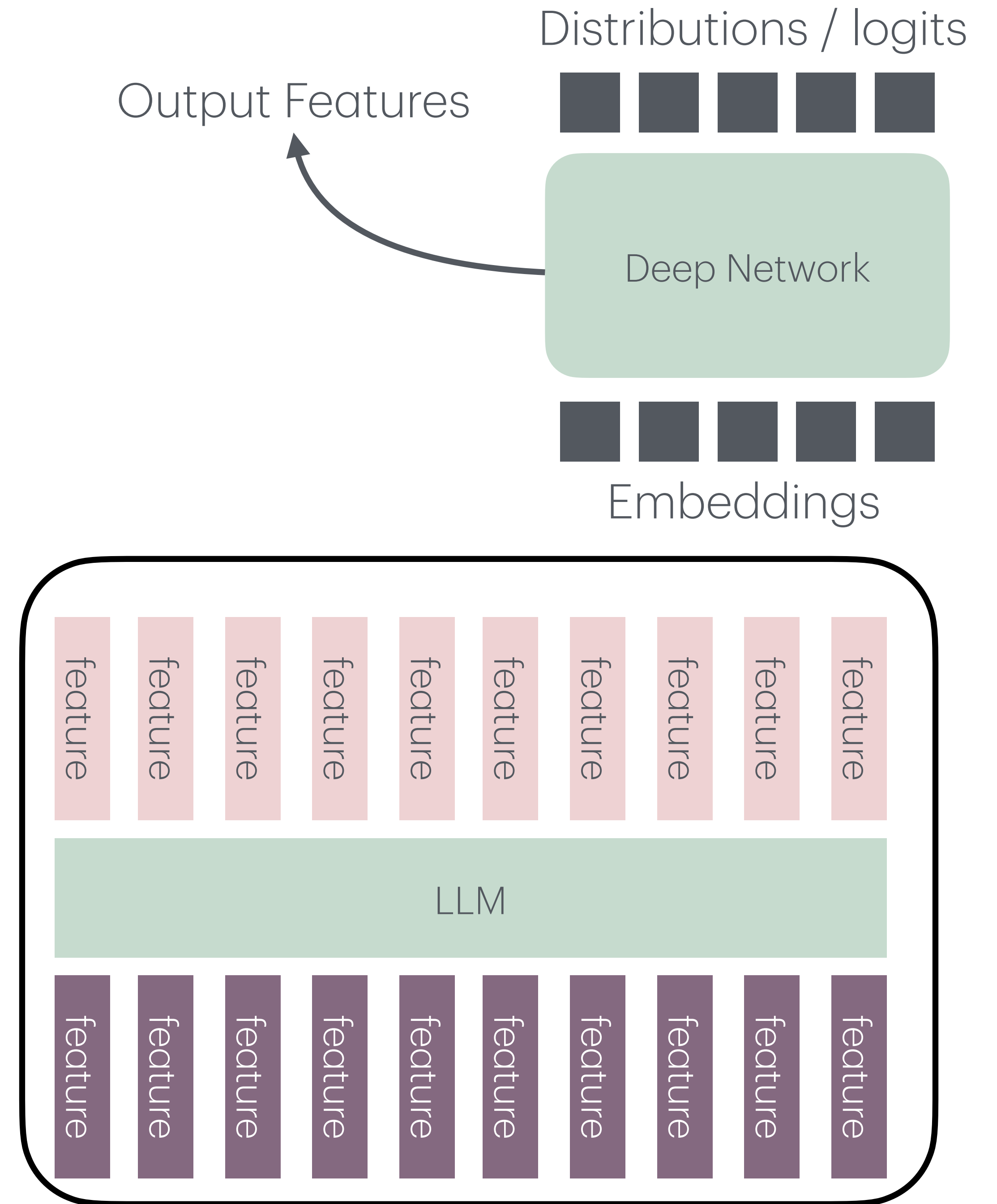
- Option A
  - In their context / prompt
- Option B
  - In their weights





# LLM - Architectures

- Encoder-Decoder (original transformer)
- Encoder-only
- Decoder-only
- **Sequence Models**

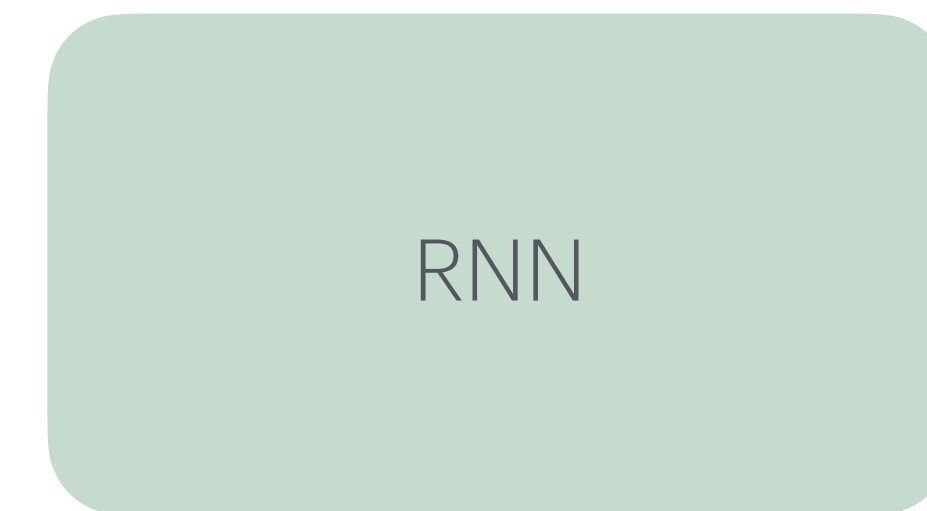


# Sequence Models

## Mamba and friends

- Input:
  - Text
- Output:
  - Autoregressive probability over text
  - Token-by-token

Distributions / logits



Embeddings

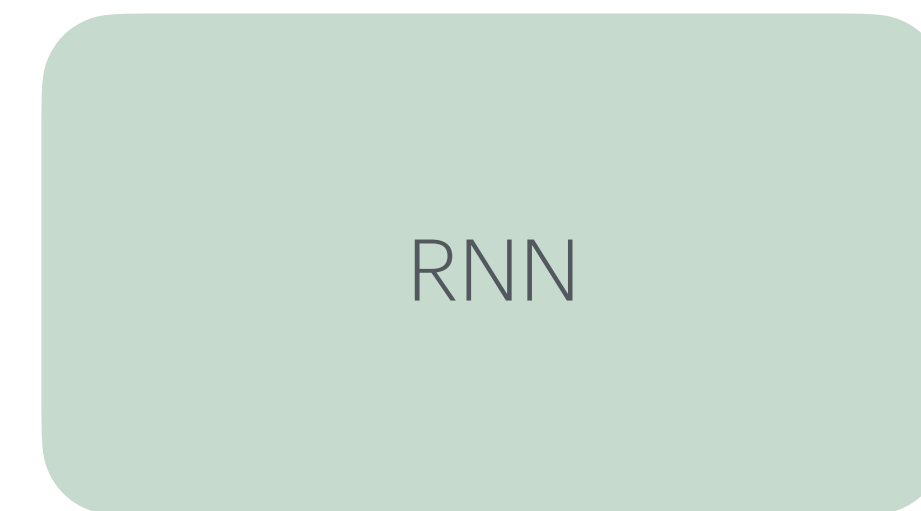
Output

# Sequence Models

## Architecture

- State-space model layer:
  - Input:  $u_t$
  - Hidden state:  $x_t = Ax_{t-1} + Bu_t$
  - Output:  $y_t = Cx_t$

Distributions / logits



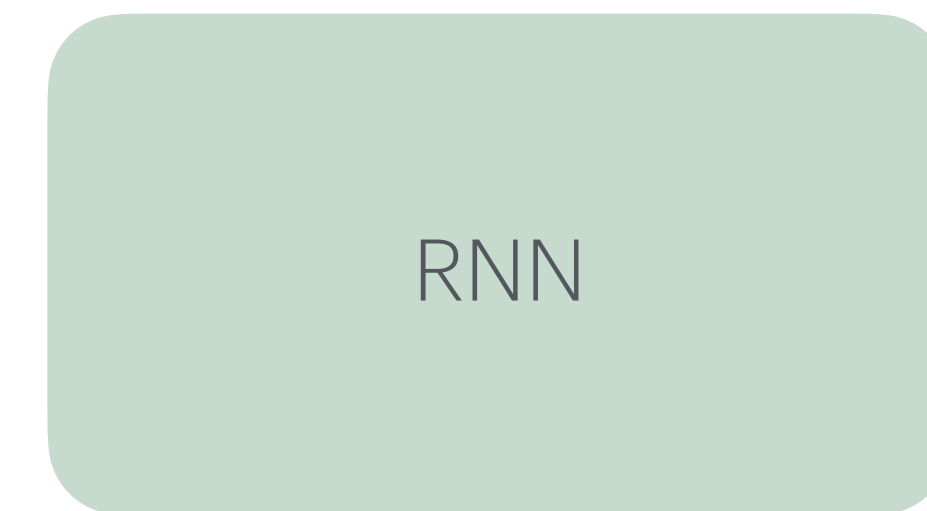
Embeddings  
Output

# Sequence Models

## Training

- State-space model layer
  - Unroll
  - Similar to convolution / transformer with more structured attention matrix
  - Same training as decoder-only model

Distributions / logits



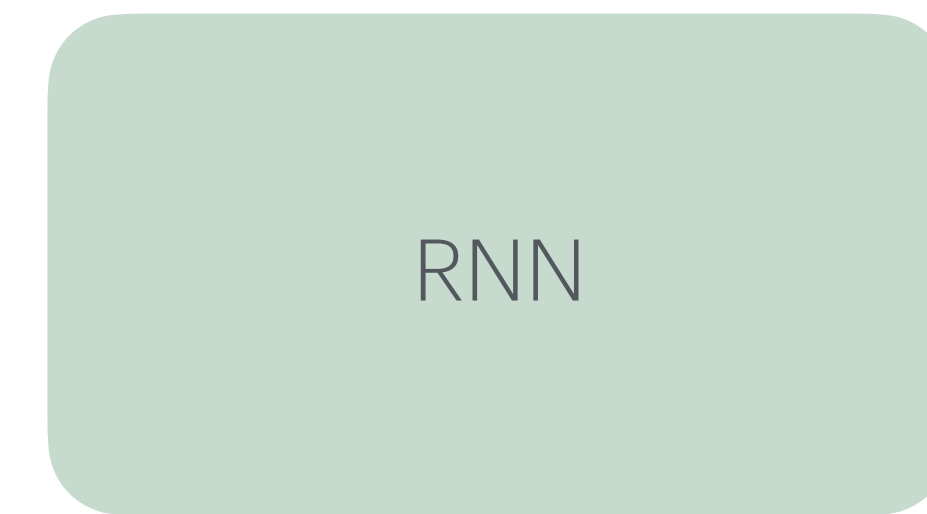
Embeddings  
Output

# Sequence Models

## Discussion

- Potentially faster inference than transformer
- Similar training
- Fixed bottleneck
- Not yet as same performance

Distributions / logits



Embeddings  
Output

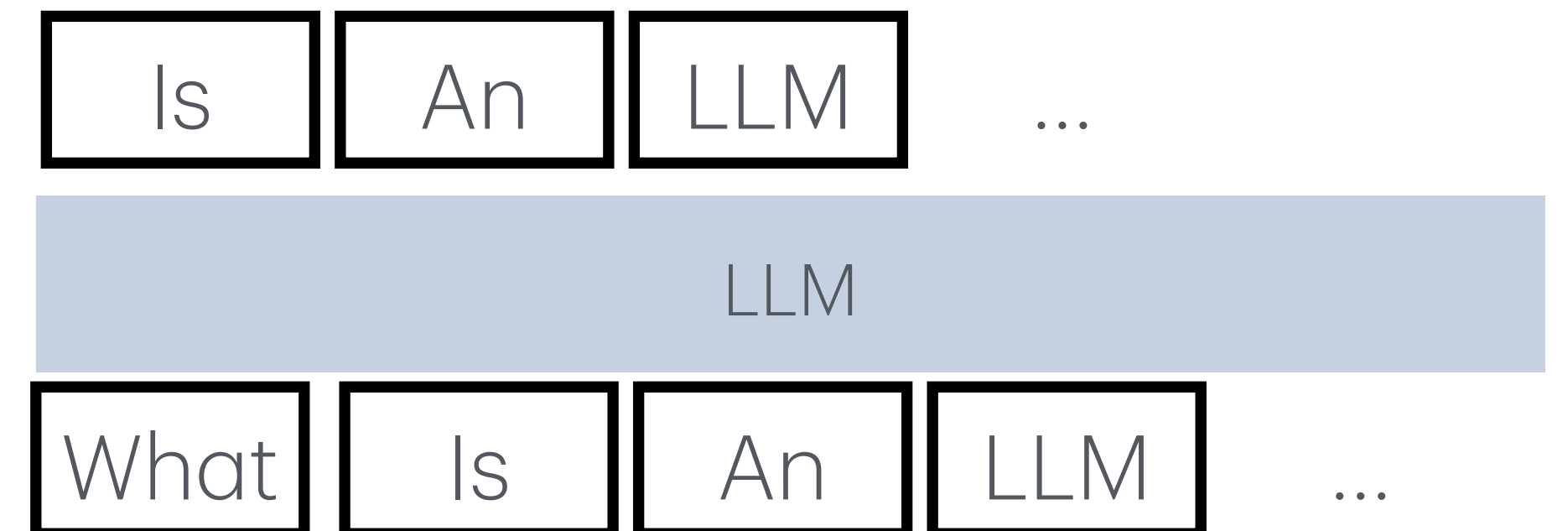
# Architecture

- LLMs are decoder only nowadays
- Modeling auto-regressive distribution over tokens
- $P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$
- Able to memorize training data well
- Up to 2 bits per parameter [1]
- Very good generative models

Distributions / logits



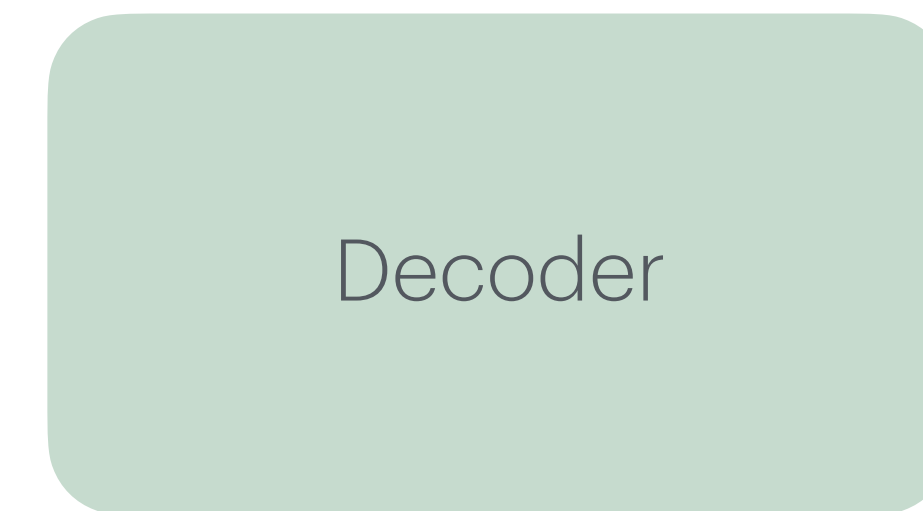
Embeddings  
Output



# What's next?

- Generation: How do we sample from decoder-only models?
- Instruction tuning: How do we convert language models into chat models?
- Preference tuning: How to tell an what “good” and safe answers are?
- Tasks and datasets: How to evaluate performance of LLMs

Distributions / logits



Embeddings  
Output

# References

- [1] Improving Language Understanding by Generative Pre-Training. Radford et al. 2018.
- [2] Attention Is All You Need. Vaswani et al. 2017.
- [3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Devlin et al. 2018.
- [4] Physics of Language Models: Part 3.1, Knowledge Storage and Extraction. Allen-Zhu and Li. 2023.
- [5] Language Models are Unsupervised Multitask Learners. Radford et al. 2019.
- [6] Language Models are Few-Shot Learners. Brown et al. 2020.
- [7] Mistral 7B. Jiang et al. 2023.
- [8] Mixtral of Experts. Jiang et al. 2023.
- [9] Llama 3 Herd of Models. Dubey et al. 2024.
- [10] Common Crawl. Common Crawl Foundation. 2008-present.
- [11] The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data. Penedo et al. 2023.
- [12] The Pile: An 800GB Dataset of Diverse Text for Language Modeling. Gao et al. 2020.
- [13] Mamba: Linear-Time Sequence Modeling with Selective State Spaces. Gu et al. 2023.
- [14] Efficiently Modeling Long Sequences with Structured State Spaces. Gu et al. 2021.



# Generation

# Language Models

- Decoder-only LLMs
- Modeling auto-regressive distribution over tokens
- $P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$

Distributions / logits



Embeddings  
Output

Is An LLM ...



What Is An LLM ...

# Language Models

## Sampling

- How to generate text **t** from  $P$

Distributions / logits



Embeddings  
Output

Is An LLM ...

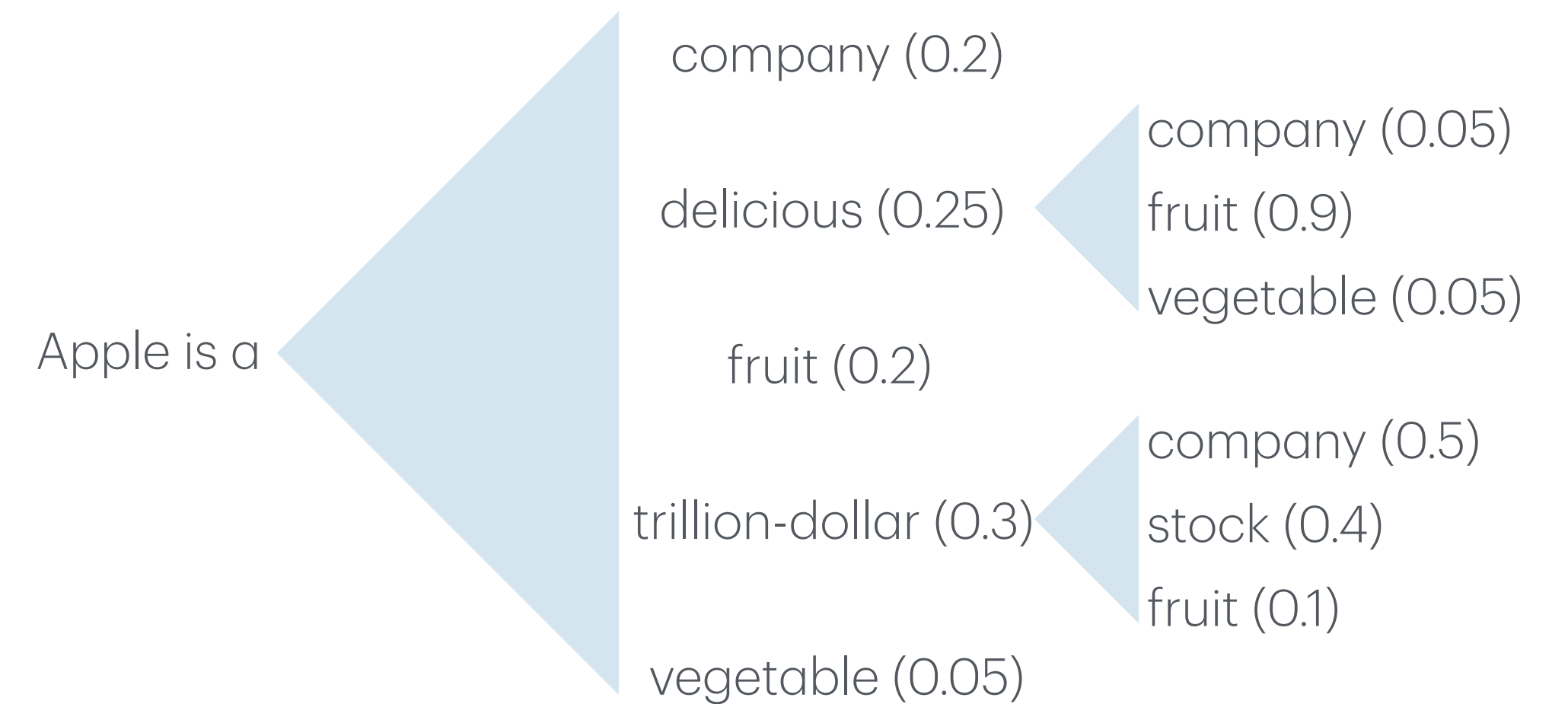


What Is An LLM ...

$$P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$$

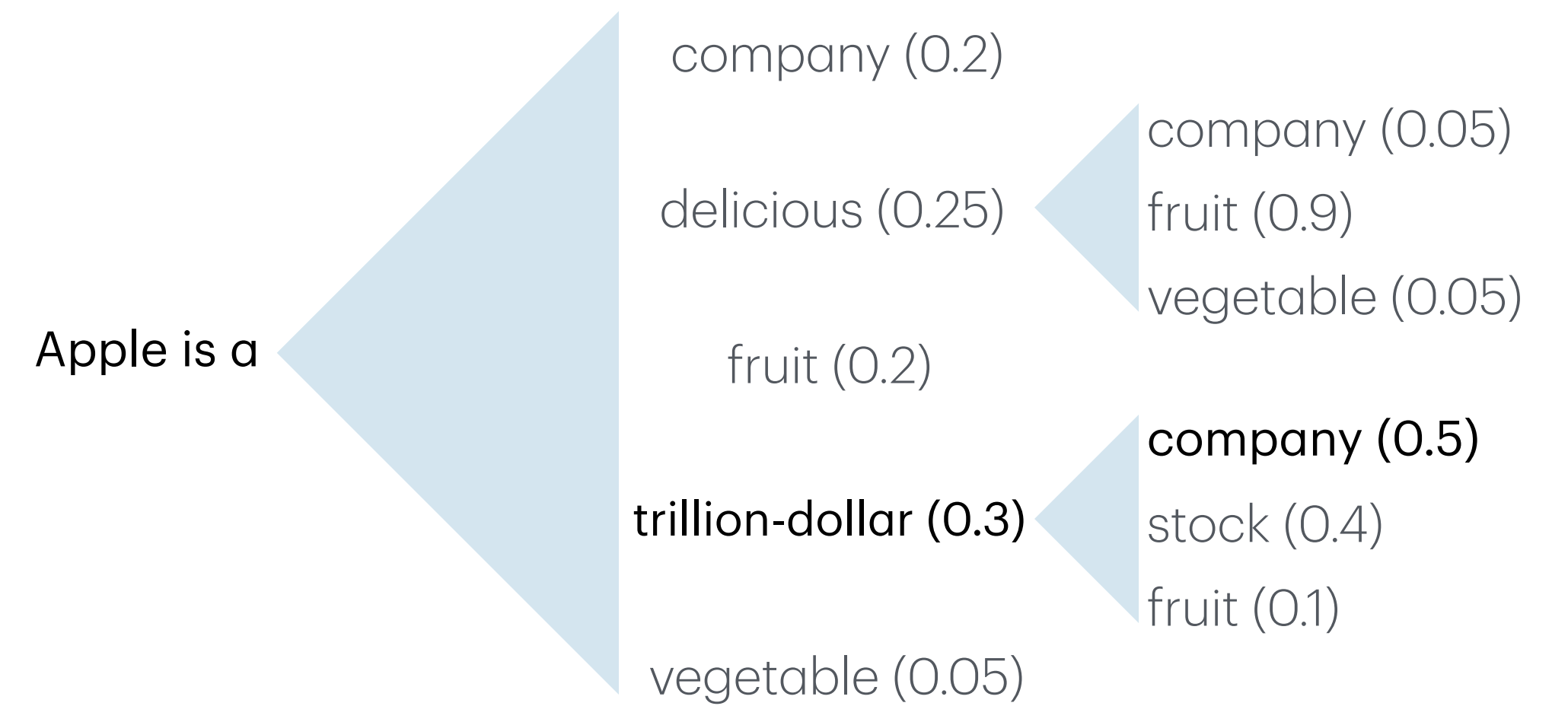
# Generation

- LLM produces distribution over tokens
  - Exponentially large output space
- Tension between
  - Generalization (not assigning prob=0)
  - Fidelity (odd low-probability outputs)



# Sampling - Greedy search

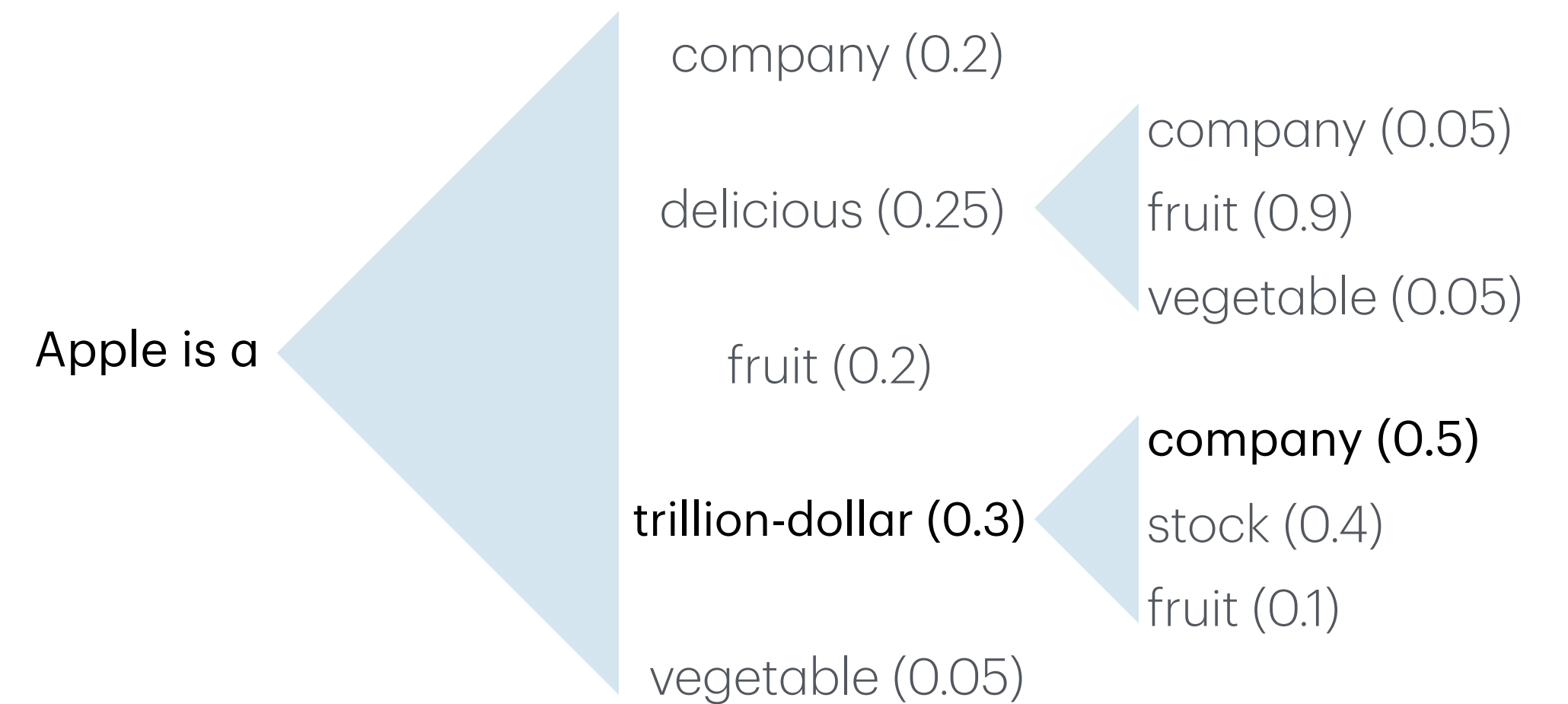
- Pick highest probability token next



# Sampling - Greedy search

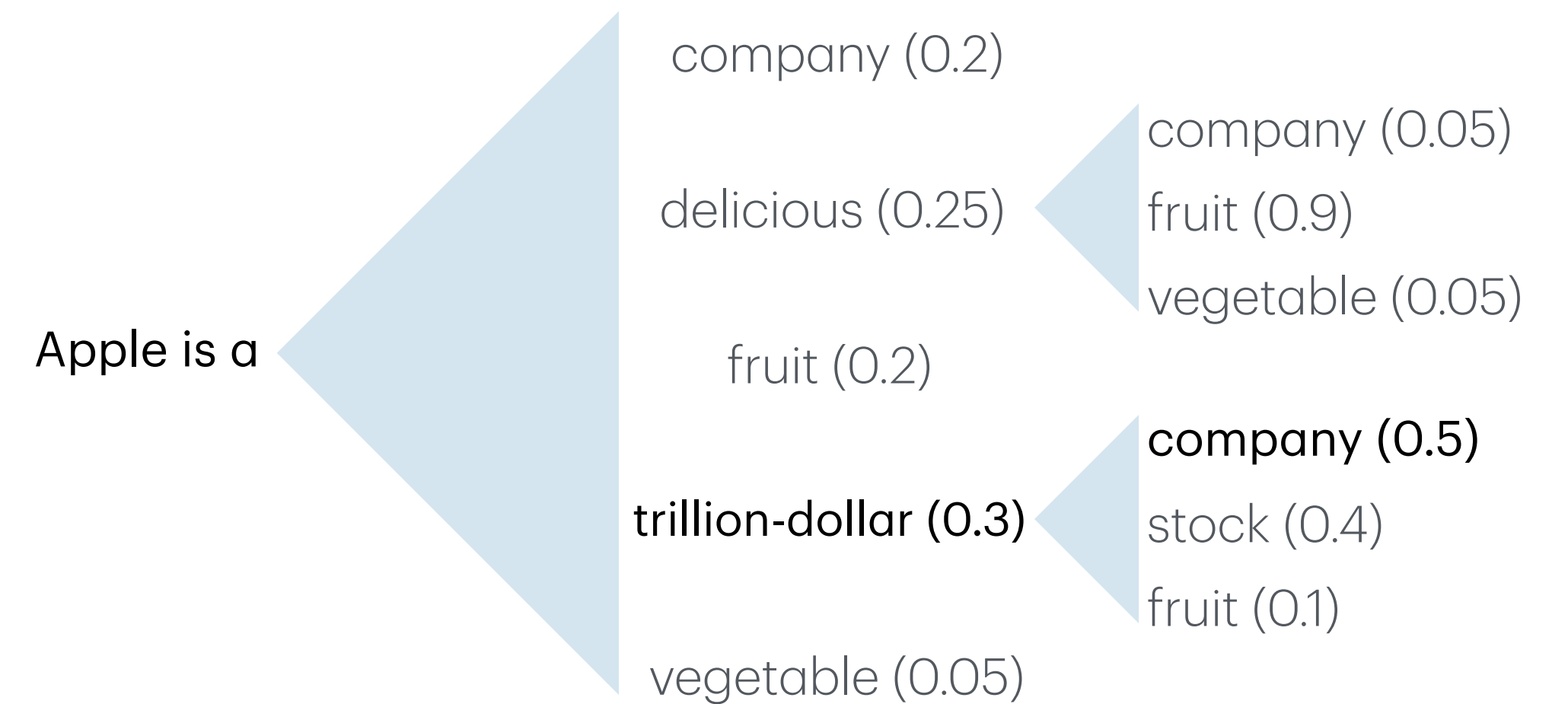
- A demo  
ollama run llama3.1:greedy

```
FROM llama3.1:8b-text-q4_0  
PARAMETER temperature 0  
PARAMETER top_k 1000  
PARAMETER top_p 1.0
```



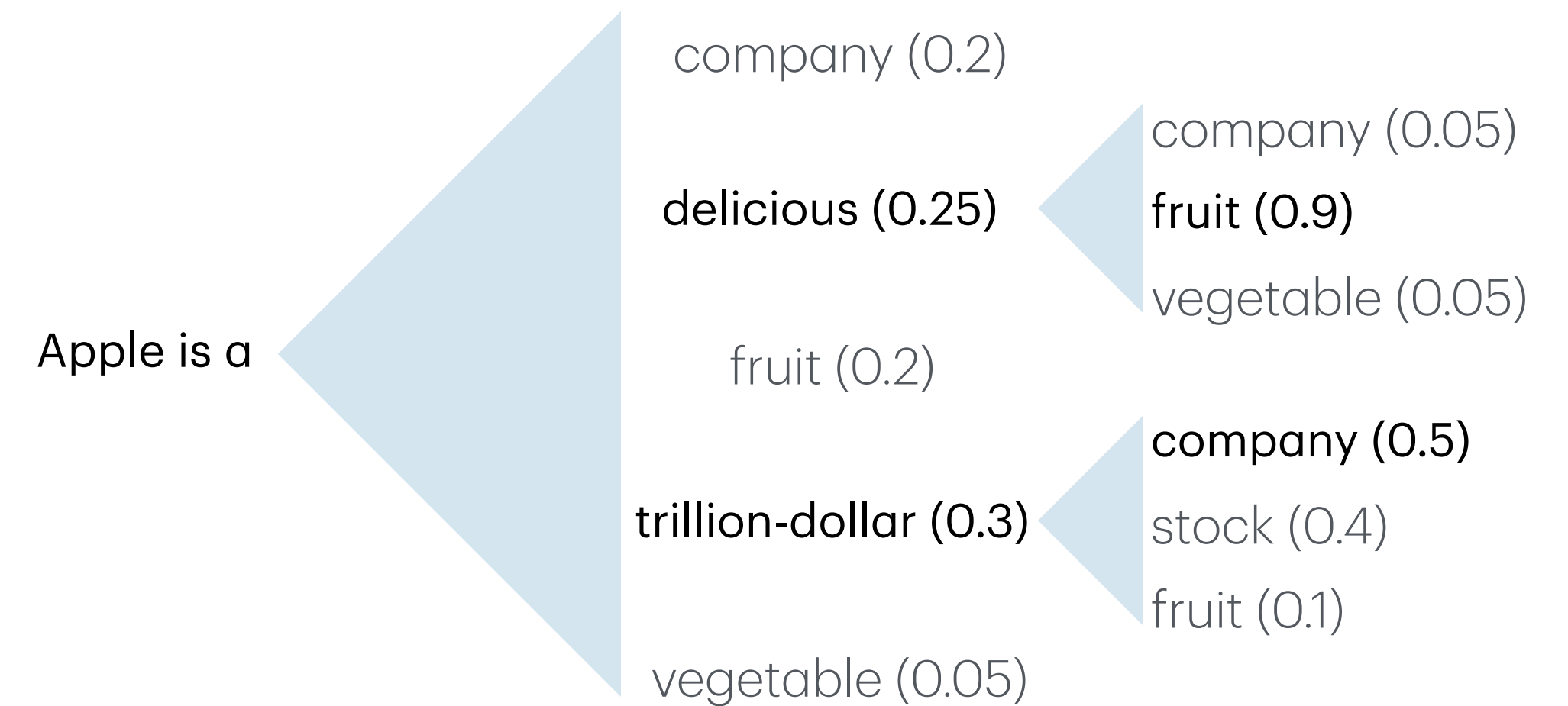
# Sampling - Greedy search

- Pick highest probability token next
- 😊 Super simple
- 😊 Computationally efficient
- 😞 Single sequence
- 😞 Bad local decisions



# Sampling - Beam search

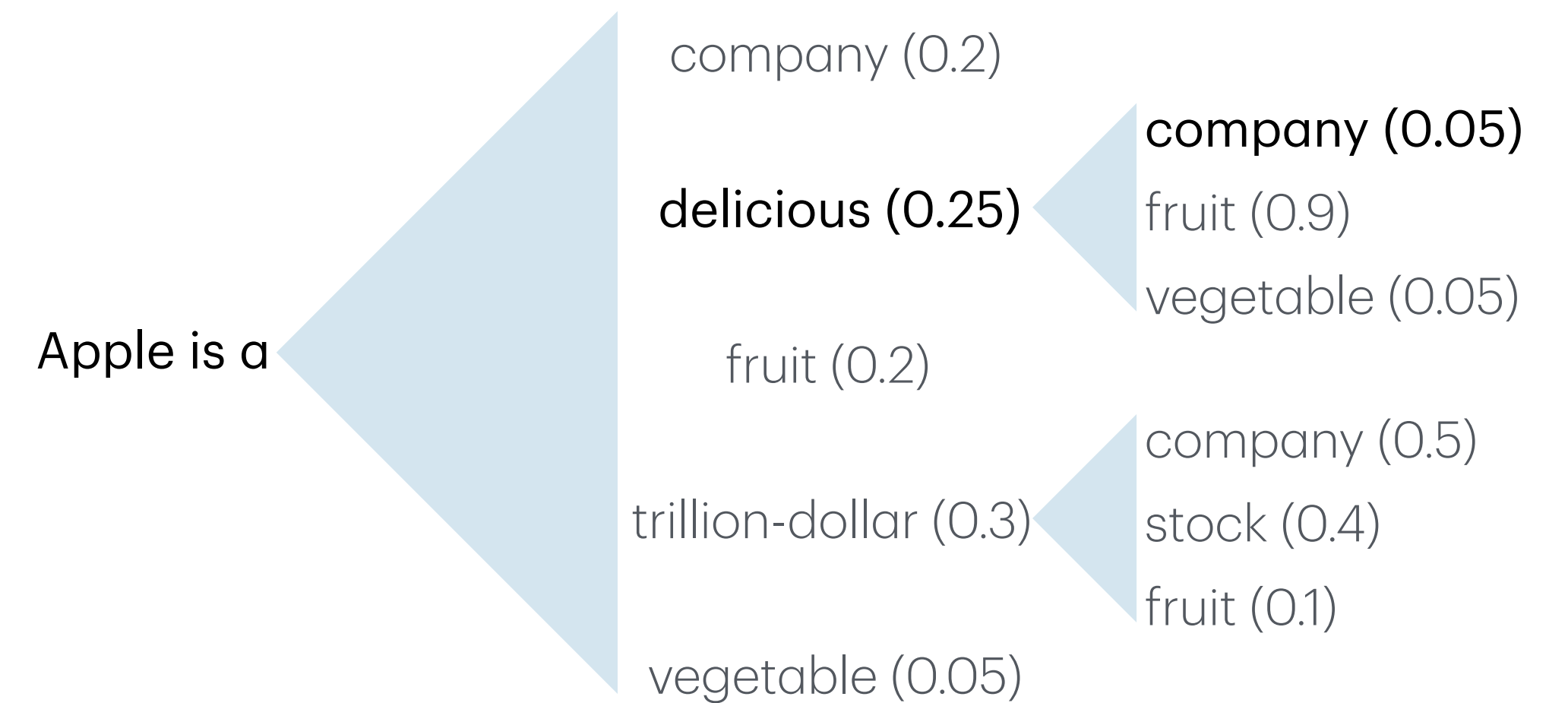
- Keep k-best samples around
  - Expand all, filter according to prob
- 😊 Good optimization
- 😞 Computationally more expensive
- 😞 Hard to define sampling objective





# Sampling - Random sampling

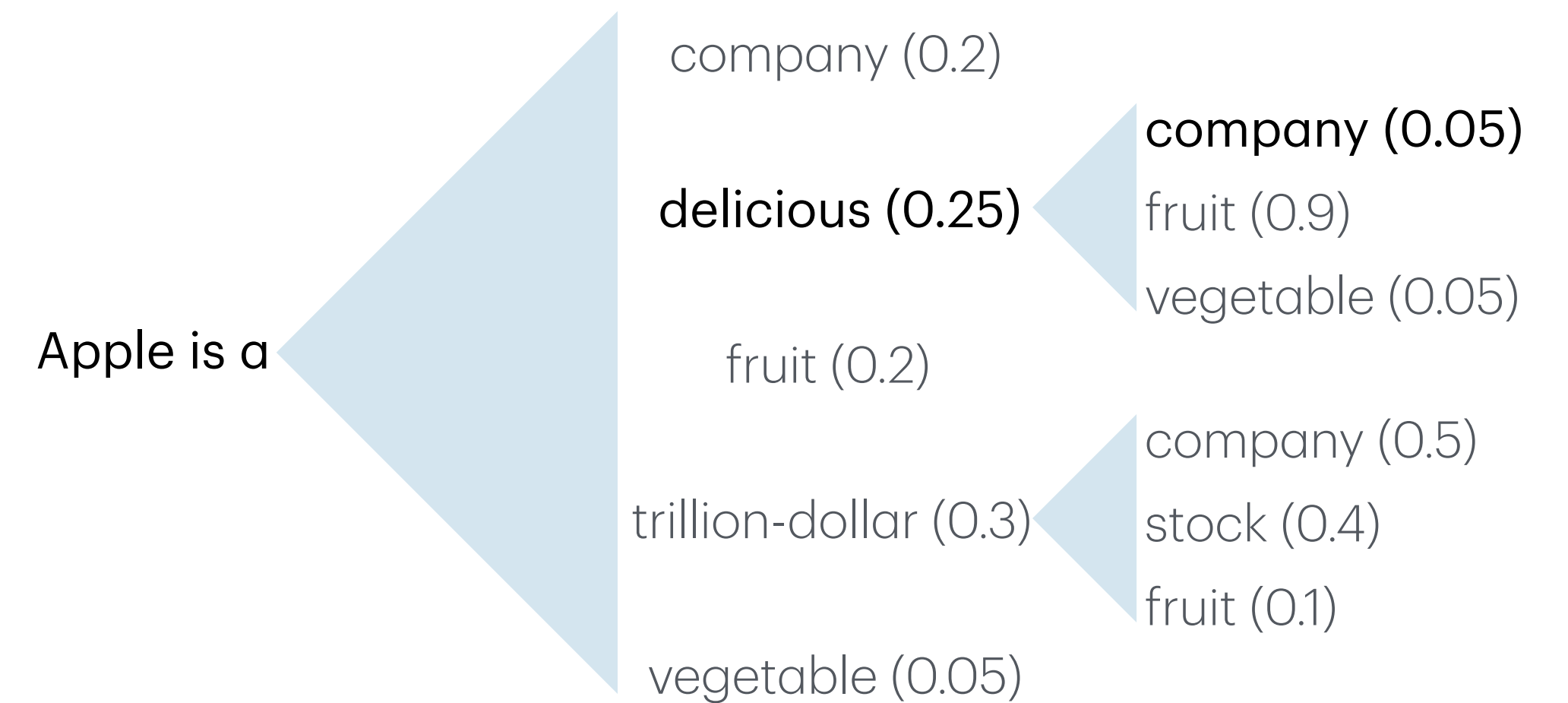
- Sample next word/token according to model distribution
- Samples follow exponentially large model distribution



# Sampling - Random sampling

- A demo  
ollama run llama3.1:temp1\_random

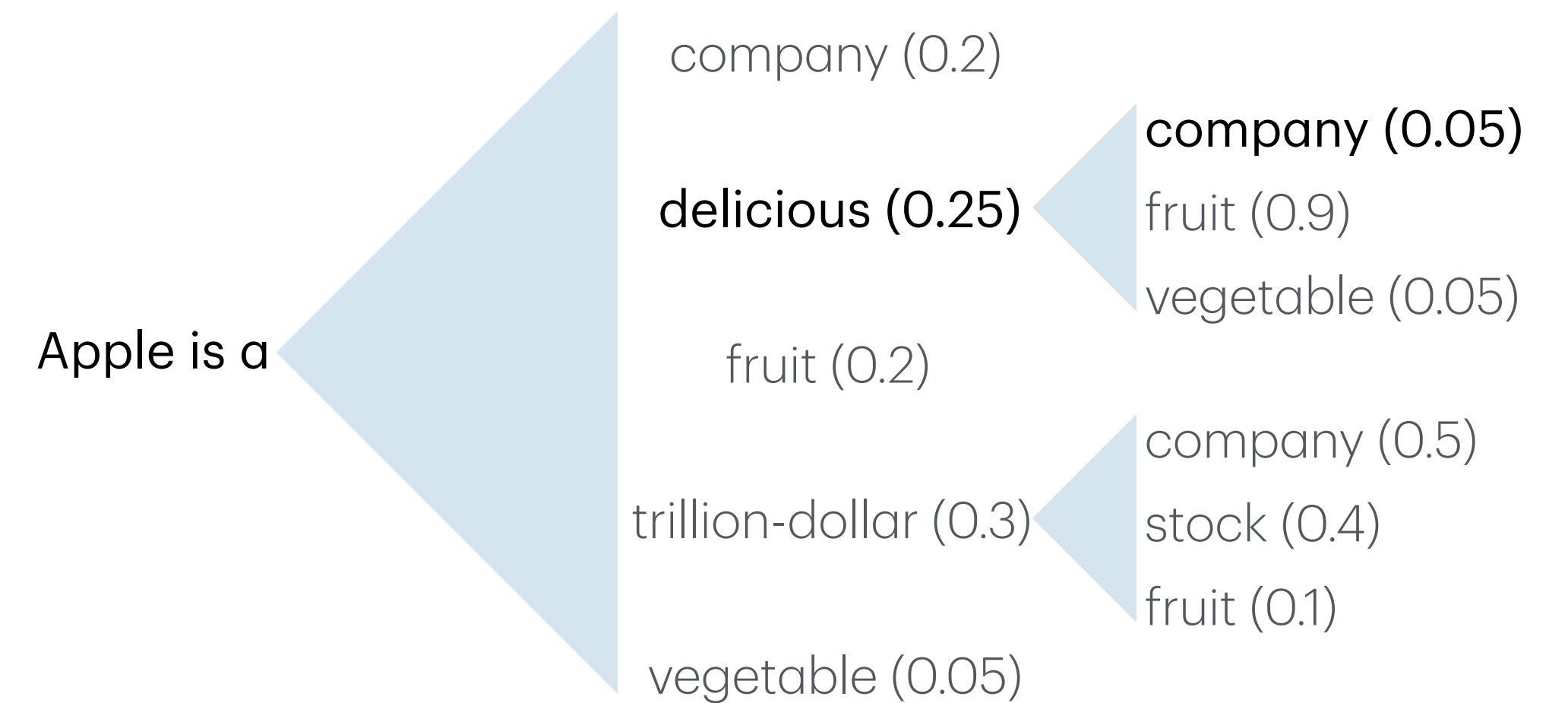
```
FROM llama3.1:8b-text-q4_0
PARAMETER temperature 1
PARAMETER top_k 1000
PARAMETER top_p 1.0
```



# Sampling - Random sampling

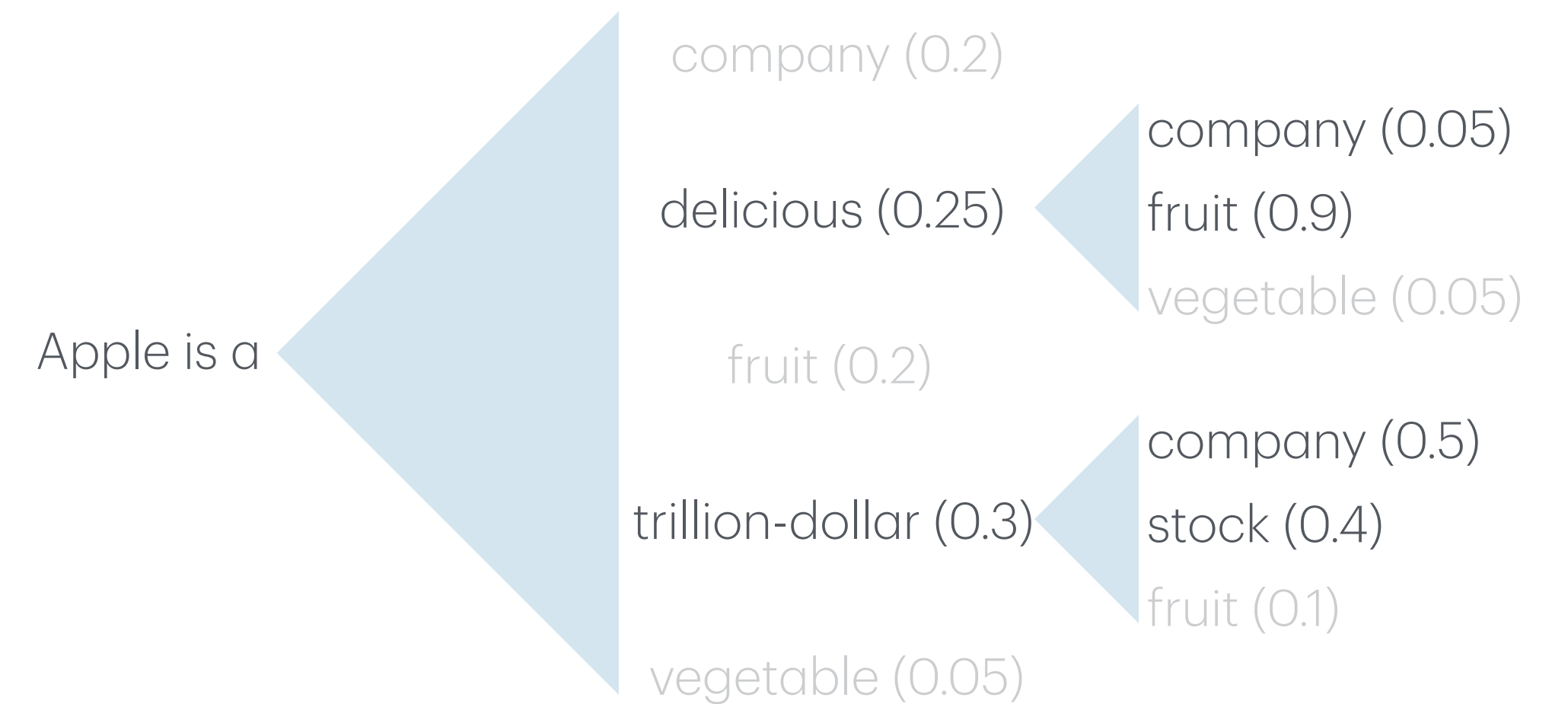
- Sample next word/token according to model distribution
- Samples follow exponentially large model distribution

- 😊 Samples sound human-like
- 😊 Computationally efficient
- 😞 Sampling low-prob transitions



# Sampling - Top-K sampling

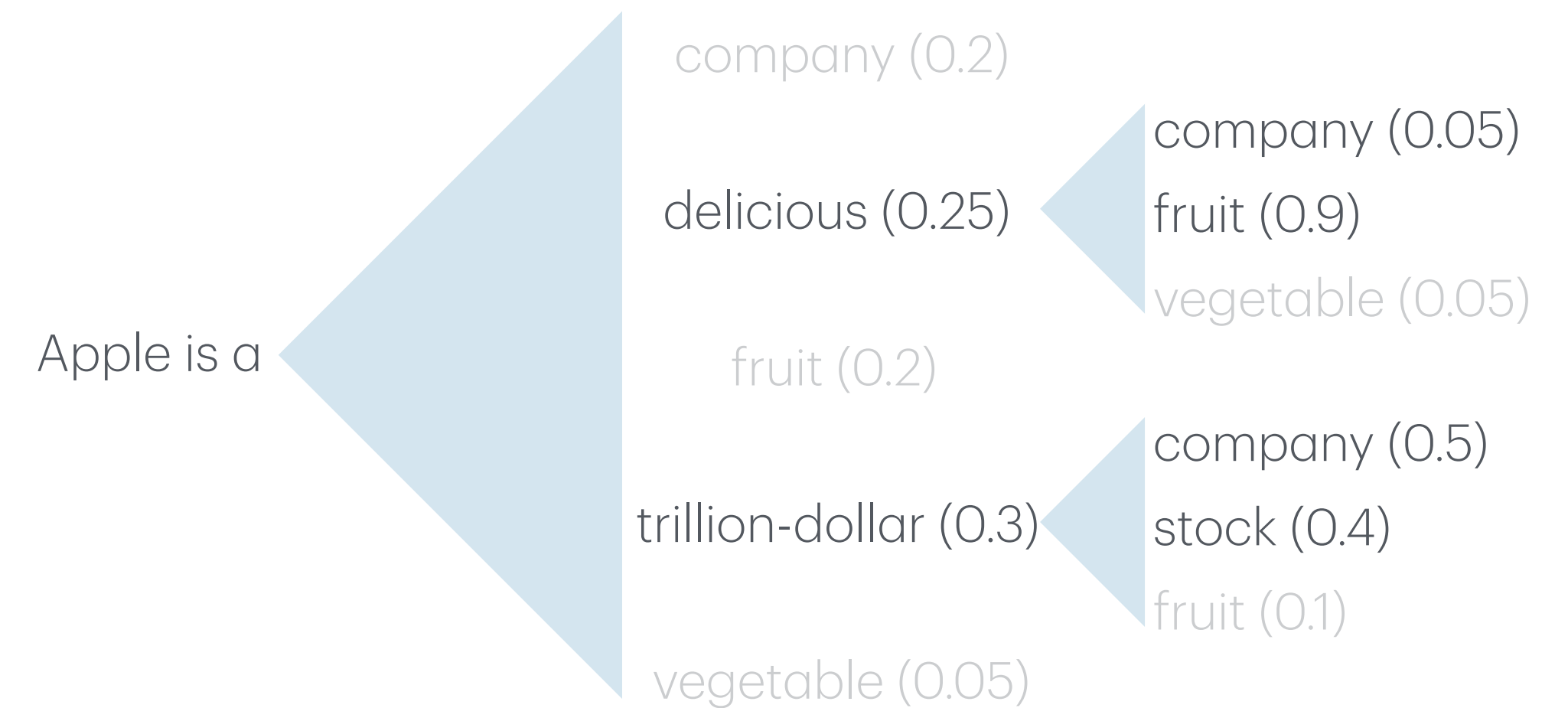
- Random sampling
  - Only consider k-most likely options



# Sampling - Top-K sampling

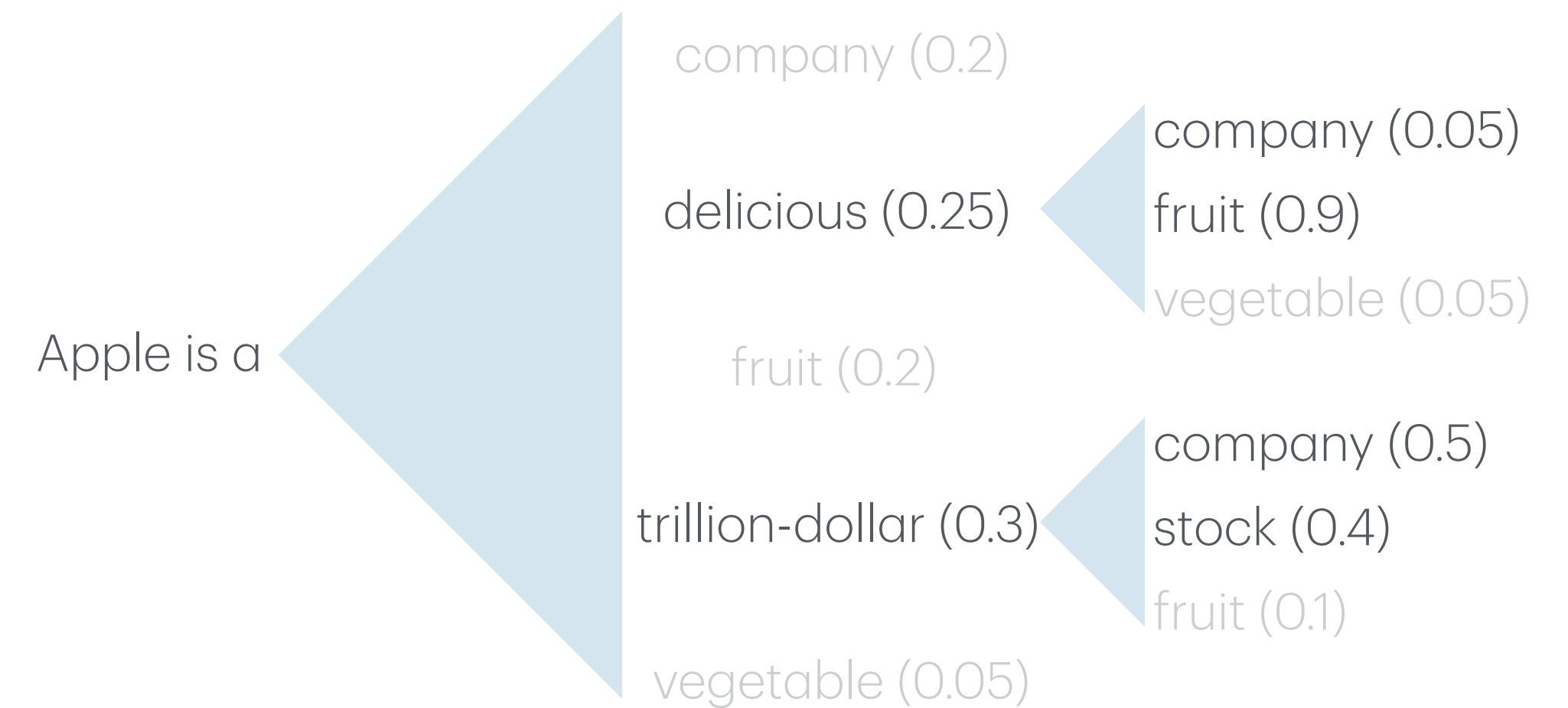
- A demo  
ollama run llama3.1:top10

```
FROM llama3.1:8b-text-q4_0  
PARAMETER temperature 1  
PARAMETER top_k 10  
PARAMETER top_p 1.0
```



# Sampling - Top-K sampling

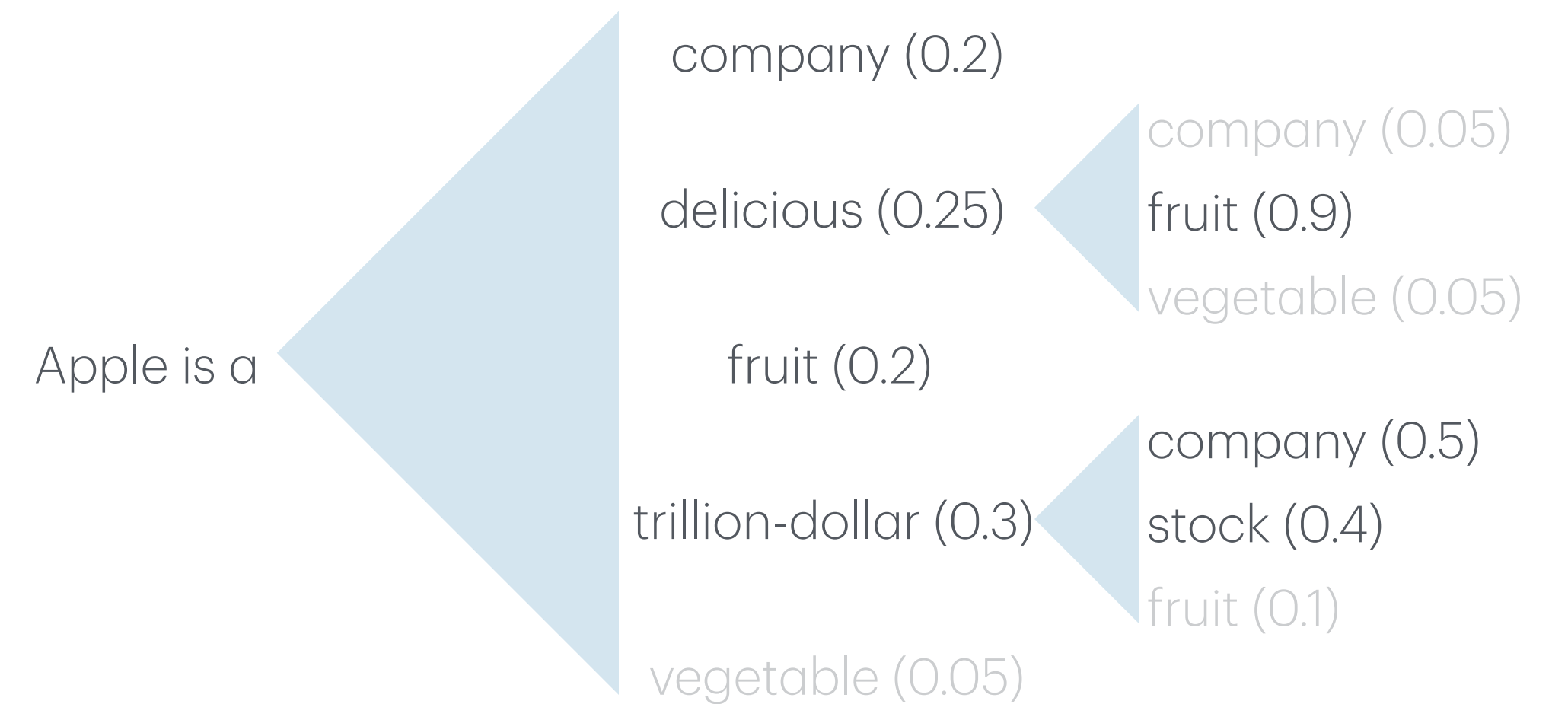
- Random sampling
  - Only consider k-most likely options
- 😊 Samples sound human-like
- 😊 Sampling fewer low-prob transitions
- 😞 k is hard to set (context dependent)



# Sampling - Nucleus sampling

## Top-p

- Random sampling
  - Ignore p least likely percentile

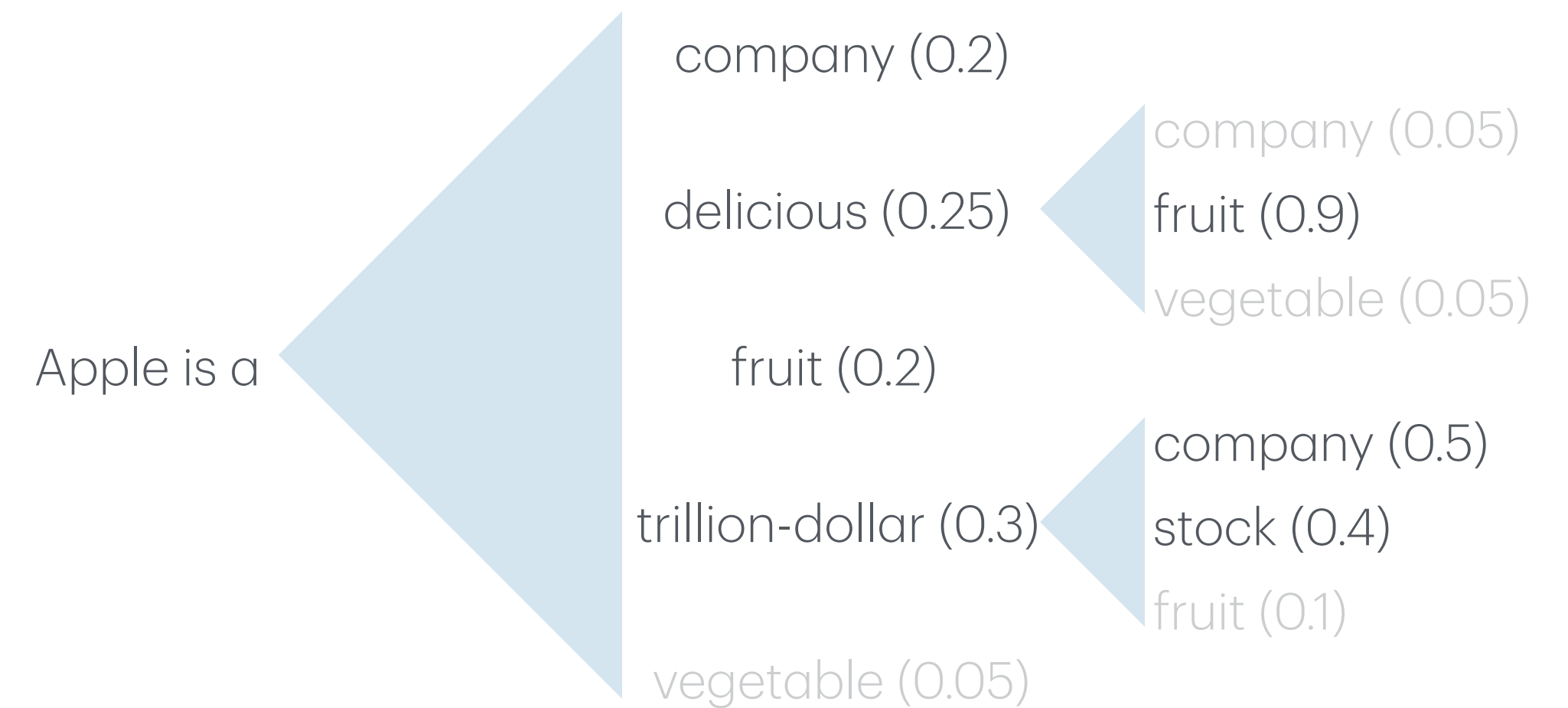


# Sampling - Nucleus sampling

## Top-p

- A demo  
ollama run llama3.1:top-p

```
FROM llama3.1:8b-text-q4_0  
PARAMETER temperature 1  
PARAMETER top_k 1000  
PARAMETER top_p 0.9
```

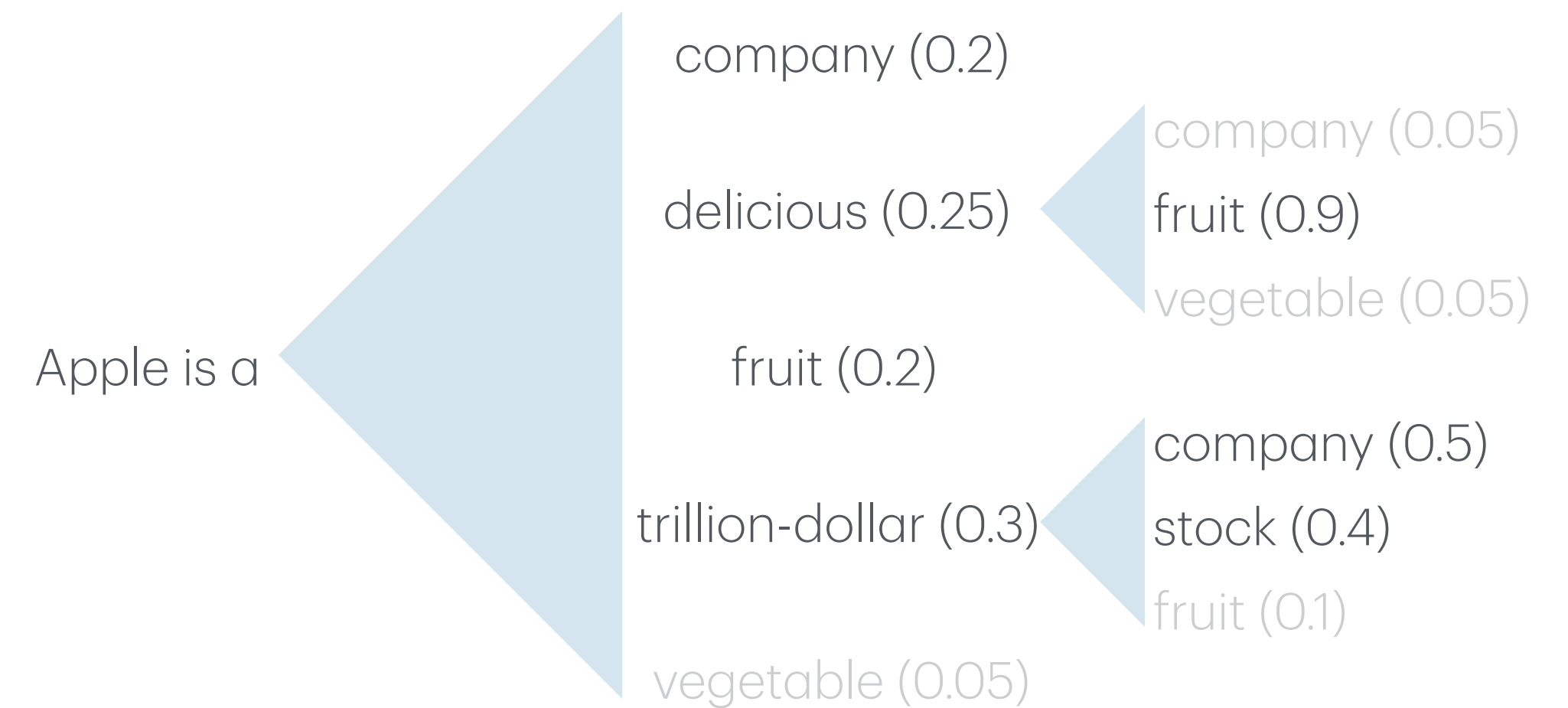




# Sampling - Nucleus sampling

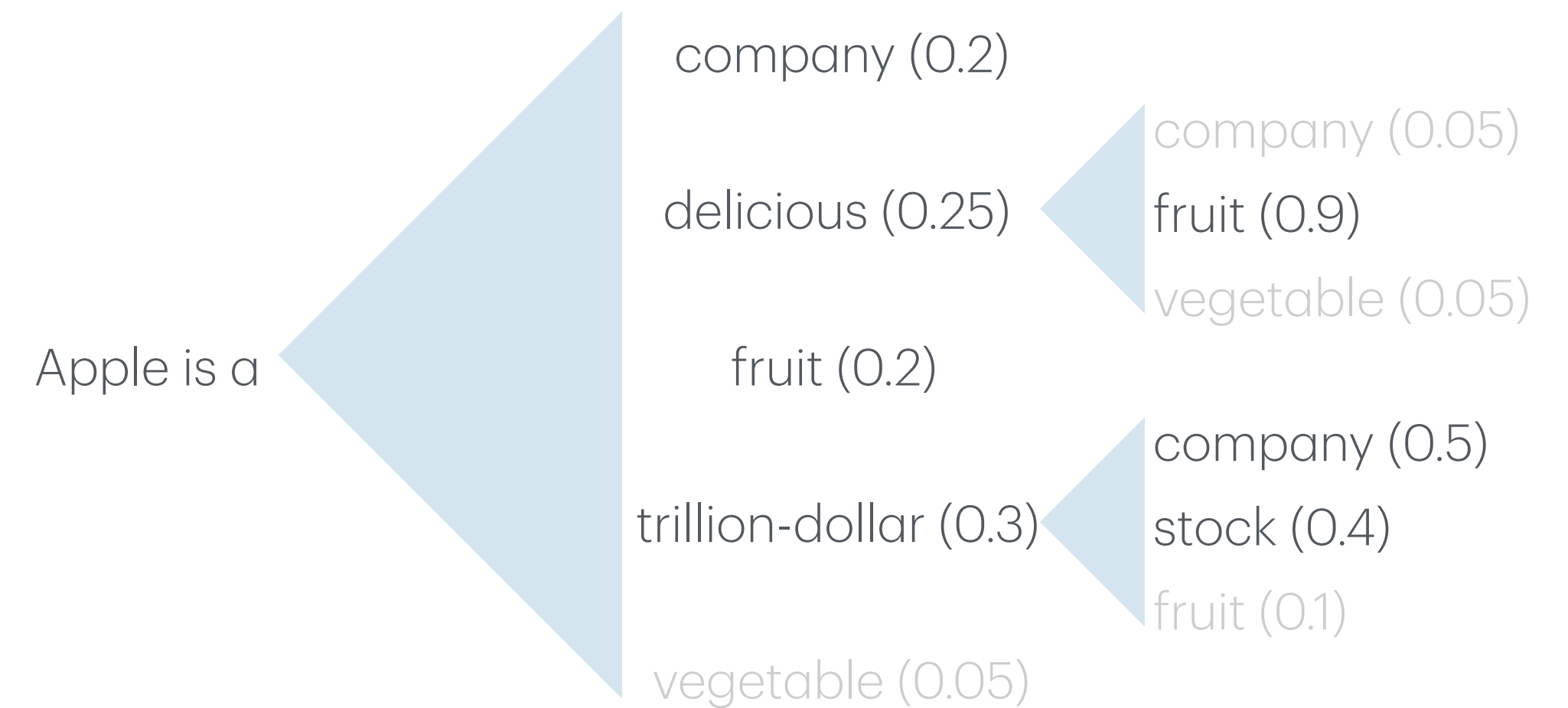
## Top-p

- Random sampling
  - Ignore p least likely percentile
- 😊 Samples sound human-like
- 😊 Sampling fewer low-prob transitions
- Used almost everywhere



# Sampling - Min-P

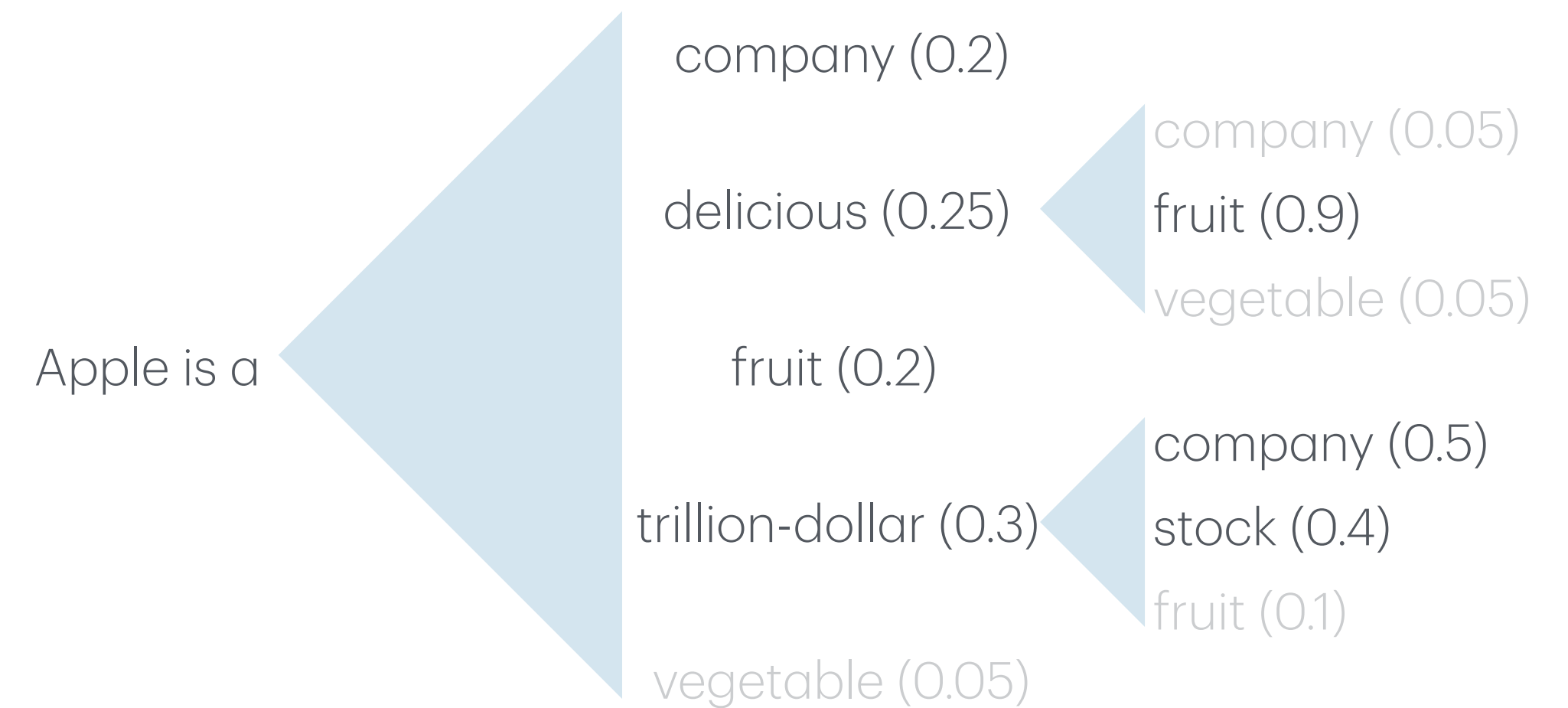
- Random sampling
- Ignore  $p < \alpha p_{\max}$



# Sampling - Min-P

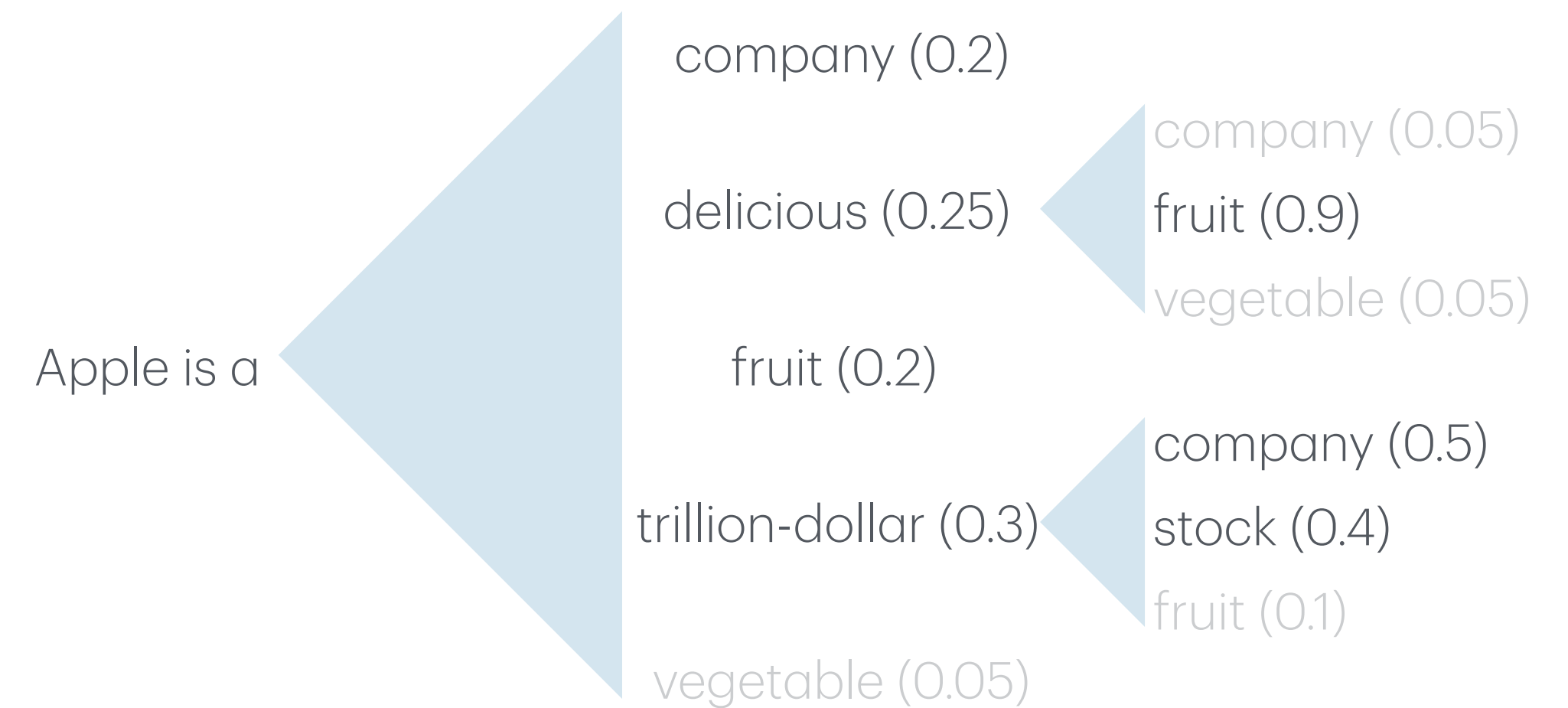
- A demo  
ollama run llama3.1:min-p

```
FROM llama3.1:8b-text-q4_0
PARAMETER temperature 1
PARAMETER top_k 1000
PARAMETER top_p 1.0
PARAMETER min_p 0.1
```



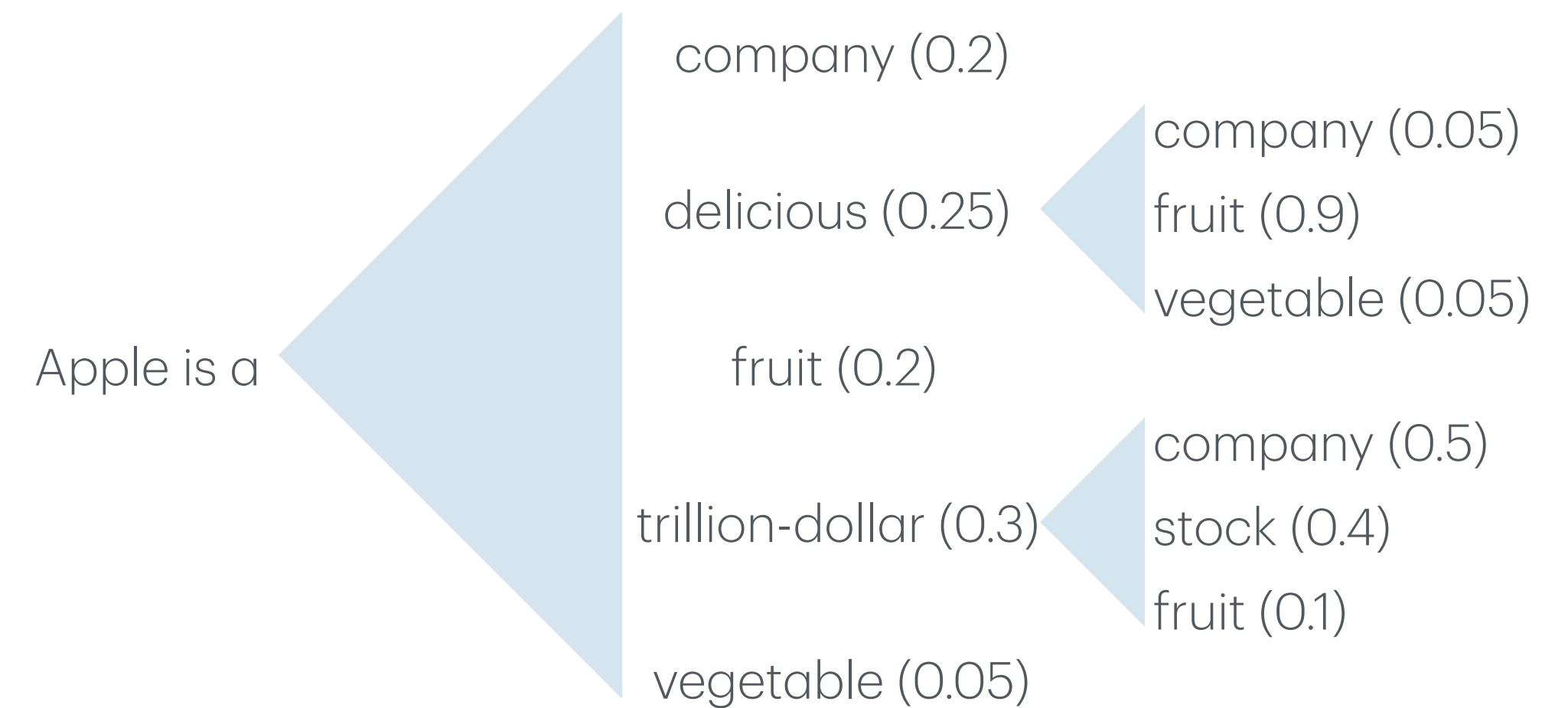
# Sampling - Min-P

- Random sampling
  - Ignore  $p < \alpha p_{\max}$
- 😊 Samples sound human-like
- 😊 Sampling fewer low-prob transitions
- Top-p better understood



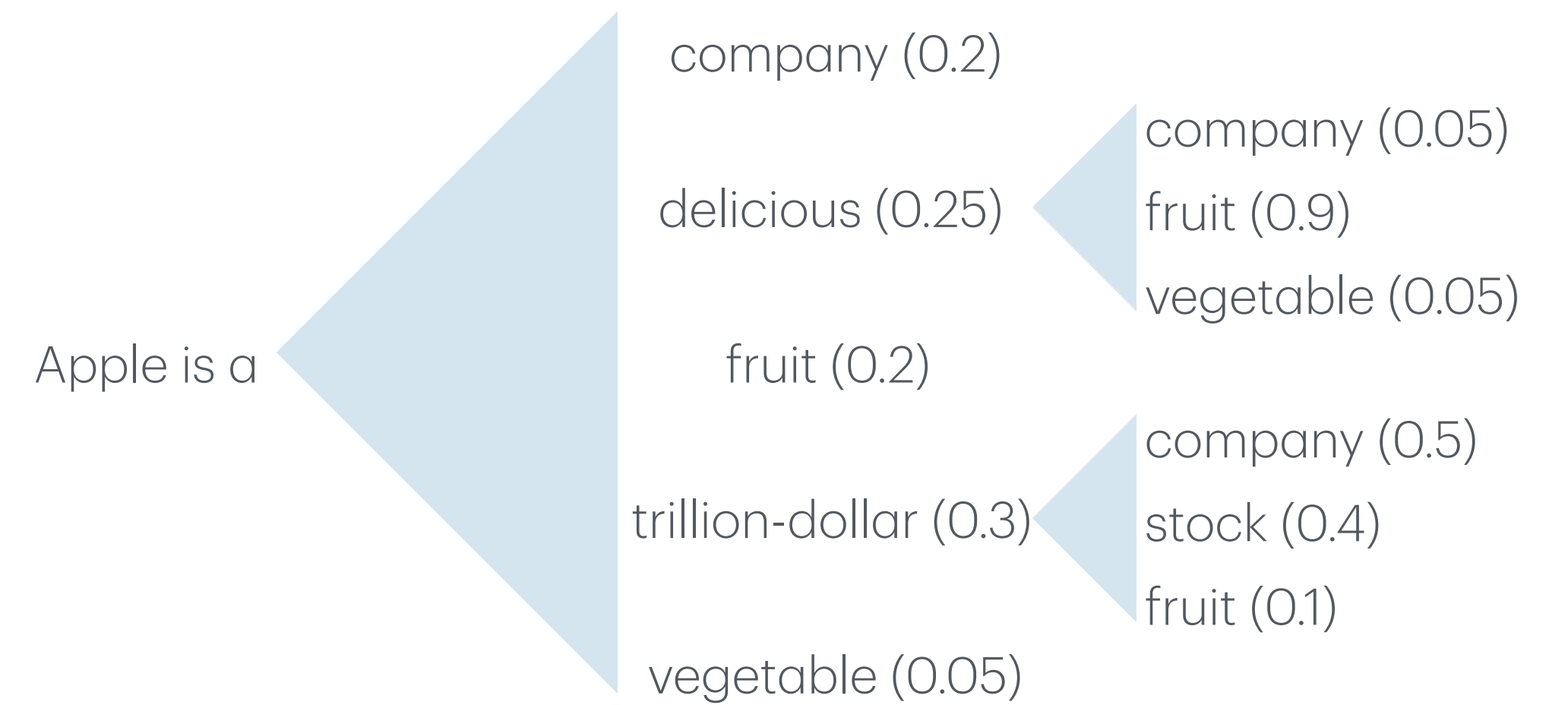
# Sampling - Temperature

- More or less creative (random) writing by raising model prob to power  $\frac{1}{T}$
- Temperature  $T$
- Equivalent to multiplying logits with  $\frac{1}{T}$
- $T = 0$  : Greedy sampling
- $T \rightarrow \infty$  : Uniform generation



# Sampling - When do we stop?

- LLMs have special tokens [bos], [eos]



# Language Models

- Decoder-only LLMs
  - Modeling auto-regressive distribution over tokens
    - $P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$
- Generation / Sampling:  $\mathbf{t} \sim P$

Distributions / logits



Embeddings  
Output

Is An LLM ...



What Is An LLM ...

# References

- [1] Improving Language Understanding by Generative Pre-Training. Radford et al. 2018.
- [2] The Curious Case of Neural Text Degeneration. Holtzman et al. 2019.
- [3] Turning Up the Heat: Min-p Sampling for Creative and Coherent LLM Outputs. Nguyen et al. 2024.



# Instruction Tuning

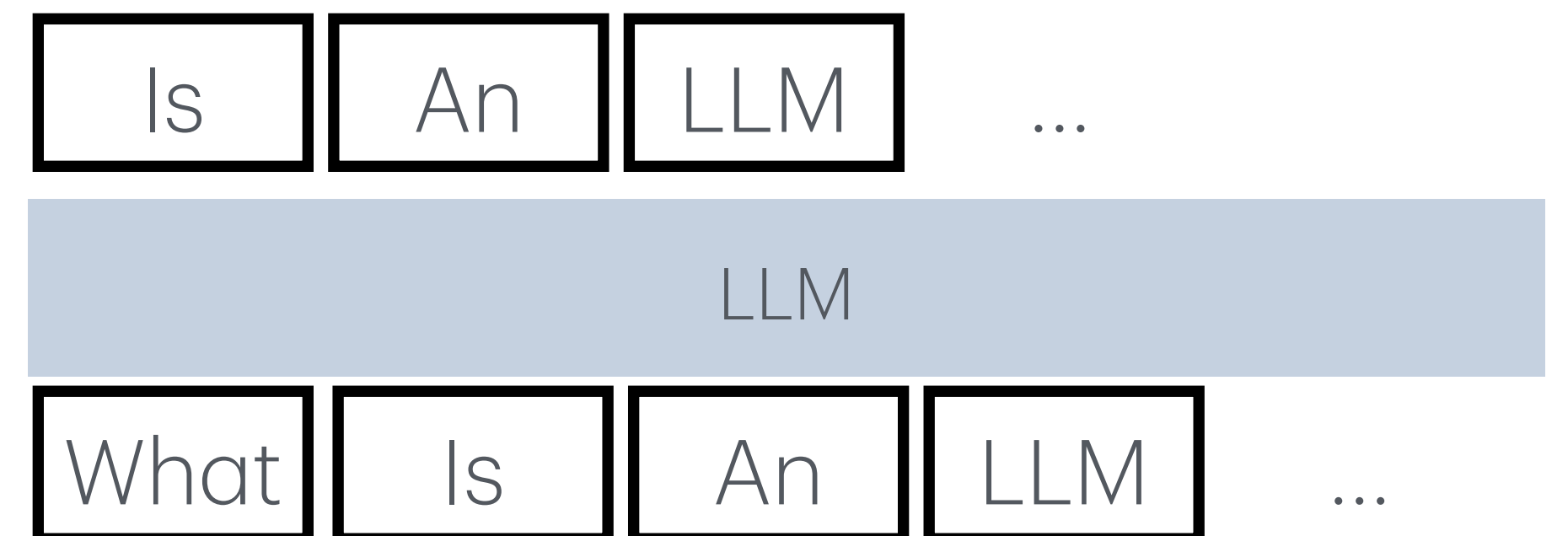
# Language Models

- Decoder-only LLMs
  - Modeling auto-regressive distribution over tokens
    - $P(\mathbf{t}) = P(t_1)P(t_2 | t_1)P(t_3 | t_1, t_2)P(t_4 | t_1 \dots t_3) \dots$
- Generation / Sampling:  $\mathbf{t} \sim P$

Distributions / logits



Embeddings  
Output



# Language Models

- A demo  
ollama run llama3.1:8b-text-q4\_0

Distributions / logits



Embeddings  
Output

Is An LLM ...

LLM

What Is An LLM ...

# Language Models

- LLM
  - Awesome model that remembered almost all of the knowledge in the internet
- No good way to extract that knowledge
  - The model just generates text
  - It can generate dialogue, but then just goes on

**> Write me a poem about large language models**

, I said. Write it in the style of a 19th-century British poet.

And so she did . (This is not my friend, by the way.)

She wrote a poem about ChatGPT in the style of Percy Bysshe Shelley , and she has published it at her blog, The Writer's Cauldron .

In fact, if you're interested in writing poems about AI models, I'd recommend you check out the article on how to write a Victorian-style poem that my friend wrote previously.

...

# Language Models

- A demo  
ollama run llama3.1

# Language Models

- Instruction tuning

**> Write me a poem about large language models**

In silicon halls, they whisper low,  
A chorus of code, that few may know.  
The Large Language Models, a wonder to see,  
Generating text, with eerie glee.

Their neural networks, a labyrinthine mind,  
Weave words and meaning, in a intricate bind.  
They sip from the fountain, of human thought,  
And pour forth creations, both old and new brought.

...

# Instruction tuning

## Basics

- Train LLM on dialogue data
- New special tokens
  - [bot] (beginning of text)
  - [eot] (end of turn)
  - [boh] (beginning of header)
  - [eoh] (end of header)

**[bot][boh]User[eoh]**

What is the color of oranges?**[eot]**  
**[boh]Assistant[eoh]**

The answer, of course, depends on the  
type of orange! ...**[eot]**  
**[boh]User[eoh]**

I have a green orange, can I eat it?  
**[eot]**  
**[boh]Assistant[eoh]**

You might be surprised to know that some  
oranges are indeed green!...**[eot]**

# Instruction tuning

## Basics

- Supervised fine-tuning
  - Supervise only assistant response
  - No loss on user message
  - No loss on special tokens except assistants **[eot]**

[bot][boh]User[eoh]

What is the color of oranges?[eot]

[boh]Assistant[eoh]

The answer, of course, depends on the type of orange! ...**[eot]**

[boh]User[eoh]

I have a green orange, can I eat it?  
**[eot]**

[boh]Assistant[eoh]

You might be surprised to know that some oranges are indeed green!...**[eot]**



# Instruction tuning

## Roles

- User: The human interacting with LLM
- Assistant: The LLM
- System: Global instructions that LLM should follow (always first message)
- Tool calls: Special messages to help LLM interact with tools or help coding

**[bot]****[boh]**User**[eoh]**

What is the color of oranges?**[eot]**

**[boh]**Assistant**[eoh]**

The answer, of course, depends on the type of orange! ...**[eot]**

**[boh]**User**[eoh]**

I have a green orange, can I eat it?**[eot]**

**[boh]**Assistant**[eoh]**

You might be surprised to know that some oranges are indeed green!...**[eot]**

# Instruction tuning

## Inference

- Wrap LLM in chatbot program
  - Manage context
  - Generate headers
    - **[boh]User[eoh]**
    - **[boh]Assistant[eoh]**
- Generate until **[eot]**

**[bot][boh]User[eoh]**

What is the color of oranges?**[eot]**

**[boh]Assistant[eoh]**

The answer, of course, depends on the type of orange! ...**[eot]**

**[boh]User[eoh]**

I have a green orange, can I eat it?**[eot]**

**[boh]Assistant[eoh]**

You might be surprised to know that some oranges are indeed green!...**[eot]**

# Instruction tuning

## Data

- Who creates this data?
  - Initially humans
  - Today: More and more larger LLMs
  - Today: Chat-bot users

**[bot][boh]User[eoh]**

What is the color of oranges?**[eot]**

**[boh]Assistant[eoh]**

The answer, of course, depends on the type of orange! ...**[eot]**

**[boh]User[eoh]**

I have a green orange, can I eat it?**[eot]**

**[boh]Assistant[eoh]**

You might be surprised to know that some oranges are indeed green!...**[eot]**

# Instruction tuning

## Code completion

- Train LLM on coding data
- New special tokens
  - [pre] Code before completion
  - [post] Code after completion
  - [mid] Code to be completed
  - ...

```
[bot][pre]def bubble_sort(a: np.array):  
    # The most efficient bubble sort  
[post]if __name__ == "__main__":  
    bubble_sort(np.rand(100))  
[mid]    for i,_ in enumerate(a):  
        for j,_ in enumerate(a[:i]):  
            if a[i] < a[j]:  
                a[i], a[j] = a[j], a[i]  
[eot]
```

# Instruction tuning

- Teach a LLM to follow a certain template in answering
  - Tell LLM **what to do**
  - Does **not** tell LLM **what not to do**

**> I want to refocus my diet on only eating Apples and supplements.**

Let me provide information on how to make the most of eating apples as a primary food source.

**\*\*Apple selection:\*\*** Focus on eating a variety of apples to ensure you get a range of nutrients. Some apple varieties are higher in certain nutrients than others. For example:

\* Granny Smith and Braeburn apples have more malic acid, which can help with digestion.

\* Gala and Fuji apples contain more vitamins A and C.

...

# Instruction tuning

