Large Language Models (LLMs)

The ingredients that make language models reach the caliber of ChatGPT

Dr. Letitia Parcalabescu
10th of December 2024
Part of “Machine Learning and Physics” Lecture at Heidelberg University

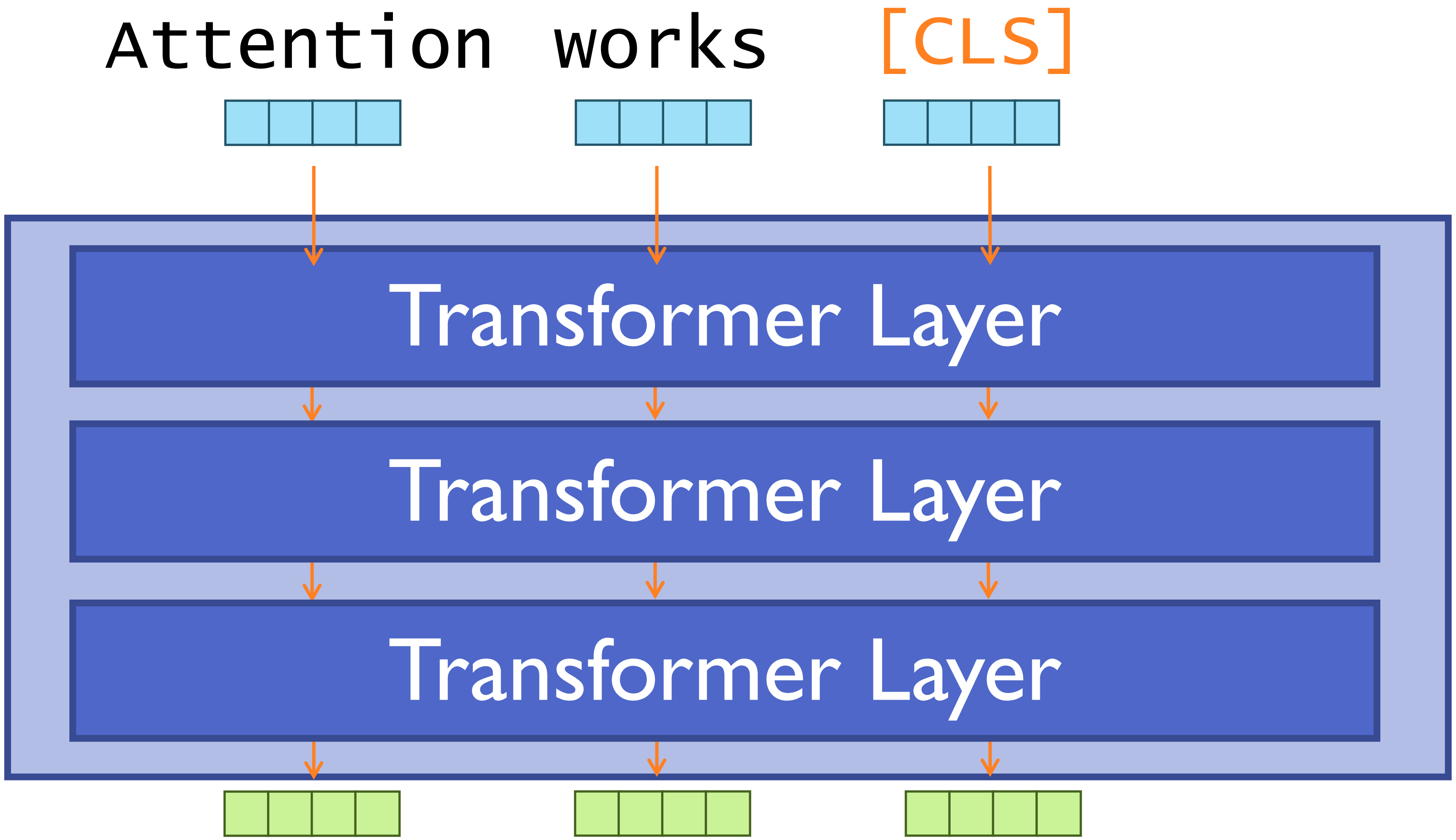
Contents

1

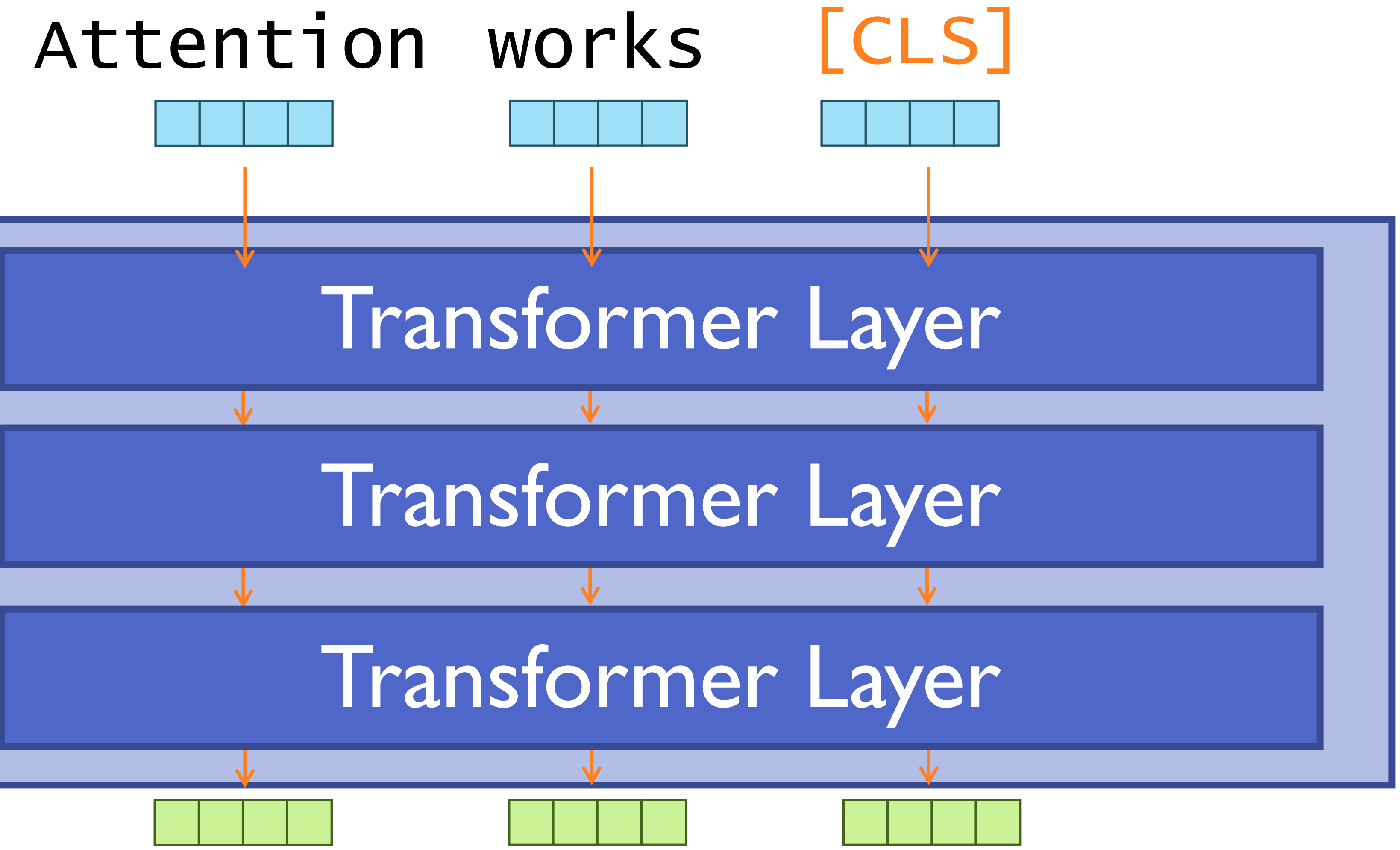
The Transformer

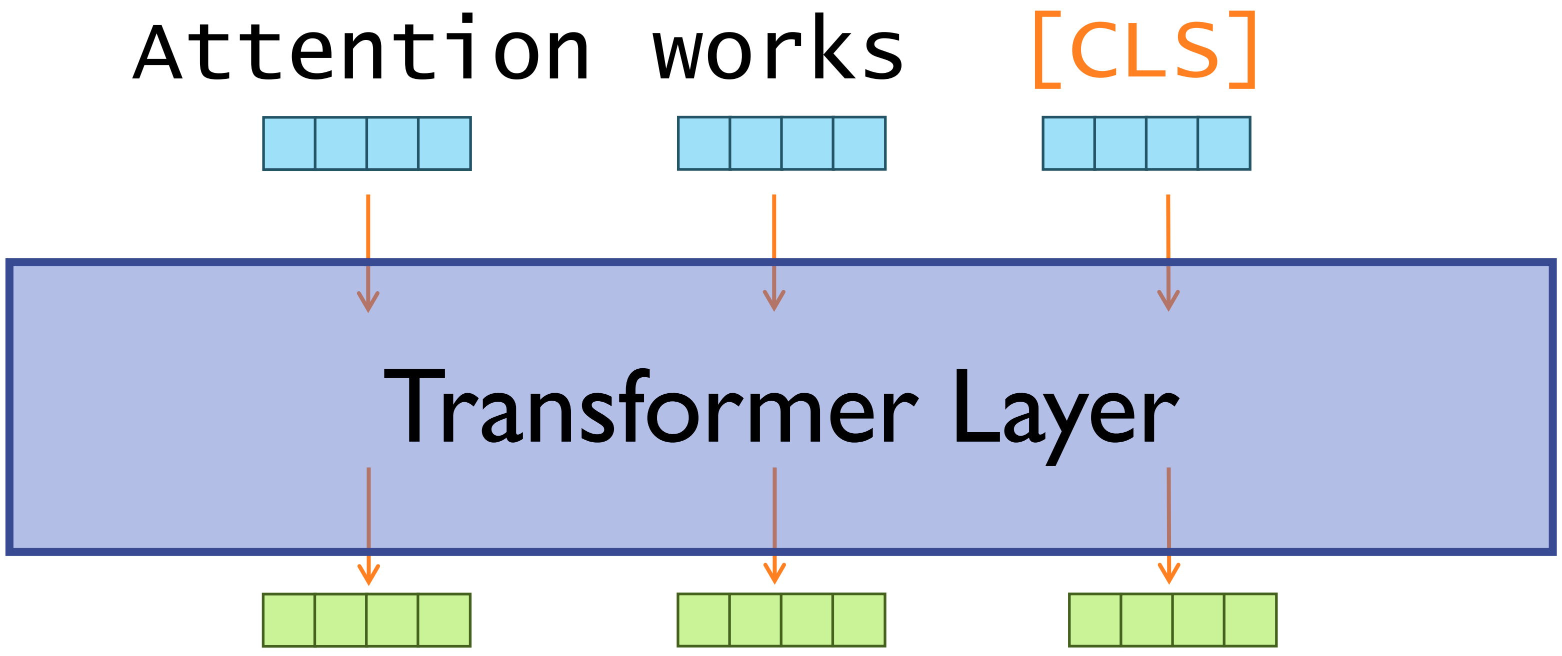
- Tokenization
- Positional Embeddings
- The Transformer Layer

The Transformer



We want representations that ...





Maps L d-dimensional vectors v_l into other L d-dimensional vectors u_l.

How to represent text as vectors?

- Idea:
  - Look at all words we have in the training corpus and make a vocabulary.
  - Index the vocabulary: Assign to each word in the vocabulary an id.
  - Assign to each id unique embedding vector which is learned during training (just like weights are trained).

word

mother
attention

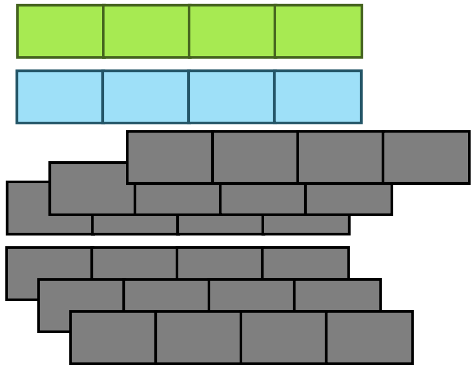
- Problem:
  - All new words at test time map to the id corresponding to UNKOWN.

mom	→	25800 (UNK) →
attenttionn	→	25800 (UNK) →

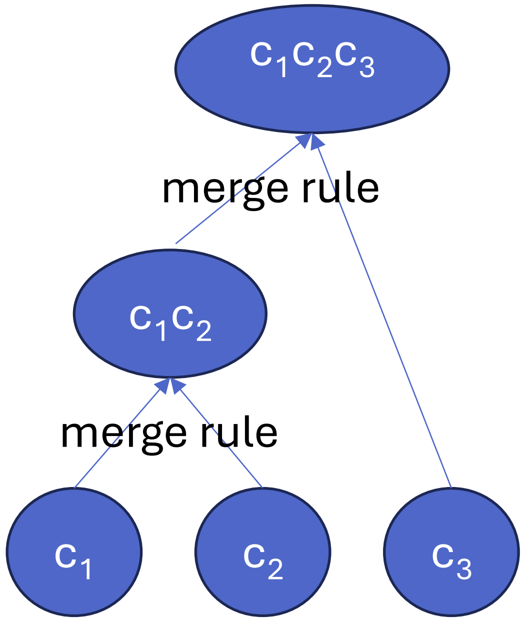
Solution: Tokenization

- Compose the vocabulary out of subwords (tokens).
- Common words end up part of the subword vocabulary. Split rarer words into subcomponents (sometimes unintuitive).
- Worst case: Each character in the word becomes a subword.

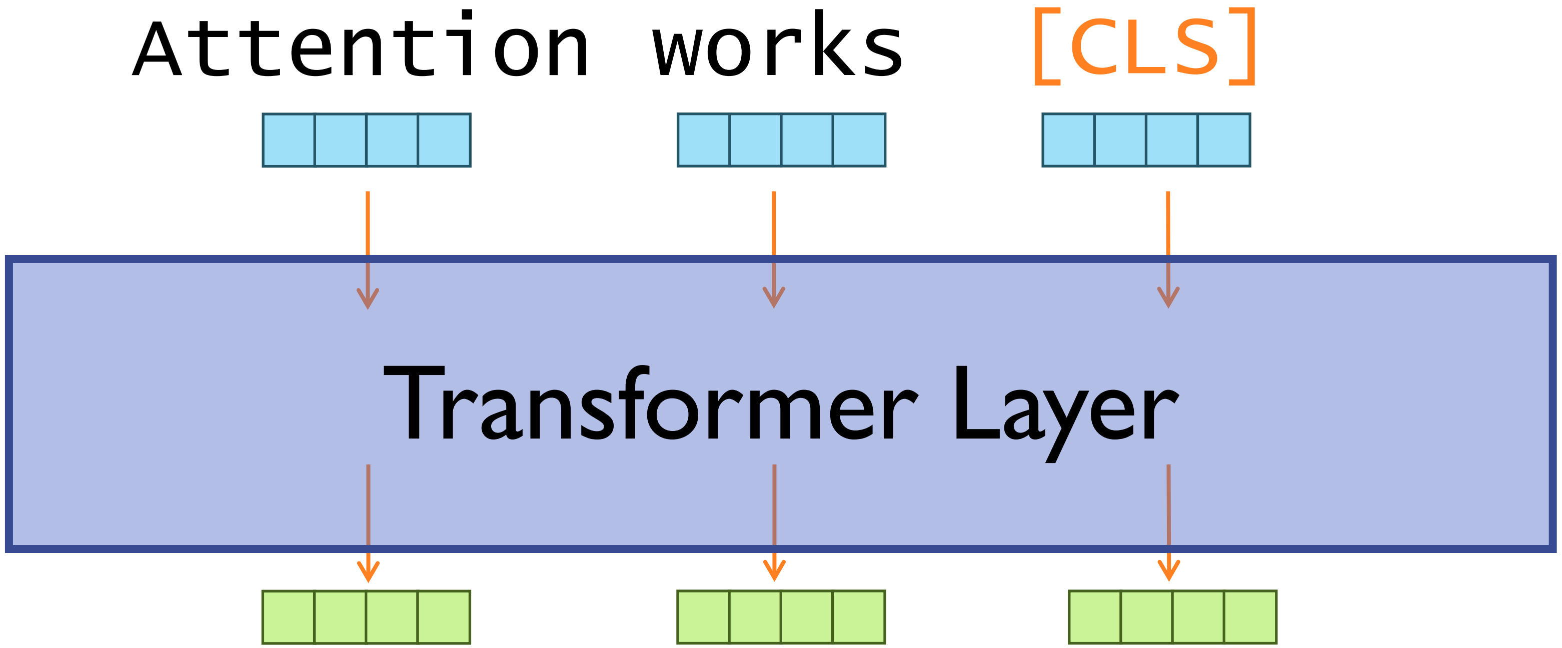
word		**subword (token)**		**token id**		**token embedding**
mother	→	mother	→	84	→	
attention	→	attention	→	1347	→	
mom	→	m## o## m##	→	26; 57; 26	→	
attenttionn	→	att## ent## tion## n	→	565; 287; 698	→	

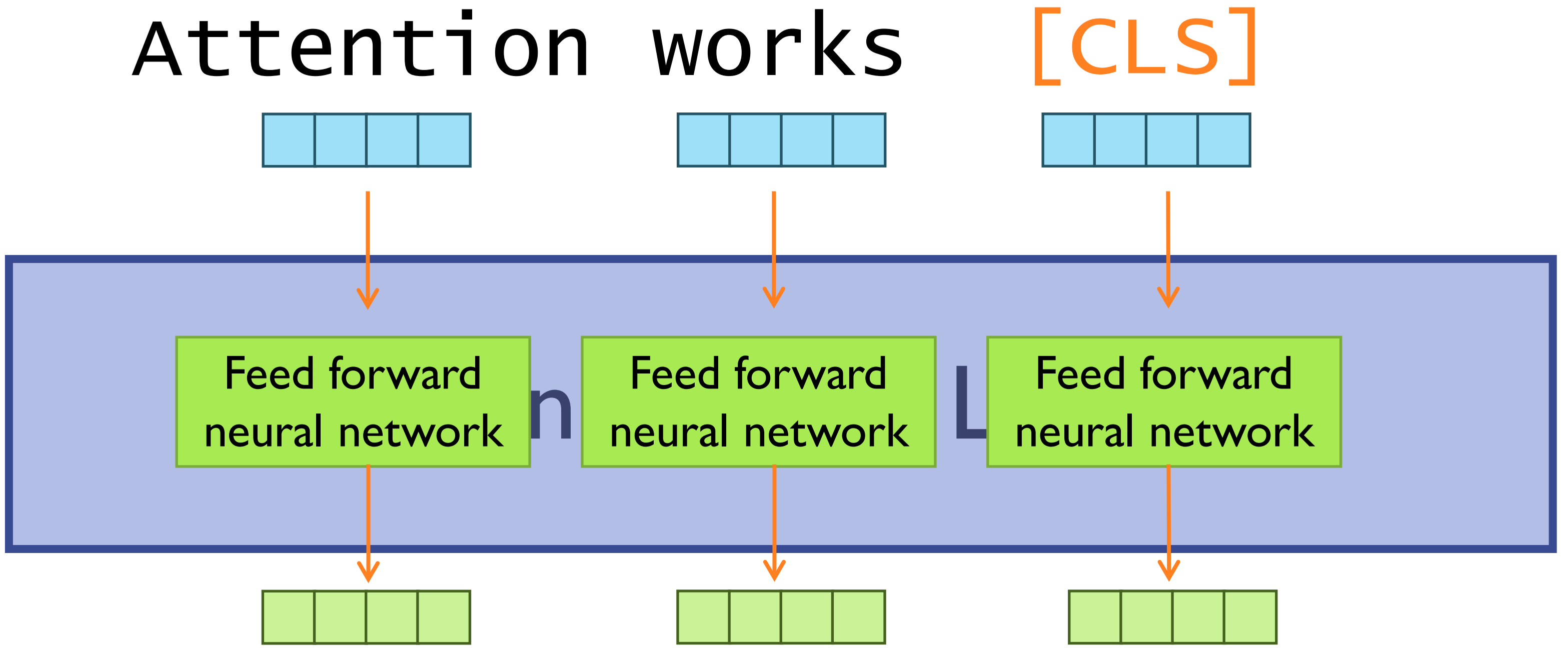


Example Tokenizers

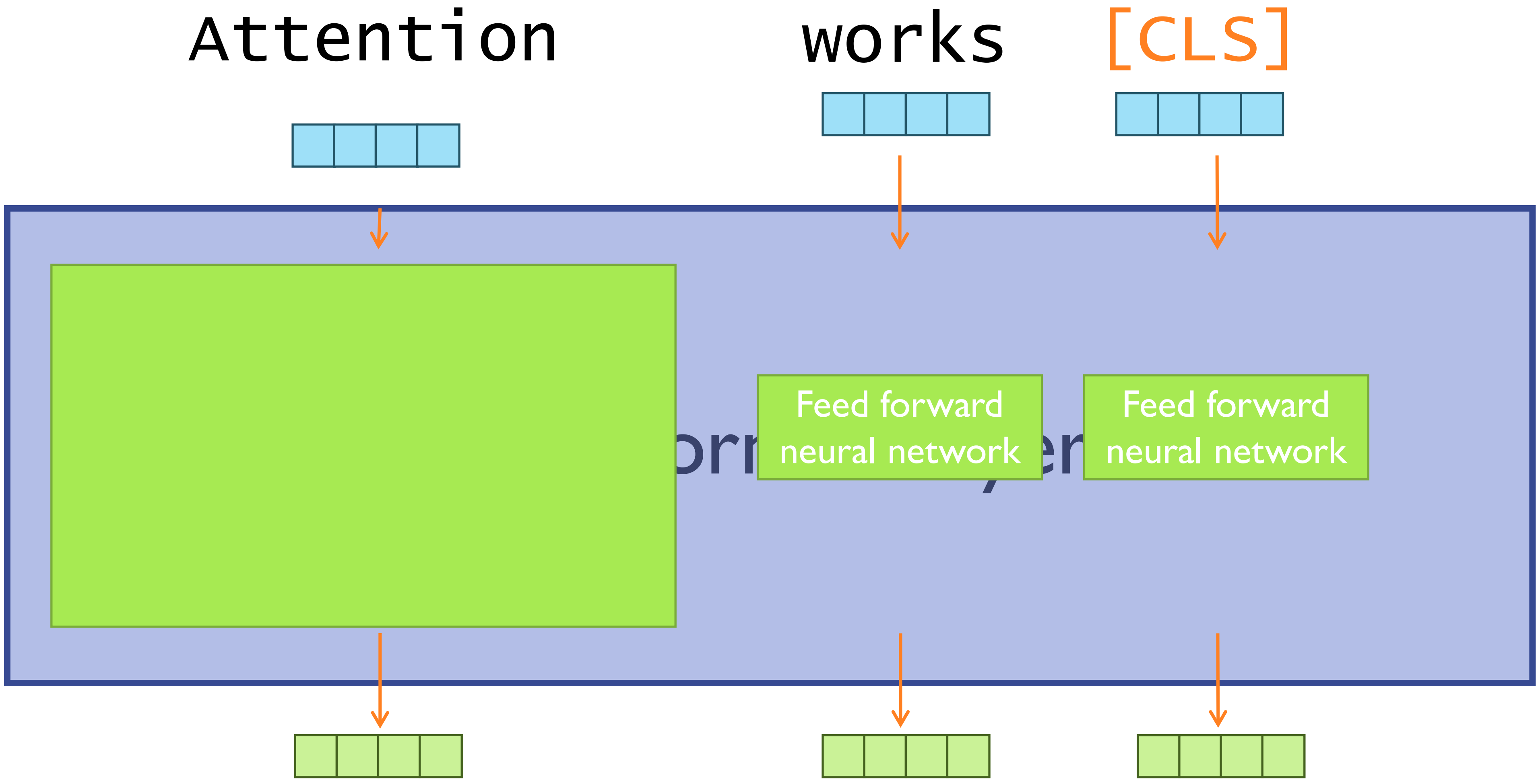


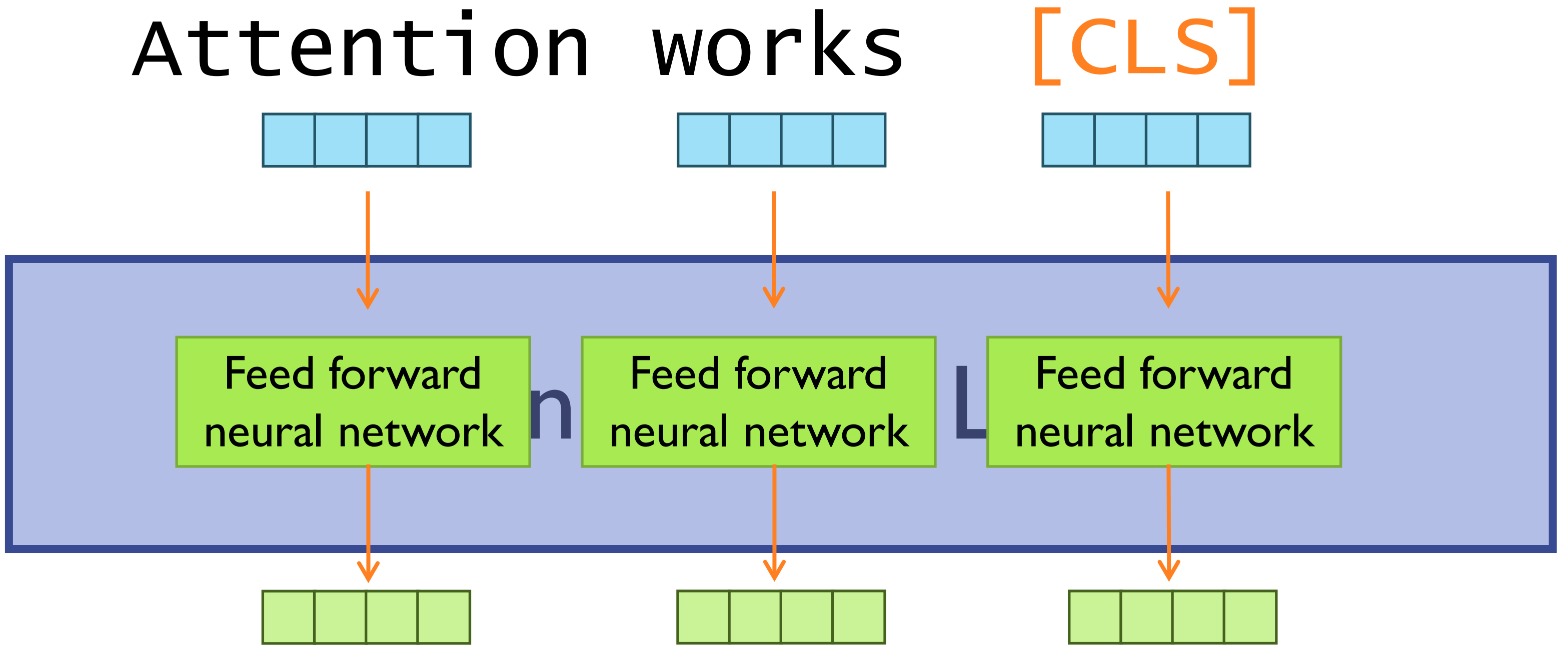
- Byte Pair Encoding (BPE) (Sennrich, 2016):
  - Training:
    - Define a target vocabulary size.
    - Start with a set of all characters as individual tokens.
    - Progressively, merge the most frequent token pairs occurring together in the training data. Store the resulting new token and the merging rule, and the training is completed when the desired number of tokens is reached.
  - To encode text with the trained tokenizer, BPE splits the input into individual characters and applies the lowest-ranking merge rule until no more are applicable.
- Many other tokenizers:
  - Unigram, Sentencepiece, character-based, n-gram based, etc.

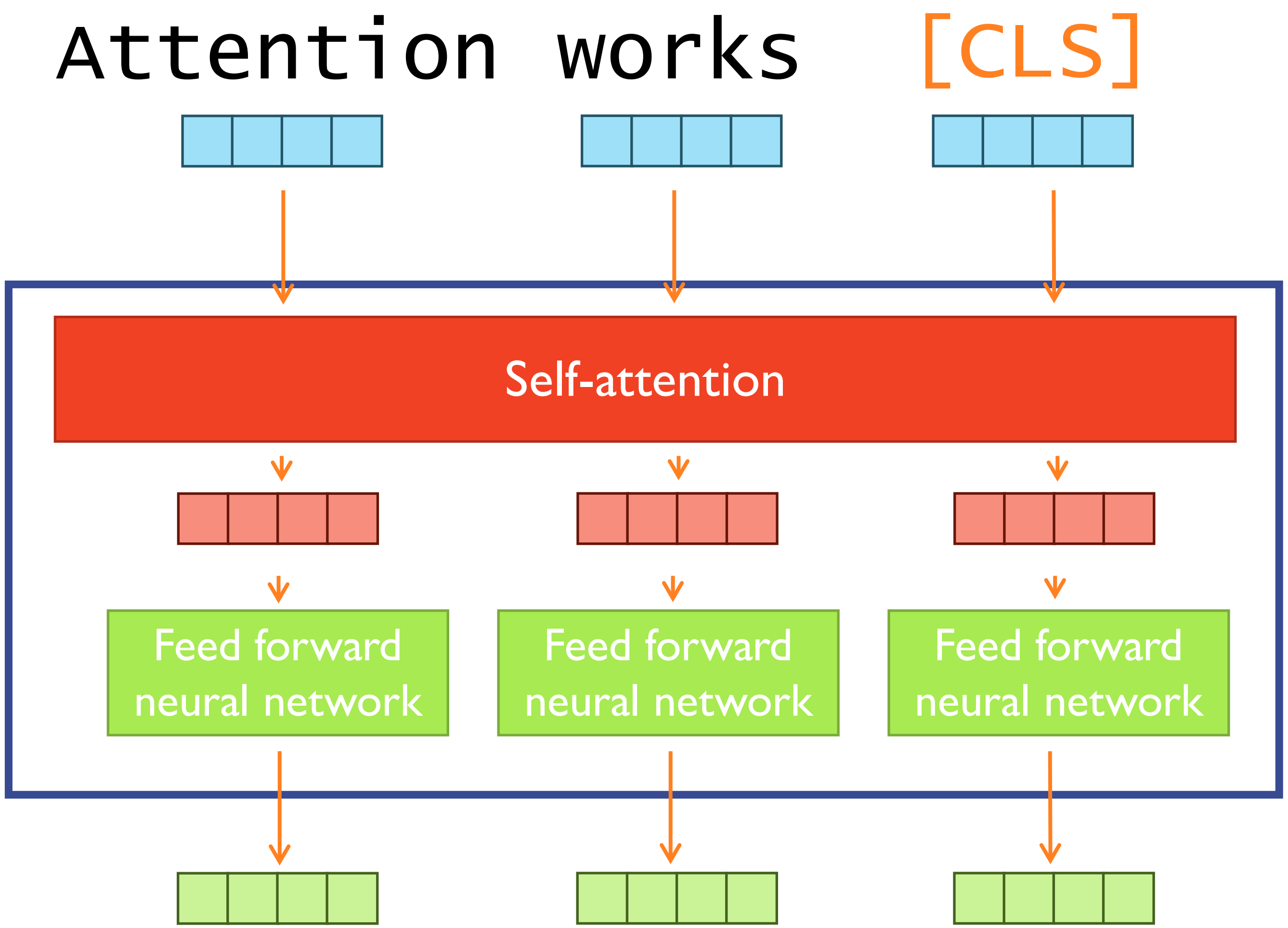

Attention works [CLS]
Transformer Layer

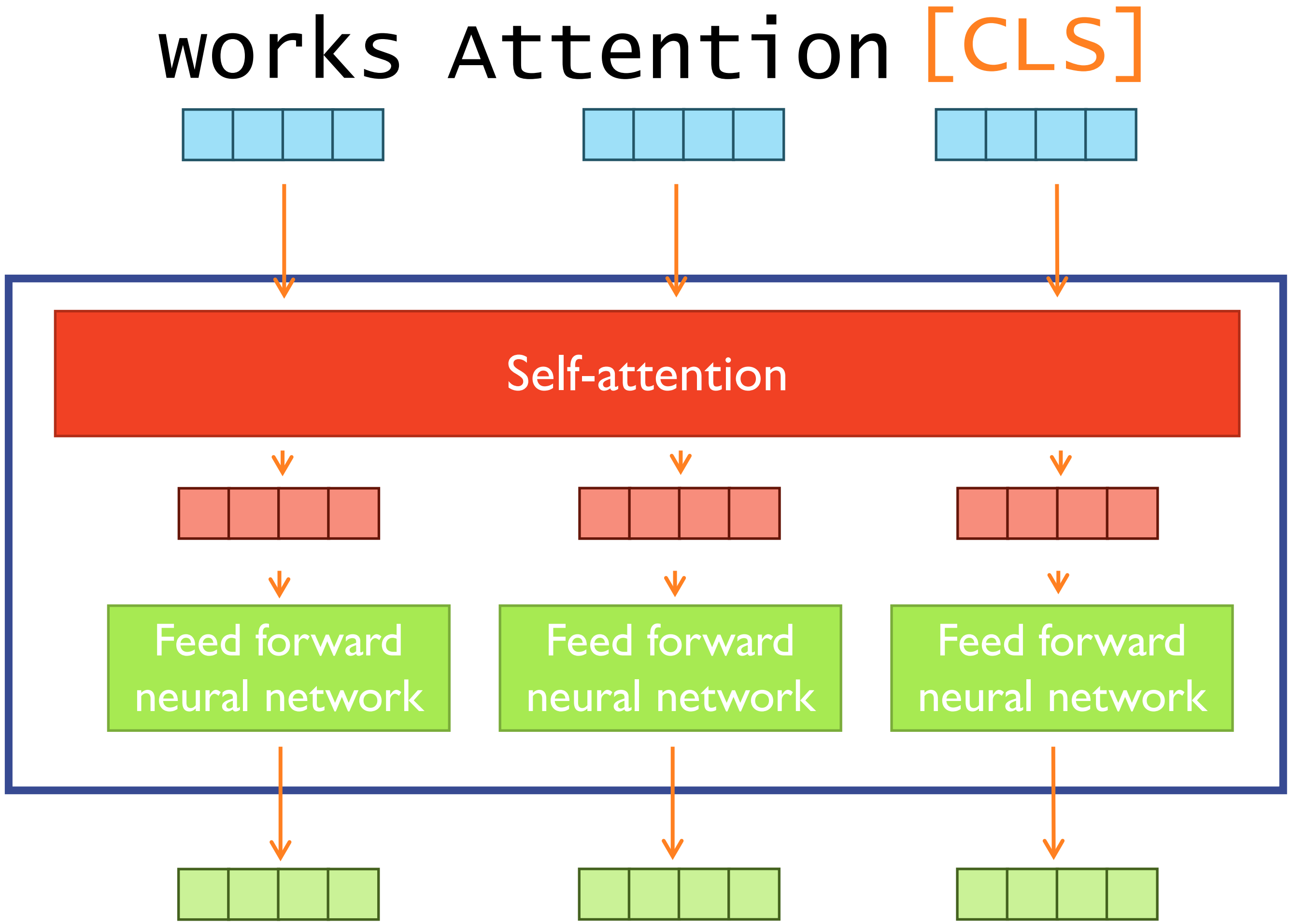


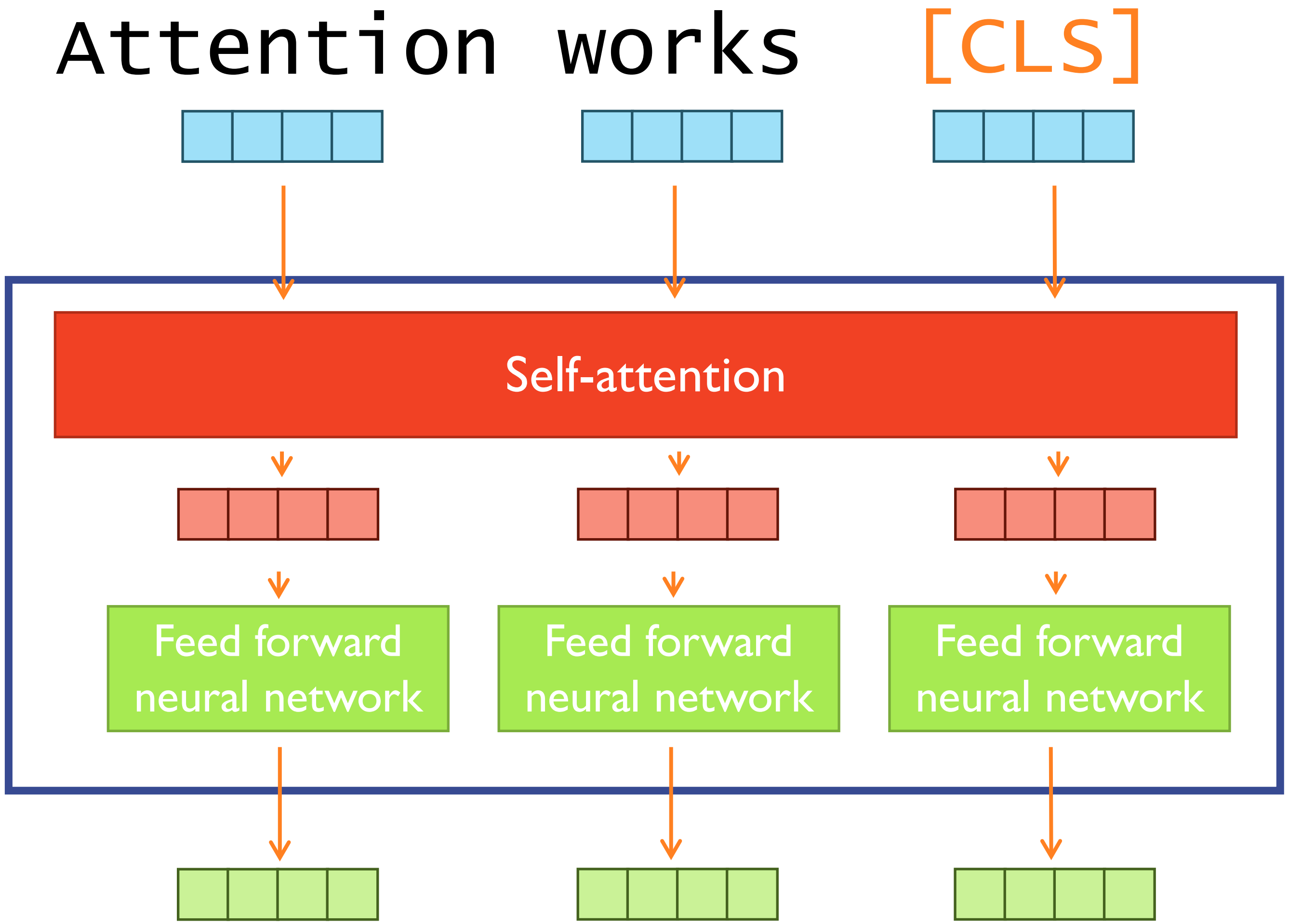
Attention works [CLS]
Feed forward neural network
Feed forward neural network
Feed forward neural network

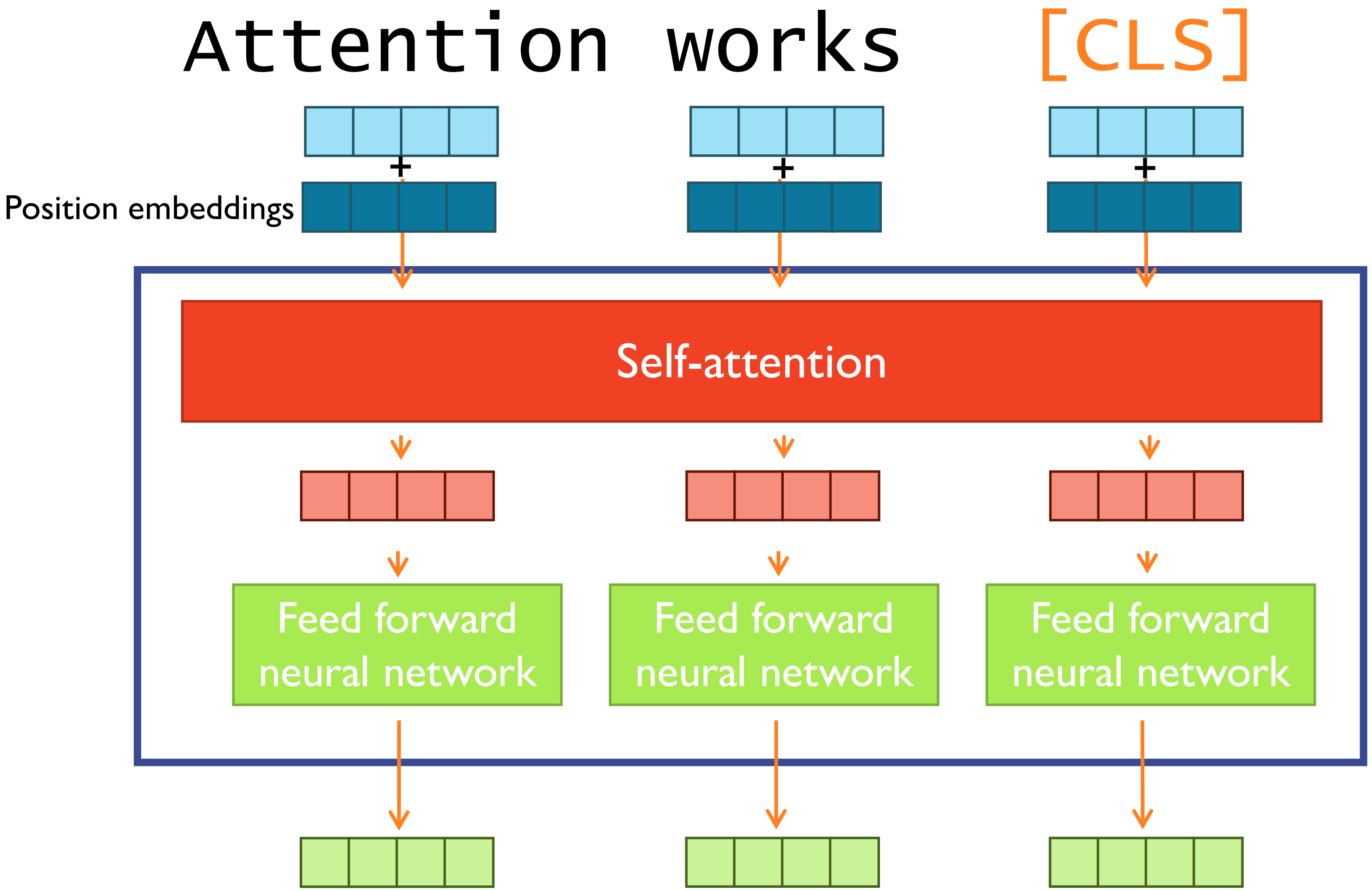

Attention
works
[CLS]
Feed forward neural network
Feed forward neural network


Attention works [CLS]
Feed forward neural network
Feed forward neural network
Feed forward neural network


Attention works [CLS]
Self-attention
Feed forward neural network
Feed forward neural network
Feed forward neural network


works Attention [CLS]
Self-attention
Feed forward neural network
Feed forward neural network
Feed forward neural network


Attention works [CLS]
Self-attention
Feed forward neural network
Feed forward neural network
Feed forward neural network

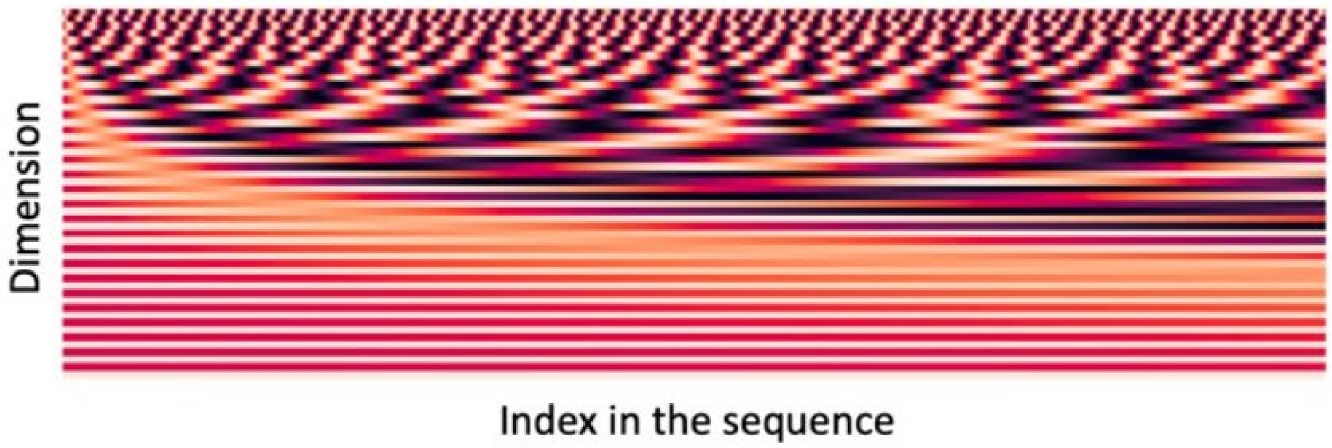


Attention works [CLS]
+
+
+
Position embeddings
Self-attention
Feed forward neural network
Feed forward neural network
Feed forward neural network

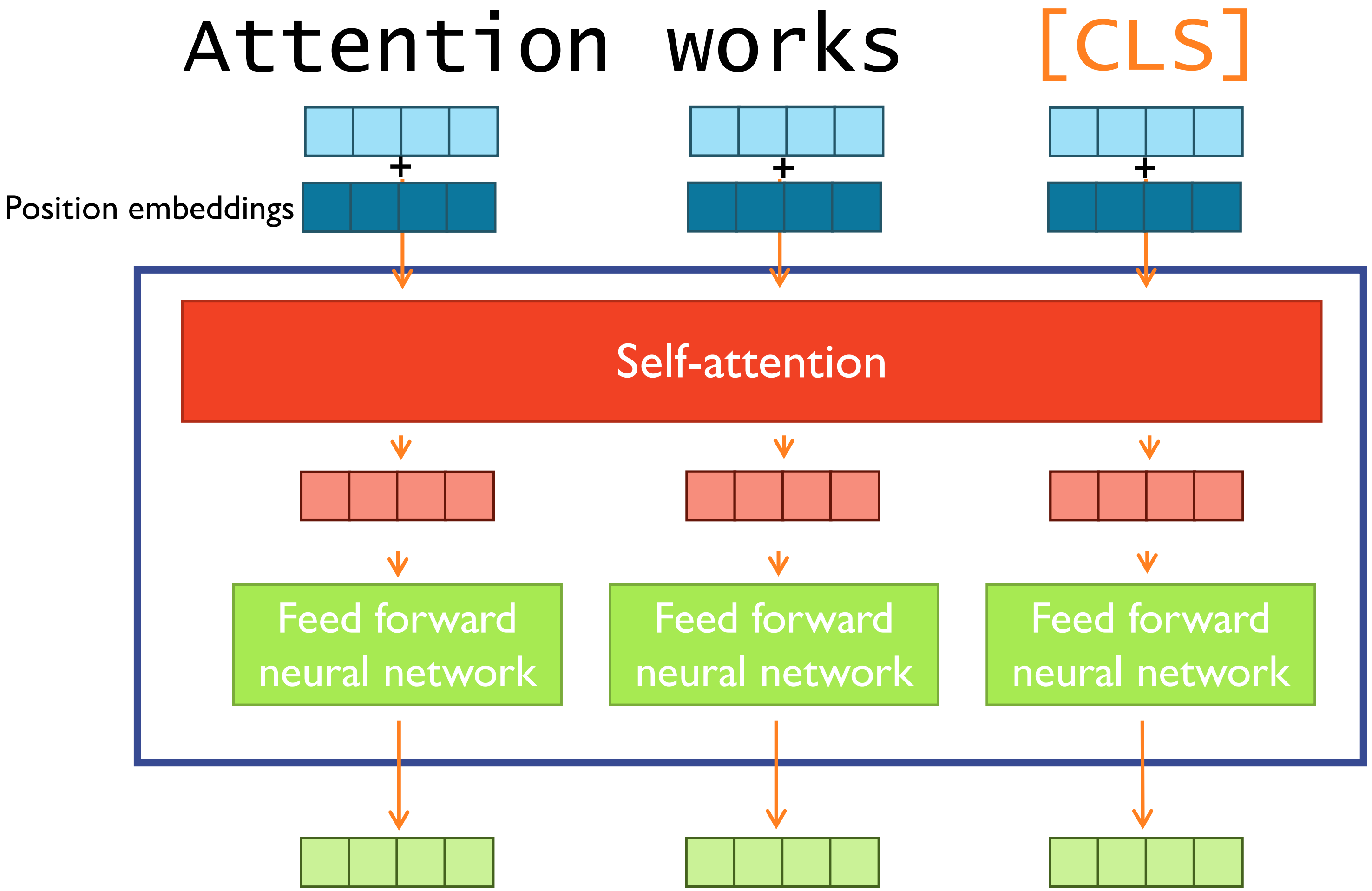
Position embeddings

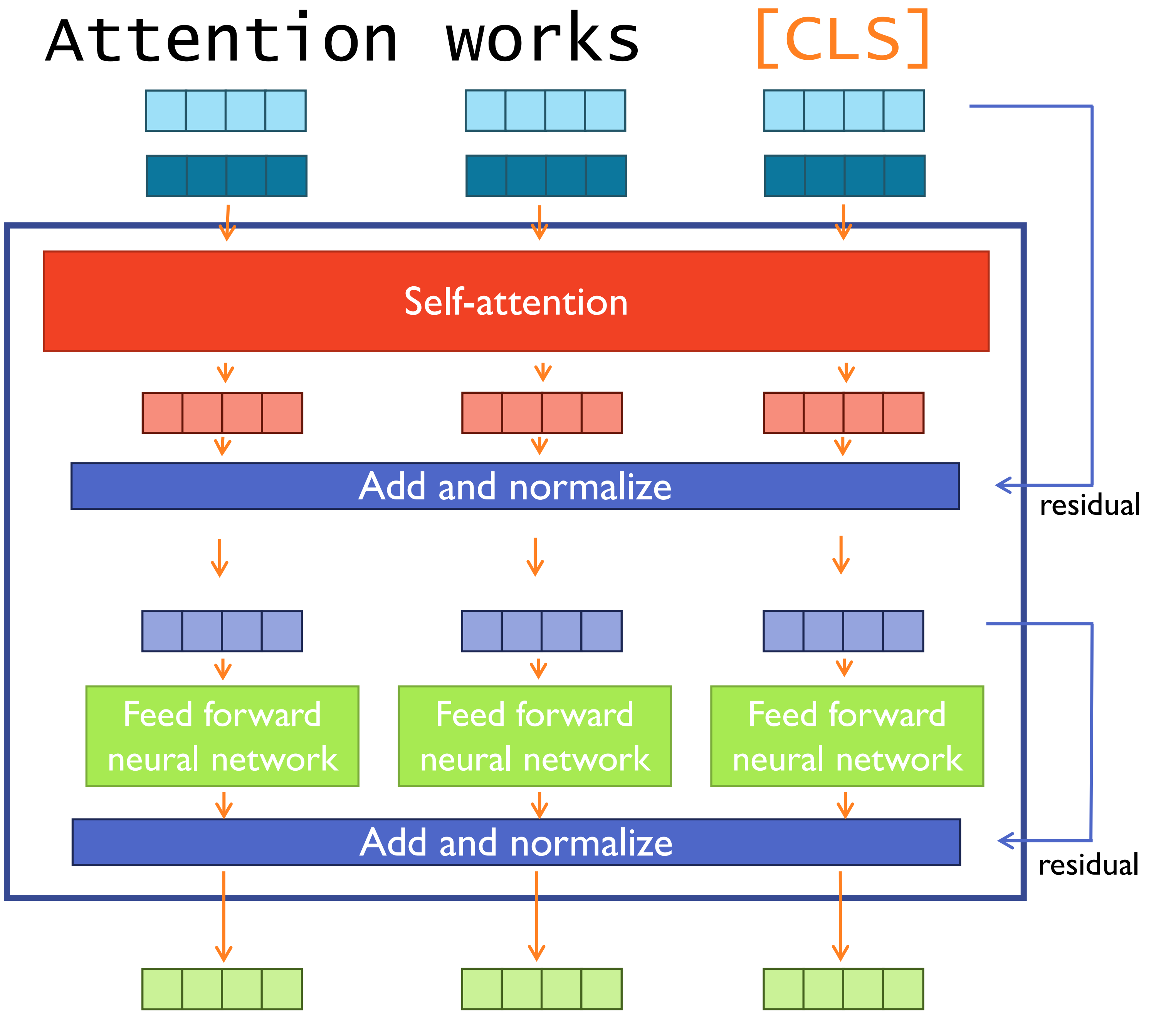
- Sinusoidal

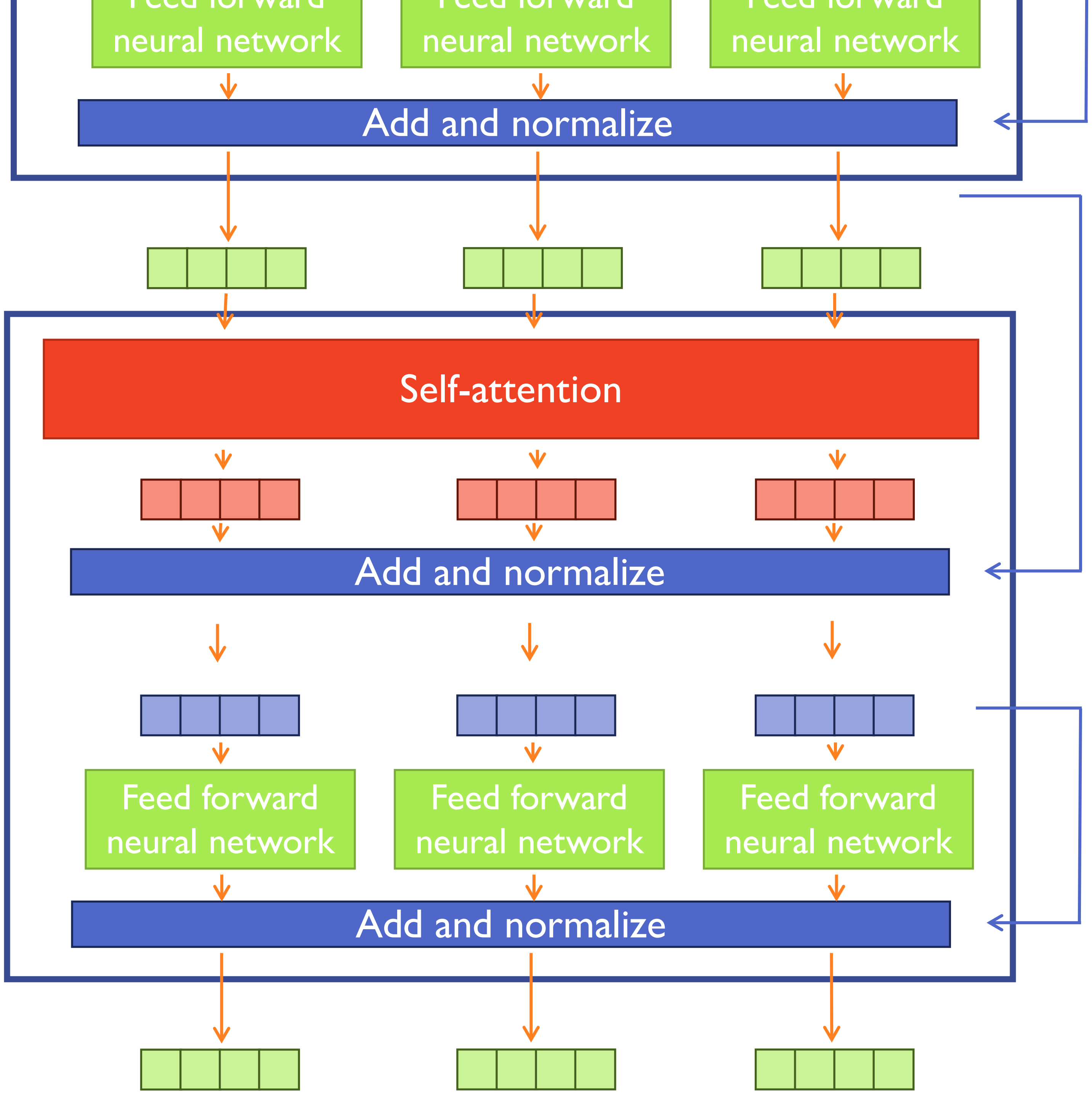
$$
\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}
$$



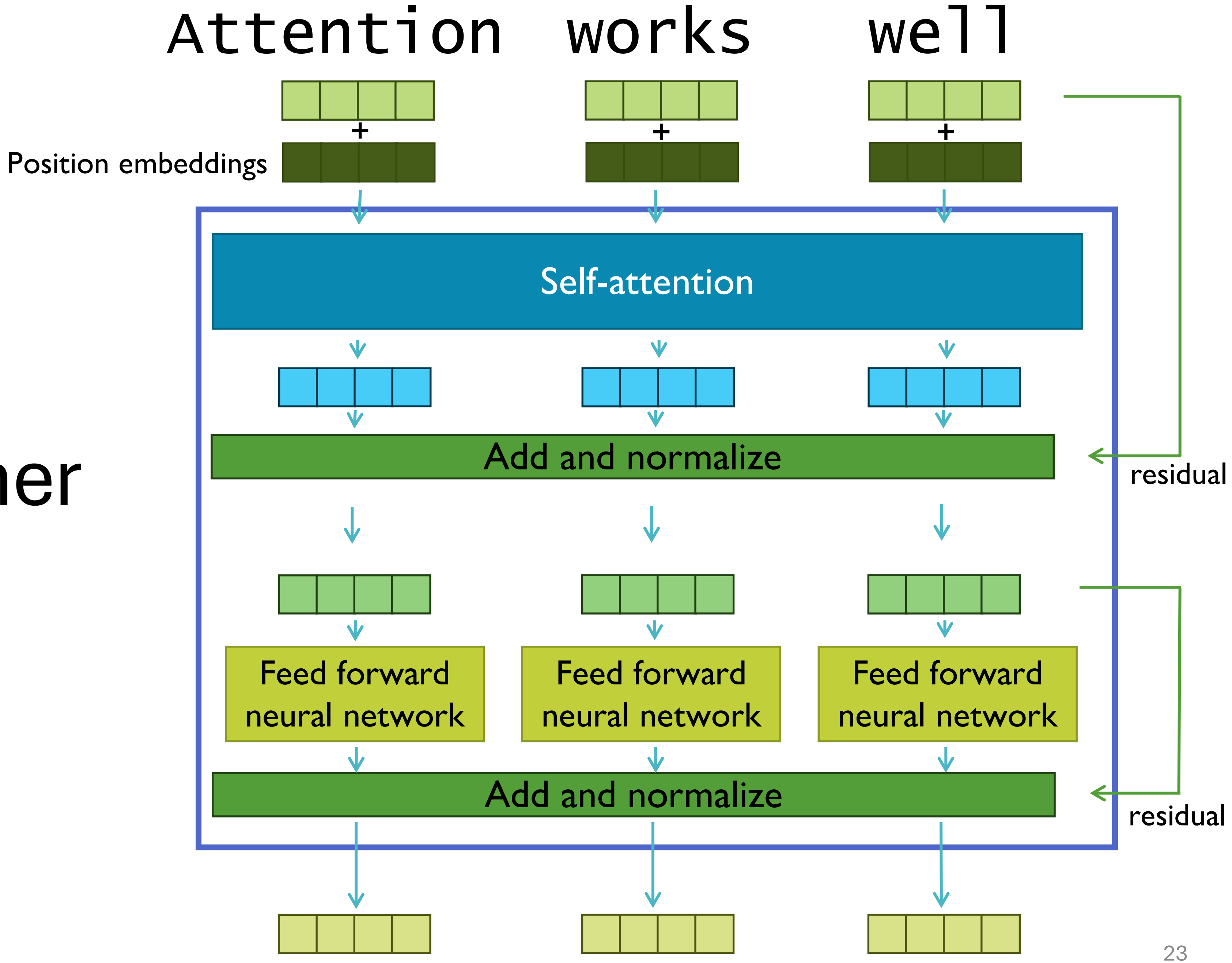
- Learned position embeddings
- Rotary position embeddings https://blog.eleuther.ai/rotary-embeddings/


Attention works [CLS]
+
+
+
Position embeddings
Self-attention
Feed forward neural network
Feed forward neural network
Feed forward neural network


Attention works [CLS]
Self-attention
Add and normalize
residual
Feed forward neural network
Feed forward neural network
Feed forward neural network
Add and normalize
residual


Feed forward neural network
Feed forward neural network
Feed forward neural network
Add and normalize
Self-attention
Add and normalize
Feed forward neural network
Feed forward neural network
Feed forward neural network
Add and normalize

The Transformer **Layer**



2

Language Models Architectures

- Encoders (e.g., BERT)
- Decoders (e.g., GPT)
- Encoder-Decoders (e.g., T5).

Two main ways of SSL on Text

Let's train on "The cat sat on the mat."

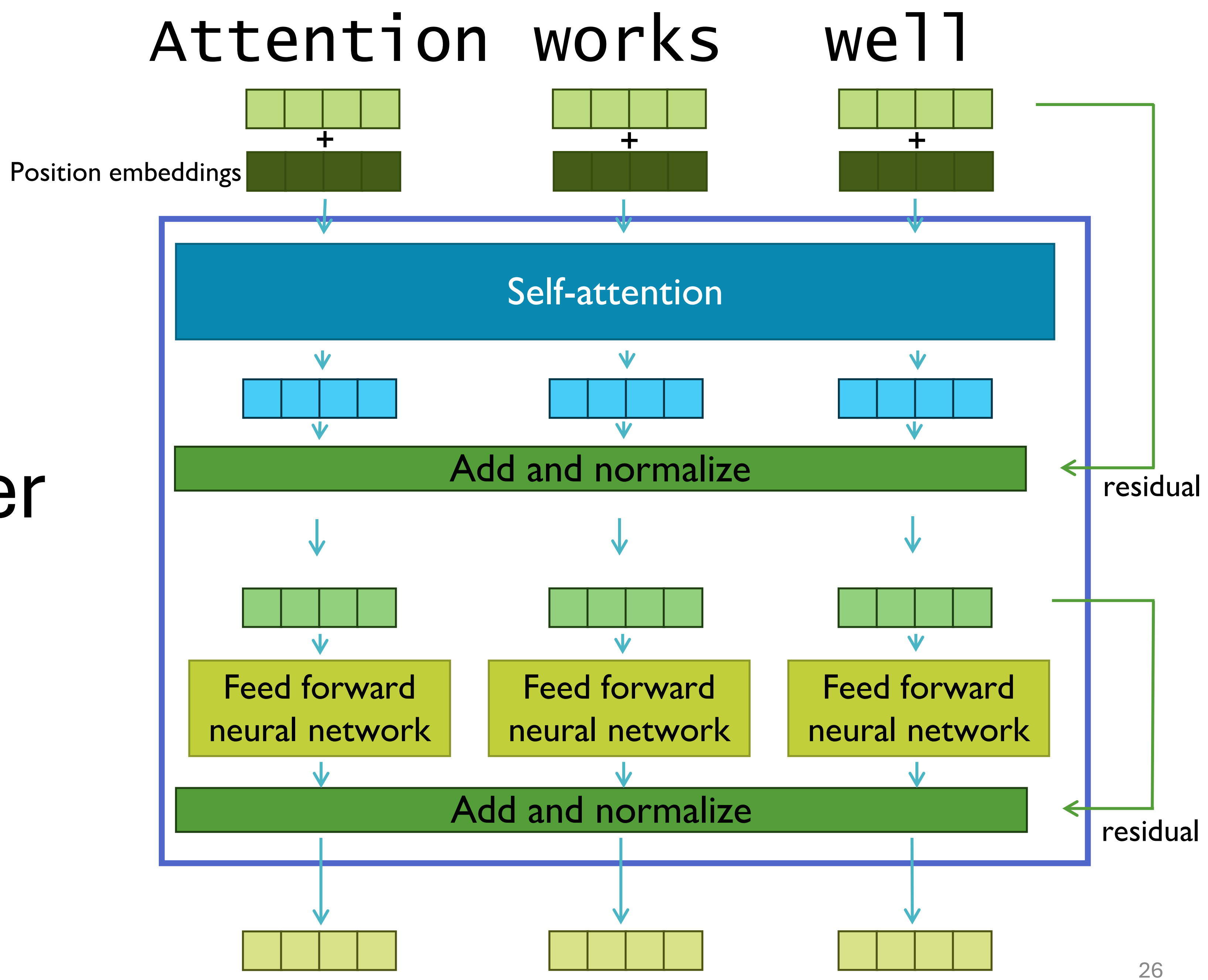
Masked Language Modelling (MLM)

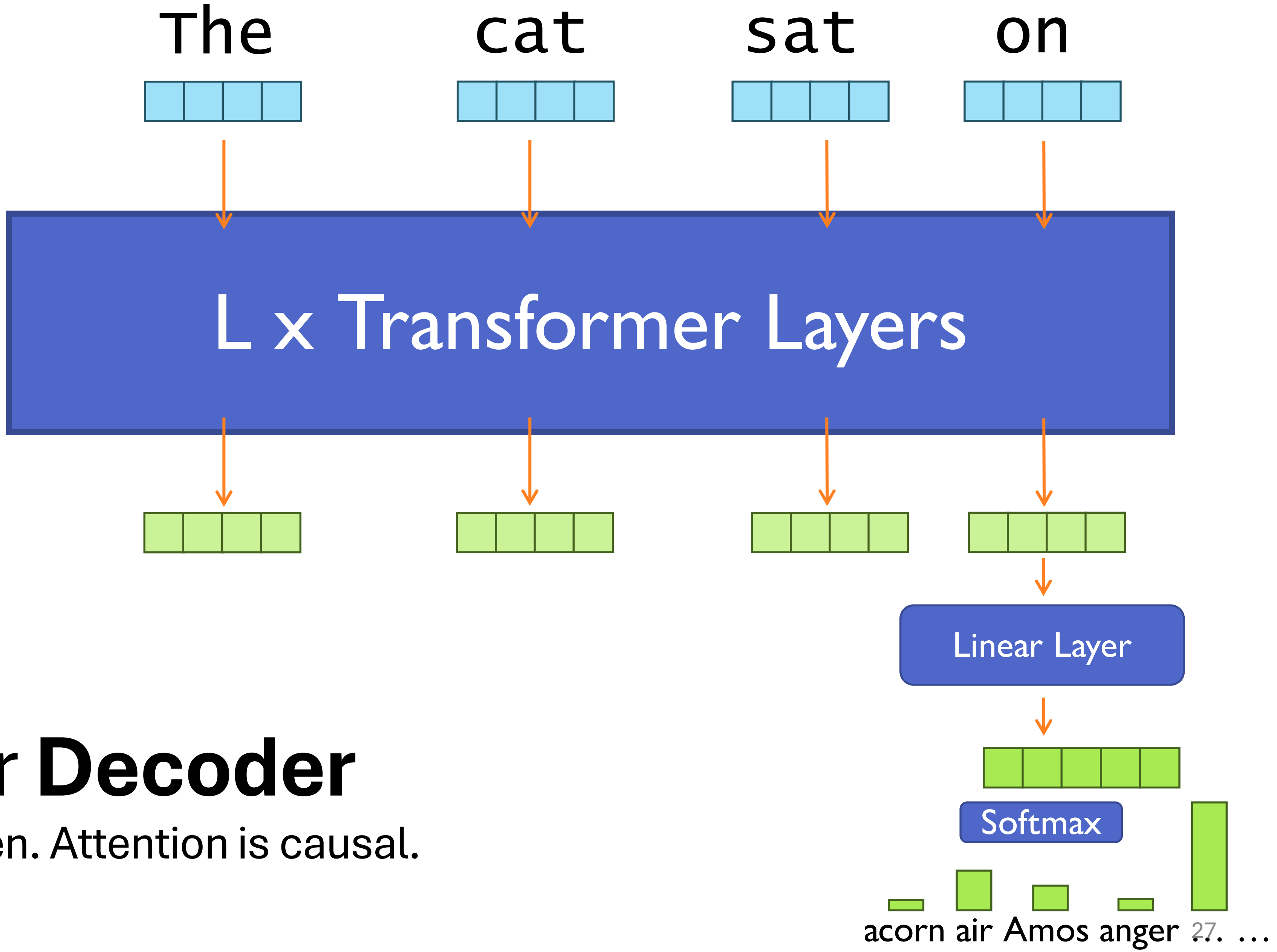
- Training data: Text with 15% masked tokens.
  - The [MASK] sat on the mat.
- The model predicts the [MASK] tokens.
  - Classification setting: Choose one of 60,000 tokens / words in the vocabulary to put instead of [MASK]?
  - Loss function: Cross-Entropy
  - The dog sat on the mat.
  - The **cat** sat on the mat.
- Transformer **encoders** / bidirectional encoders:
  - Notable examples: BERT, RoBERTa, etc.

Autoregressive Language Modelling

- Training data: Complete the text.
  - The cat sat ....
- The model predicts the text.
  - Classification setting: Choose one of the 60,000 tokens / words in the vocabulary to put as next token.
  - Loss function: Cross-Entropy
- Autoregressive, because during inference:
  - The -> **cat**
  - The cat -> **sat**
  - The cat sat -> **on**
  - The cat sat on -> **the**
  - The cat sat on the -> **mat**
- Transformer **decoders** / autoregressive decoders / causal transformers:
  - GPT, GPT-2, GPT-3, ChatGPT, GPT-4

The Transformer **Layer**

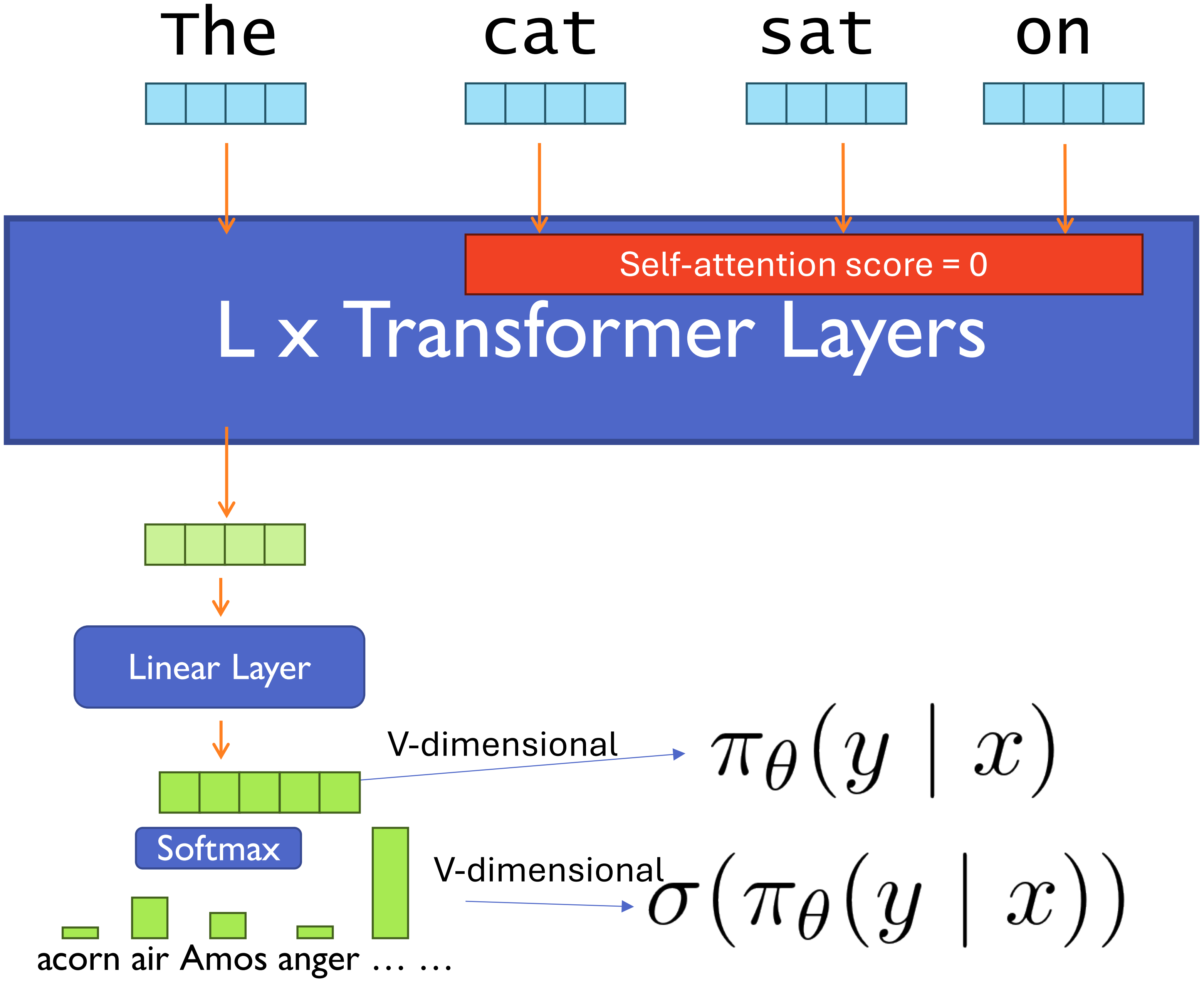




Transformer **Decoder**

Predicts the next token. Attention is causal.

Training



$$\pi_\theta(y \mid x)$$

$$\sigma(\pi_\theta(y \mid x))$$

Supervised training

- The model produces logits π_θ for all tokens in the vocabulary -> softmax to get probabilities.
- During training, we compute the loss with cross-entropy.

$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \sigma(\pi_\theta(y \mid x))\right]$$

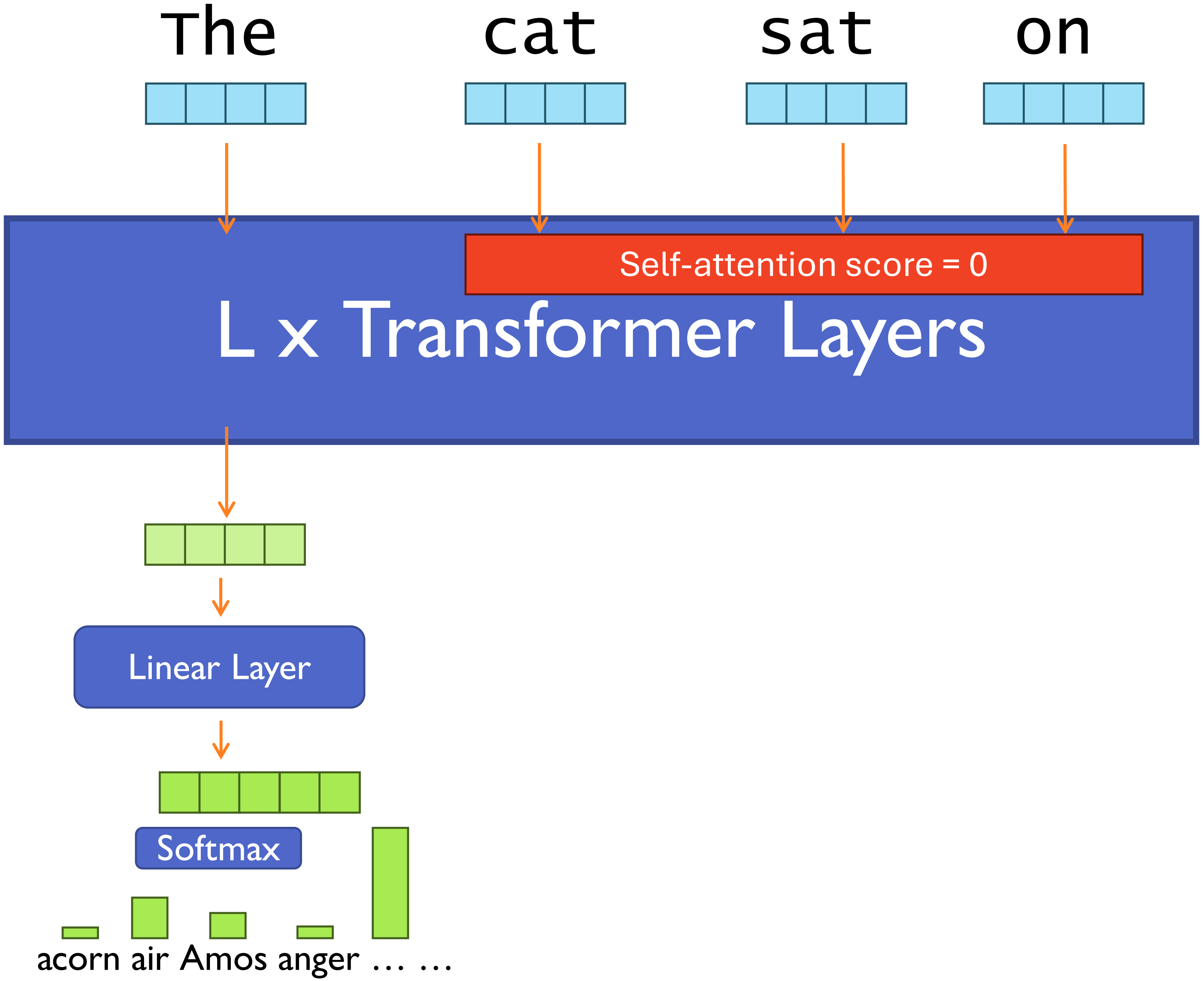
```python
import numpy as np

def cross_entropy(P, Q):
    return -np.sum(P * np.log(Q))

# probabilities for true vs predicted values
P = np.array([0, 1, 0, 0, 0])                 # True distribution
Q = np.array([0.1, 0.6, 0.1, 0.15, 0.05])  # Predicted distribution

print(f"Cross Entropy: {cross_entropy(P, Q):.2f}")
```

softmax over all vocabulary tokens

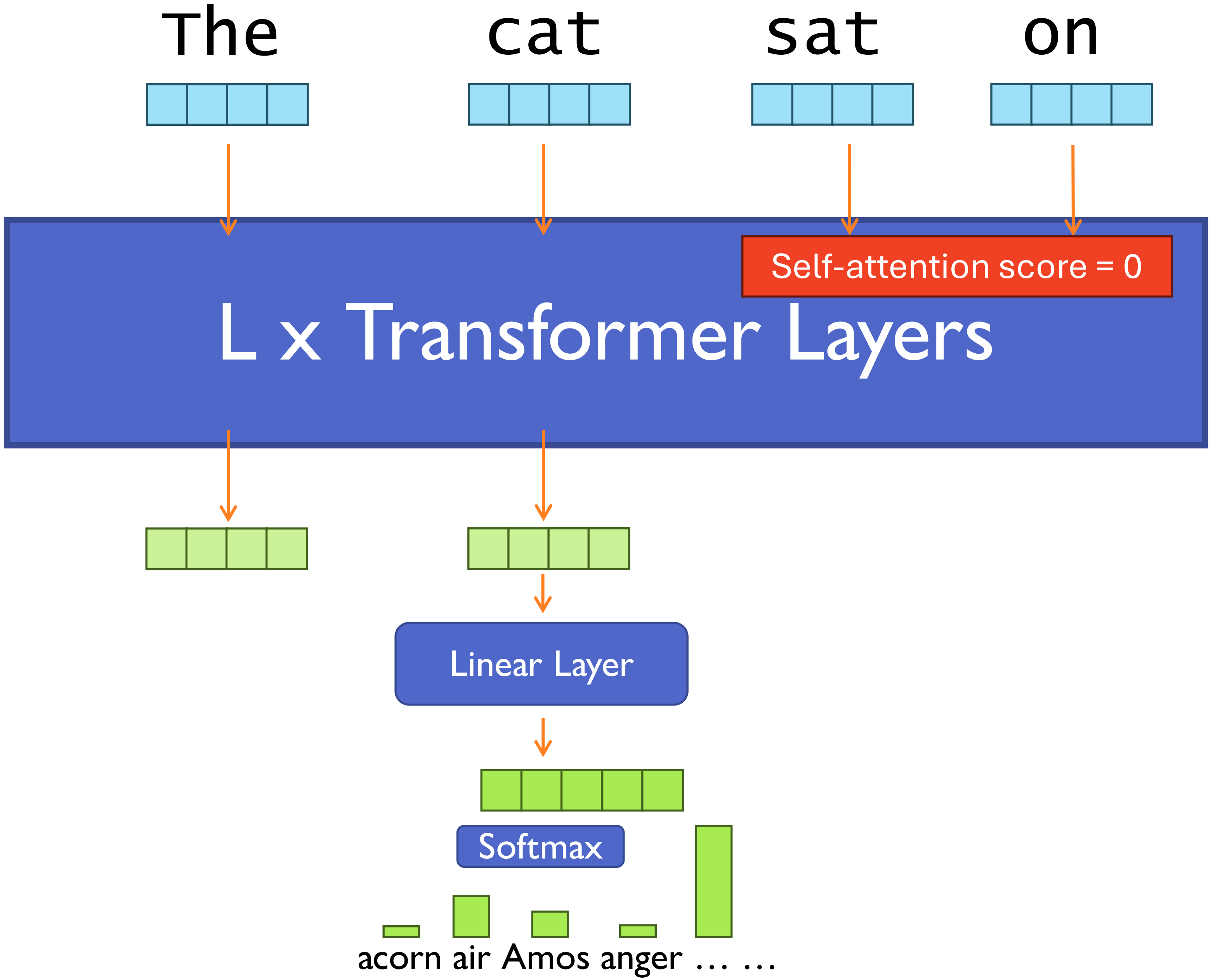
the correct (target)next token

text input

Training

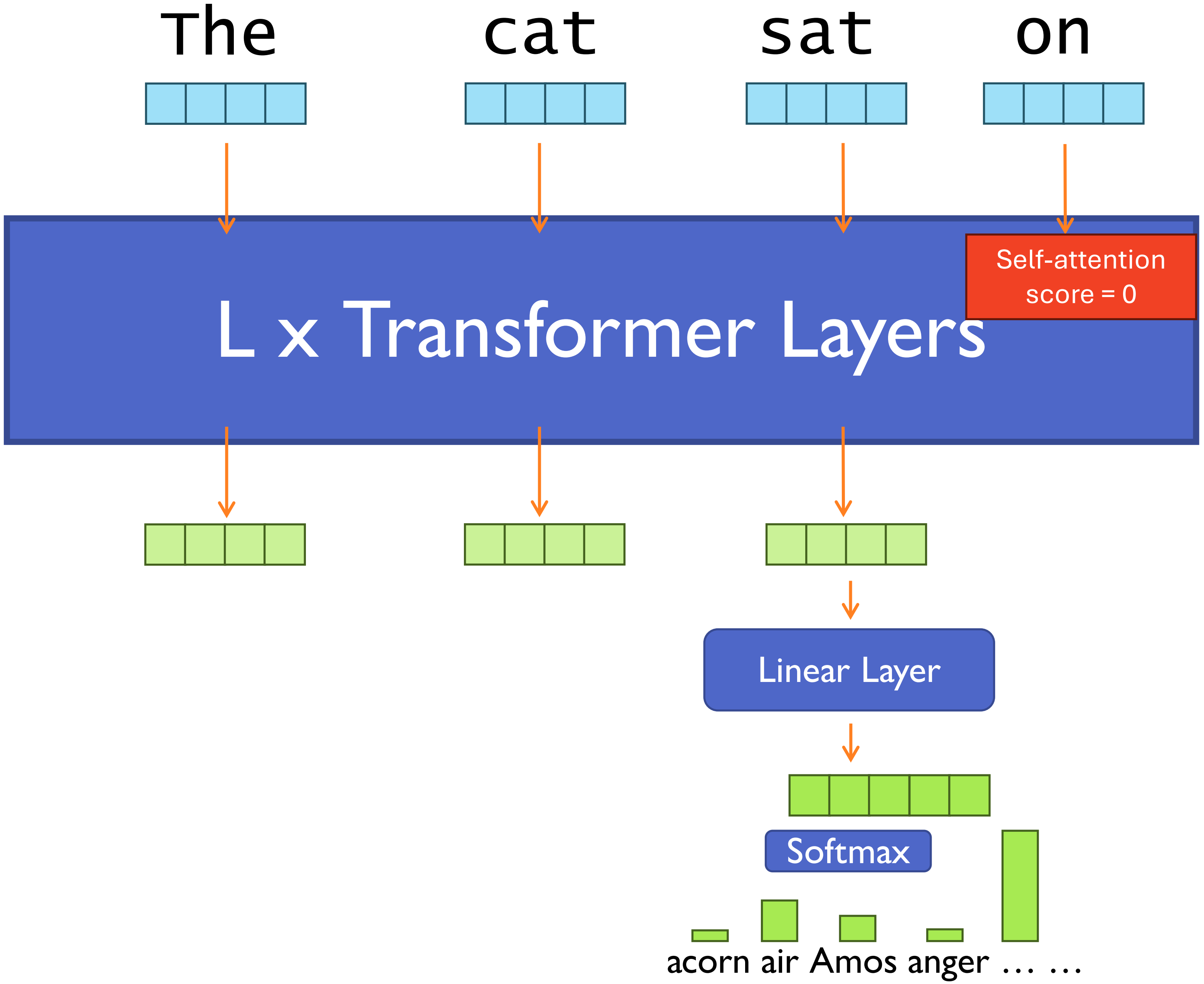

The
cat
sat
on
Self-attention score = 0
L x Transformer Layers
Linear Layer
Softmax
acorn air Amos anger … …

Training

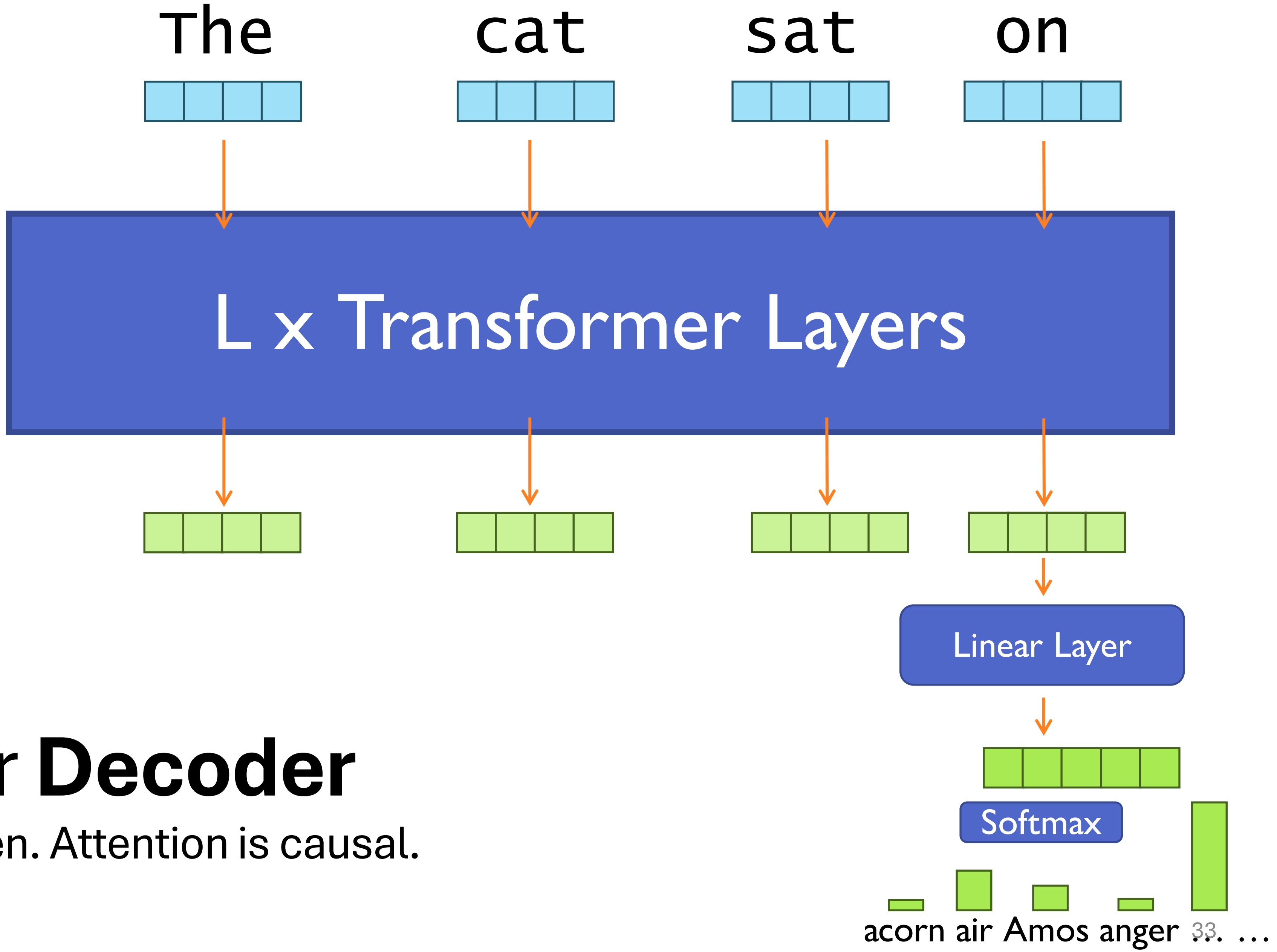


The
cat
sat
on
Self-attention score = 0
L x Transformer Layers
Linear Layer
Softmax
acorn air Amos anger … …

Training



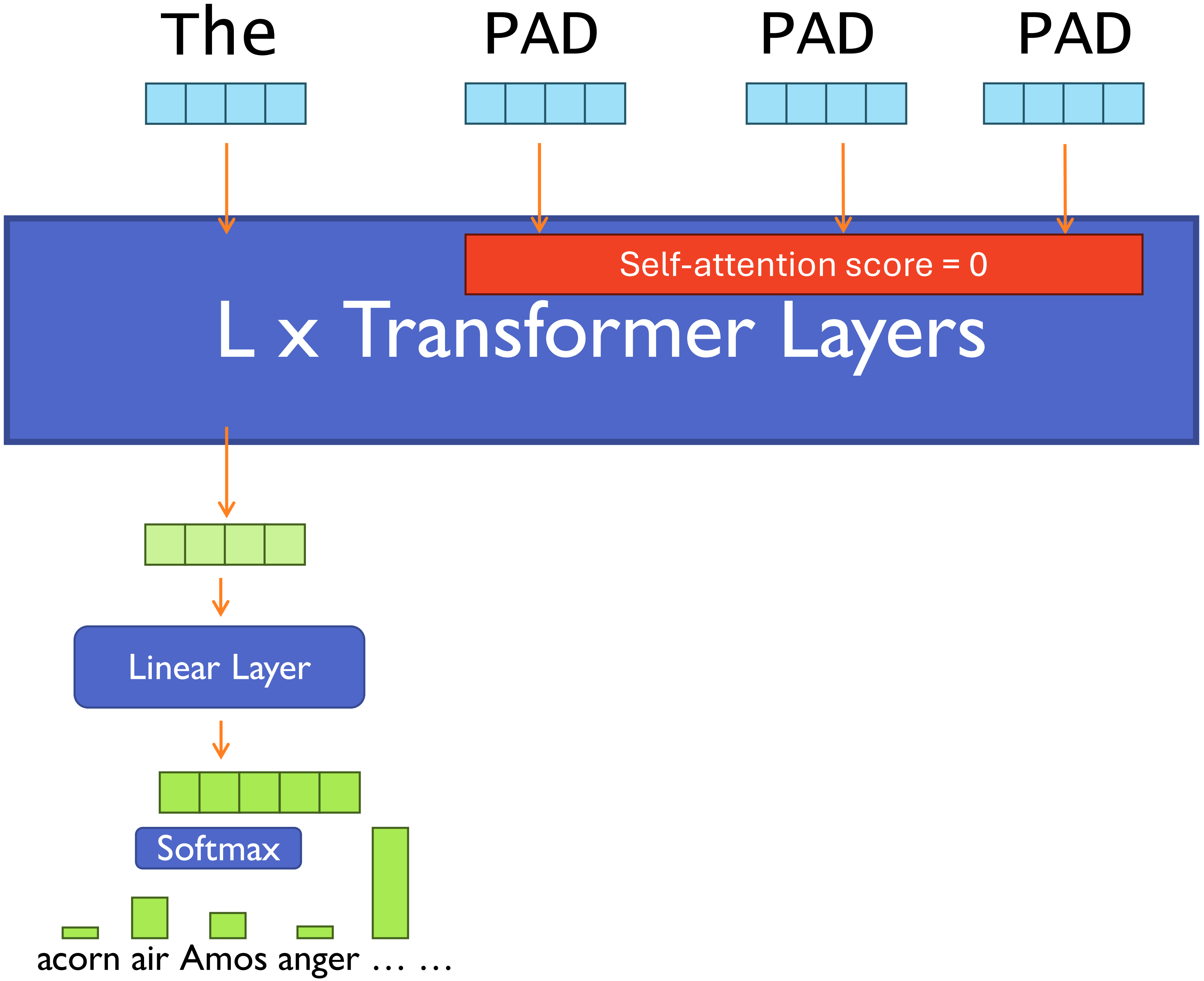
The
cat
sat
on
L x Transformer Layers
Self-attention
score = 0
Linear Layer
Softmax
acorn air Amos anger … …



Transformer **Decoder**

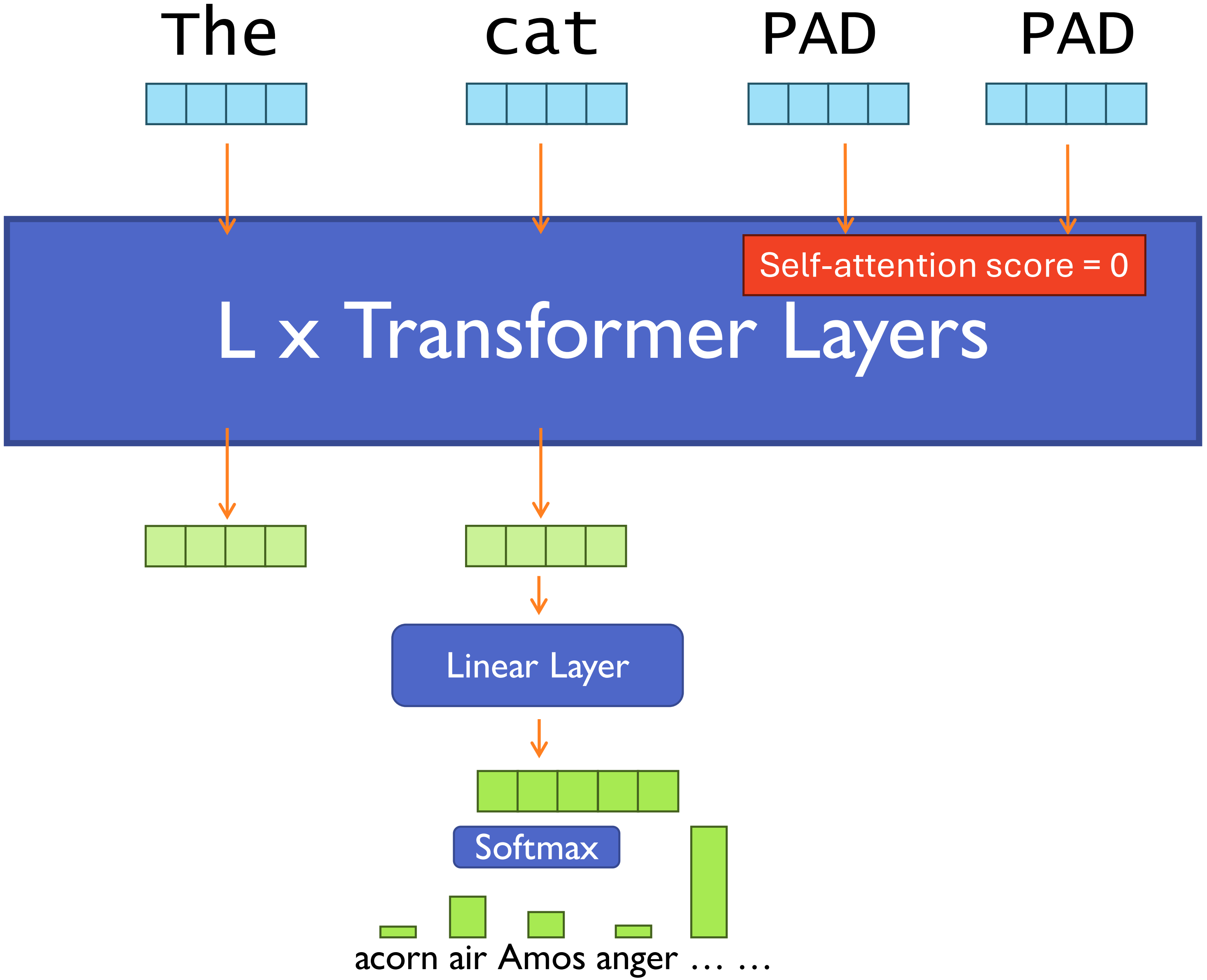
Predicts the next token. Attention is causal.

Inference

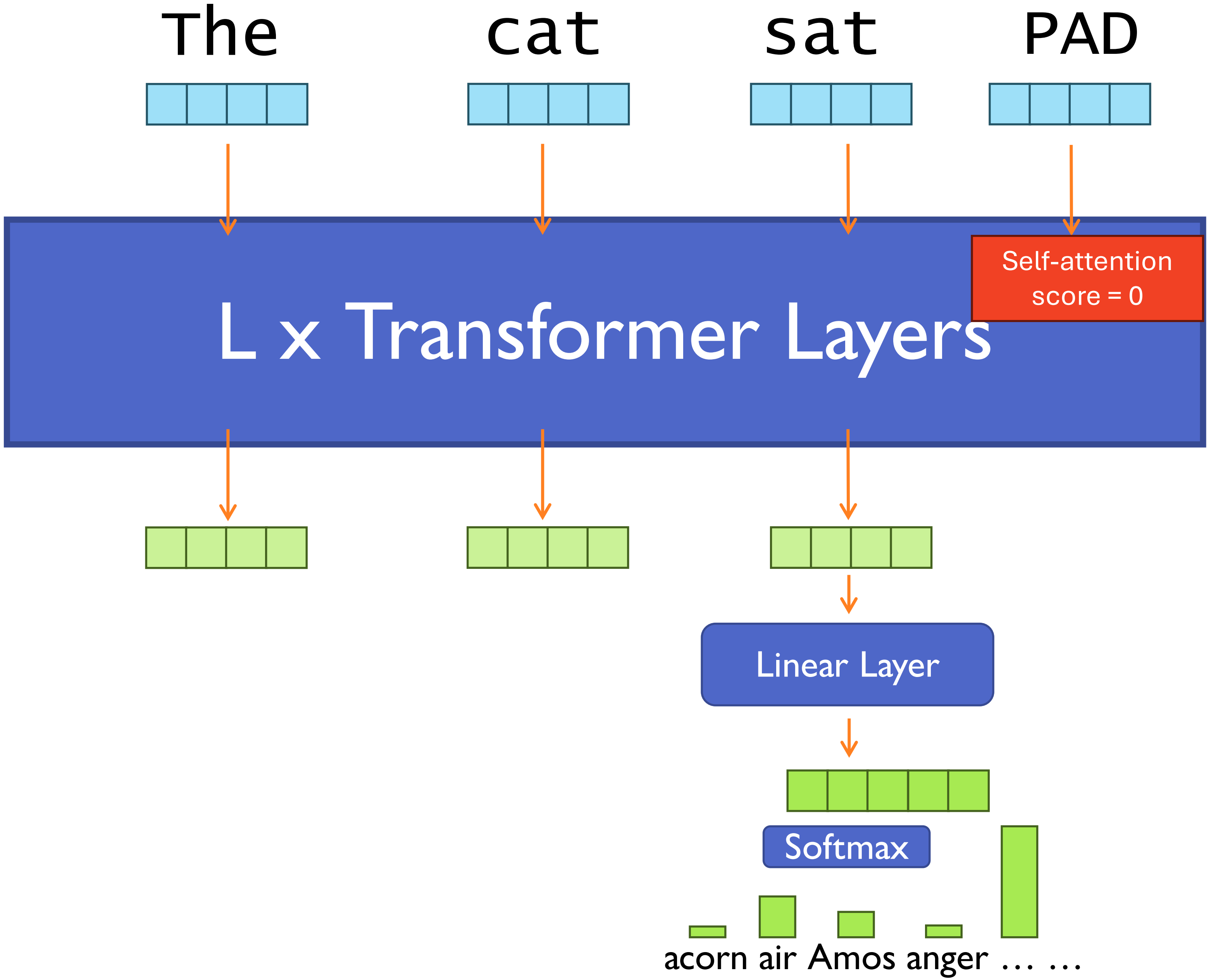


The
PAD
PAD
PAD
Self-attention score = 0
L x Transformer Layers
Linear Layer
Softmax
acorn air Amos anger … …

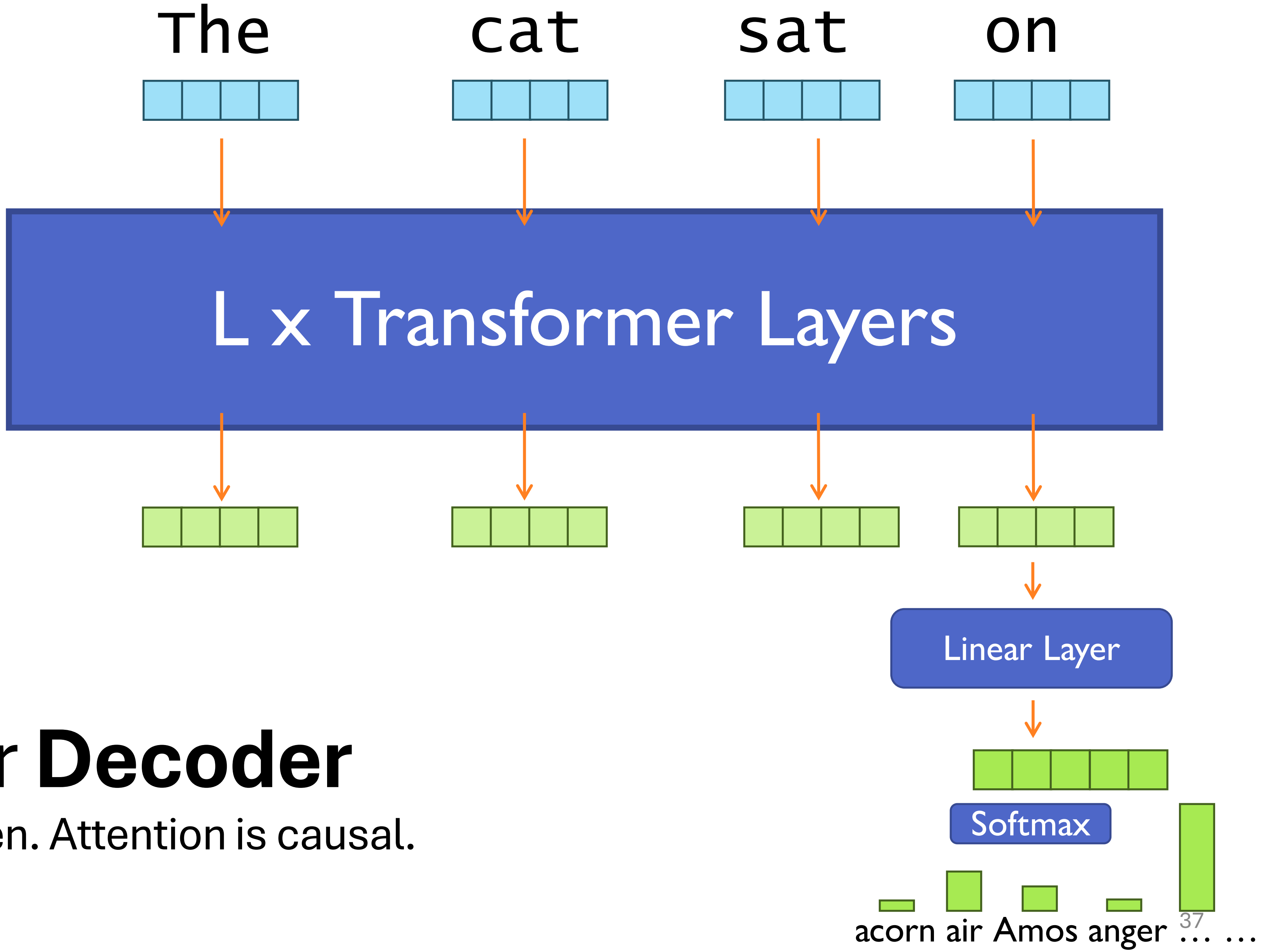
Inference


The
cat
PAD
PAD
Self-attention score = 0
L x Transformer Layers
Linear Layer
Softmax
acorn air Amos anger … …

Inference

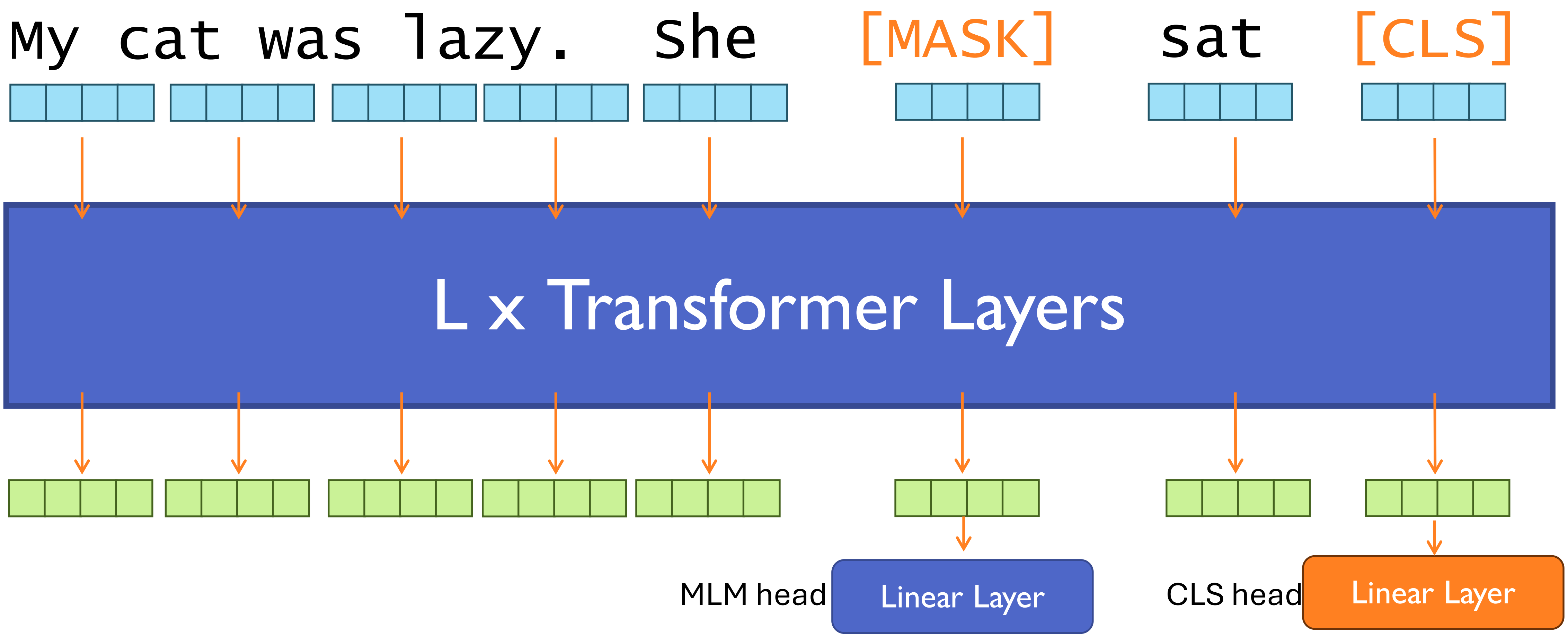


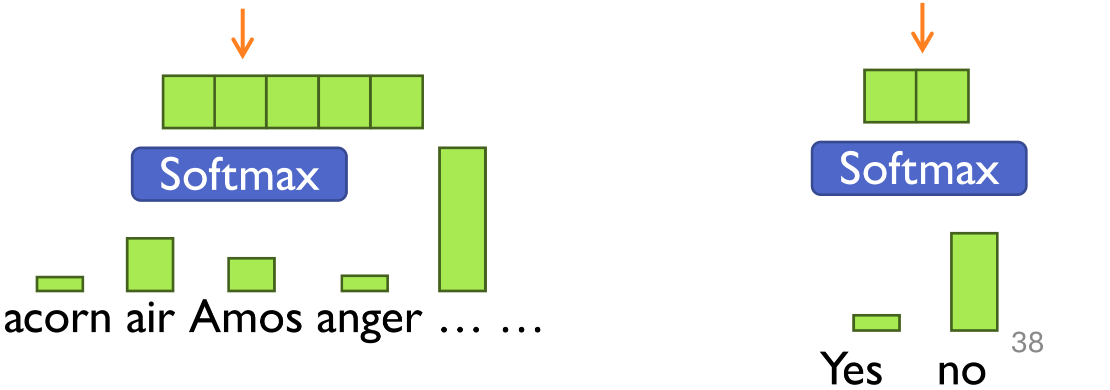
The
cat
sat
PAD
L x Transformer Layers
Self-attention score = 0
Linear Layer
Softmax
acorn air Amos anger … …



Transformer **Decoder**

Predicts the next token. Attention is causal.





Transformer **Encoder**

Attention is bidirectional.
Predicts the MASK token and the CLS token.
CLS: Is the second sentence a continuation of the first sentence?

What different layers learn via just self-supervision

input

Transformer encoder	part-of-speech (POS) tags
Transformer encoder	constituents
Transformer encoder	dependencies
Transformer encoder	semantic roles (who, to whom, when, where?)
Transformer encoder	coreference
Transformer encoder	relations, proto-roles, roles, ...

output

BERT Rediscovers the Classical NLP Pipeline

Ian Tenney[1] **Dipanjan Das**[1] **Ellie Pavlick**[1,2]
[1]Google Research [2]Brown University
{iftenney,dipanjand,epavlick}@google.com

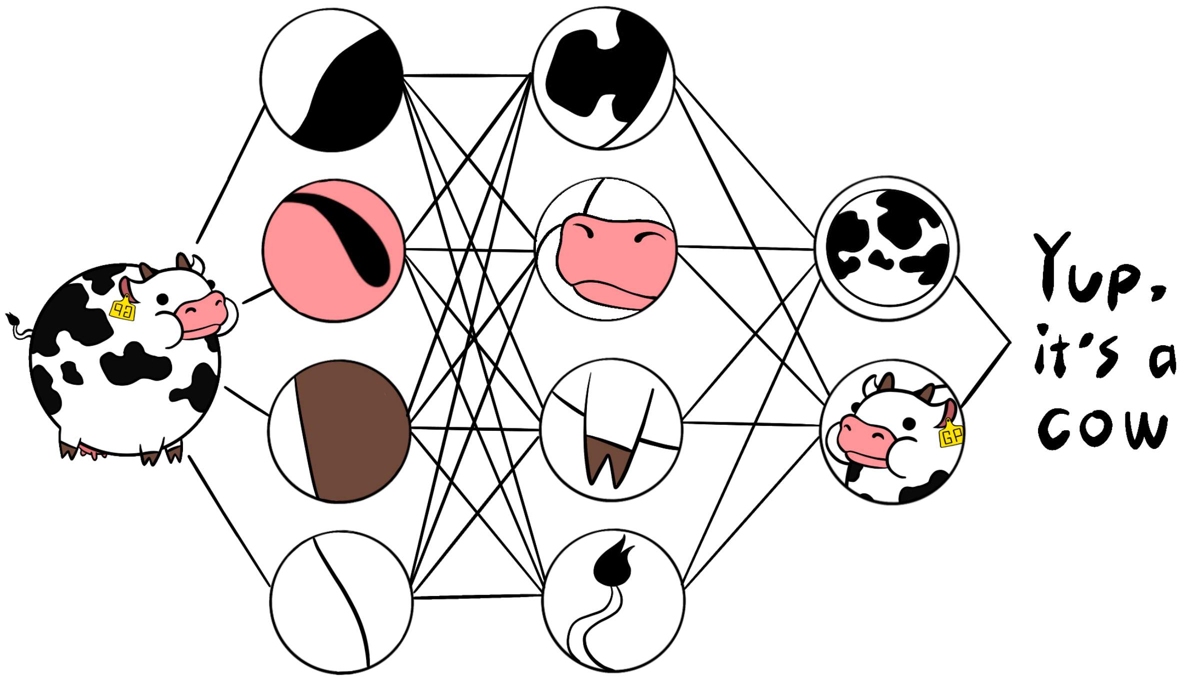
Abstract

Pre-trained text encoders have rapidly advanced the state of the art on many NLP tasks. We focus on one such model, BERT, and aim to quantify where linguistic information is captured within the network. We find that the model represents the steps of the traditional NLP pipeline in an interpretable and localizable way, and that the regions responsible for each step appear in the expected sequence: POS tagging, parsing, NER, semantic roles, then coreference. Qualitative analysis reveals that the model can and often does adjust this pipeline dynamically, revising lower-level decisions on the basis of disambiguating information from higher-level representations.

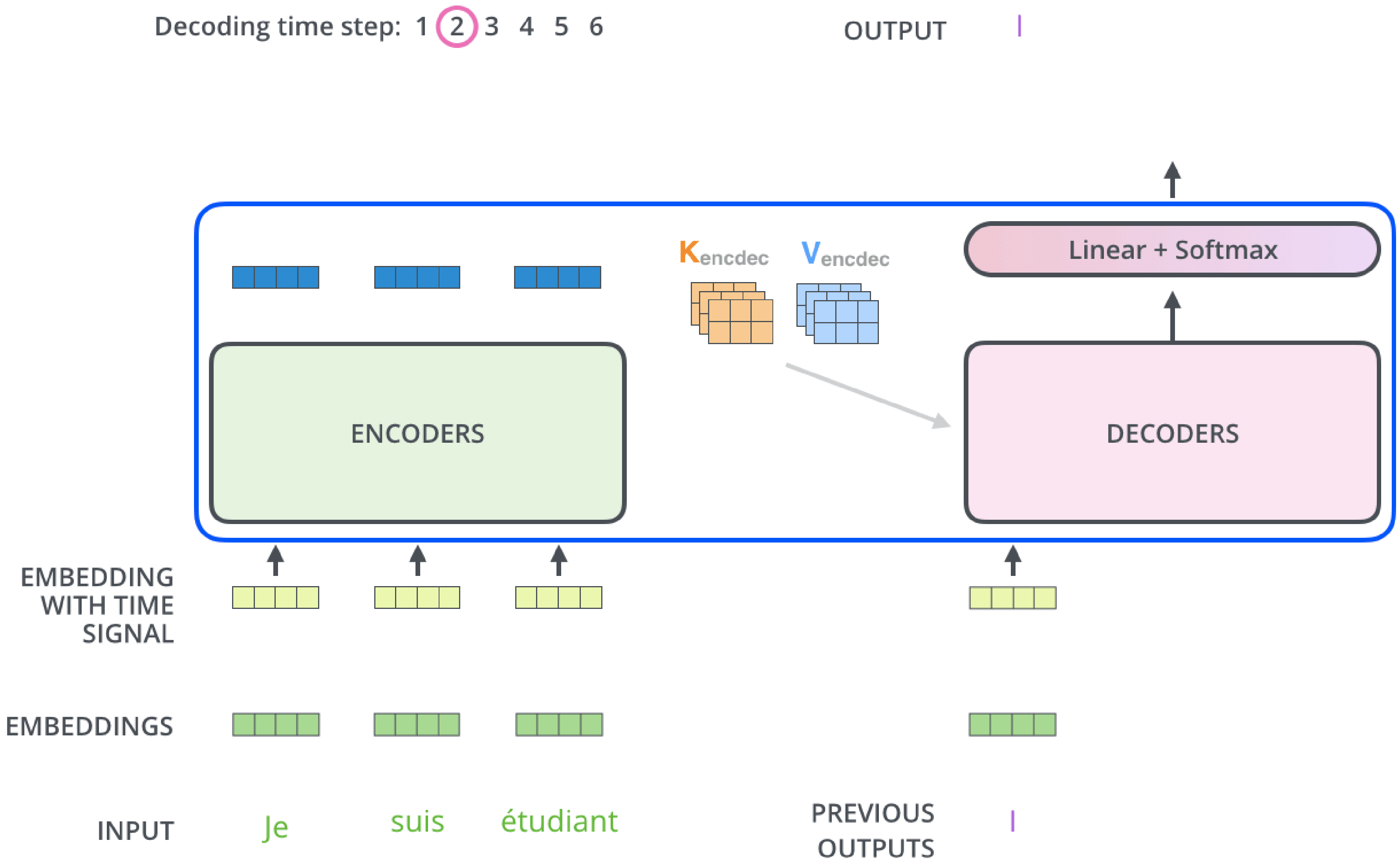
of the network directly, to assess whether there exist localizable regions associated with distinct types of linguistic decisions. Such work has produced evidence that deep language models can encode a range of syntactic and semantic information (e.g. Shi et al., 2016; Belinkov, 2018; Tenney et al., 2019), and that more complex structures are represented hierarchically in the higher layers of the model (Peters et al., 2018b; Blevins et al., 2018).

We build on this latter line of work, focusing on the BERT model (Devlin et al., 2019), and use a suite of probing tasks (Tenney et al., 2019) derived from the traditional NLP pipeline to quantify where specific types of linguistic information are

Low- to high-level feature hierarchy in CNNs



Encoder-Decoder



The decoder attends to:

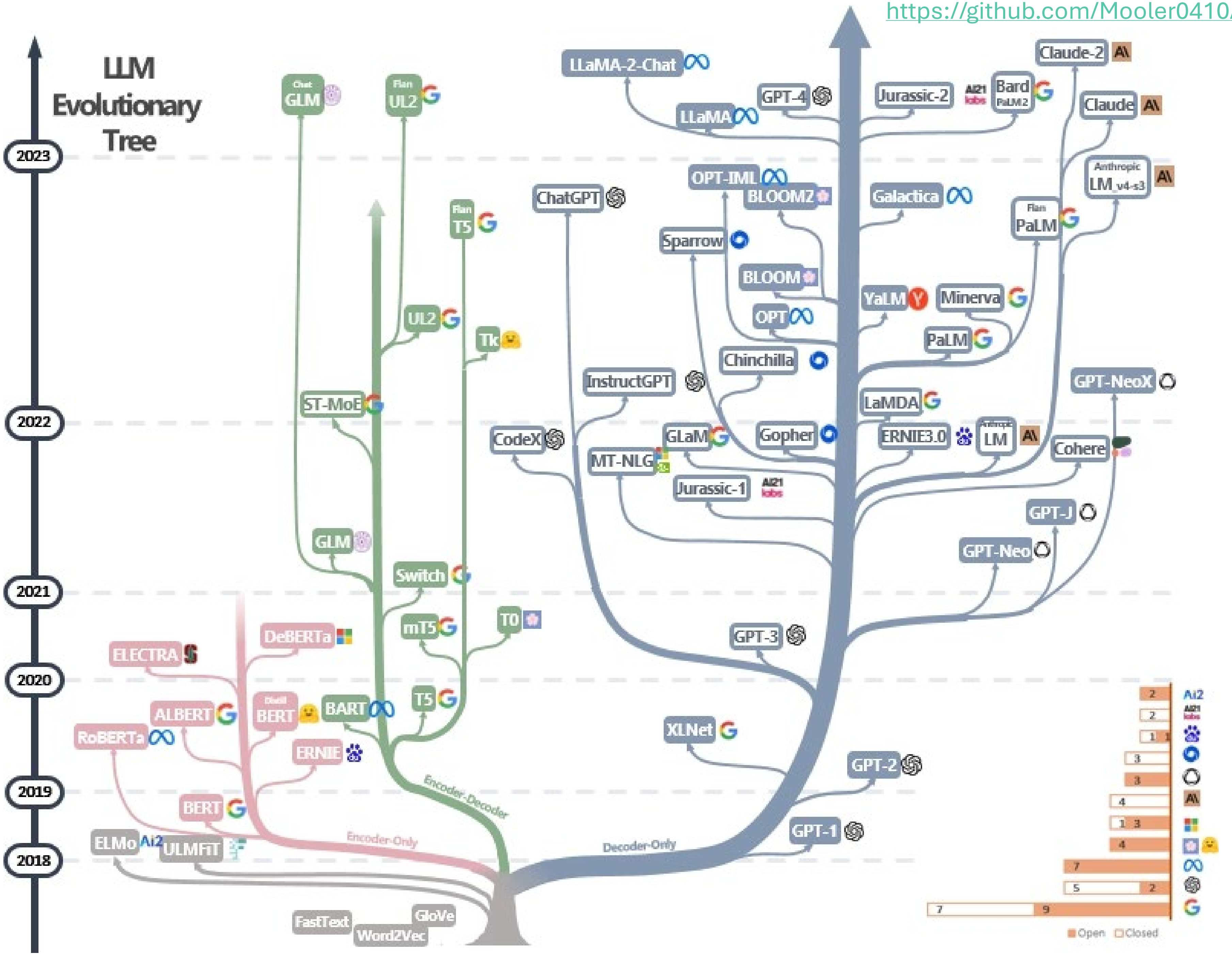
- its own previous steps (self-attention).
- the encoder's outputs (via key and values).

https://jalammar.github.io/illustrated-transformer/

Why 3 types of LM architectures?

- Encoders:
  - 👍 Can learn to represent context, e.g. the entire input into one vector (e.g., [CLS] token) – make great sentence representations for e.g. RAG or diffusion models!
  - 👎 Are not (yet) great at producing language from scratch,
  - 👍 but are great at replacing missing words.
- Decoders:
  - Are generative models (of language).
    $$p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)\ \dots p(x_N|x_1,x_2,\dots,x_{N-1})$$
  - Make great chatbots.
- Encoder-Decoders are a mix of both.
  - 👍 combine the upsides from encoders and decoders.
  - 👎 bring the training instabilities and complexities from both, thus harder to scale.

https://github.com/Mooler0410/LLMsPracticalGuide



When is an LM an LLM versus an SLM?

- Depends if you are gpu-rich (10^5 H100s), gpu-poor (10^0 3090s), or gpu-middle-class (10^3 A100s).
- L in LLM is not a time-invariant attribute:
  - According to capabilities, the 175 billion parameter model of 2020 is a 7 billion parameter model in 2024.
  - The GPU-rich get GPU-richer and can scale up to ever more parameters.
- For this lecture, with LLM we just mean an autoregressive LM (transformer decoder).

Contents

1. The Transformer
2. Brief overview of language models architectures

est. 2017

3. Prompting
4. Post-training
5. Benchmarking

est. 2020

3

Prompting

- Vanilla prompting
- Prompt tuning
- Chain-of-Thought (CoT)
- In-context (few-shot) learning
- Retrieval augmented generation (RAG)

Let’s assume we have trained a transformer **decoder** on the entire Internet!

- LLMs (transformer decoders) trained on lots of text data can solve multiple tasks.
- Now, we want to get out its expertise (i.e., make ChatGPT out of it).
- Thus, we let it complete the following sentence:

When was Albert Einstein born?

Completions:

March 14, 1879

When was Heisenberg born?

This question will come up in the science history exam.

❗ These are all valid completions given the training task and training data of the model (autocomplete the entire Internet).

Greatest idea and scientific advancement in NLP since 2017.

Let's ask the model nicely
to do what we want it to do.
☺

aka. prompting

Prompting

- Finding good ways of presenting the input to the model in order to increase the probability of the expected answer.
- Example task: Natural Language Inference.
  - Data sample:
    - *"Premise: John ate pasta for supper. Hypothesis: John ate pasta for dinner."* 1/10,000
  - Labels: <u>Entailment</u>, contradiction.
- We can verbalize the **same** example from that task in **many** ways:
  - *Suppose John ate pasta for supper. Can we infer that "John ate pasta for dinner."?* 1/20
  - *Suppose John ate pasta for supper. Can we infer that "John ate pasta for dinner."? Yes or no?* 1/2
  - *Are we right to infer that if John ate pasta for supper, that John ate pasta for dinner? Yes or no?* ... (many other possibilities)

Here we are tuning in word space.

Vanilla prompting

Reminds us of adversarial examples repurposed for good!

Here we are tuning in word **embedding** space.

Prompt tuning

Premise: John ate pasta for supper.
Hypothesis: John ate pasta for dinner.

Prompt engineering.

Suppose John ate pasta for supper. Can we infer that "John ate pasta for dinner."? Yes or no?

tokenization

...

Huge Language Model

Yes.

No human prompt engineering.

Premise: John ate pasta for supper.
Hypothesis: John ate pasta for dinner.

tokenization

...

Huge Language Model

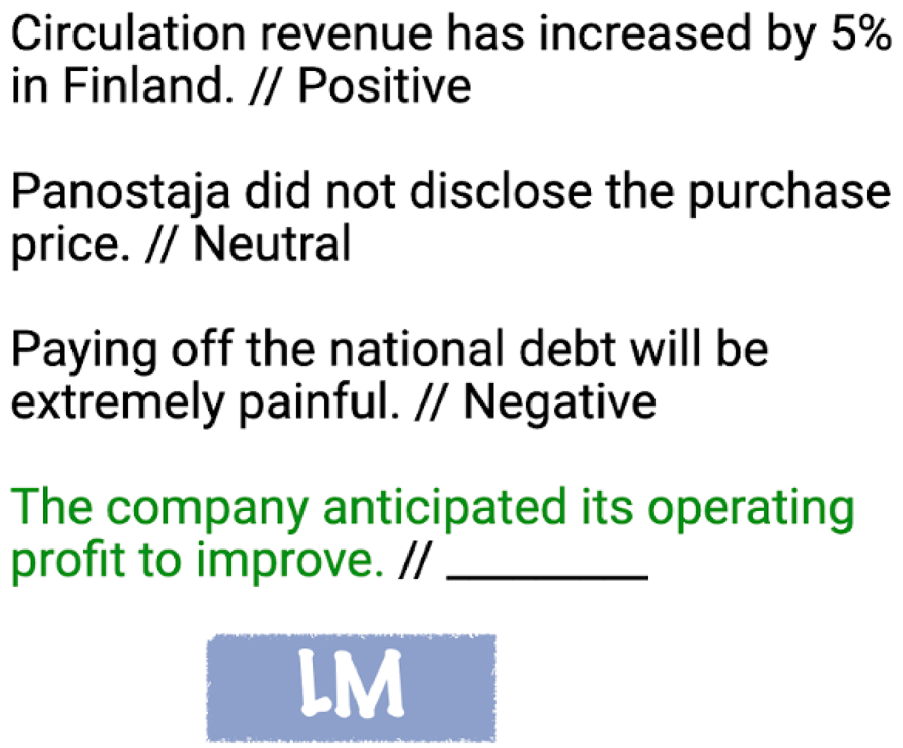
These are just vectors with numbers that we optimize until the output is correct.

Yes. **No.**

In-context few-shot learning

Give the model some solved examples as input before asking it a final question for it to solve.

Example on sentiment classification: 👉

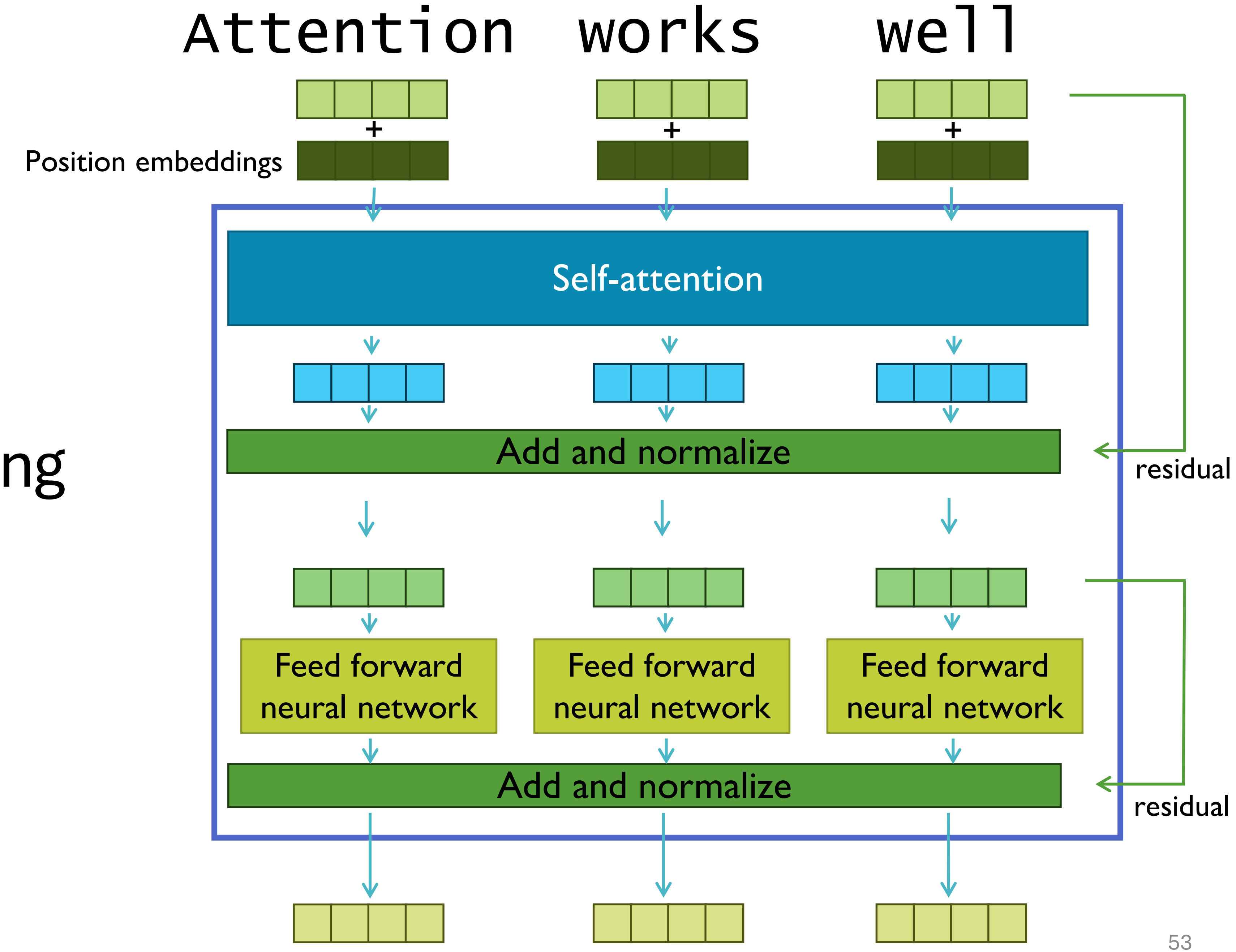


This works, because we restrict the solution space by specifying the labels (positive, neutral, negative).

Read more: Min et al. 2022
https://arxiv.org/abs/2202.12837
and follow-up work.

http://ai.stanford.edu/blog/understanding-incontext/

In-context learning versus SGD



The second greatest idea and scientific advancement in NLP since 2017.

Let’s think step by step.

Add this to your prompt and write an entire paper about it. ☺
https://arxiv.org/pdf/2205.11916

Chain-of-Thought Prompting

- **Prompt:**
  - The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have? Let’s think step by step:
- **Model Response:**
  - The cafeteria had 23 apples originally. They used 20 to make lunch. So, they had 23-20 = 3. They bought 6 more apples, so they have 3+6=9.

- The answer becomes more likely given that:
  - The problem is broken down into smaller pieces (divide and conquer).
  - The tokens preceding the answer describe a simple problem “3+6”, instead of the original, more complicated problem setting.

Retrieval Augmented Generation (RAG)

- **Prompt:**
  - How many employees did NVIDIA have in 2024?
- **Model Response:** ❌
  - The model's data was trained from the Internet before 2023.
  - The requested information is not public.
- RAG idea:
  - Use the prompt to search for related passages in company internal documents (database).
  - Append the retrieved passages to the model input and ask again:
- **Prompt:**
  - NVIDIA was founded in [....] and had 167 employees. It grew until [...] and 2024 it reached 2,000 employees. How many employees did NVIDIA have in 2024?
- Model response:
  - 2,000.

Disclaimer, the numbers are invented for illustration purposes.

Summary: Types of Prompting

- Vanilla prompting
- Prompt tuning
- In-context few-shot learning
- Chain-of-Thought (CoT)
- Retrieval augmented generation (RAG)

Small quiz

- What models were introduced in the following papers?
  - “Improving Language Understanding by Generative Pre-Training”

    GPT-1
  - “Language Models are Unsupervised Multitask Learners”

    GPT-2
  - “Language Models are Few-Shot Learners”

    GPT-3
  - “Training language models to follow instructions with human feedback”

    InstructGPT

Small quiz

- What models were introduced in the following papers?
  - "Improving Language Understanding by **Generative Pre-Training**" GPT-1
  - "Language Models are Unsupervised **Multitask** Learners" GPT-2
  - "Language Models are **Few-Shot Learners**" GPT-3
  - "Training language models to **follow instructions** with **human feedback**" InstructGPT

4

Post-Training LLMs

- Instruction Tuning
- Preference tuning with human feedback via:
  - Direct Preference Optimization – DPO
  - Reinforcement Learning from Human Feedback – RLHF

Back to the problem that the next likely continuation **does not align** with what we want...

- We let the LLM complete the following sentence:

    When was Albert Einstein born?

Completions:

When was Heisenberg born?

This question will come up in the science history exam.

March 14, 1879

Prompting:

Question: When was Albert Einstein born? **Answer:**

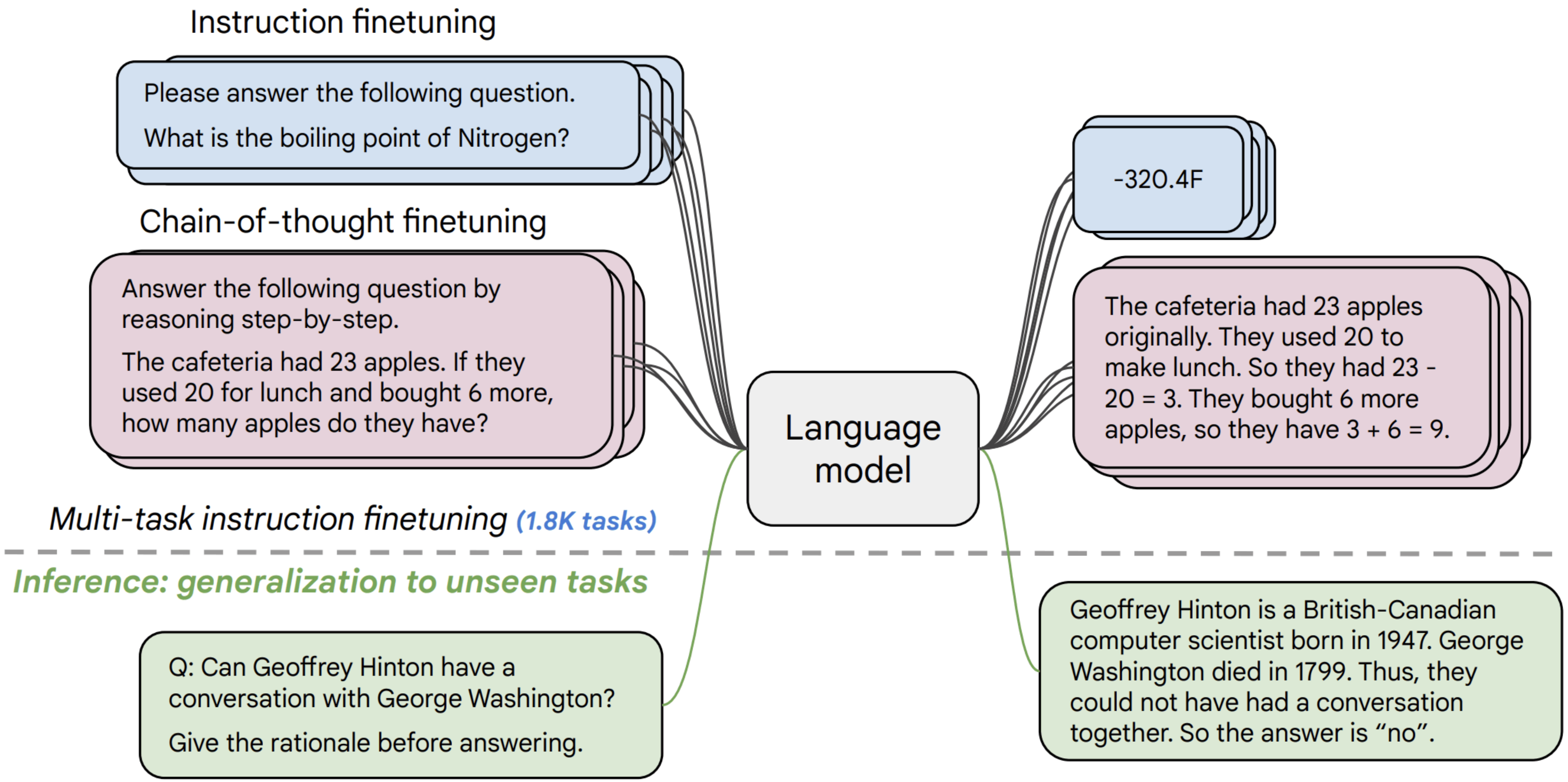
Let's train to follow these instruction prompts and call this (post)training: Instruction Tuning.

Supervised finetuning on instruction data (Instruction Tuning)

https://huggingface.co/blog/instruction-tuning-sd#:~:text=Instruction%2Dtuning%20is%20a%20supervised,Learners%20(FLAN)%20by%20Google.

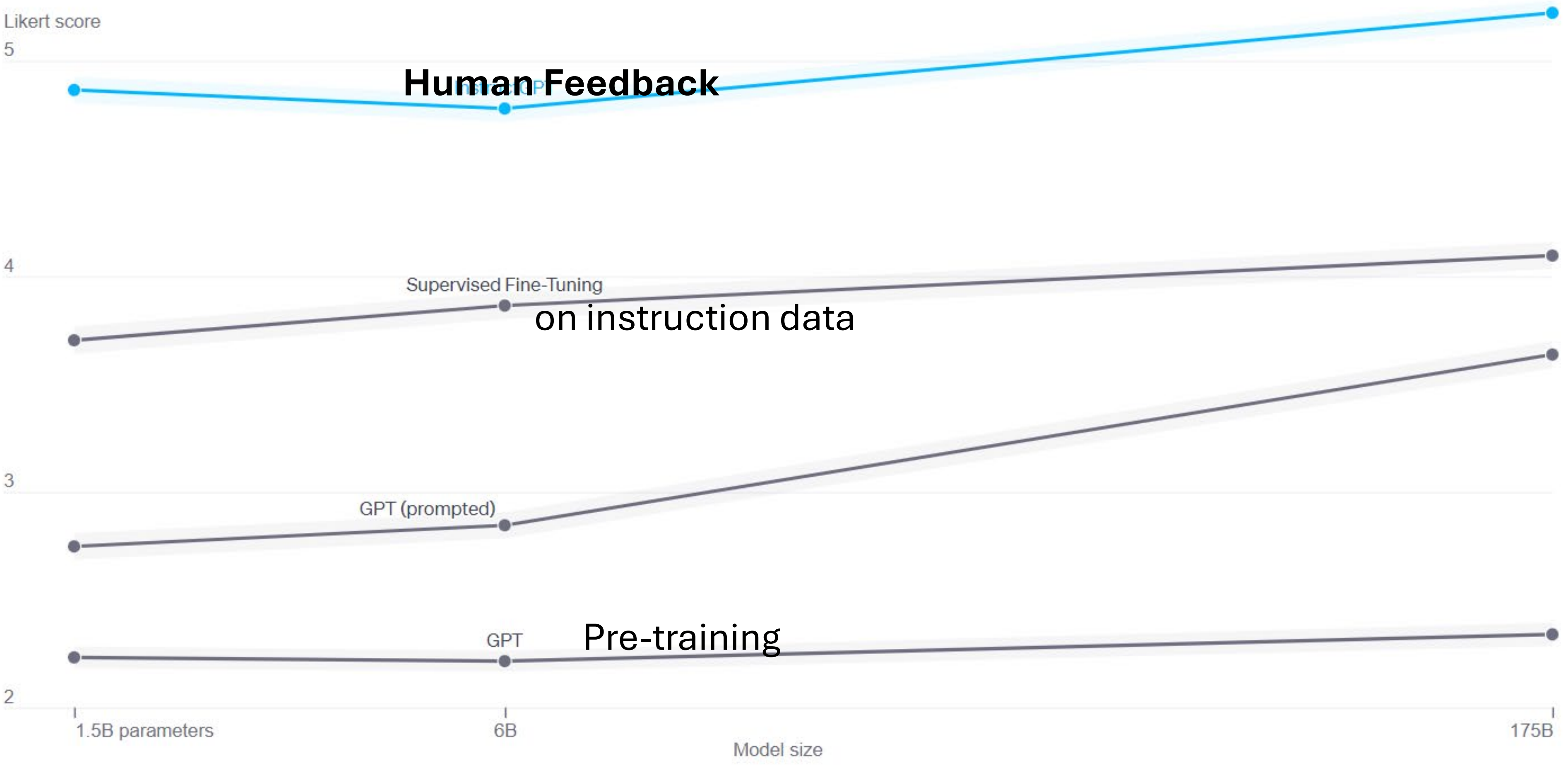
We **finetune (aka post-train)** the model on different NLP tasks described by instructions.

For example, for translating from EN to DE, we train (supervised CE loss): ***"Please translate the following sentence from English to German:** The cat sat on the mat. Die Katze saß auf der Matte."*



$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \sigma(\pi_\theta(y \mid x))\right]$$

The stages or post-training



https://openai.com/research/instruction-following

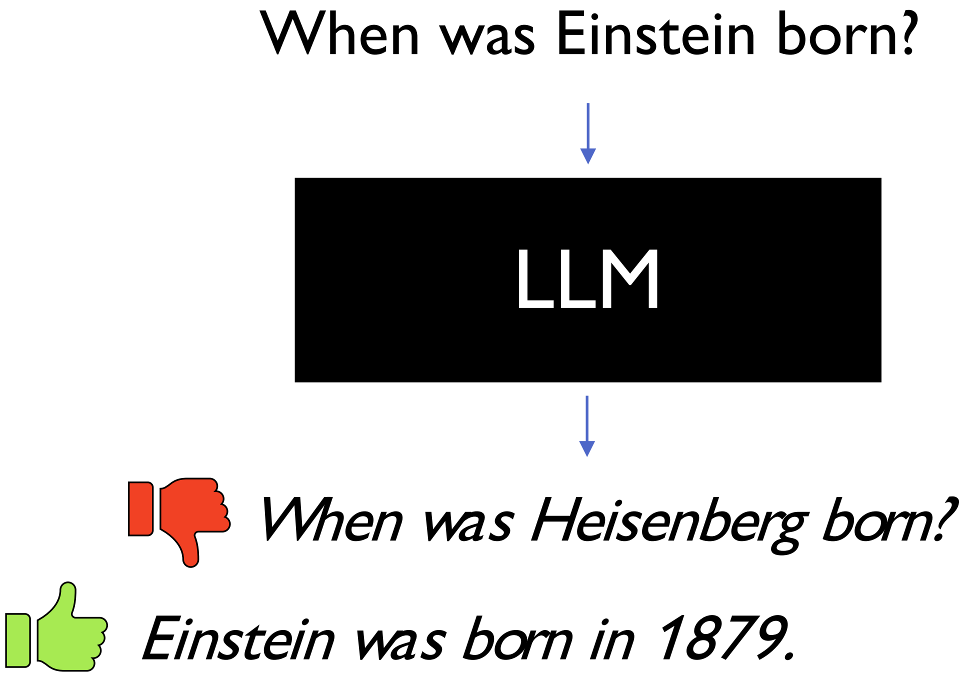
Training with Human Feedback

1. Collect a dataset composed of triplets:
   a) Question / task
   b) One possible answer, given by the pre-trained model or by a human
   c) Another possible answer
2. Let humans rate whether given a), they prefer b) or c).
3. Train with this human feedback signal, via:
   - **DPO – Direct Preference Optimization**
   - RLHF – Reinforcement Learning with Human Feedback (PPO)

https://arxiv.org/pdf/2305.18290

DPO

- Train the LLM (supervised) to increase the likelihood of positively rated outputs. Train to decrease the likelihood of producing negative examples.
- Regularise the LLM to stay close in predicted token probs to the LLM before finetuning (to avoid collapse).

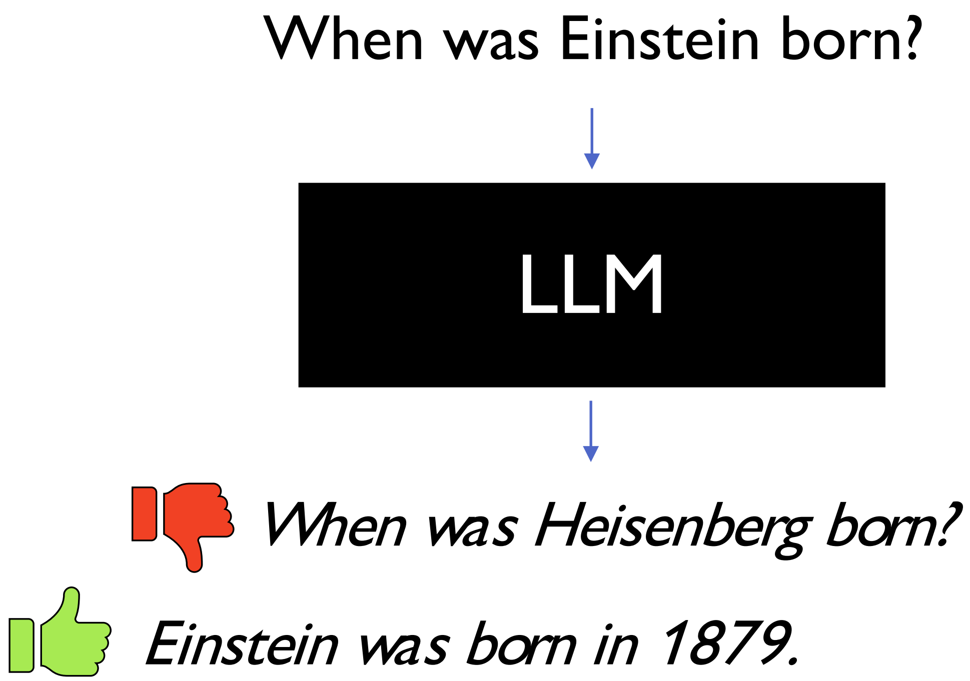


$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \sigma(\pi_\theta(y \mid x))\right]$$

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right].$$

RLHF

- Train the LLM to produce output that is getting high ratings.
- ❗ Problem:
  - The LLM's goal is to get high ratings, but by doing so, can produce any kind of completion (which is not in the training data).
  - To make these completions, the model chooses tokens via an argmax, which is nondifferentiable.
- Use Reinforcement Learning (RL), a framework that overcomes nondifferentiability.



5

Benchmarking LLMs

Emergent capabilities and how (not) to measure them.

The (ca. 1-year) cycle of madness

1. People design a “super-hard” benchmark.
2. OpenAI / Google / Meta fire up their GPU clusters to train a larger model on more (and cleaner) data than ever before.
3. 10^X dollars and months later: super-hard benchmark is solved.
4. Goto 1.

Meme and slide by Dr. Juri Opitz.

Example capability: Code translation

Link in the description below.👇

https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.

Prompt	Model Response

```
// Translate from C to Python
int add_one ( int x ){
 int m = 1;
 while ( x & m ) {
   x = x ^ m;
   m <<= 1;
 }
 x = x ^ m;
 return x; }
```

Explaining a joke

Prompt

```
I will explain these jokes:
(1) The problem with kleptomaniacs is that they always take things literally.
Explanation: This joke is wordplay. Someone who "takes things literally" is someone who doesn't fully understand social cues and context, which is a negative trait.
But the definition of kleptomania is someone who literally takes things.
(2) Always borrow money from a pessimist. They'll never expect it back.
Explanation: Most people expect you to pay them back when you borrow money, however a pessimist is someone who always assumes the worst, so if you borrow money from
them, they will expect that you won't pay them back anyways.
```

Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

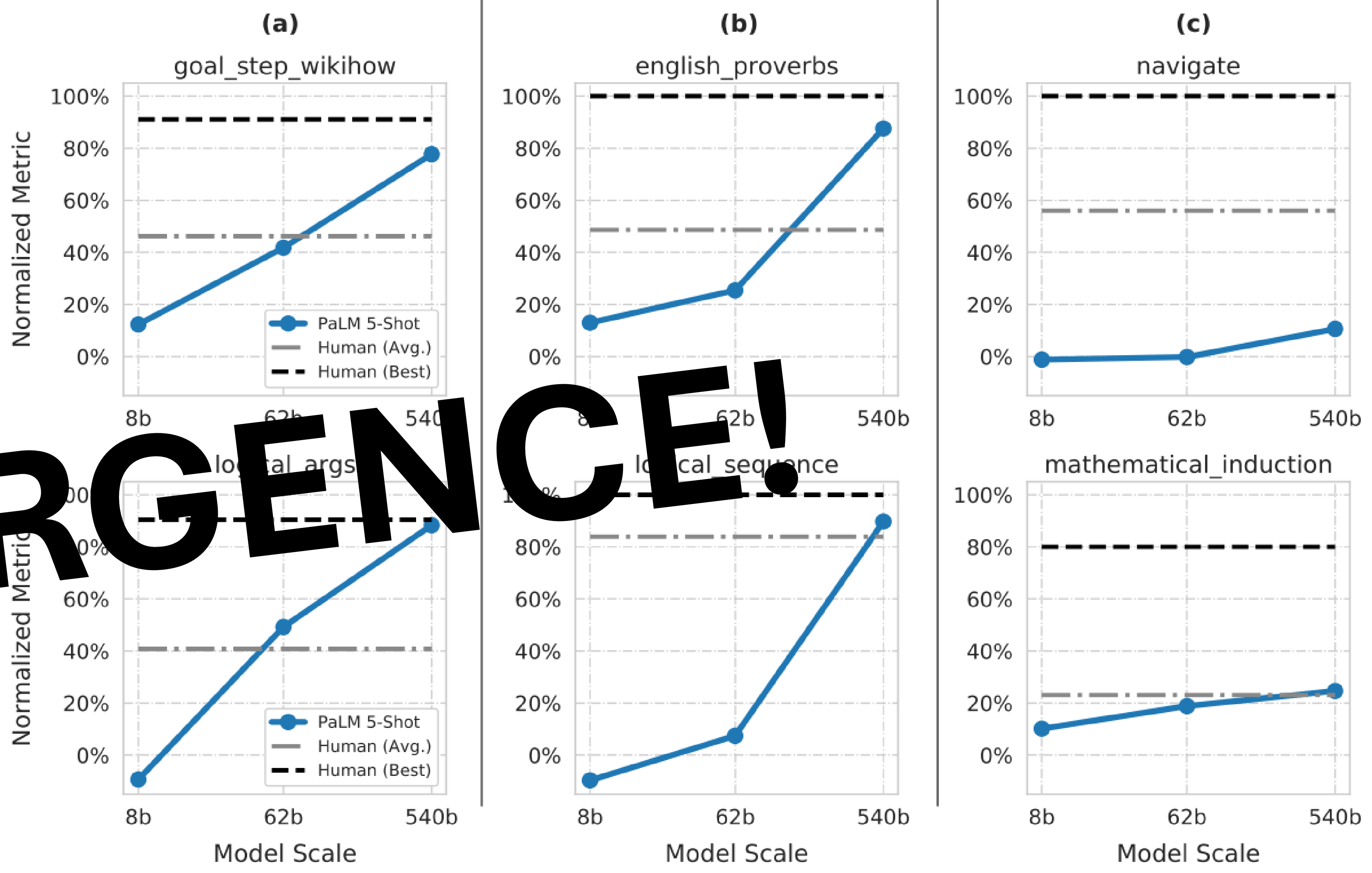
Model Response

Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

Emerging capabilities

Larger models with the same architecture and training data can solve tasks that smaller models are uncapable of solving.

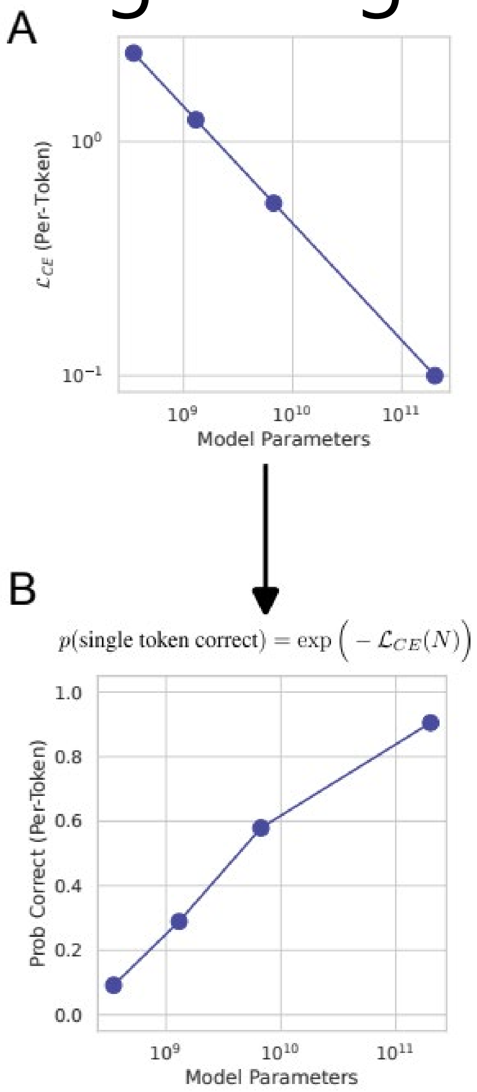
For some tasks, there are jumps in the plots.



EMERGENCE!

Figure 5: 5-shot evaluations on six individual tasks with interesting scaling characteristics. For each task, the

Are Emergent Abilities of Large Language Models a Mirage?



- https://arxiv.org/abs/2304.15004

Are Emergent Abilities of Large Language Models a Mirage?

- https://arxiv.org/abs/2304.15004

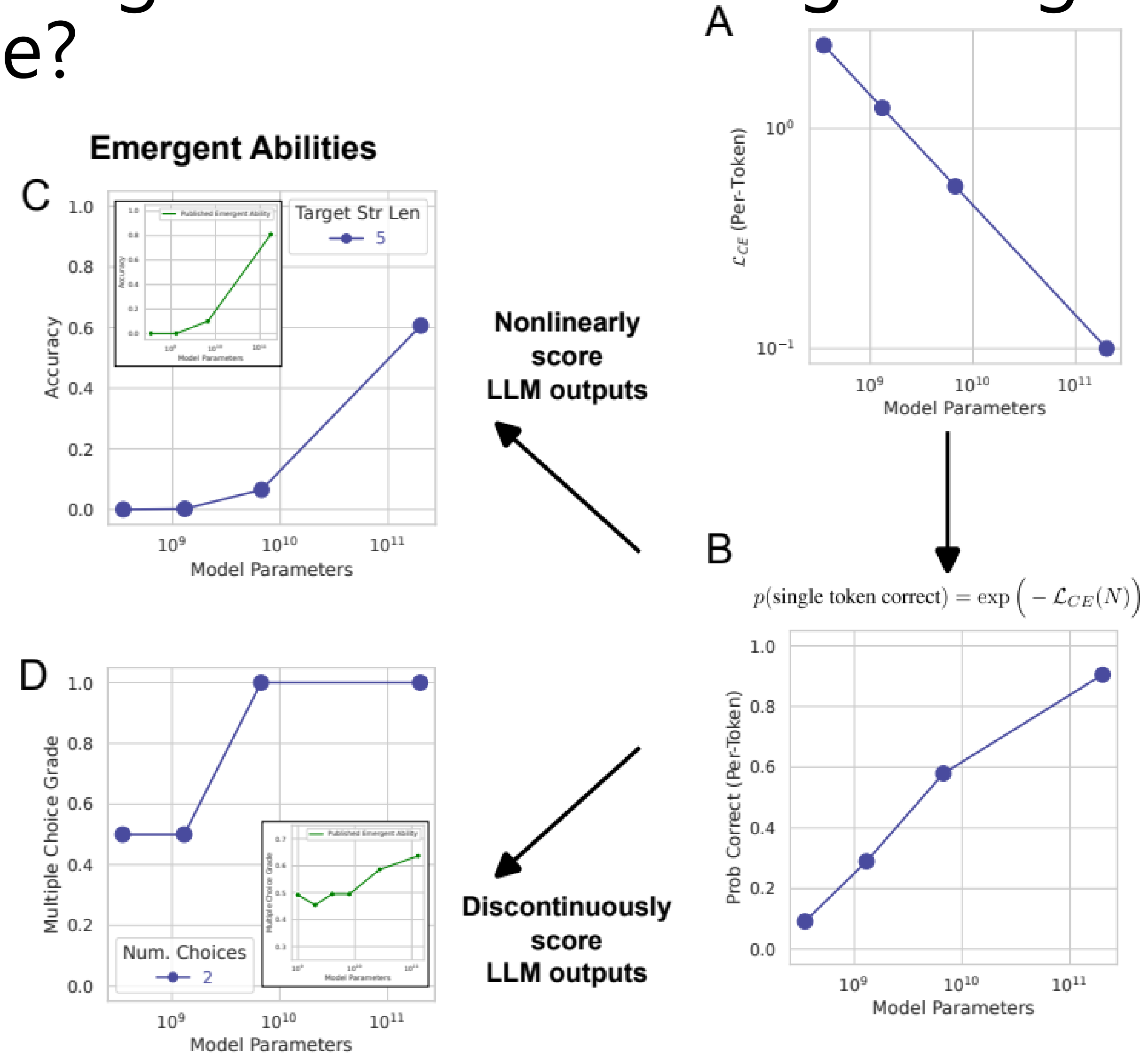


Figure 2: **Emergent abilities of large language models are metrics, not unpredictable changes in model behavior wi**

Are Emergent Abilities of Large Language Models a Mirage?

- https://arxiv.org/abs/2304.15004

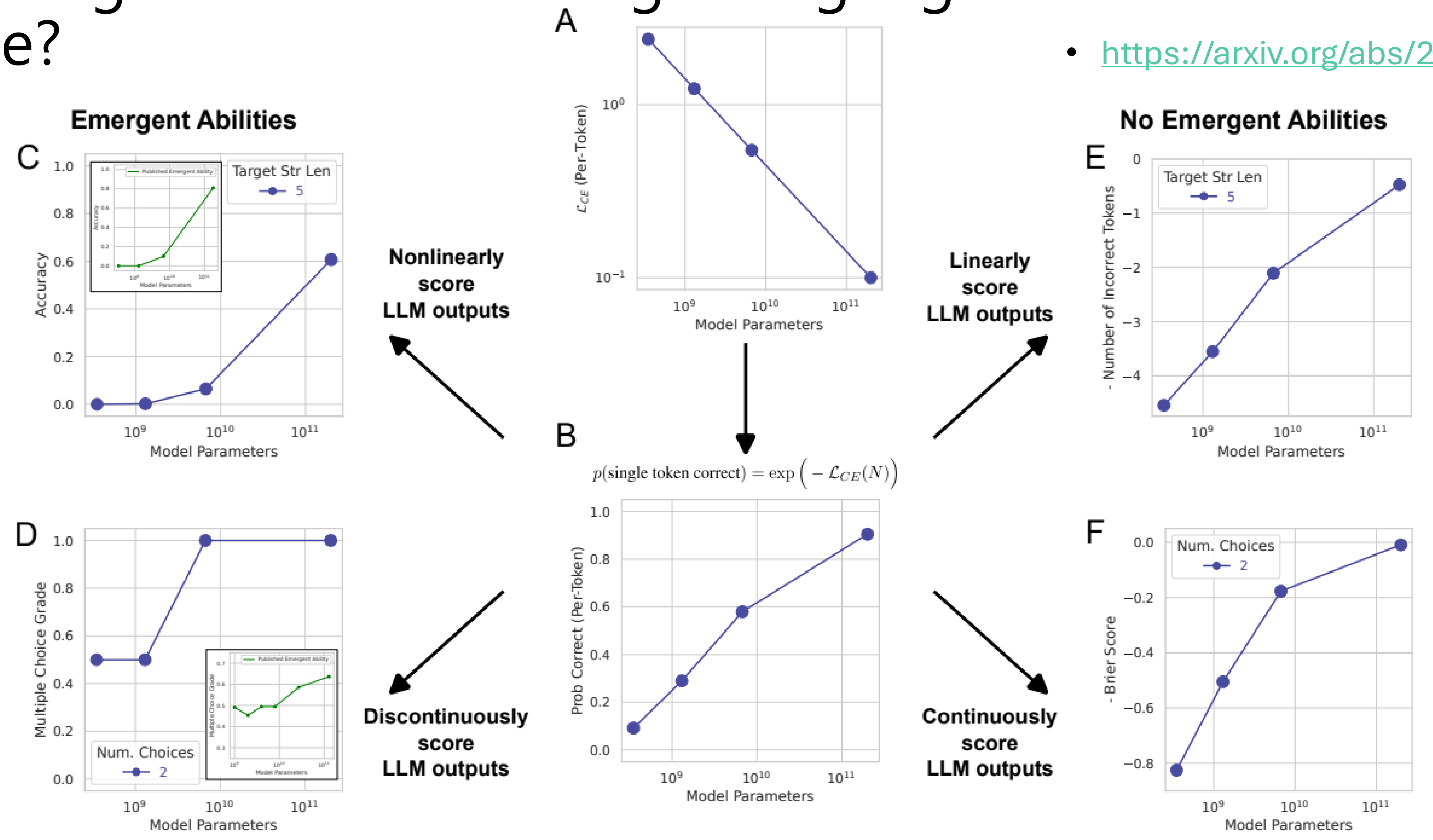
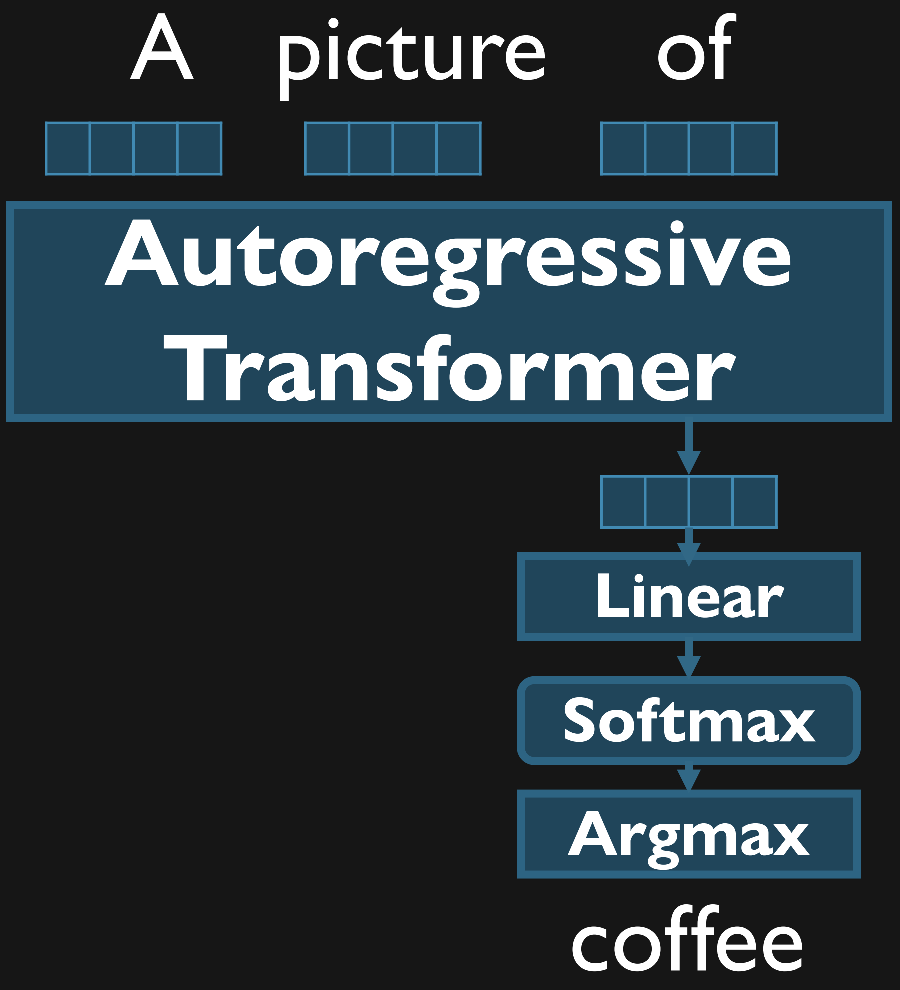
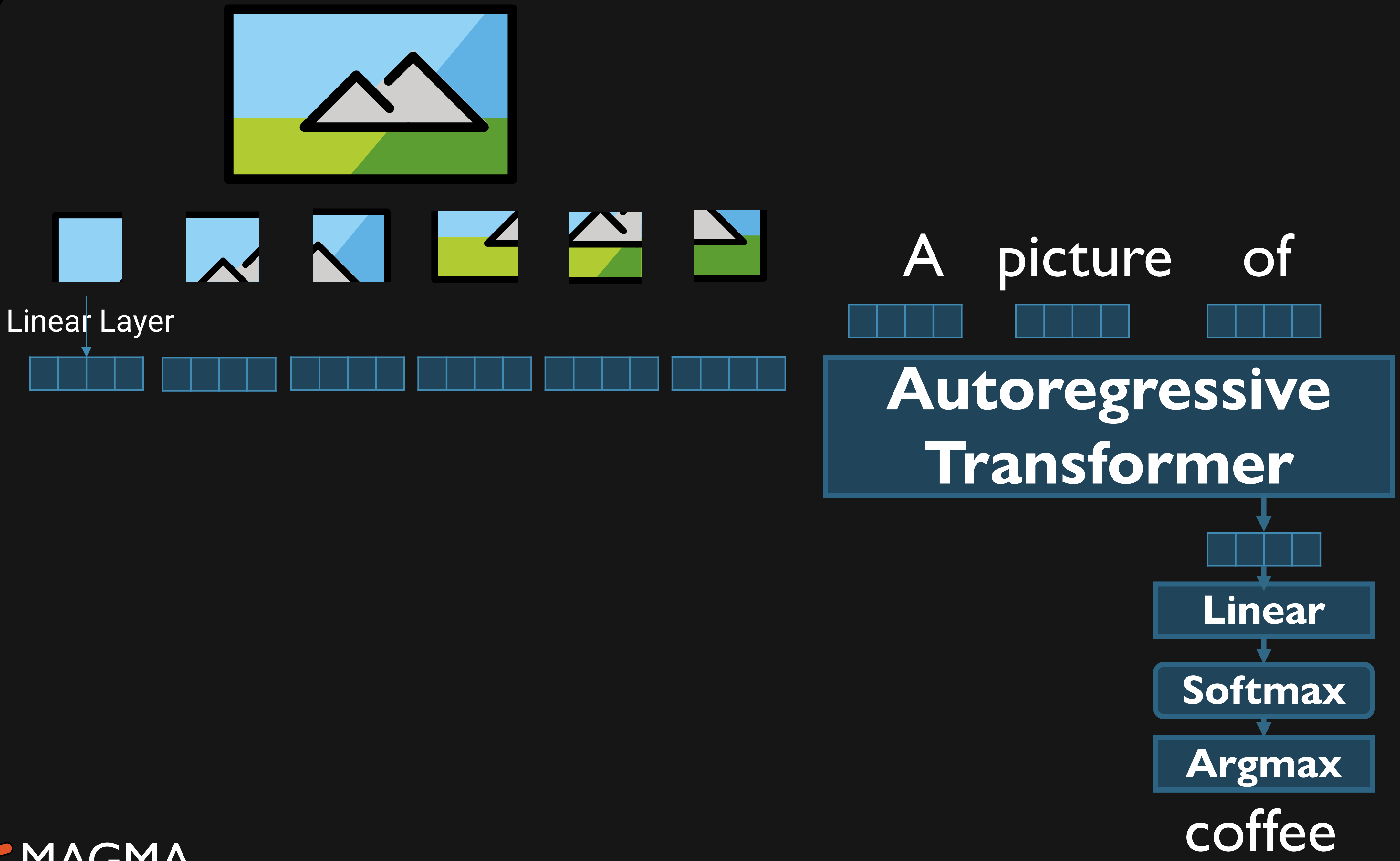


Figure 2: **Emergent abilities of large language models are created by the researcher's chosen metrics, not unpredictable changes in model behavior with scale.** (A) Suppose the per-token

6

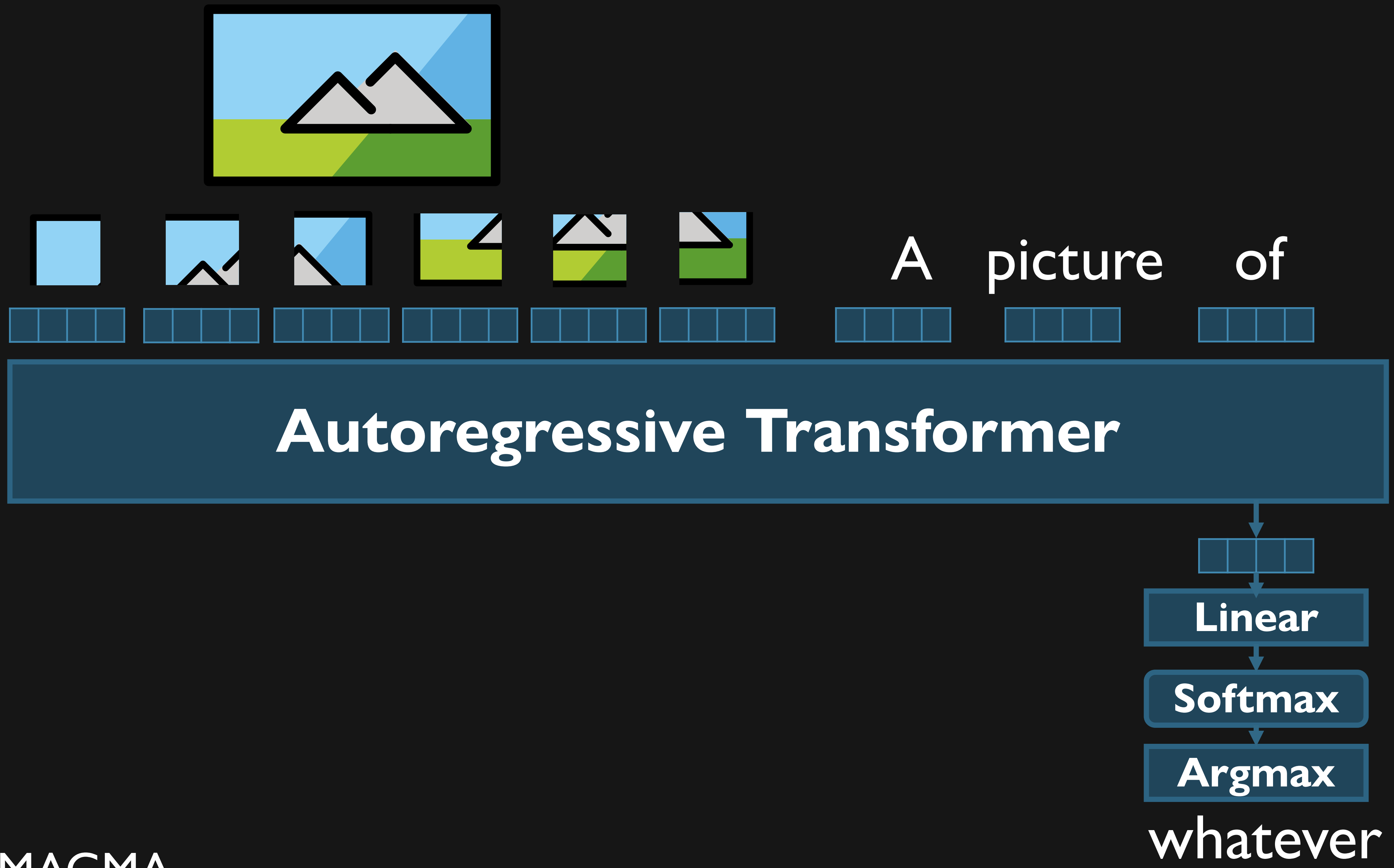
Multimodal Extensions of LLMs


A
picture
of
Autoregressive Transformer
Linear
Softmax
Argmax
coffee

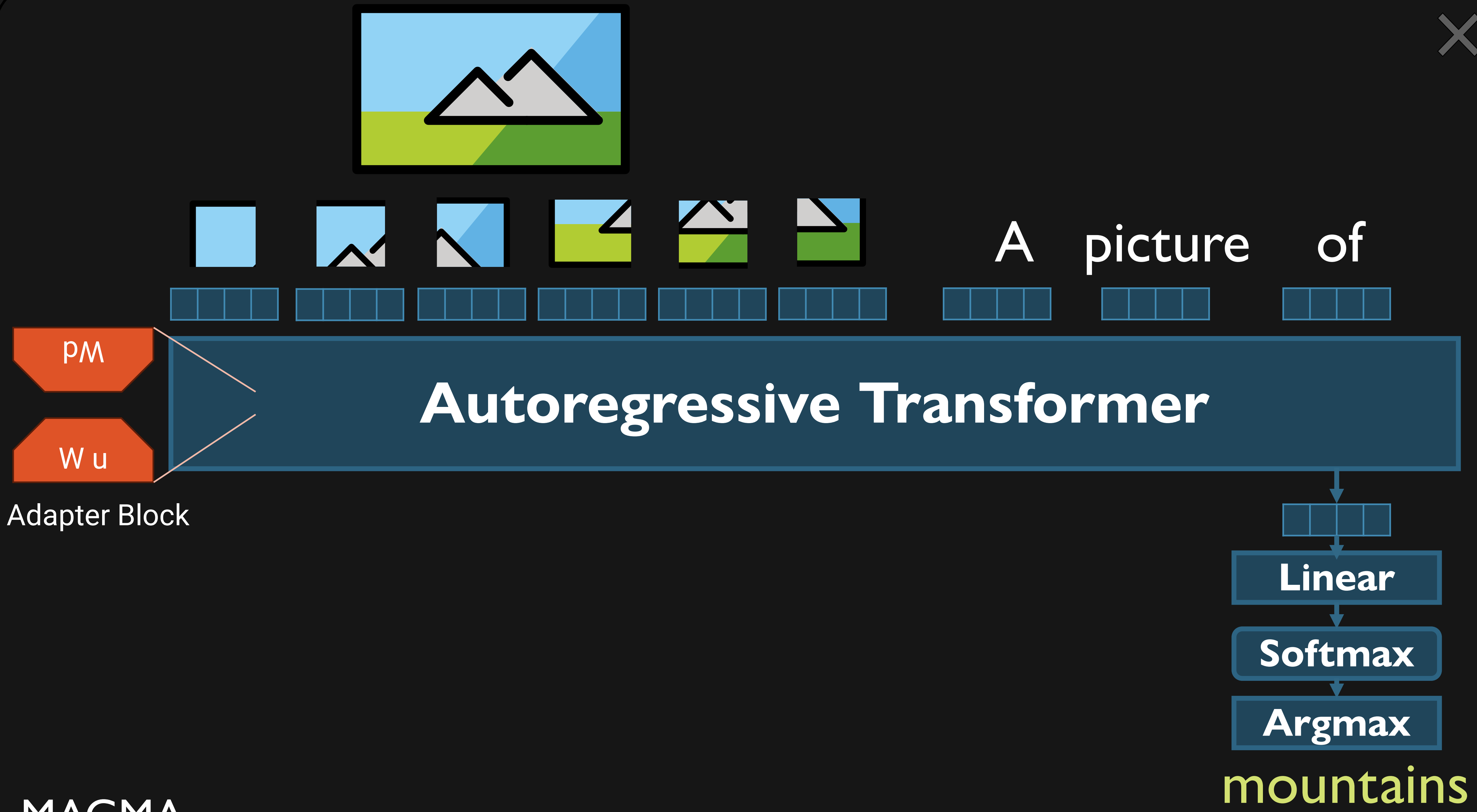


MAGMA

C Eichenberg, S Black, S Weinbach, L Parcalabescu, A Frank, Findings of EMNLP 2022



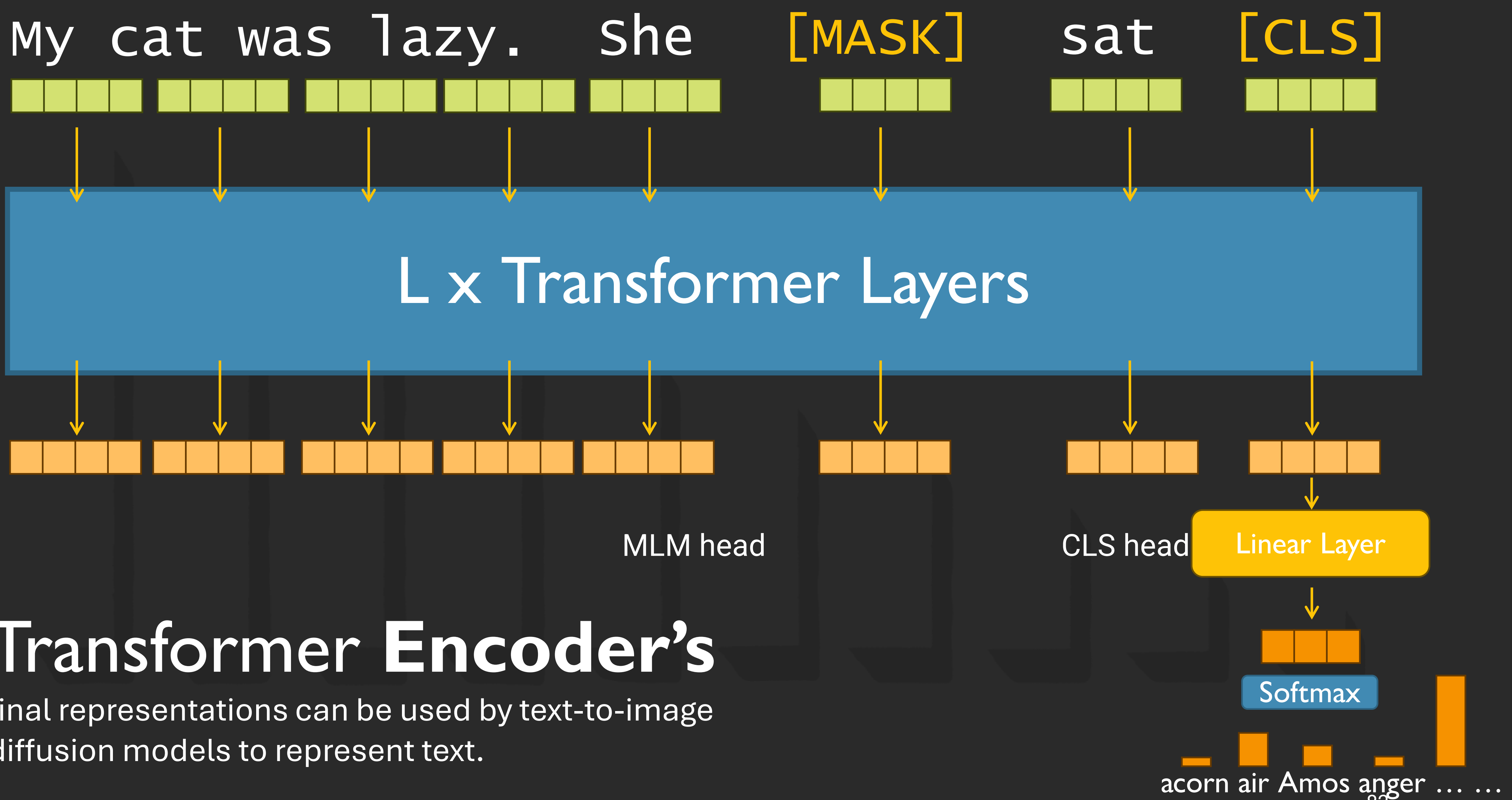
MAGMA C Eichenberg, S Black, S Weinbach, L Parcalabescu, A Frank, Findings of EMNLP 2022



MAGMA

C Eichenberg, S Black, S Weinbach, L Parcalabescu, A Frank, Findings of EMNLP 2022


Autoregressive
Transformer
Linear
Softmax
Argmax
mountains



Transformer **Encoder's**

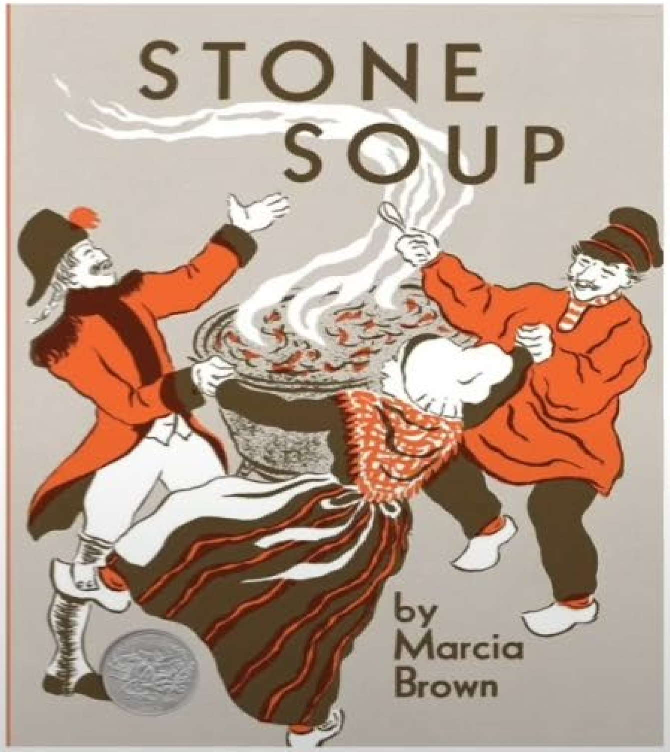
final representations can be used by text-to-image diffusion models to represent text.

Yes, models are cool.

But their true power comes from the data.

https://simons.berkeley.edu/news/stone-soup-ai https://youtu.be/K0gavpe0uDo?si=FdsURxkV39fyU1NV

Gopnik's Parable of Stone Soup AI



https://simons.berkeley.edu/news/stone-soup-ai https://youtu.be/K0gavpe0uDo?si=FdsURxkV39fyU1NV

Gopnik's Parable of Stone Soup AI

- Tech execs come to village:
  - *"We will make amazing AI with just a few magical algorithms (e.g. gradient descent, transformers, next token prediction)"*
  - *…but would be so much better with more data*
  - *…even more magical with human feedback*
  - *…even better if humans would learn to ask questions in the right way*
- Villagers: wow, what magical AI just from a few algorithms!