CS-461

# Foundation Models and Generative AI

## **Generative Models III**:
Recap on Generative Models and Generalizations

**Charlotte Bunne**, Fall Semester 2025/26

# Announcements

- Half of the lecture next week is a guest lecture!

Imanol Schlag
ETH Zurich

**Language Foundation Model:**
From GPT to Apertus

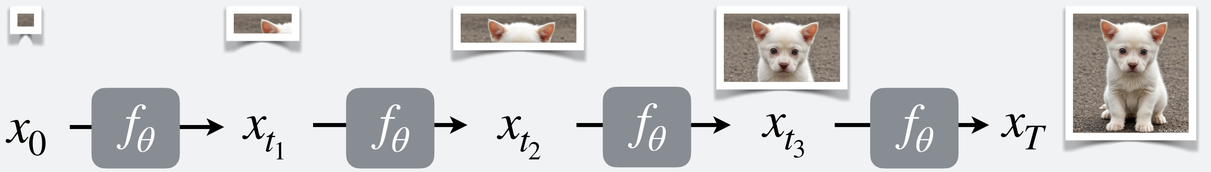
- Traveling for Lecture 13 and 14:
  - For **Lecture 13**: Guest Lecture on Zoom from **2 to 3 pm**!
  - For **Lecture 14**: PostDoc Linus Bleistein will give the class.

- **Assignment 1** is due **next Wednesday at 23:59 pm**!

# Four Philosophical Approaches

## 1. Autoregressive Models

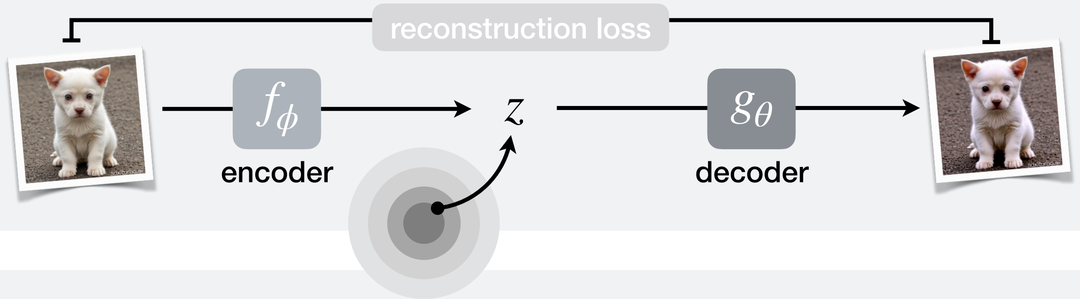
*"One step at a time."*




## 2. Autoencoders

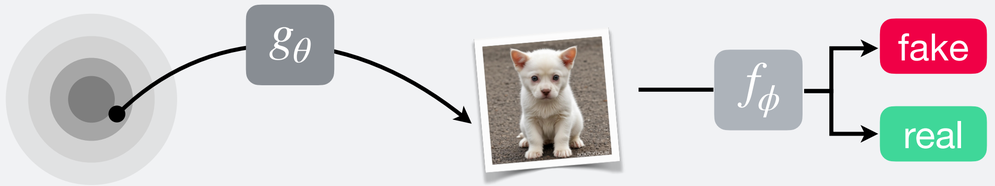
*"Reduce to the essence and rebuild."*

*Variational Autoencoder*



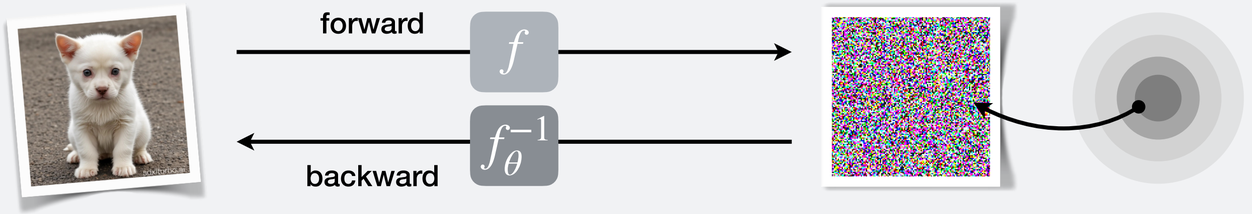

## 3. Generative Adversarial Models

*"Fake it till you make it."*



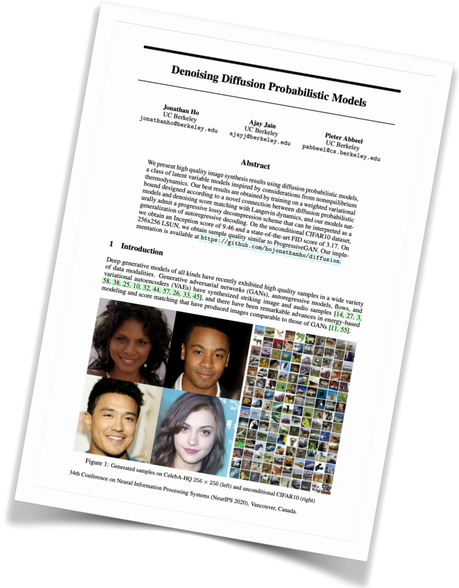

## 4. Diffusion Models

*"Mess it up, then* `Ctrl+Z`*."*



# This Week's Papers

Papers are linked in Moodle.




Ho et al., "Denoising Diffusion Probabilistic Models." Advances in Neural Information Processing Systems 33 (2020).


Song et al., "Score-Based Generative Modeling through Stochastic Differential Equations." International Conference on Learning Representations (ICLR) (2021).


Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022).

# Week 5's Exercise Sheet

Exercise 4 · Task 1

**Denoising Diffusion Probabilistic Models:**

Derives the DDPM training objective from the ELBO, showing how Gaussian forward/posterior kernels lead to a simple noise-prediction MSE, the epsilon-parameterizations, sampling updates, and why uniform ("simple") loss weighting improves low-SNR steps.

Exercise 4 · Task 2

**From Forward Diffusion to Denoising Score Matching:**

Takes the small-step noising limit to a continuous SDE, states the reverse-time SDE with the (intractable) marginal score, then replaces it with the tractable conditional score to get the denoising score matching (DSM) objective, and contrasts SDE with discrete DDPM training.

# Week 5's Code Demonstration

Code Notebook 4 · Task 1

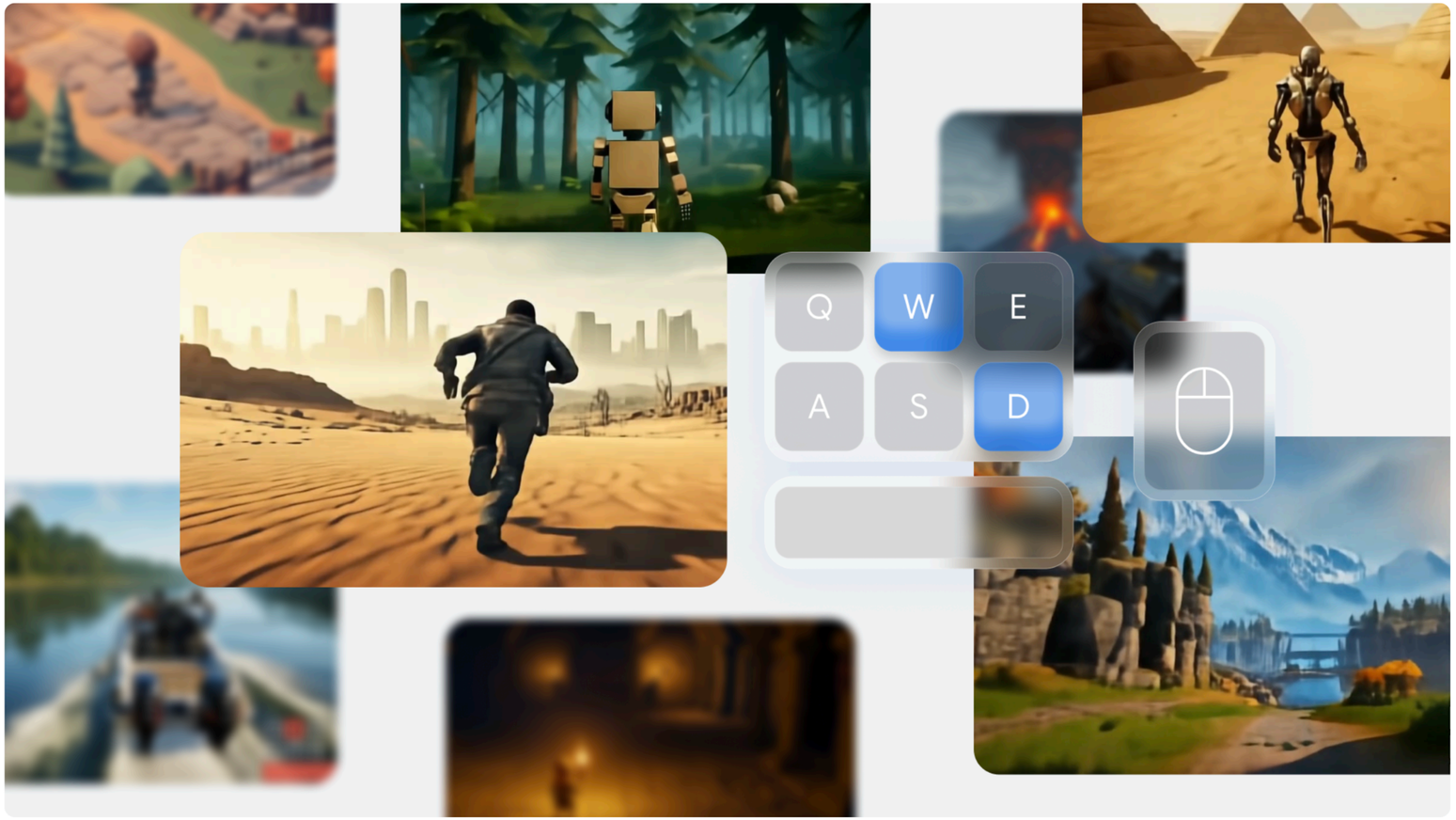
**Implementing DDPMs from Scratch:** Build and train a noise-prediction DDPM on 2-D moons (with and without class conditioning); visualize forward/backward processes and sampling quality.
→ Jupyter notebook exercise

Code Notebook 4 · Task 2

**Stable Diffusion & CFG with *diffusers*:** Recreate the SD pipeline (VAE, UNet, text encoder, scheduler), then run classifier-free guidance and study how the guidance scale $s$ steers generations.
→ Jupyter notebook exercise

# Generative AI and Foundation Models

## **Toward** (Generative) **World Models**




e.g., **DeepMinds's Genie 2 or 3**

Proto-engine for generating video games *on-the-fly*.

**Lecture 13: FM in Robotics**

**Lecture 12: World Models**

CS-461

# Foundation Models and Generative AI

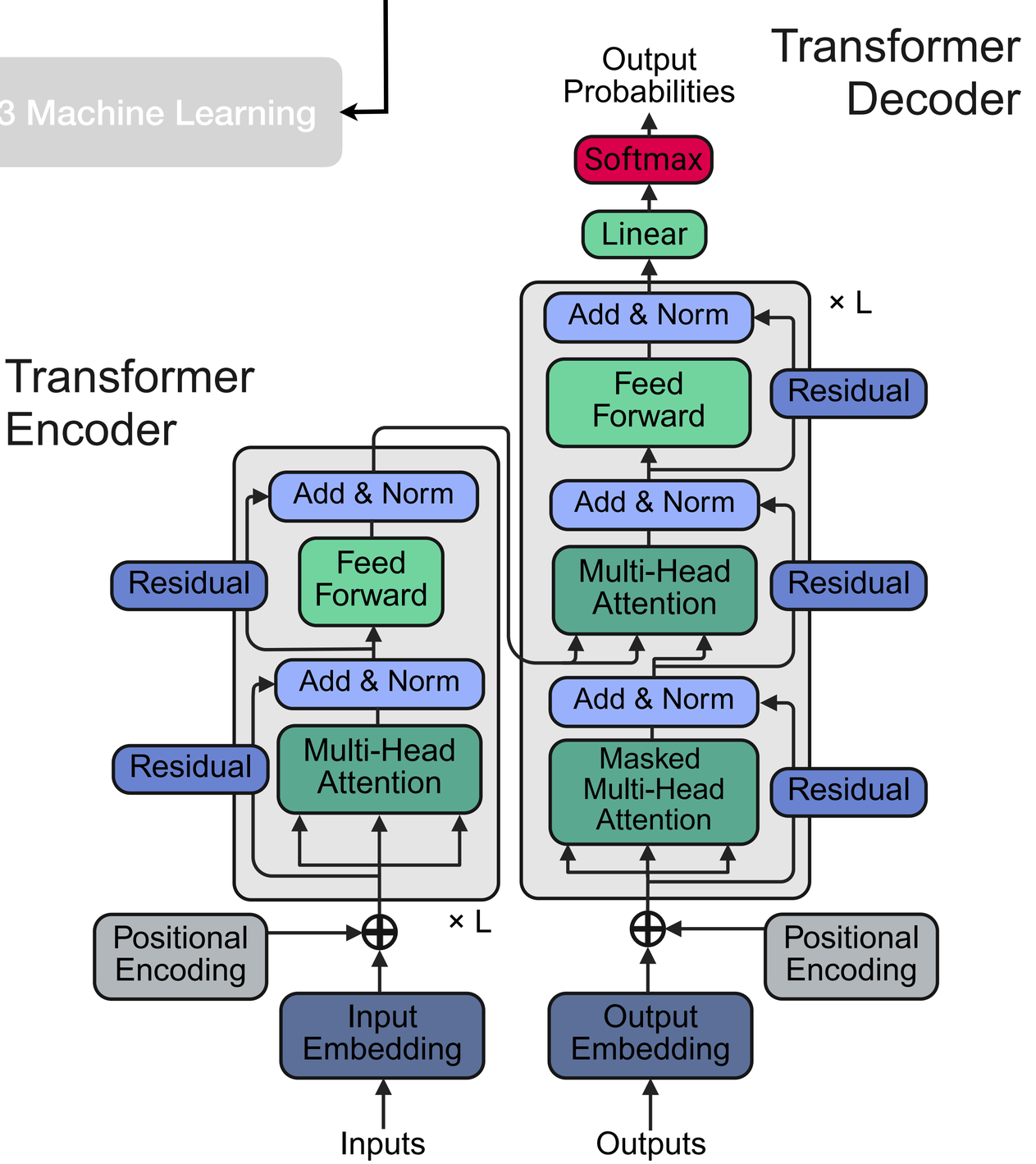
## Tokenization and Building Blocks of Foundation Models

**Charlotte Bunne**, Fall Semester 2025/26

# Building Blocks & Components

CS-433 Machine Learning

1. Tokenization
2. Positional / Structural Encodings
3. Linear Projections
4. Context Mixing Operator
5. Nonlinearities
6. Normalization & Stabilizers
7. Residual Connections



# Building Blocks & Components

## 1 Tokenization

`raw input → token`

Segment raw input into tokens suitable for the model.

**Transformers:** subwords, image patches, etc.

## 2 Positional / Structural Encodings

`pos/structure → encoding`

Inject order or topology information.

**Transformers:** sinusoidal, learned, rotational, etc.
**CNNs an RNNs:** spatial/sequence order implicit.

## 3 Linear Projections

`y = Wx + b`

Apply dense projections to transform hidden states.

**Transformers:** Q/K/V/O projections, FFN expansion/projection.

## 4 Context Mixing Operator

`context-aware interactions`

Exchange information across tokens.

**Transformers:** multi-head self-attention, cross-attention.
**CNNs:** convolution (local receptive fields).
**RNNs:** recurrence/state updates.

## 5 Nonlinearities

`σ(x)`

Introduce non-linearity and gating.

**Transformers:** GeLU, SiLU, etc.
**Other architectures:** ReLU, etc.

## 6 Normalization & Stabilizers

`normalize(x)`

Stabilize training and enable deep stacks.

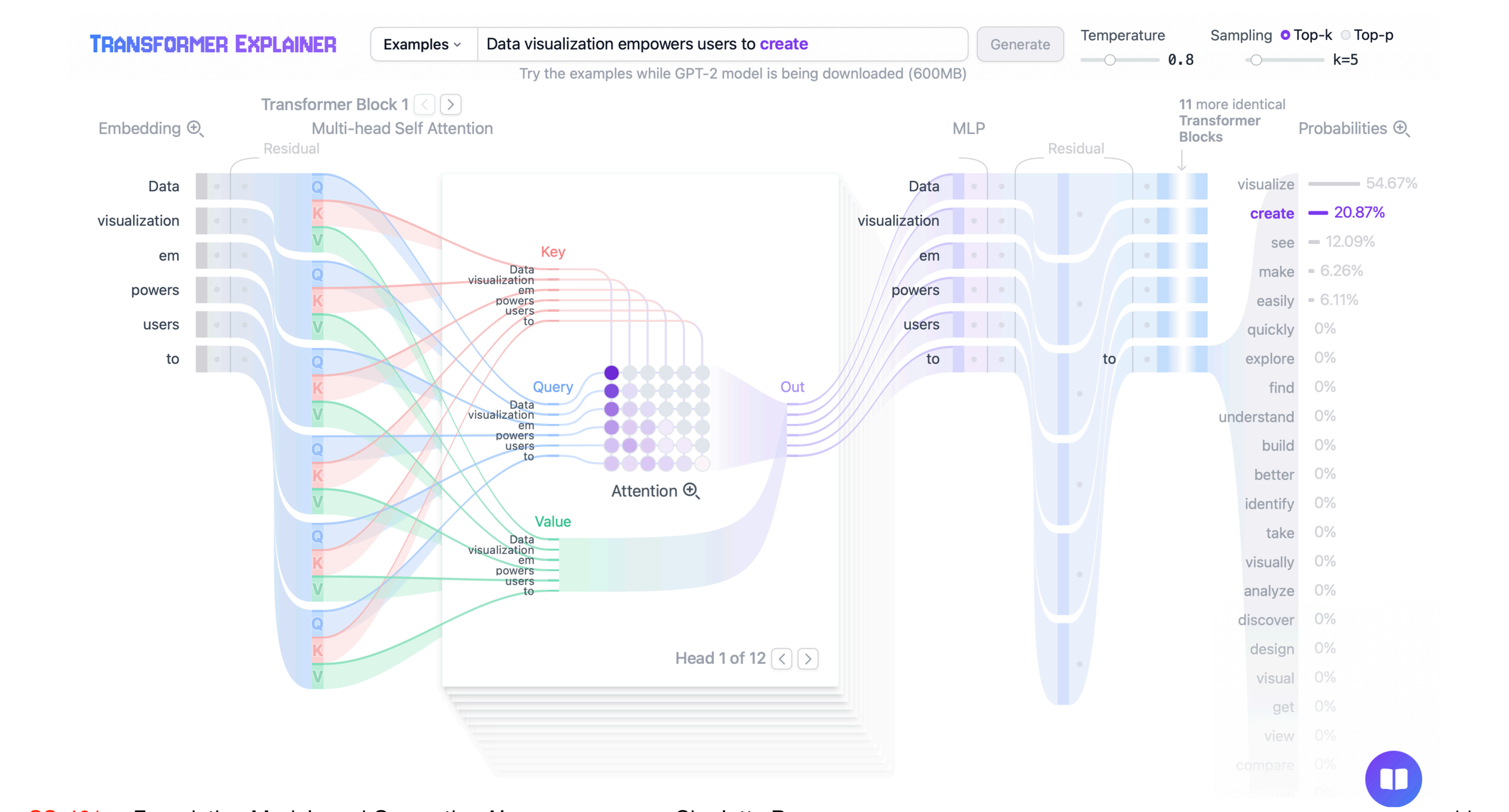
**Transformers:** LayerNorm, RMSNorm, dropout, residual scaling.
**CNNs:** BatchNorm, GroupNorm.

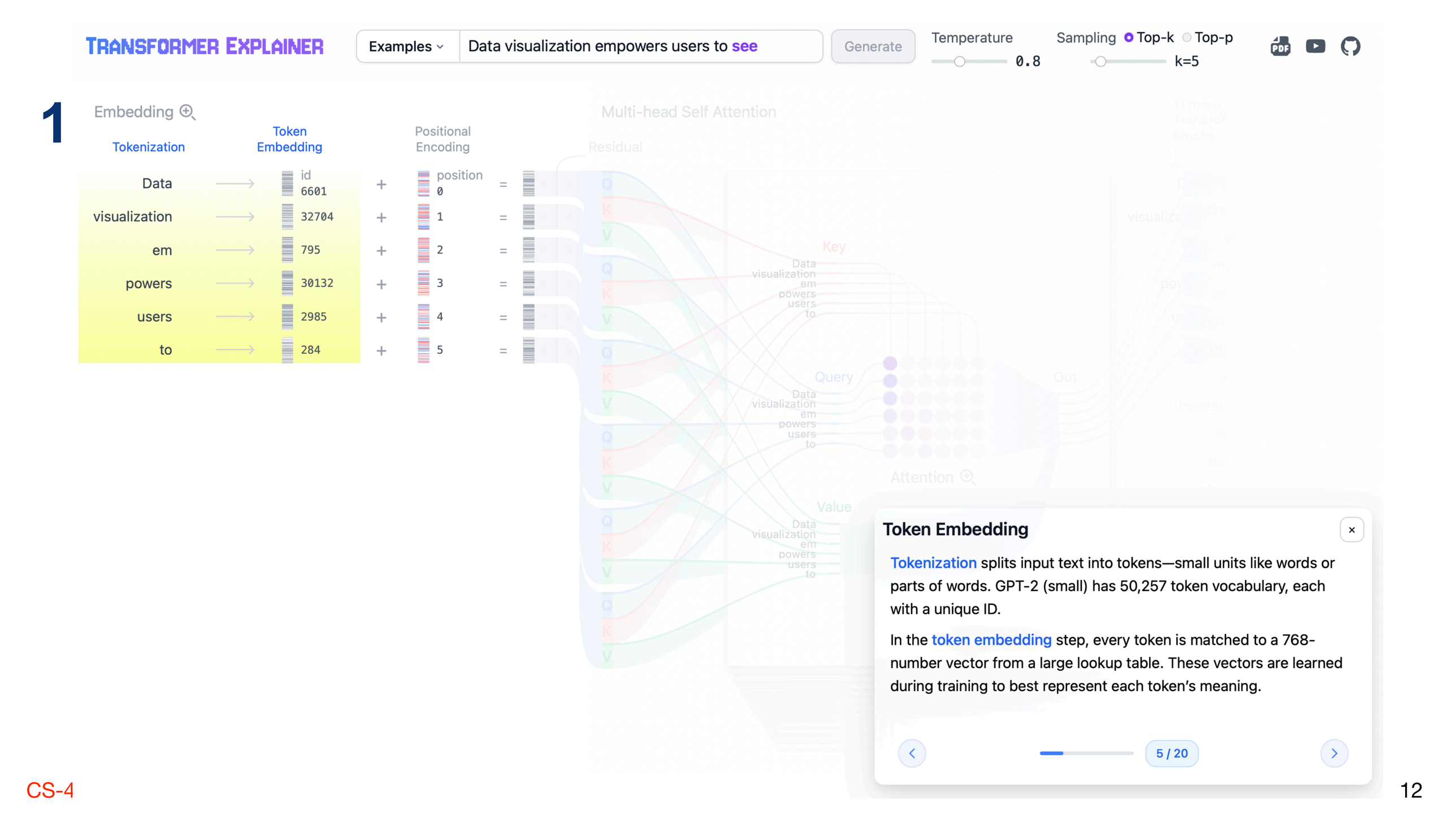
## 7 Residual Connections

`y = x + F(x)`

Provide gradient highways and enable very deep networks.

**Transformers:** residuals everywhere.
**CNNs:** ResNet-style skips.


TRANSFORMER EXPLAINER
Examples
Data visualization empowers users to create
Generate
Temperature
0.8
Sampling
Top-k
Top-p
k=5
Try the examples while GPT-2 model is being downloaded (600MB)
Transformer Block 1
Embedding
Multi-head Self Attention
Residual
MLP
Residual
11 more identical Transformer Blocks
Probabilities
Data
visualization
em
powers
users
to
Q
K
V
Key
Query
Value
Attention
Out
Head 1 of 12
visualize 54.67%
create 20.87%
see 12.09%
make 6.26%
easily 6.11%
quickly 0%
explore 0%
find 0%
understand 0%
build 0%
better 0%
identify 0%
take 0%
visually 0%
analyze 0%
discover 0%
design 0%
visual 0%
get 0%
view 0%
compare 0%


TRANSFORMER EXPLAINER
Examples
Data visualization empowers users to see
Generate
Temperature
0.8
Sampling
Top-k
Top-p
k=5
1
Embedding
Tokenization
Token Embedding
Positional Encoding
Data
visualization
em
powers
users
to
id
6601
32704
795
30132
2985
284
position
0
1
2
3
4
5
Multi-head Self Attention
Residual
Key
Query
Value
Attention
Out
Token Embedding
Tokenization splits input text into tokens—small units like words or parts of words. GPT-2 (small) has 50,257 token vocabulary, each with a unique ID.
In the token embedding step, every token is matched to a 768-number vector from a large lookup table. These vectors are learned during training to best represent each token's meaning.
5 / 20

# What is a Token?



**Token**

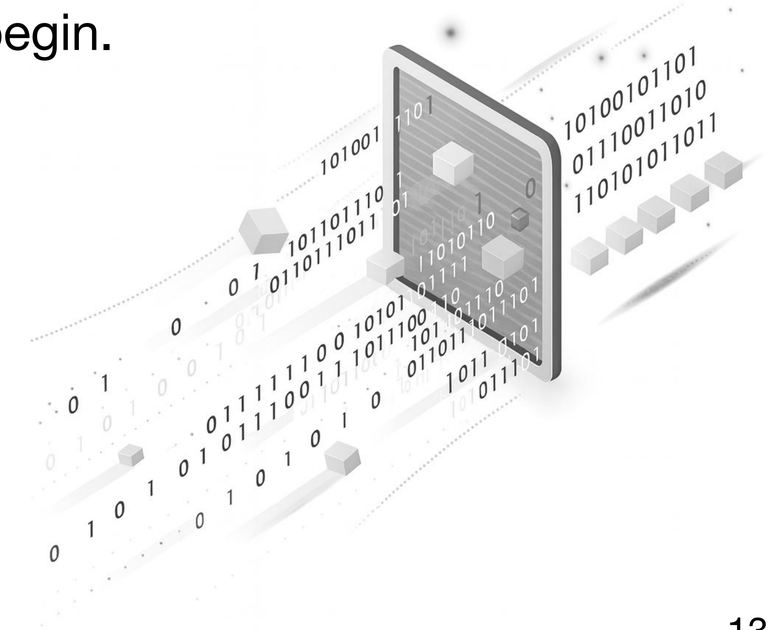
A token in AI is a unit of information that serves as the *basic processable element* in a system.

200’000 pixels

**Why can’t neural networks just process raw data directly?**

- Raw inputs are *extremely high-dimensional* → massive parameter count, poor generalization.
- Input needs to be number-based and consistent.

→ Neural nets need structure and compression before reasoning can begin.



**Tokenization provides this structure!**

Tokens group raw signals into meaningful units and

act as the interface between raw data and learned representations.

# Tokenization Concepts

1

We define a tokenization function $\tau : X \to Z$, where $X$ is our input space (e.g., text, images, etc.) and $Z$ is our token space ($\mathbb{Z}$ for discrete, $\mathbb{R}^d$ for continuous).

1. **Pre-Defined Tokenization**, i.e., fixed rules that define tokenization.

   e.g.,

   for **text** with vocabulary $V$: $\tau : \texttt{string} \to \{1,2,\ldots,|V|\}^n$,

   for **images** with patch size $p$: $\tau : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{N\times(p^2\cdot C)}$, $N = \frac{HW}{p^2}$ denotes the number of patches.

2. **Learned Tokenization**, i.e., the model discovers its own units.

   e.g., VQ-VAE codes, adaptive patches, tokenizer-free models, etc.

   **Key Idea**: From hand-crafted discretization → to self-discovered abstractions.

# Requirements on Tokenization Functions



Tokenization as a bijective function (for lossless tokenization):

$$\text{Encode: } \tau : \Sigma^* \to V^*$$
$$\text{Decode: } \tau^{-1} : V^* \to \Sigma^*$$

where $\Sigma$ is our original alphabet and $V$ is our vocabulary.

For lossy tokenization (like image patches), we have:

$$\text{Encode: } \tau : X \to Z$$
$$\text{Decode: } \tau^{-1} : Z \to X$$

where $\hat{X} \approx X$ but information is lost.

**Key Properties We Want:**

**Stability**:
Small changes in input → small changes in tokenization.

**Efficiency**:
Average token sequence length is minimized.

**Universality**:
Can handle any and even new inputs.

**Learnability**:
Token boundaries align with semantic units.

# Role of Tokens in Foundation Models



Tokenization is not a new concept, the term goes back to **compiler design** in the **1960s**, and to i**nformation retrieval in the 1970s**.

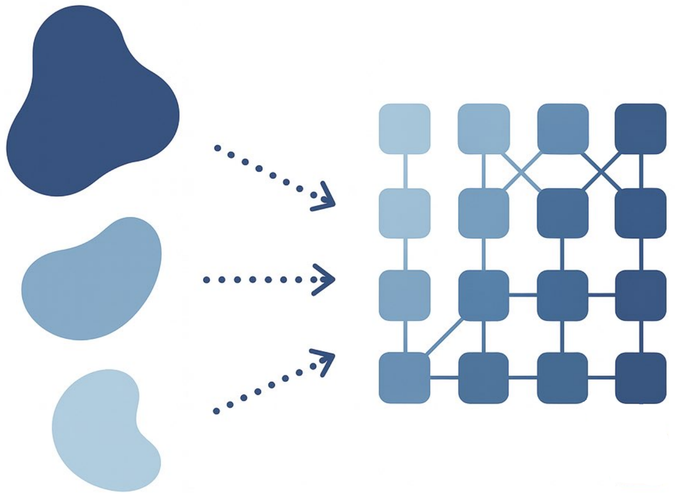
In early deep learning, tokenization was an *engineering step* to make data compatible with models.

*Dogma Shift!*

In foundation models, tokenization became **part of the model's representation design**.

It no longer only prepares data: it defines how different modalities can be expressed and *connected*.

〉 By translating all modalities into a shared set of tokens, models learn a common discrete language of the world
→ a prerequisite for multimodal alignment and joint reasoning.



Lecture 8: Multimodality

# Language Tokenizers



e.g., “don’t”

Traditionally, operated over **words**

… but you have ambiguity of the word boundaries,

… it is not typically applicable to languages without spaces, and

… it treats different forms of the same word as separate types.

e.g., “talk” “talks” “talked” “talking”

*How to deal with out-of-vocabulary words?*

<UNK>

We can operate over **characters**

… small vocabulary and solves the out-of-vocabulary problem.

*But, the length of the input increases rapidly!*

Instead, we can operate over **subwords**

… with more meaningful individual tokens,

… reduced impact of the out-of-vocabulary problem, and

… a manageable vocabulary size.

*A solution between word- and character-based tokenization!*

CS-552 Modern NLP ←

# **Language Tokenizers:** Byte Pair Encoding



**1990**: Data compression algorithm.

**2015**: Tokenizer for neural machine translation.

**How?**

1. Start with characters as the base vocabulary.
2. **Iteratively merge** the most frequent character pairs.
3. **Stop** after a predetermined number of merges.

*Always merge greedily based on raw frequency*

Neural Machine Translation of Rare Words with Subword Units

Rico Sennrich and Barry Haddow and Alexandra Birch
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk,bhaddow@inf.ed.ac.uk

Abstract

Neural machine translation (NMT) models typically operate with a fixed vocabulary, but translation is an open-vocabulary problem. Previous work addresses the translation of out-of-vocabulary words by backing off to a dictionary. In this paper, we introduce a simpler and more effective approach, making the NMT model capable of open-vocabulary translation by encoding rare and unknown words as sequences of subword units. This is based on the intuition that various word classes are translatable via smaller units than words, for instance names (via character copying or transliteration), compounds (via compositional translation), and cognates and loanwords (via phonological and morphological transformation).

lem, and especially for languages with productive word formation processes such as agglutination and compounding, translation models require mechanisms that go below the word level. As an example, consider compounds such as the German *Abwasser|behandlungs|anlange* 'sewage water treatment plant', for which a segmented, variable-length representation is intuitively more appealing than encoding the word as a fixed-length vector.

For word-level NMT models, the translation of out-of-vocabulary words has been addressed through a back-off to a dictionary look-up (Jean et al., 2015; Luong et al., 2015b). We note that such techniques make assumptions that often do not hold true in practice. For instance, there is not always a 1-to-1 correspondence between source and target words because of variance in the degree of morphological synthesis between languages, like productory compounding example. Also, unable to translate or gen-

… used in GPT-2 and GPT-3, LLaMA, Mistral and Mixtral

Senrich et al., (2015)

```
Tokenizer
[30642, 7509]

Charlotte Bunne
[24453, 11404, 28515, 710]
```

# Language Tokenizers: WordPiece



**Byte pair encoding** is essentially a **greedy algorithm for grammar induction**.

*Problem!*

This results in inconsistent tokenization. For example, each digit sequence is tokenized differently.

*One reason why LLMs struggle with arithmetic!*

```
123 12345 1234 45 45
[10163, 17031, 2231, 1105, 2682, 4153, 4153]
```

**WordPiece** … used in BERT (Wu et al., 2016)

Merge the pairs based not only on frequency, but also how frequent elements are individually.

**Pointwise mutual information**

$$\text{score}(a, b) = \frac{\text{freq}(ab)}{\text{freq}(a) \times \text{freq}(b)}$$

PMI

`##` indicate continuation

```
[CLS]123123##45123##44545[SEP]
[101, 13414, 13414, 21336, 13414, 1527, 2532, 2532, 102]
```

→ BPE uses frequency, WordPiece uses likelihood.

WordPiece tends to yield smaller, more semantically consistent vocabularies but is slower to train.

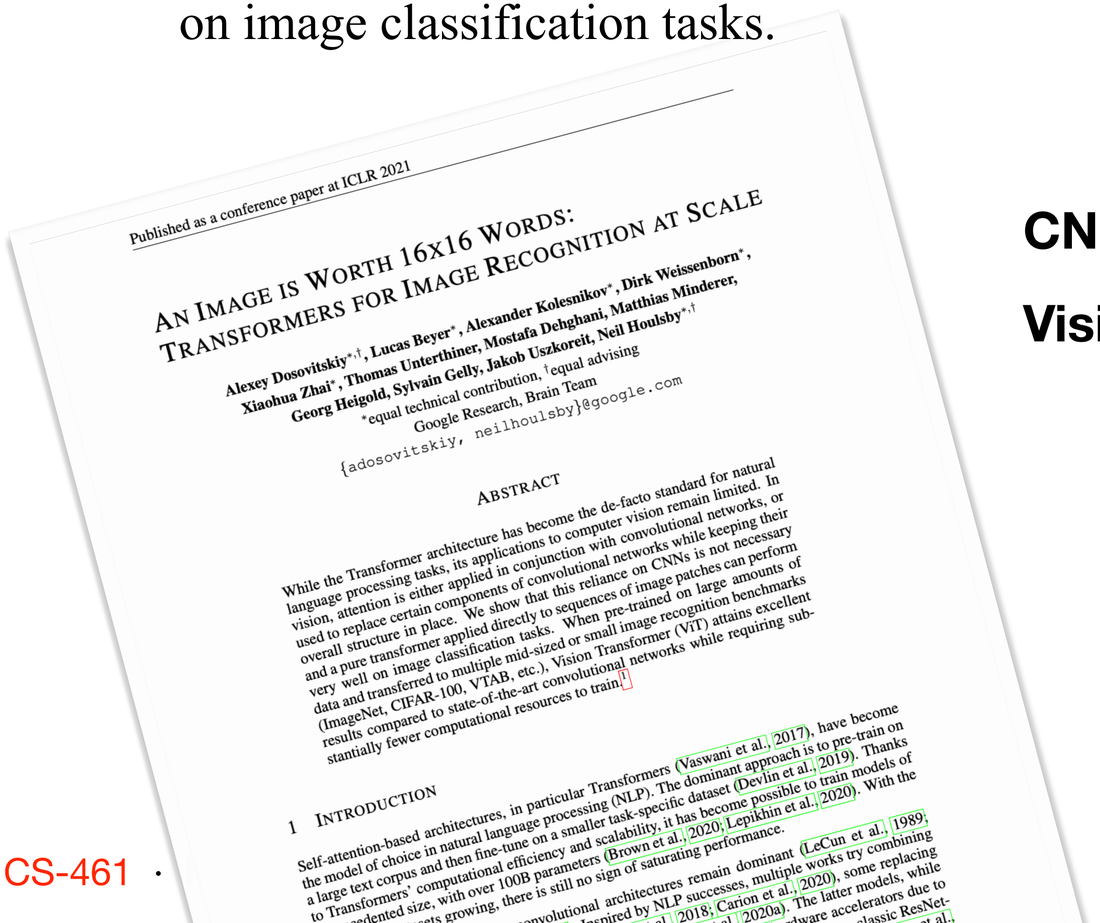
# Language Tokenizers

## Hugging Face

# **Vision Tokenizers:** Fixed Patch Tokenization



> While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks.



Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

**Alexey Dosovitskiy**[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

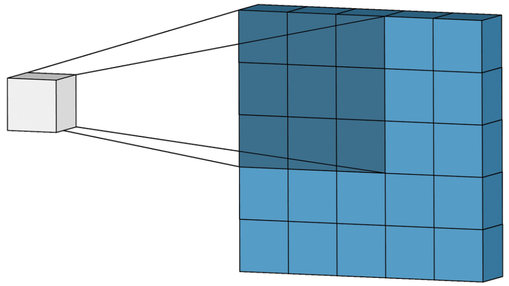
ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

1 INTRODUCTION

Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP). The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019). Thanks to Transformers' computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters (Brown et al., 2020; Lepikhin et al., 2020). With the models and datasets growing, there is still no sign of saturating performance.

In computer vision, convolutional architectures remain dominant (LeCun et al., 1989; Krizhevsky et al., 2012; He et al., 2016). Inspired by NLP successes, multiple works try combining CNN-like architectures with self-attention (Wang et al., 2018; Carion et al., 2020), some replacing the convolutions entirely (Ramachandran et al., 2019; Wang et al., 2020a). The latter models, while theoretically efficient, have not yet been scaled effectively on modern hardware accelerators due to the use of specialized attention patterns. Therefore, in large-scale image recognition, classic ResNet-



**CNNs**: Operate on raw pixels.

**Vision Transformers**: Require tokens!

*Radical Idea!*

Treat an image like a sequence of words.

Dosovitskiy et al., (2021)

# **Vision Tokenizers:** Fixed Patch Tokenization

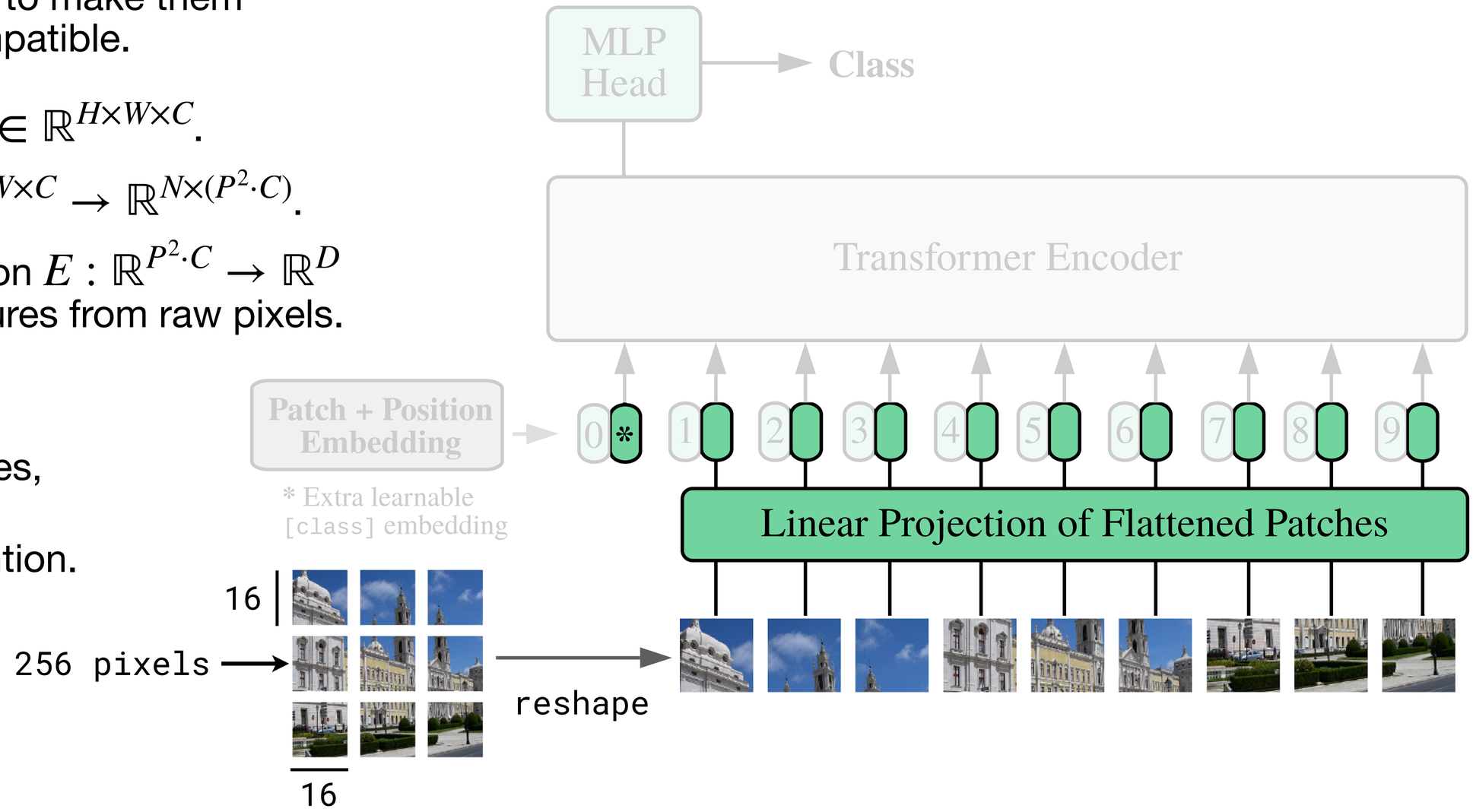


Tokenize images to make them Transformer-compatible.

> Given image $I \in \mathbb{R}^{H \times W \times C}$.
>
> Reshape $\mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{N \times (P^2 \cdot C)}$.
>
> Linear projection $E : \mathbb{R}^{P^2 \cdot C} \rightarrow \mathbb{R}^{D}$ to extract features from raw pixels.

$$N = \frac{HW}{P^2} \text{ patches,}$$

with $\mathcal{O}(N^2)$ attention.



# Vision Tokenizers: VQ-VAE



Vector-Quantized Variational Autoencoder (VQ-VAE) represents a fundamentally different approach:

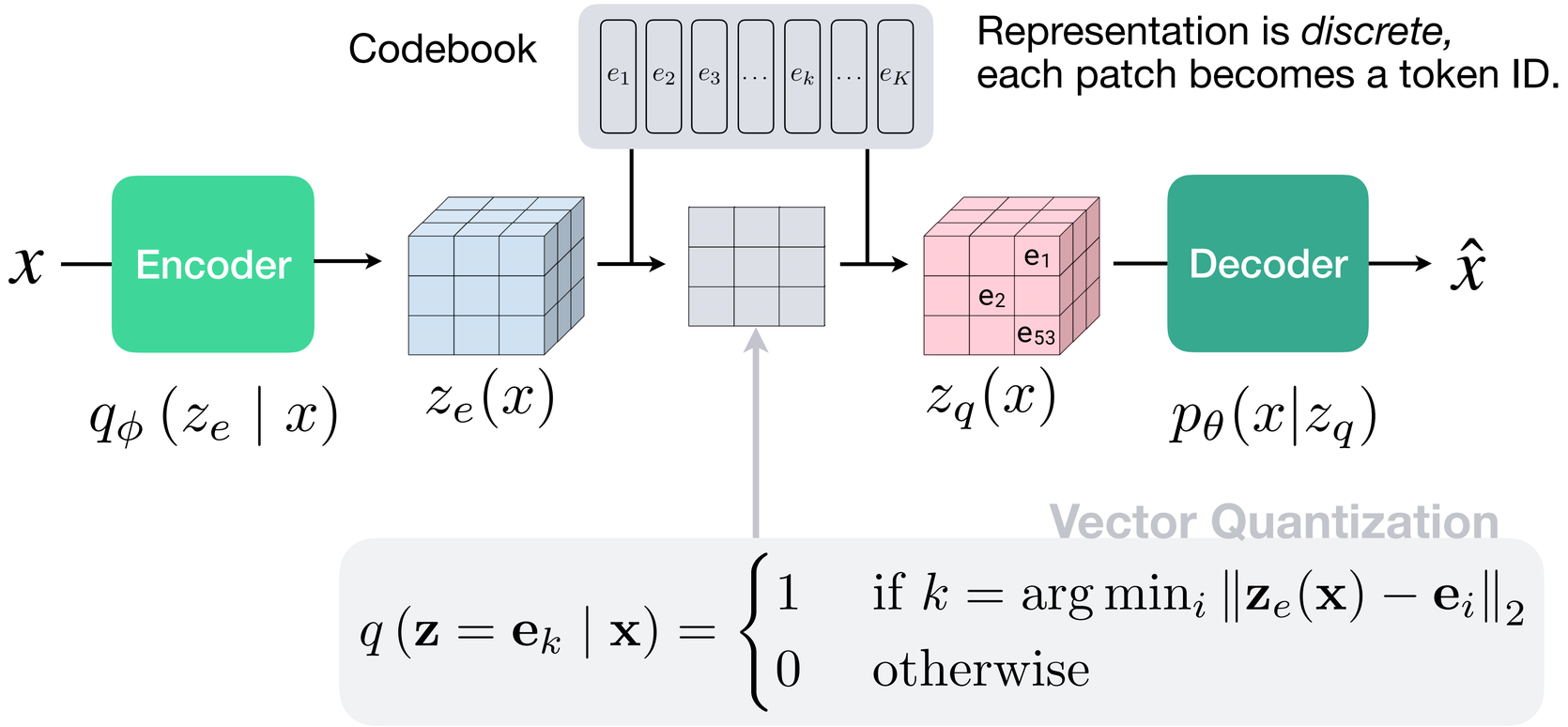
**Goal**: Instead of fixed patches, compress a continuous image $x$ into a sequence of discrete latent tokens, i.e., a **learned vocabulary of visual concepts**.

**Encoder**:
Maps $x$ into continuous latent features $z_e$.

**Vector Quantization**:
$z_e$ is replaced by its nearest codebook entry $e_k$.

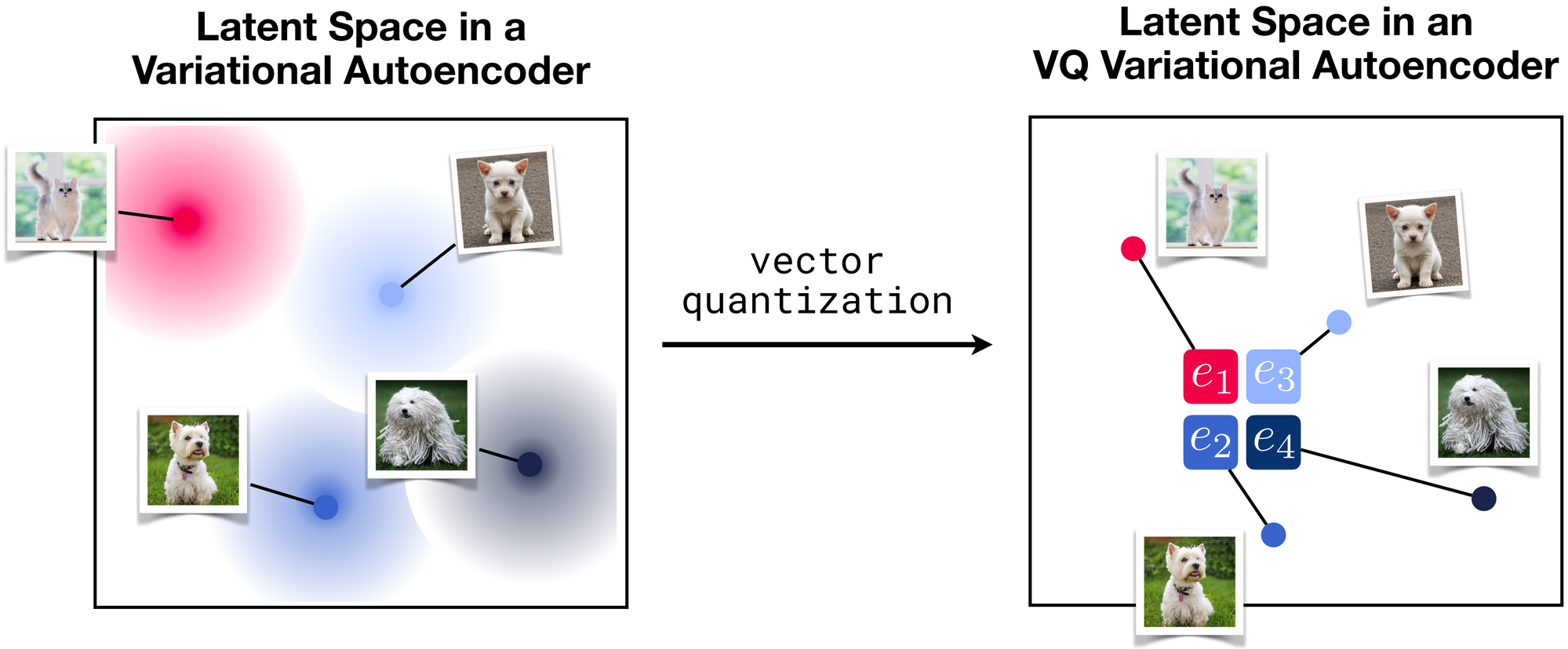
**Decoder**:
Reconstucts $x$ from quantized representation $z_q$.

(van den Oord et al., 2017)



# Vision Tokenizers: VQ-VAE





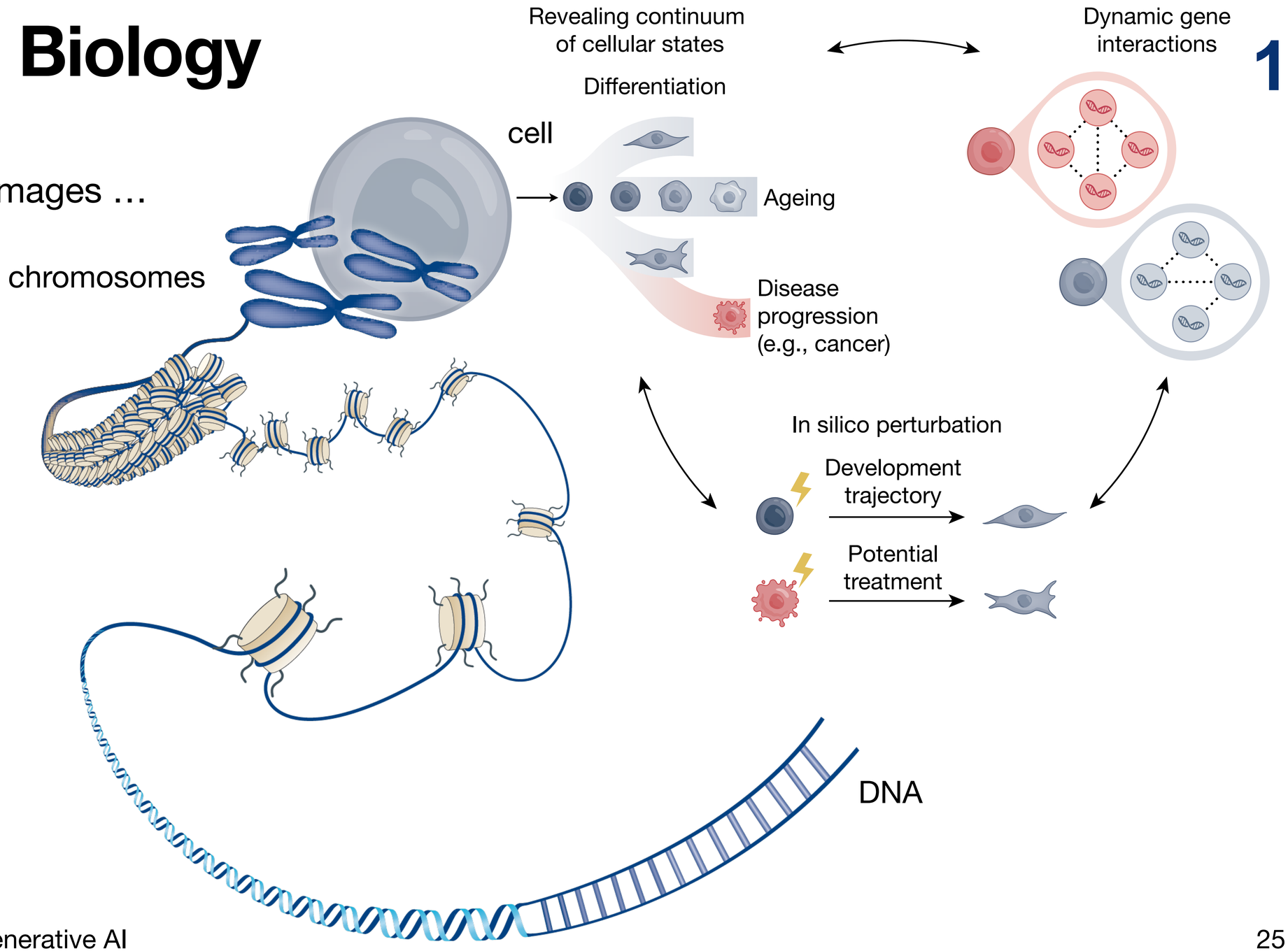

Ramesh et al. (2021) in DALL·E the first to use the VQ-VAE code indices as a tokenizer for images.

Lecture 8: Multimodality

# Tokenizers in Biology

So far, we have covered
how to tokenize text and images …

But how to tokenize
**complex systems**
such as biology?



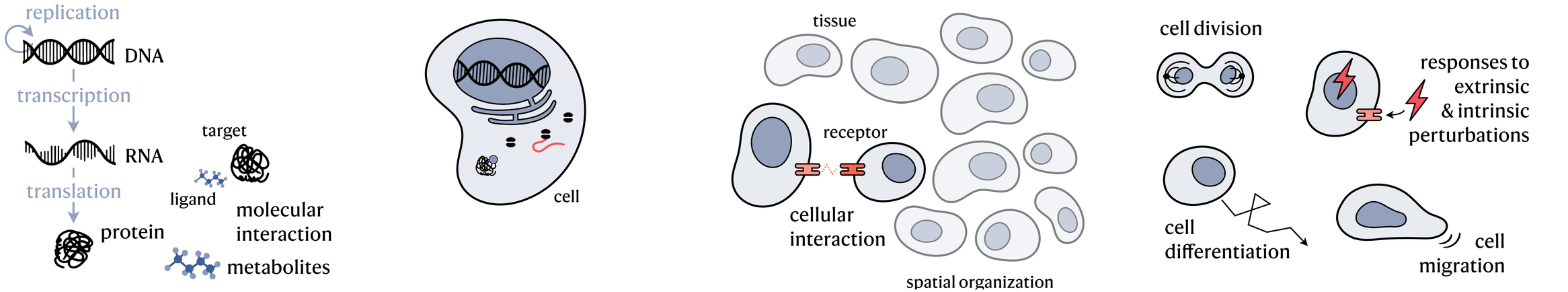
# Tokenization in Biology

(Bunne et al., 2024)



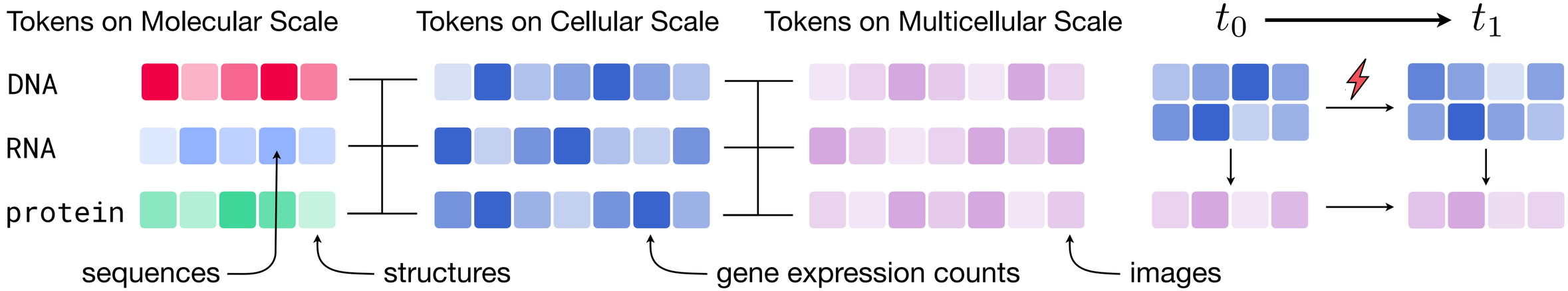
**Cellular building blocks, environments, …**

**… behavior, and dynamics.**




**Tokenize across Biological Scales and Modalities …**

**… and Time**



# Tokenizers in Biology

BUNNE LAB

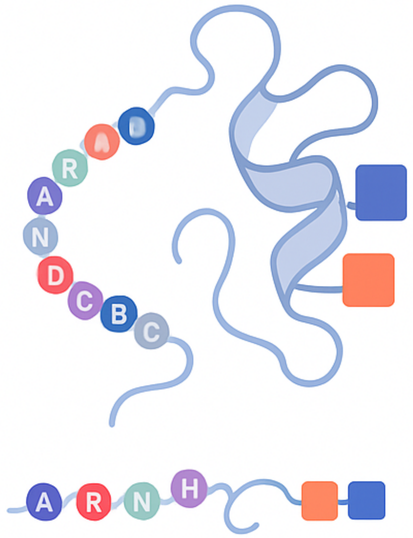
Join Us!



**DNA**

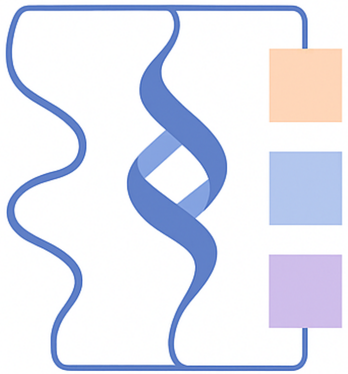

$k$-mers tokens

**Proteins**

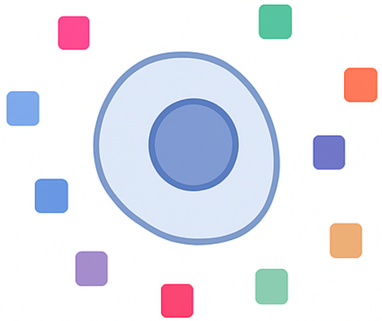



amino acid tokens

structure tokens

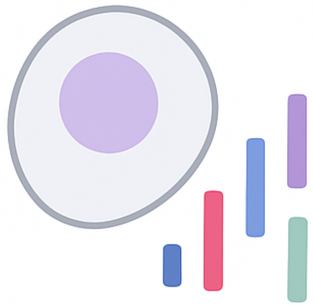


**Cells**



protein imaging intensity tokens

gene expression tokens



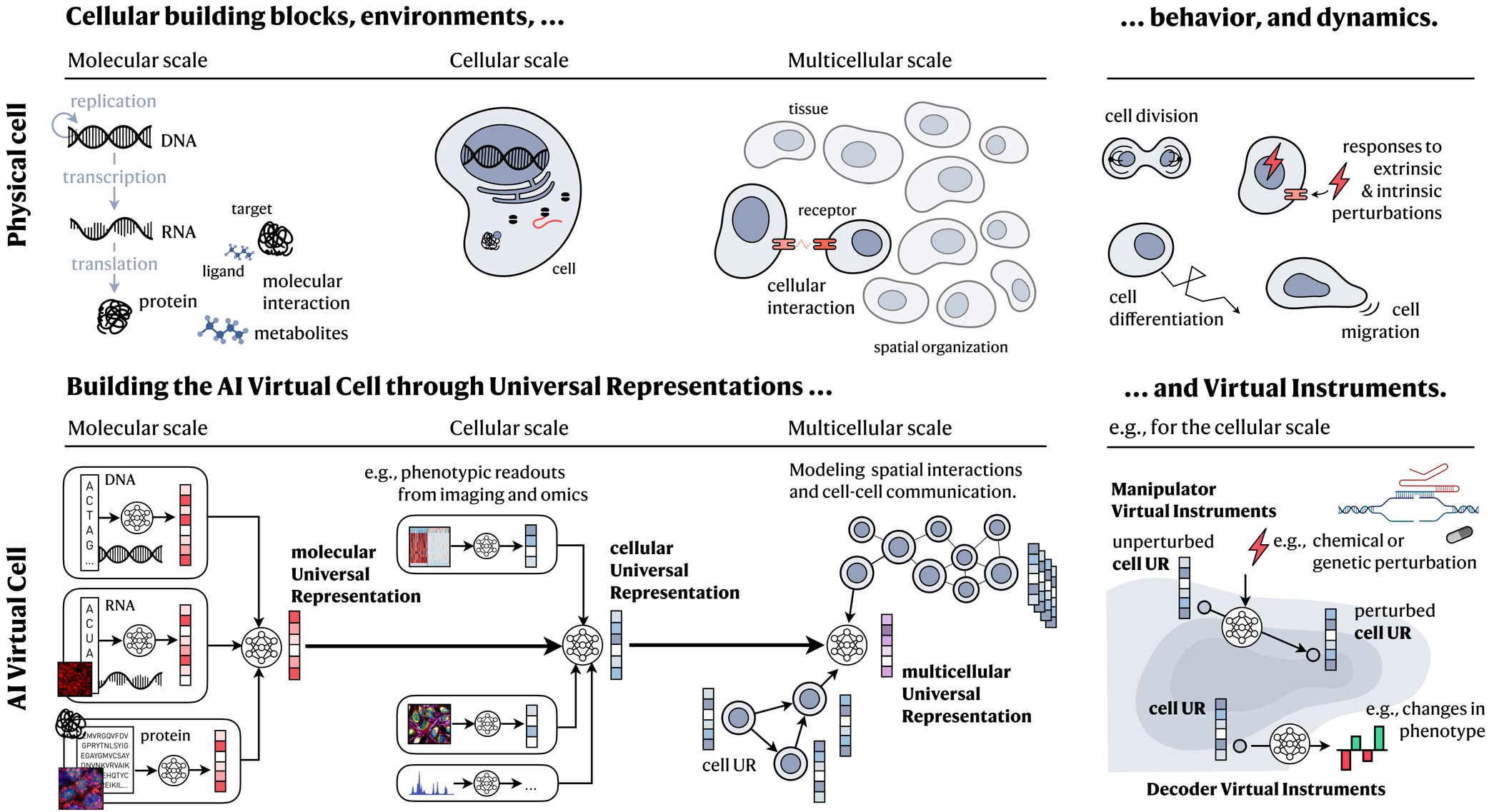
CS-502 Deep Learning in Biomedicine

Lecture 9: FM in Biology

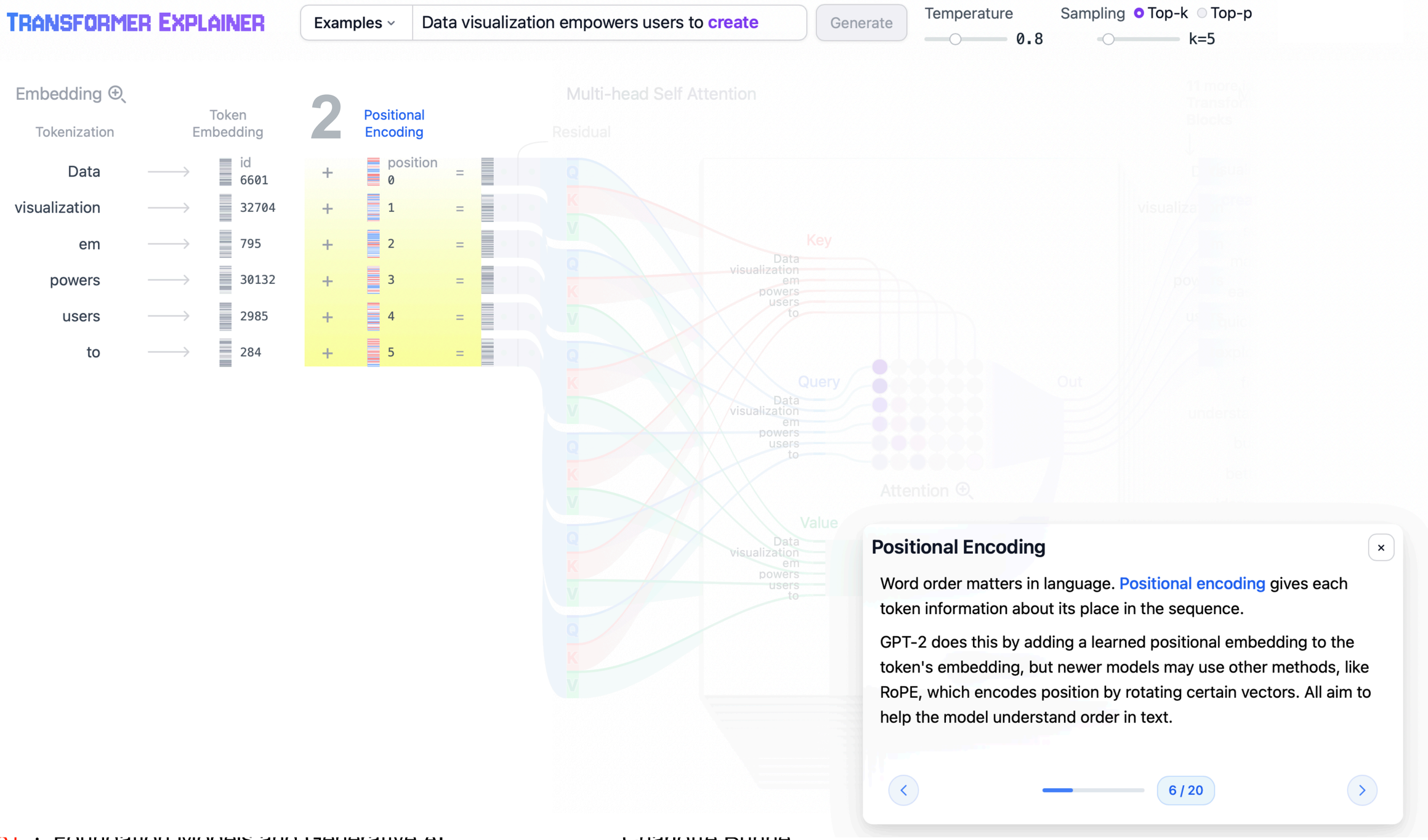
# Toward an AI Virtual Cell

(Bunne et al., 2024)






Lecture 9: FM in Biology


TRANSFORMER EXPLAINER
Examples
Data visualization empowers users to create
Generate
Temperature
0.8
Sampling
Top-k
Top-p
k=5
Embedding
Tokenization
Token Embedding
2
Positional Encoding
Multi-head Self Attention
Residual
Data
visualization
em
powers
users
to
id
6601
32704
795
30132
2985
284
position
0
1
2
3
4
5
Key
Query
Value
Attention
Out
Positional Encoding
Word order matters in language. Positional encoding gives each token information about its place in the sequence.
GPT-2 does this by adding a learned positional embedding to the token's embedding, but newer models may use other methods, like RoPE, which encodes position by rotating certain vectors. All aim to help the model understand order in text.
6 / 20

# Why Positional Encodings?



**Problem**: Neural networks are order-blind.

Networks treat inputs as a "bag of tokens" with no inherent position awareness:

```
"The cat ate the mouse" = {"cat", "mouse", "ate", "the", "the"}
"The mouse ate the cat" = {"cat", "mouse", "ate", "the", "the"}
```

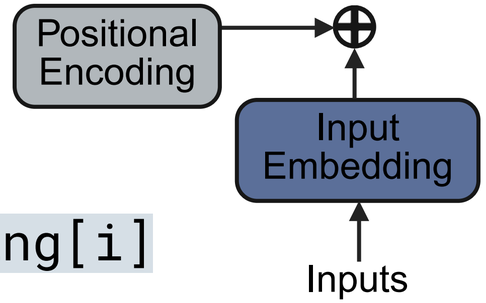
→ Same tokens, same processing.

→ Most neural operations are **permutation equivariant**, network cannot distinguish different orderings.

*Who ate whom?*

**Idea**: Add unique position information to each token.

```
final_embedding[i] = token_embedding[i] + position_encoding[i]
```



# Why Positional Encodings?



**How?**

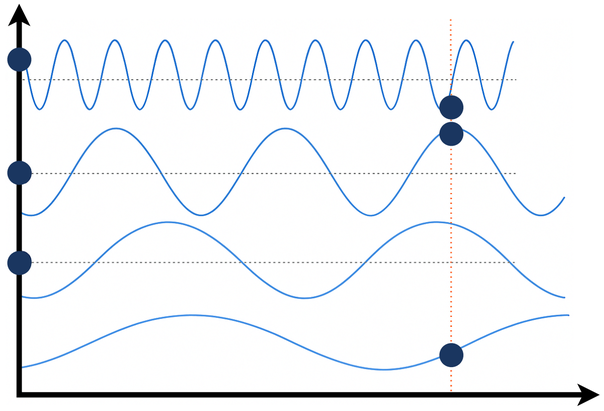
`simple index`

✗ → unbound growth.

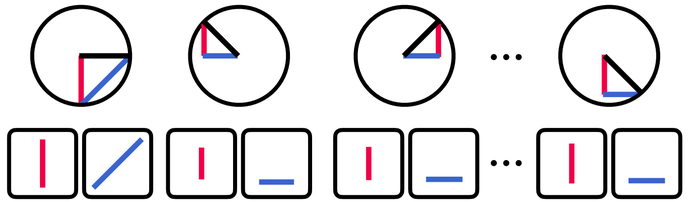
`one-hot`

✗ → cannot extrapolate to different lengths.

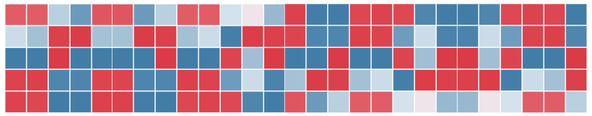
`sinusoidal`



`rotary`



`learned`



**Requirements**:

1. Unique: $PE(i) \neq PE(j)$.
2. Bounded: $\|PE(1)\| \approx \|PE(1000)\|$.
3. Relative: Distance-aware.
4. Extrapolatable: Works beyond training length.

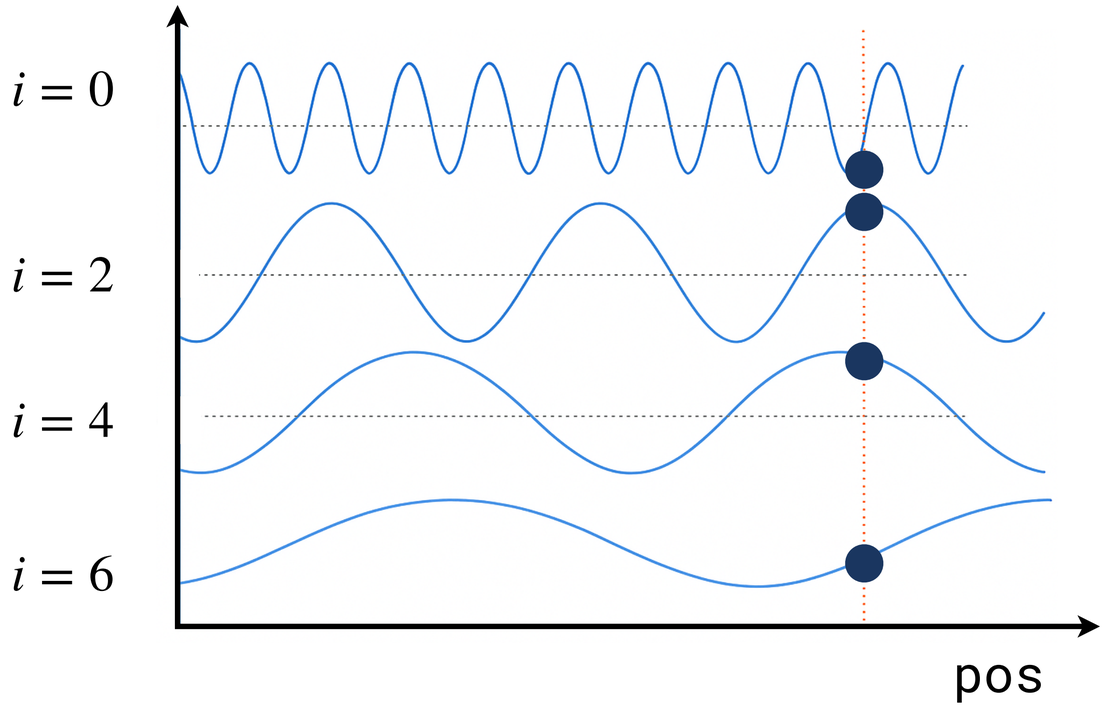
# Sinusoidal **Positional Encodings** `sinusoidal`



**Idea**: Use a combination of sine and cosine functions at different frequencies to encode position.

Each dimension operates at a different frequency, creating a unique "fingerprint" for each position.

For every position in the input `pos`
and `index` $i$




$$PE_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{model}}}\right)$$

$$PE_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{model}}}\right)$$

… the frequencies are decreasing along the vector dimension

… forms a geometric progression from $2\pi$ to $10000 \cdot 2\pi$ on the wavelengths

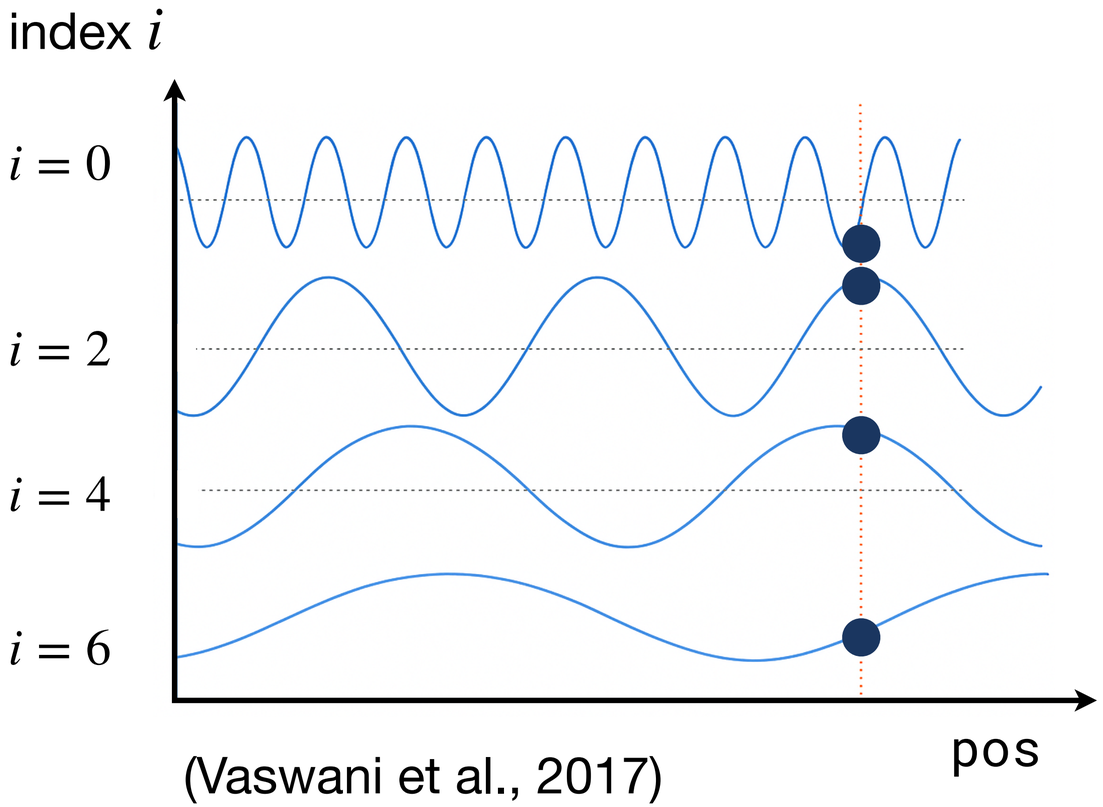
# Sinusoidal **Positional Encodings**

`sinusoidal`

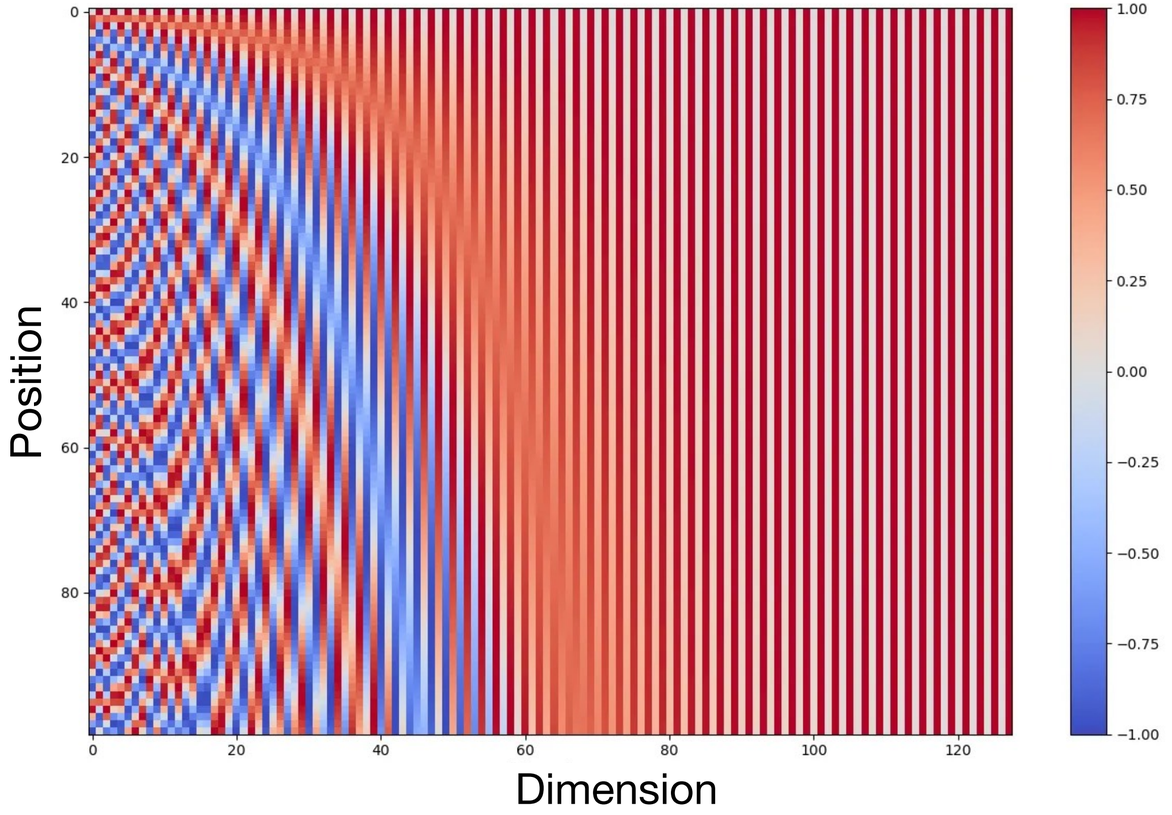


> “ We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset $k$, $PE(\text{pos} + k)$ can be represented as a linear function of $PE(\text{pos})$.

… $d$-dimensional vector that contains information about a specific position




(Vaswani et al., 2017)



# Rotary **Positional Encodings**

`rotary` (Su et al., 2023) 2

Rotary position (RoPe) encoding takes a very different approach to encoding positional information:

**Idea**:

Instead of adding a positional encoding,
RoPE applies a rotation to the existing token embeddings.

The rotation angle is a function of both the token's position in the sequence and the dimension of the embedding.

Rotation preserves the norm of the embeddings while encoding positional information.

**Step 1.** Define rotation frequencies:

$$\theta_i = 10000^{-2(i-1)/d}$$

**Step 2.** Apply position to get rotation angle.

$$\text{angle}_i = m \cdot \theta_i$$

**Step 3.** Build rotation matrix $R_m$.

e.g., for two dimensions $\begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

# Rotary **Positional Encodings**

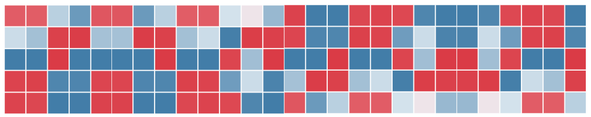
`rotary` (Su et al., 2023) 2

**Key Property**:
When computing attention between positions $m$ and $n$, $\text{score} = (R_m \mathbf{q})^\top (R_n \mathbf{k}) = \mathbf{q}^\top R_{n-m} \mathbf{k}$
the dot product depends only on relative position $(n - m)$!

*Naturally relative!*

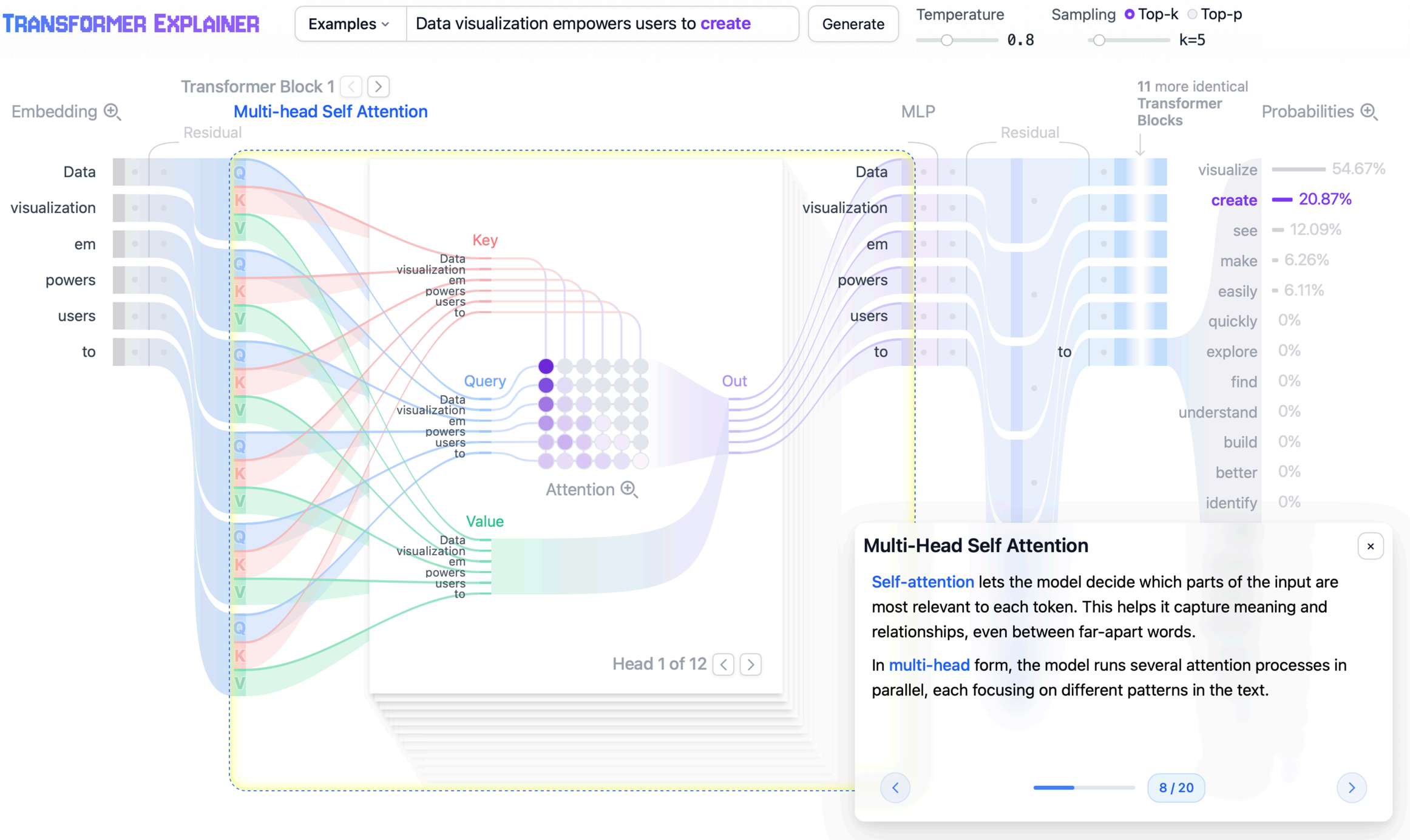
**Intuition:**
Each position rotates vectors in embedding space. Tokens compare based on the angle between their rotations, which only depends on how far apart they are!

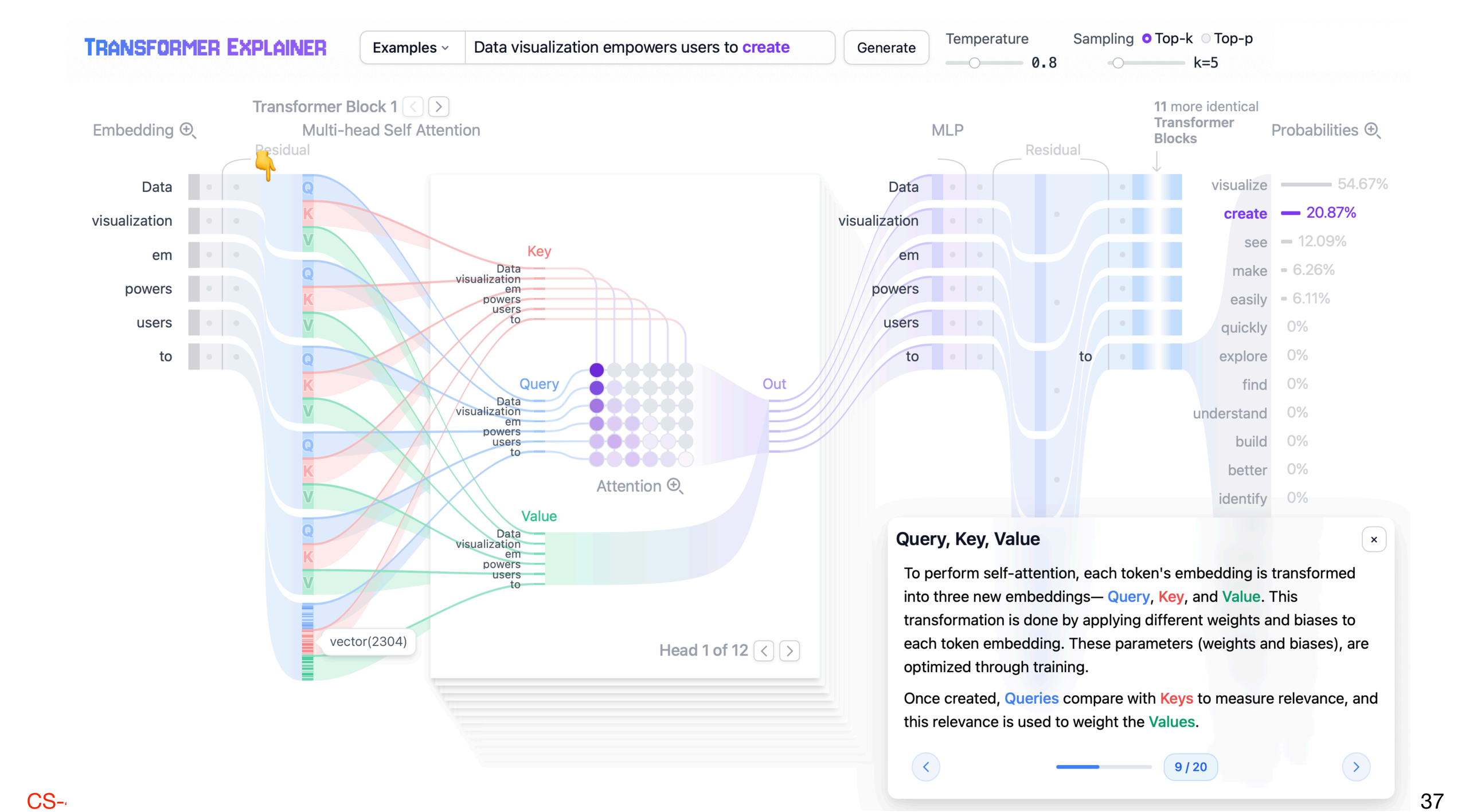
**Advantage** over `learned`?



→ Poor generalisation to inputs with more dimensions longer than the maximum position embedding encountered during training.

*Position embeddings remain an active area of research!*


TRANSFORMER EXPLAINER
Examples
Data visualization empowers users to create
Generate
Temperature
0.8
Sampling
Top-k
Top-p
k=5
Transformer Block 1
Embedding
Multi-head Self Attention
Residual
MLP
Residual
11 more identical Transformer Blocks
Probabilities
Data
visualization
em
powers
users
to
Q
K
V
Key
Query
Value
Attention
Out
Head 1 of 12
visualize 54.67%
create 20.87%
see 12.09%
make 6.26%
easily 6.11%
quickly 0%
explore 0%
find 0%
understand 0%
build 0%
better 0%
identify 0%
Multi-Head Self Attention
Self-attention lets the model decide which parts of the input are most relevant to each token. This helps it capture meaning and relationships, even between far-apart words.
In multi-head form, the model runs several attention processes in parallel, each focusing on different patterns in the text.
8 / 20


TRANSFORMER EXPLAINER
Examples
Data visualization empowers users to create
Generate
Temperature
0.8
Sampling
Top-k
Top-p
k=5
Transformer Block 1
Embedding
Multi-head Self Attention
Residual
MLP
Residual
11 more identical Transformer Blocks
Probabilities
Data
visualization
em
powers
users
to
Q
K
V
Key
Query
Value
Attention
Out
vector(2304)
Head 1 of 12
visualize 54.67%
create 20.87%
see 12.09%
make 6.26%
easily 6.11%
quickly 0%
explore 0%
find 0%
understand 0%
build 0%
better 0%
identify 0%
Query, Key, Value
To perform self-attention, each token's embedding is transformed into three new embeddings— Query, Key, and Value. This transformation is done by applying different weights and biases to each token embedding. These parameters (weights and biases), are optimized through training.
Once created, Queries compare with Keys to measure relevance, and this relevance is used to weight the Values.
9 / 20

# What is Attention?

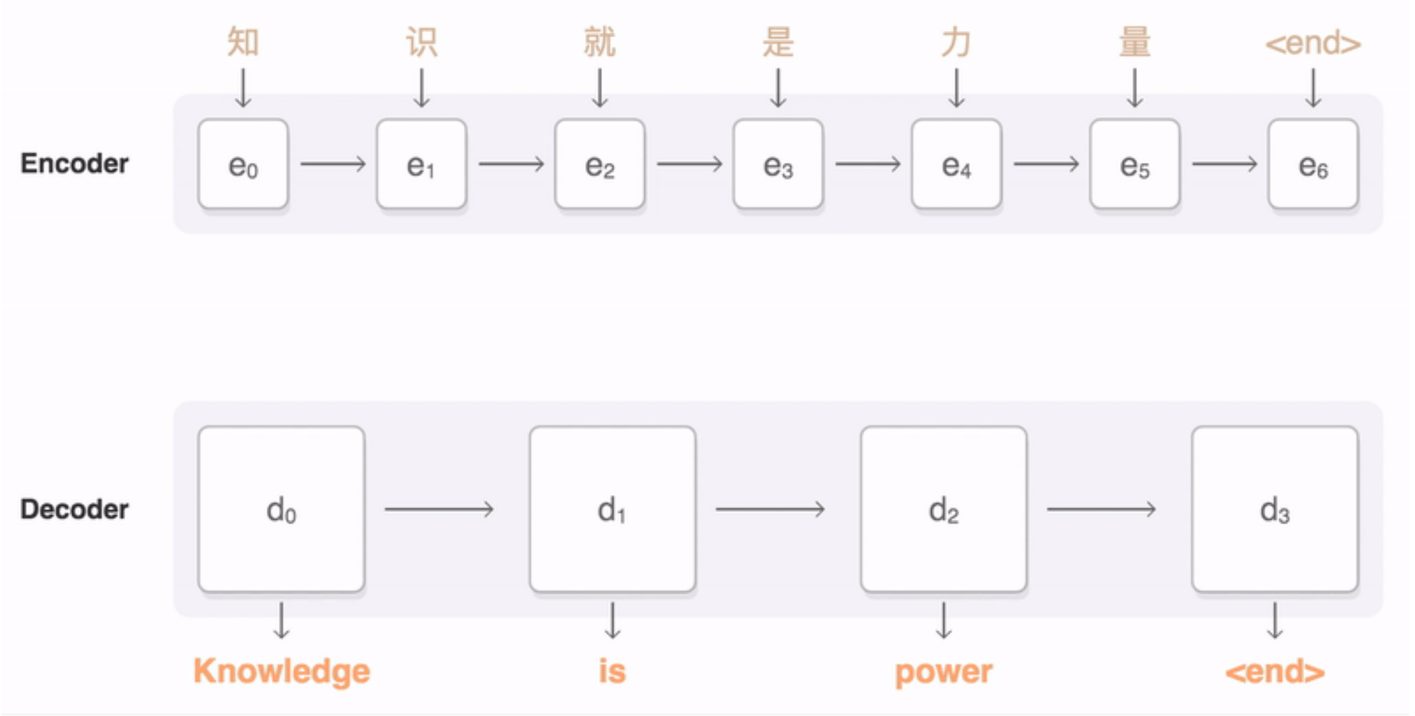


… a mechanism that lets each element of a sequence compute a **context-dependent representation** by focusing on the **most relevant parts** of the input and **learning the connections between them**.

**Key Idea**:

- Earlier architectures (CNNs, RNNs) process information locally or sequentially.
- But in many problems, any part of the input might depend on any other.
- We want a mechanism that lets the model decide what to focus on dynamically.

→ Attention allows information to flow flexibly between tokens.

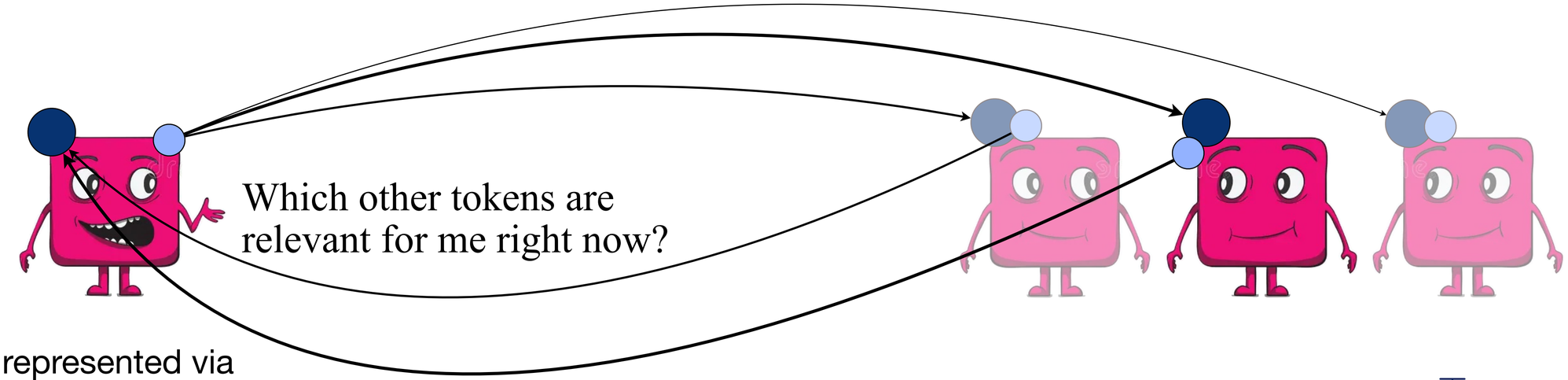


# What is Attention?



**Attention = Information Lookup**

**Intuition**




Each token is represented via

- QUERY — what it wants
- KEY — what it offers
- VALUE — the actual information

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

dot product to compare all key-query pairs

Similarity between QUERY-KEY pairs determines how much of each VALUE is used.

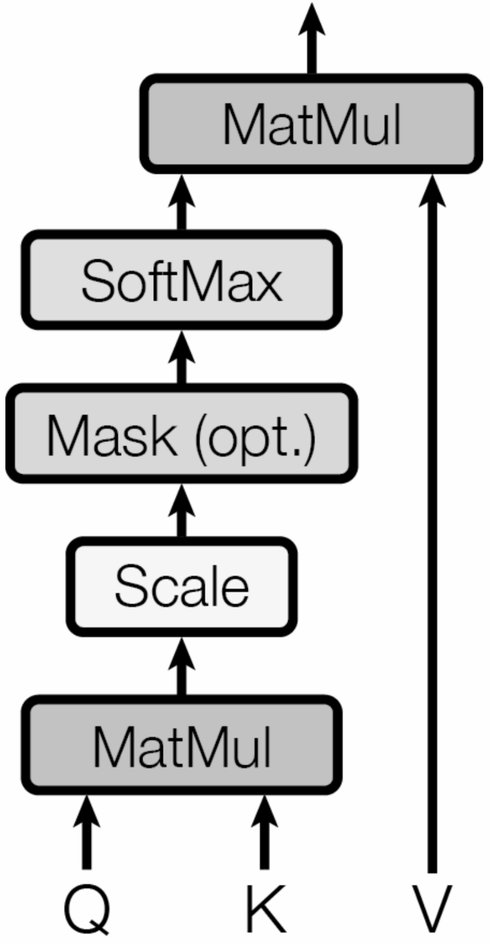
→ Attention is soft information retrieval based on learned similarity.

$Q, K, V$ : learnable weights.

# **Attention:** Scaled Dot-Product Attention

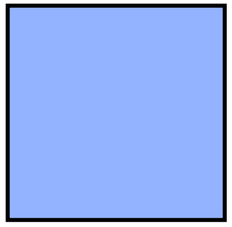
Extensively covered in 4

CS-433 Machine Learning

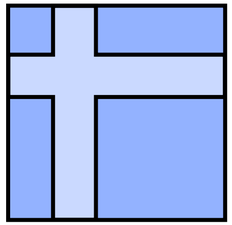



**Attention**

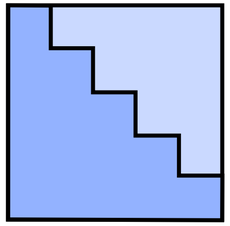
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$



full attention



bidirectional masked attention



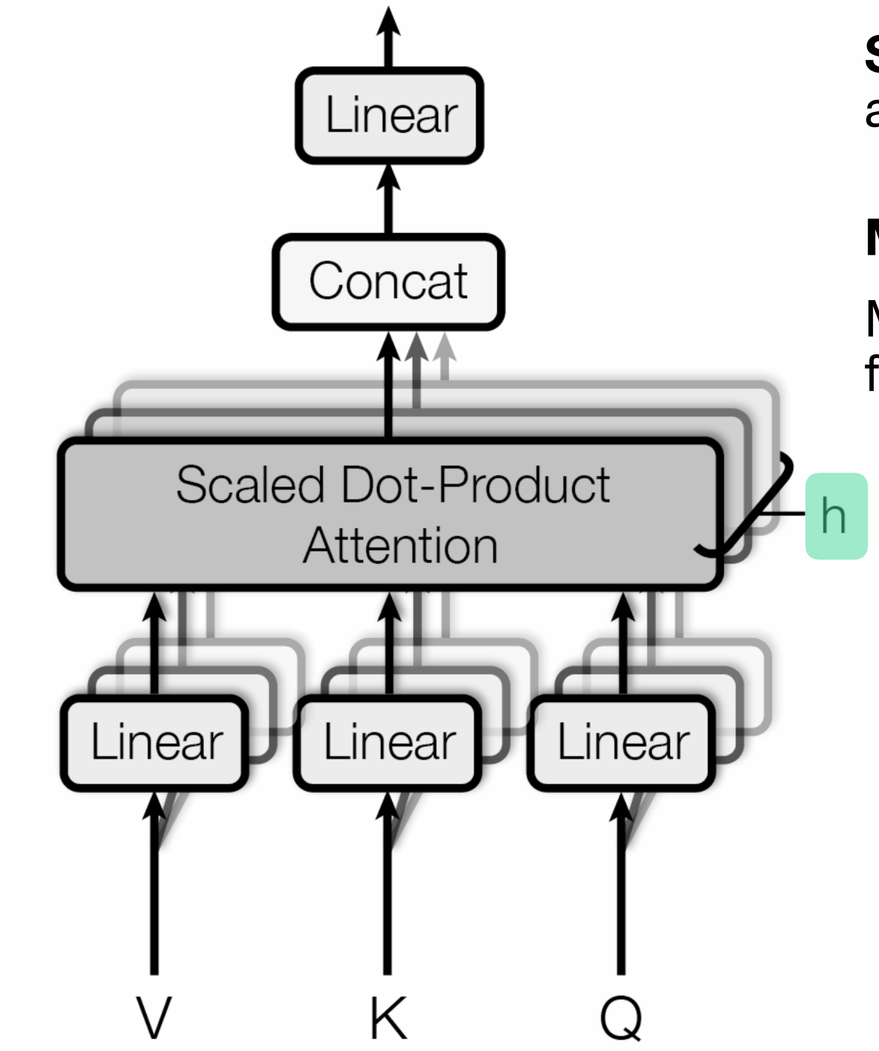
causal attention

→ Depending on architecture and learning principle, mask is applied.

$$= \text{softmax}\left(M + \frac{QK^\top}{\sqrt{d_k}}\right)V$$

# **Attention:** Multi-Head Attention






**Self-Attention** lets the model decide which parts of the input are most relevant to each token.

**Multi-Head Attention:**

Model runs several attention processes in parallel, each focusing on different patterns in the text.

- One attention map may miss certain relationships.
- Multiple heads learn independent subspaces focusing on different patterns, e.g., syntax, position, entity type, etc.
- Each head learns its own $Q$, $K$, $V$ projections.
- Their outputs are concatenated and linearly combined.

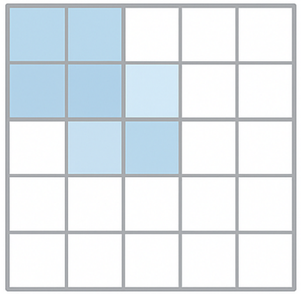
# Attention Implementations



Standard attention is $\mathcal{O}(n^2)$ in both time and memory, where $n$ is the input length.

*What do we do?*

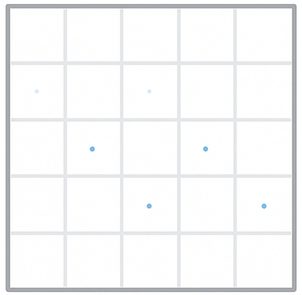
Speed and Memory Efficiency



**Flash Attention** Still $\mathcal{O}(n^2)$ but GPU-optimized.

Computes attention in chunks to fit GPU memory and avoid redundant softmax ops.

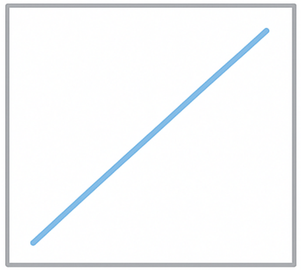
Efficiency in Long Contexts



**Sparse Attention** Reduces complexity to $\mathcal{O}(n\sqrt{n})$ or less.

Attends only to a subset of tokens (e.g., local + global pattern).
No need to compute all pairwise interactions.

Scalability to Real-Time Settings



**Linear Attention** True $\mathcal{O}(n)$ scaling but less expressive.

Approximates softmax with kernel tricks to make computation linear in sequence length.

# **Attention:** Cross-Attention



**Goal**: Enable communication between different representation spaces
e.g., text ↔ image, gene ↔ tissue, cell ↔ microenvironment.

**How?**

**Cross-Attention**

$$\mathrm{CrossAttn}\left(Q_1, K_2, V_2\right) = \mathrm{softmax}\left(\frac{Q_1 K_2^\top}{\sqrt{d}}\right) V_2$$

- Queries $Q$ come from one modality.
- Keys $K$ and Values $V$ come from another.
- The attention weights determine which parts of one modality are most relevant to the other.

**Why?**

- Self-attention: builds coherence within a modality.
- Cross-attention: builds alignment across modalities.
- Creates a shared representational space between heterogeneous data types.
- It's the core mechanism behind multimodal foundation models.

Lecture 8: Multimodality

# **Attention:** Why is attention *a* universal interface?

4

*Why does it work across so many domains?*

Attention **learns the structure from data**, not from a fixed topology.

Defines relations dynamically via similarity, not through hard-wired order or adjacency.

→ It therefore generalizes as a “universal interface” for reasoning over sets.

*But* it’s inefficient: all-to-all comparisons scale quadratically;
the brain and physical systems use sparse, hierarchicalconnectivity instead.

Tokenization limits: universality only extends to what can be discretized into tokens.

Also, attention reasons over states, not processes. It sees correlations, not causation.

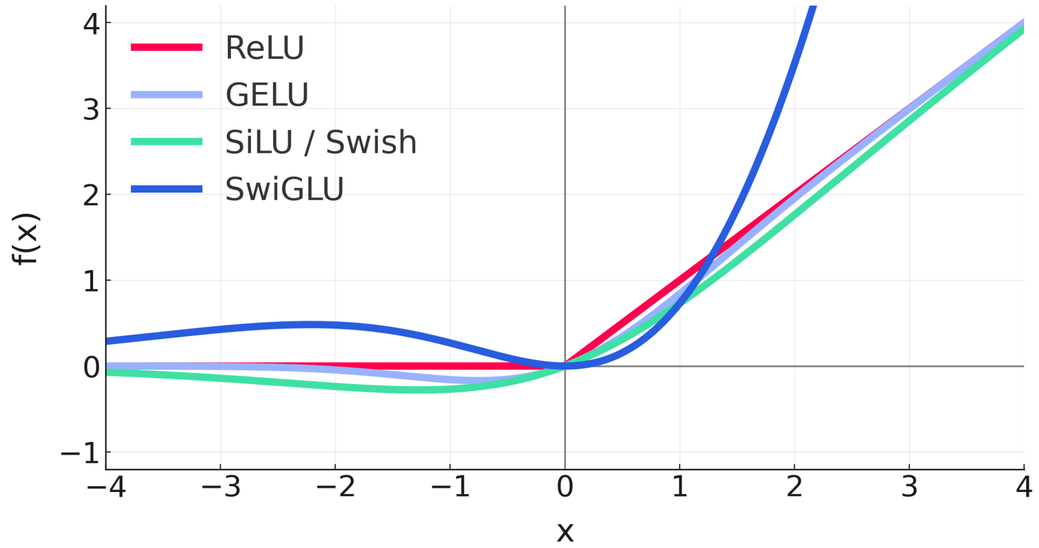
# Transformer Block

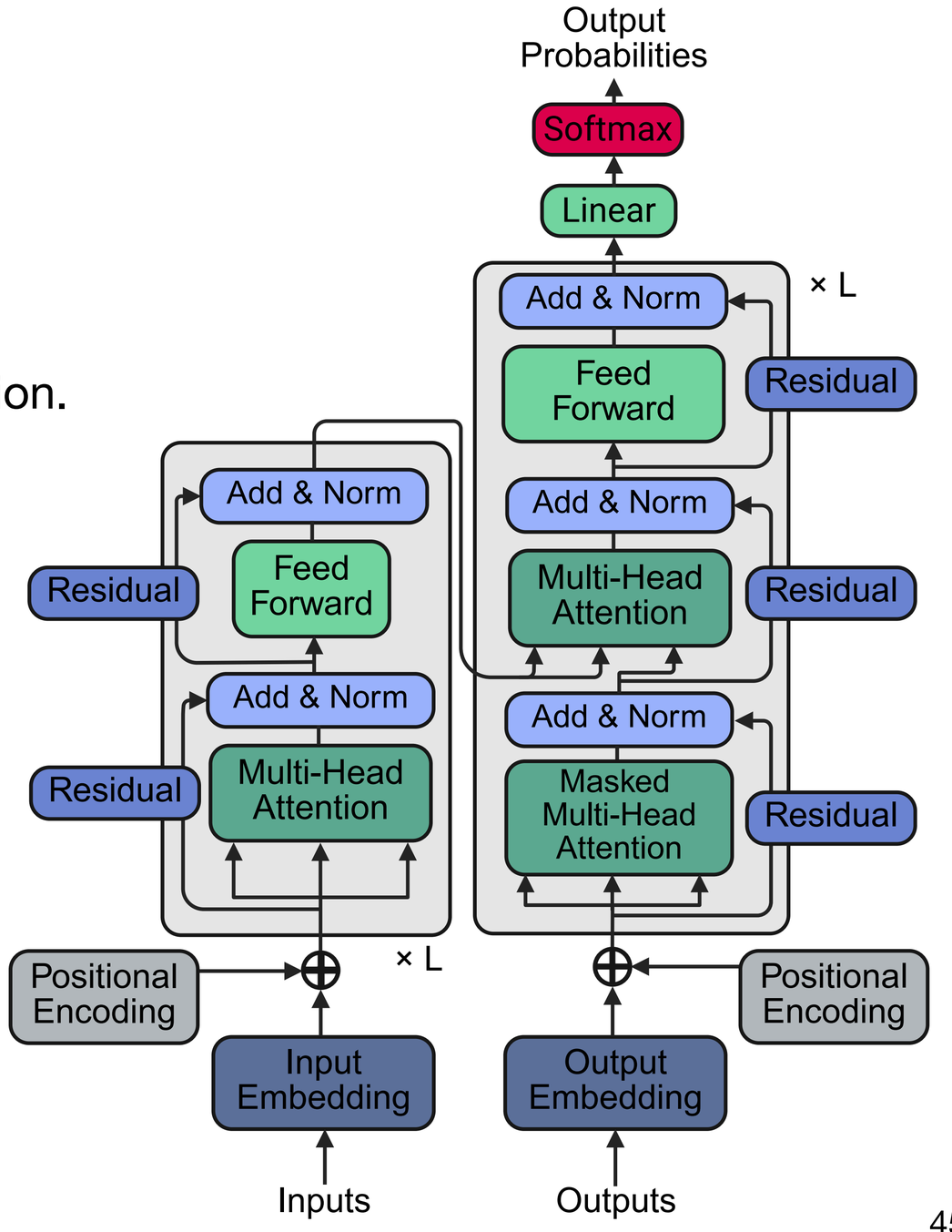
Main unit processing in the model consisting of:

3 **Feed-forward layers**: Refines each token's details.

4 **Multi-head self attention**: Lets tokens share information.

5 **Activation functions inside 3:** Smooth, non-saturating activations as they improve gradient flow, training stability, and model expressivity.



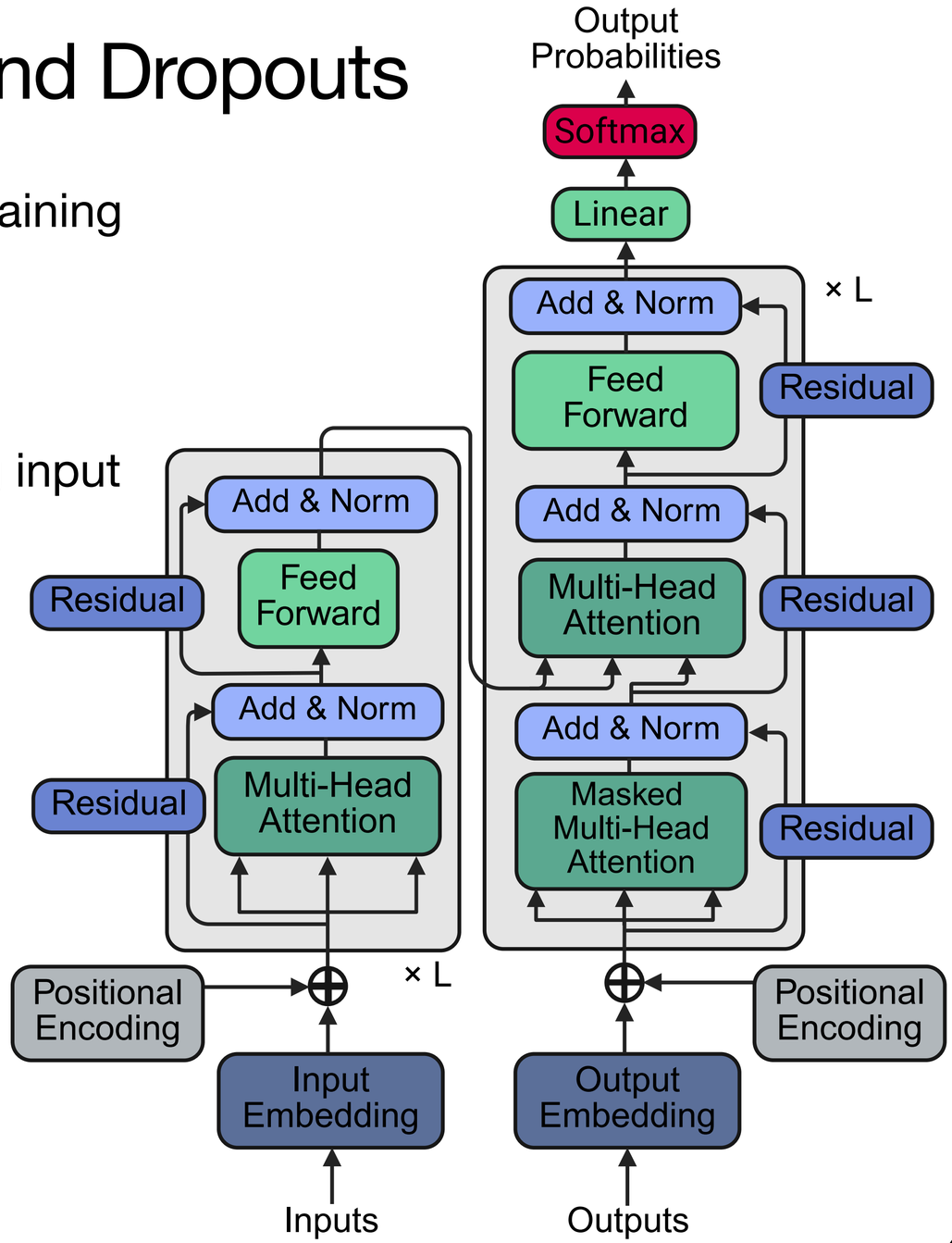
# **Architectural Patterns:** Residuals and Dropouts

The following architectural patterns stabilize Transformer training and preserve information across layers.

**6** **Layer Normalization**:
Helps stabilize both training and inference by adjusting input numbers so their mean and variance stay consistent.

**Dropout**:
During training, dropout randomly turns off some connections between numbers so the model does not overfit to specific patterns.

**7** **Residual Connections**:
Adds a layer's input to its output, keeping information and learning signals from fading through many layers.

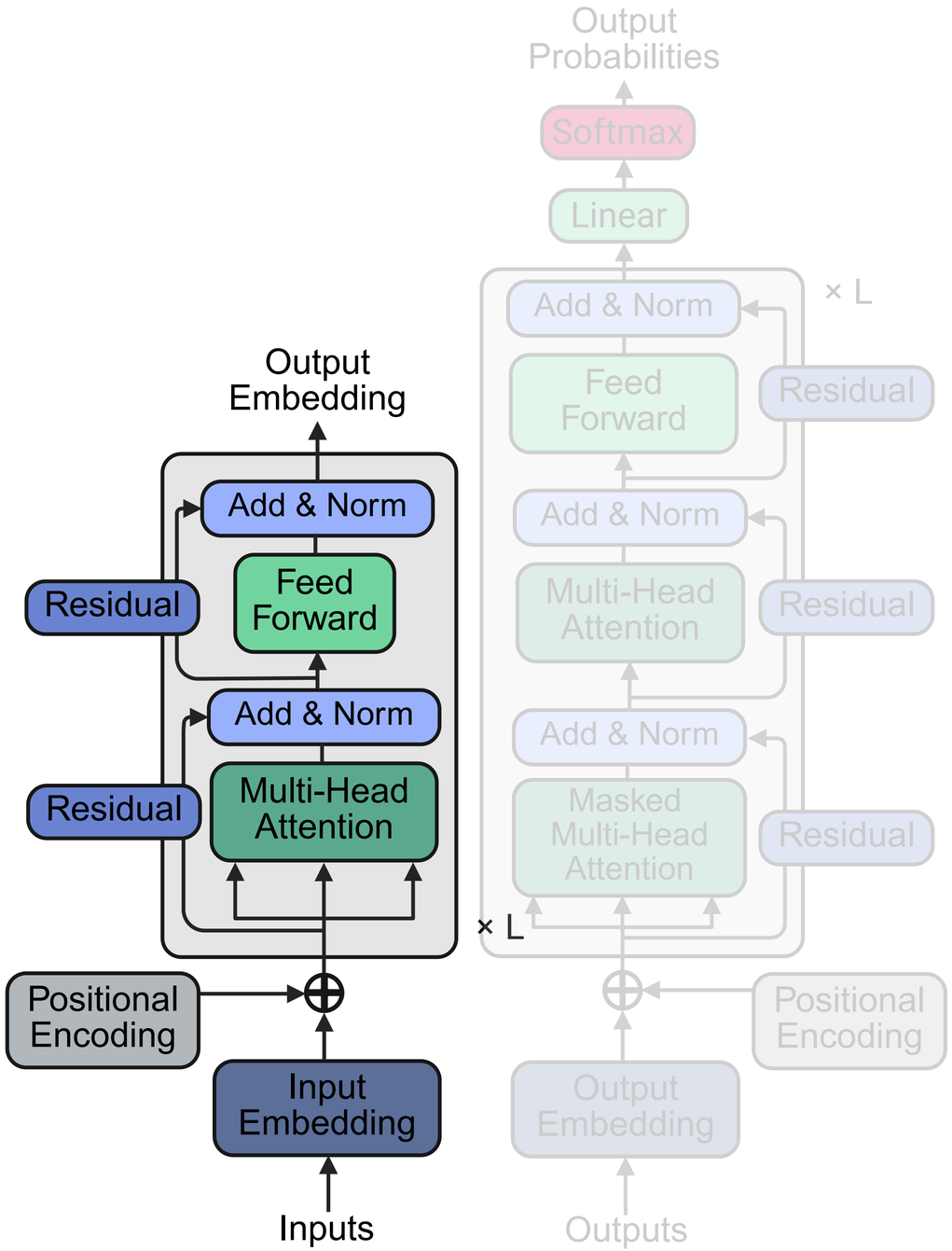


# Transformer Architecture

## Encoder

Processes the entire input in parallel to build *contextual representations*.

Produces **high-level embeddings** that can be **used for understanding**, **comparison**, or as **input to downstream modules**.



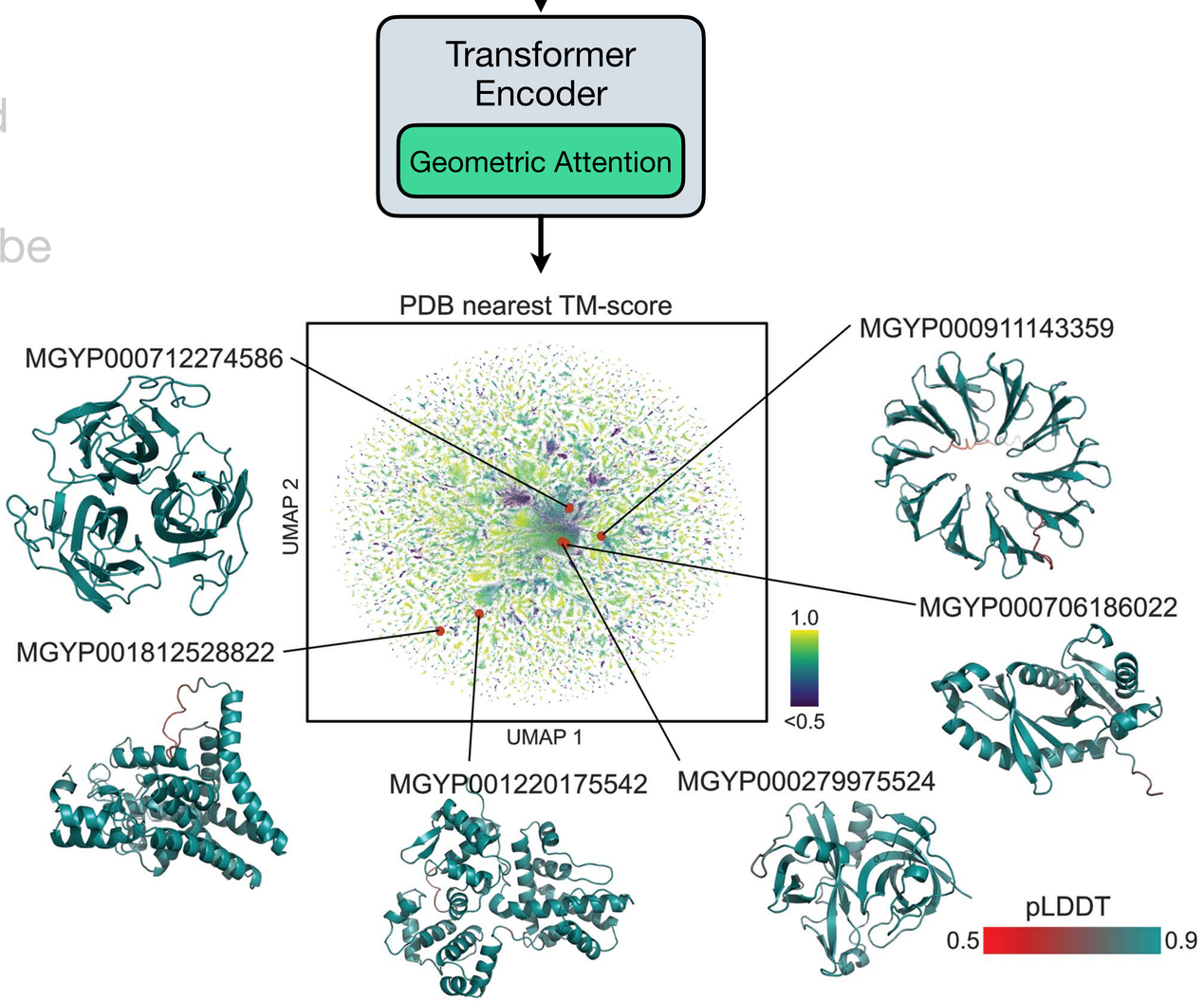
# Transformer Architecture

## Encoder

Processes the entire input in parallel to build *contextual representations*.

Produces **high-level embeddings** that can be **used for understanding**, **comparison**, or as **input to downstream modules**.

protein data base ~250 million protein sequences!




→ ESM encoders learn a protein "language" that captures *structure*, *function*, and *evolution*, allowing reasoning directly from sequence.

→ Lecture 10: Emergent Behaviors

# Transformer Architecture

## Decoder

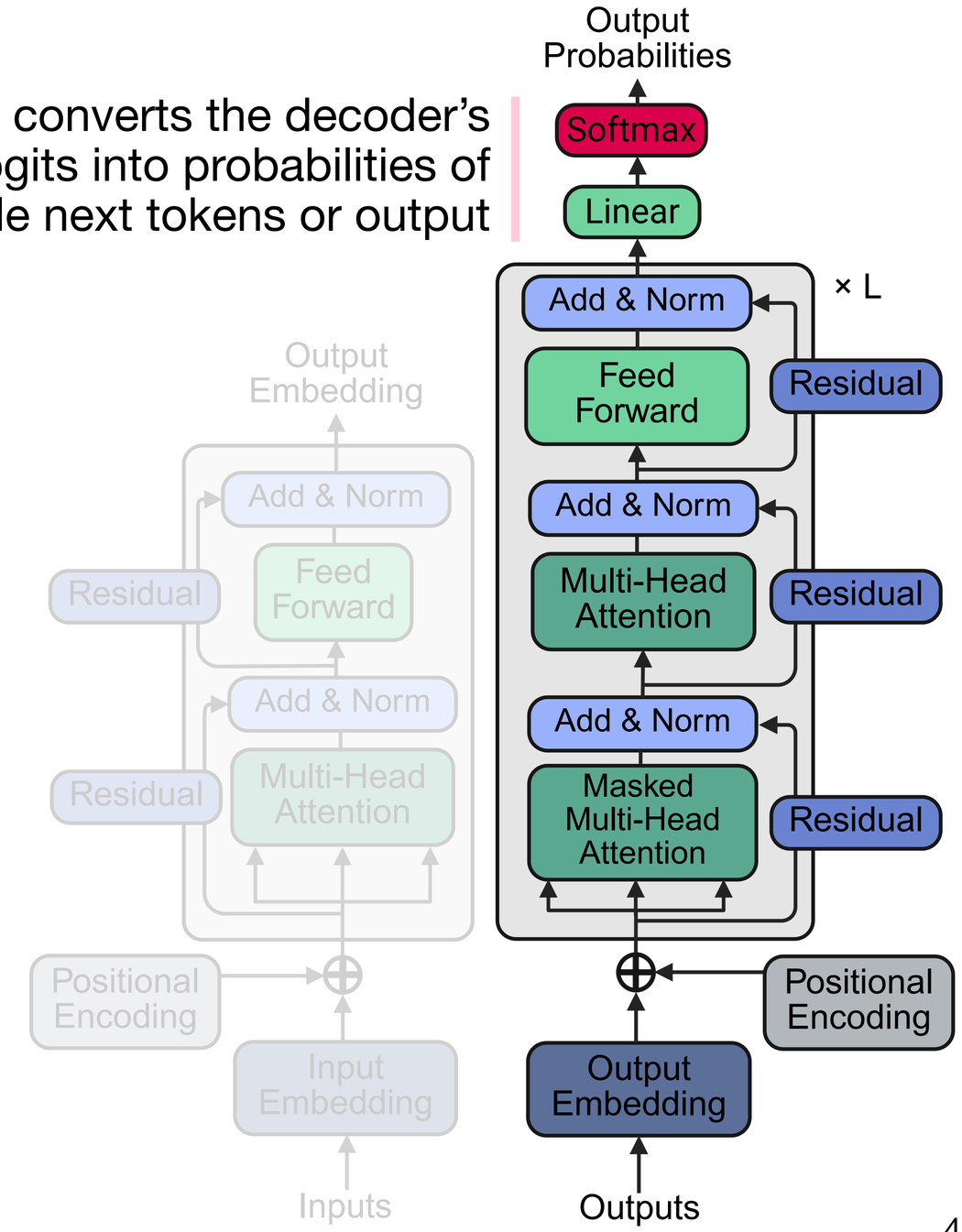
Processes and transforms representations, optionally conditioned on encoder outputs.

In **encoder–decoder models**, generation is conditional, attending to encoder representations (e.g., translation, summarization).

In **decoder-only models**, generation is autonomous, relying solely on prior outputs (e.g., GPT-style models).

Decoders enable the model to generate, transform, or predict based on learned representations.

**5** converts the decoder's logits into probabilities of possible next tokens or output



# Outlook

## Language Foundation Models

CS-552 Modern NLP

Lecture 6: Vision Foundation Models … Transformers in BERT, GPT, and more.

## Vision Foundation Models

CS-503 Visual Intelligence

Lecture 6: Vision Foundation Models … Vision Transformers in MAE, DinoV2, and more.

## Foundation Models in Science

Lecture 9: FM in Biology

Lecture 9: FM in Chemistry

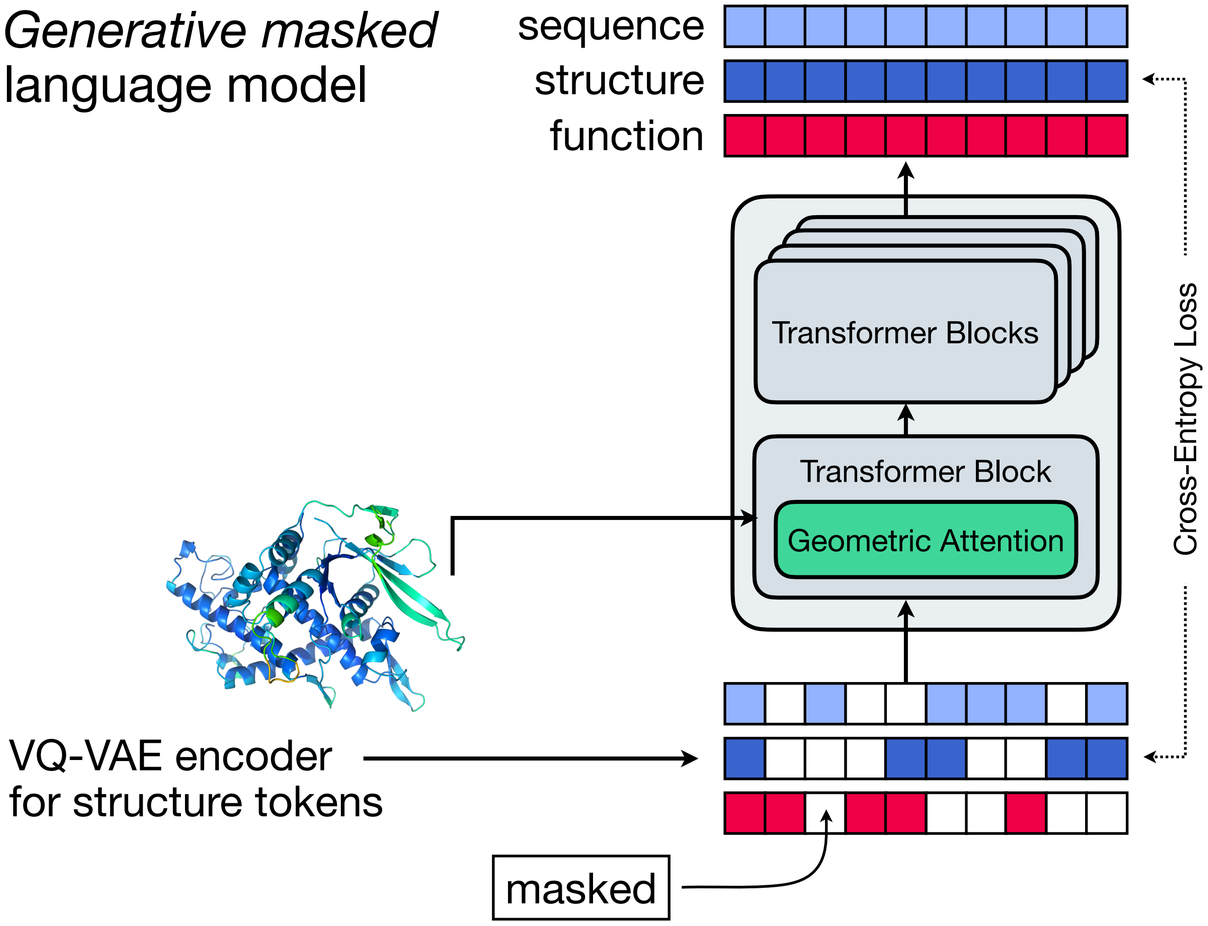
Lecture 9: FM in Earth Sciences

# **Transformer** in Protein Foundation Models

## ESM3 Transformer

(Hayes et al., 2025)

*Generative masked* language model




- **Multimodal and geometry-aware**: integrates 1D sequence, 3D structure, and function data using geometry-conditioned attention.
- **Unified cross-modal objective**: trained to generate or predict sequence, structure, or function within a shared latent space.
- **Hybrid backbone**: Transformer layers fused with SE(3)-equivariant blocks for spatial reasoning.

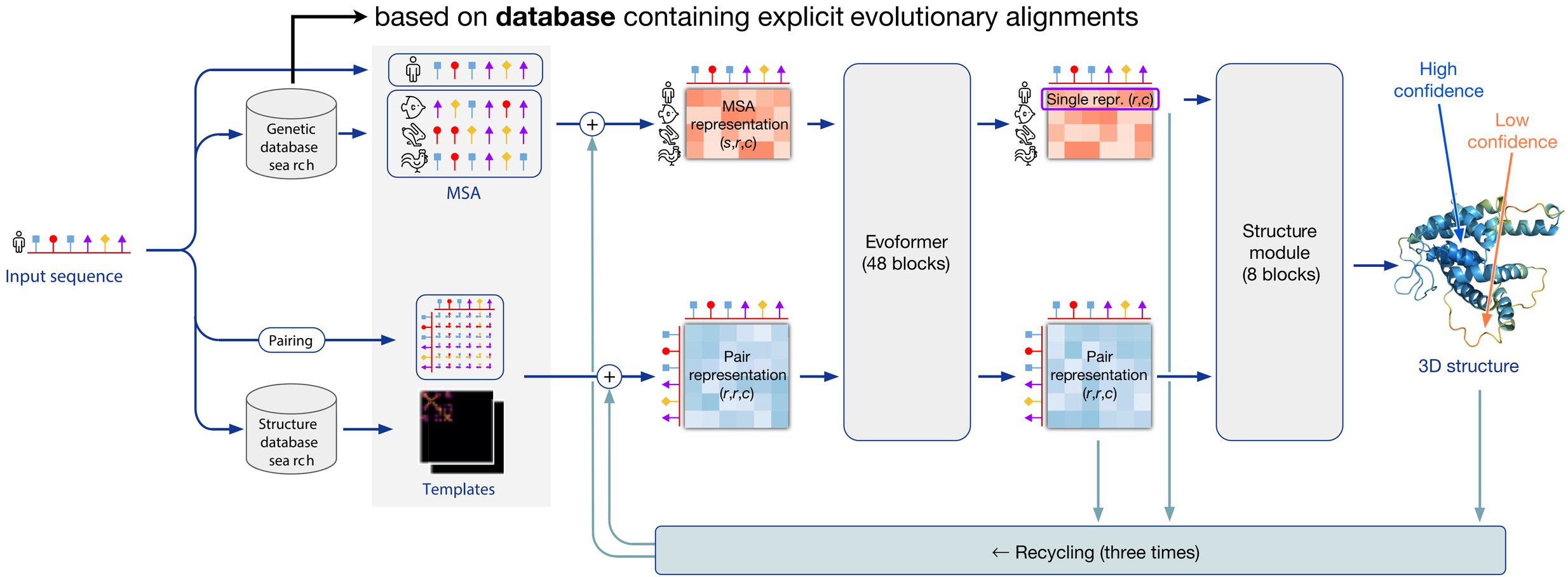
→ However, much more of a classic Transformer compared to …

# **Transformer** in Protein Folding Models

## `Evoformer` of AlphaFold 1

(Jumper et al., 2021)

→ While ESM3 keeps the core Transformer, AlphaFold's Evoformer **re-engineers attention** for pairwise and evolutionary reasoning.

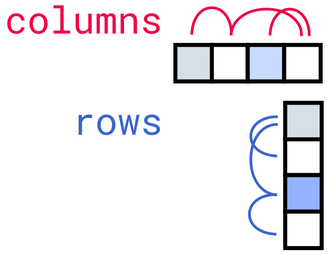


# **Transformer** in Protein Folding Models

## `Evoformer` of AlphaFold 1

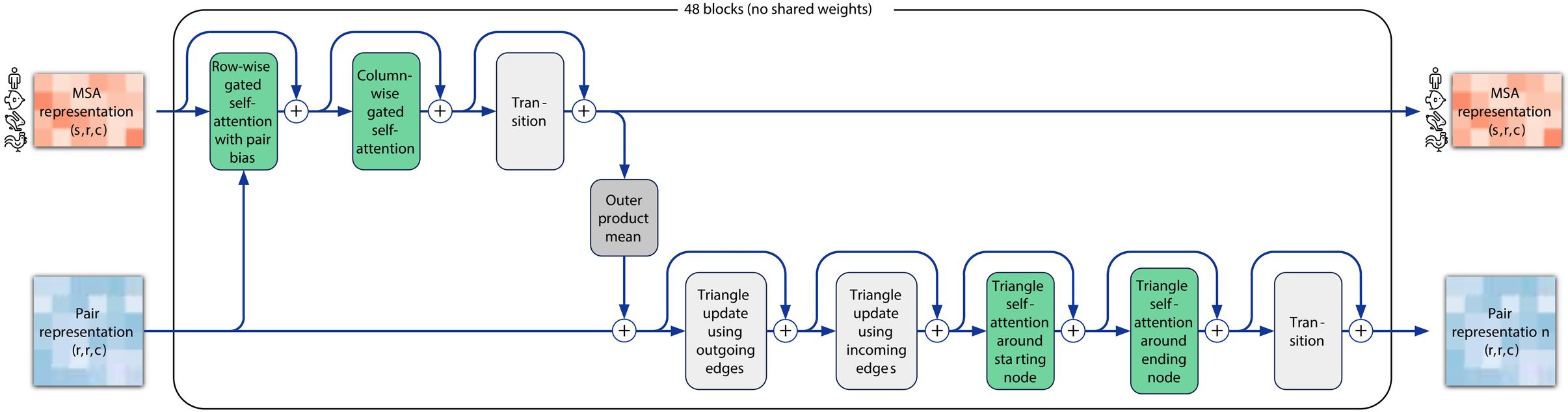
(Jumper et al., 2021)

Special attention blocks including attetion across …




residue positions along the target protein sequence → context within each protein sequence

different homologous sequences → captures evolutionary covariation



# **Transformer** in Protein Folding Models

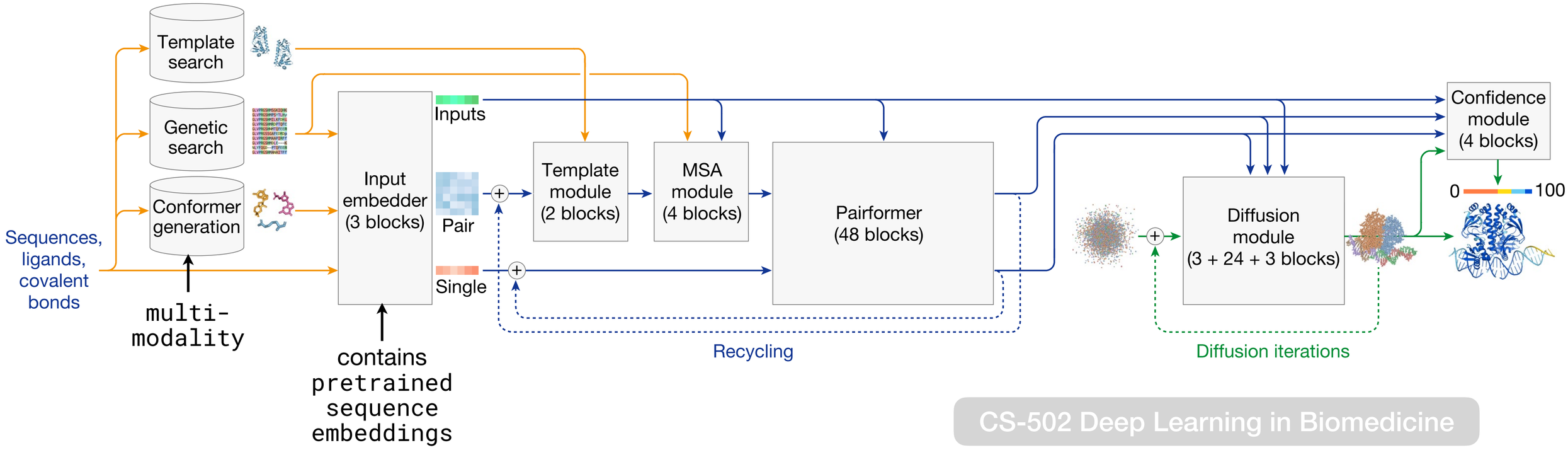
## `Pairformer` of AlphaFold 3

(Abramson et al., 2025)

AlphaFold 3 removes the MSA representation, evolutionary information comes from pretrained sequence embeddings.

Two representations: one for each residue (*single*) and one for residue-residue relationships (*pair*) to capture 3D structure.

Multimodal inputs, i.e., proteins, ligands, nucleic acids, and cofactors within the same transformer framework.




CS-502 Deep Learning in Biomedicine

Lecture 9: FM in Biology

# **Transformer** in Protein Folding Models

## Pairformer of AlphaFold 3

(Abramson et al., 2025)

Richard Sutton (2019)
“The Bitter Lesson”

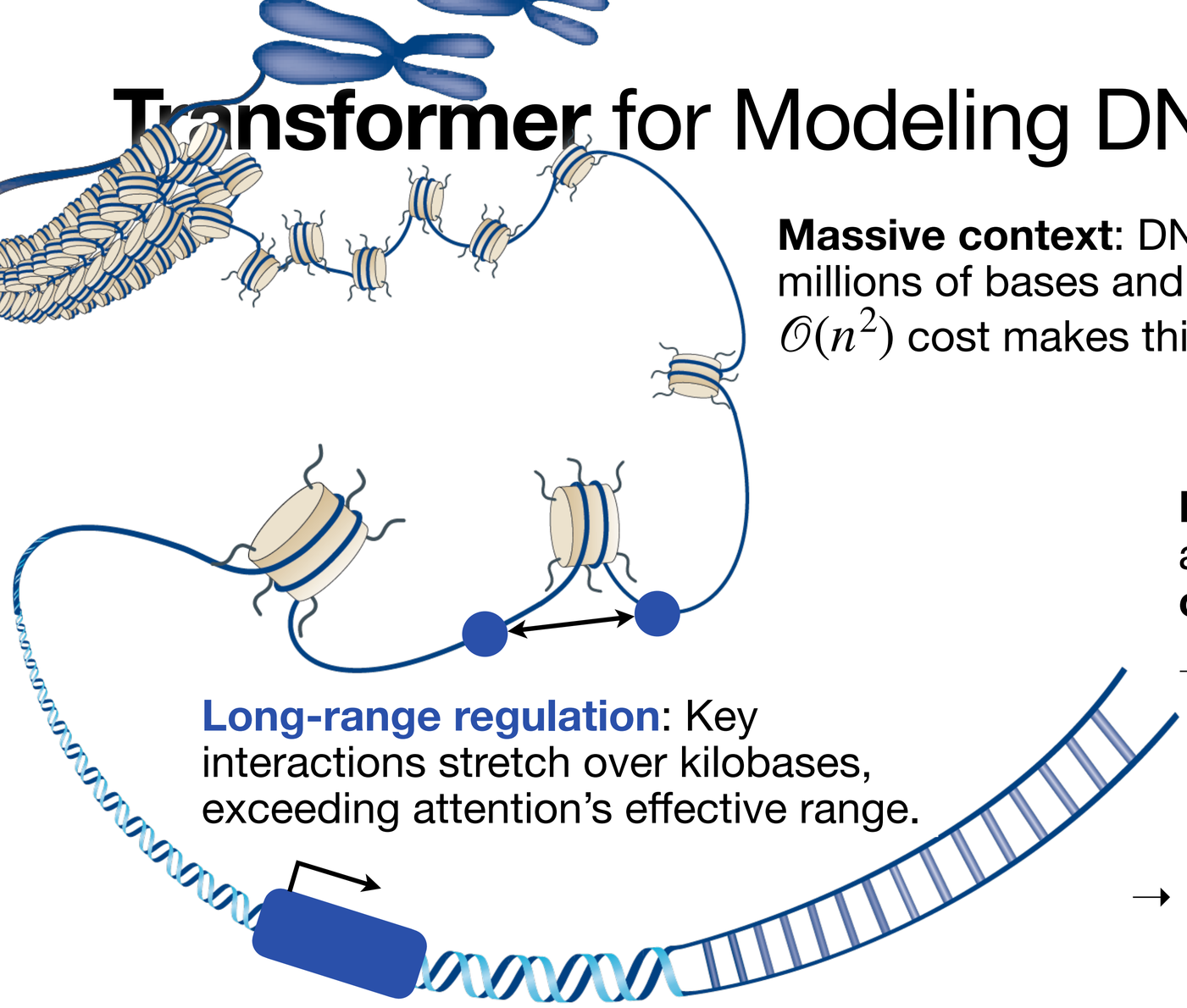
removed specialized attention blocks
but triangle updates retained
for geometric reasoning




Lesson learned!

More **general**, more **scalable**, closer to *original* Transformer design, and stripped of *handcrafted* evolutionary machinery.

# Transformer for Modeling DNA



**Massive context**: DNA spans millions of bases and attention's $\mathcal{O}(n^2)$ cost makes this infeasible.

**Long-range regulation**: Key interactions stretch over kilobases, exceeding attention's effective range.

**FMs for DNA** *replace* the Transformer's costly attention mechanism with **fast long convolutions and lightweight recurrence**:

→ allows efficient modeling of million-base DNA sequences while preserving long-range dependencies.

→ *in some settings, attention is not all you need.*

**Local structure**: Regulatory motifs are local, but attention wastes capacity on all-pair comparisons.

(Nguyen et al., 2025)

# Transformer and That's It?

Transformers proved *remarkably* flexible:
a single mechanism that scales well across text, vision, biology, and more.

*Yet* "**universality**" **comes with trade-offs**:

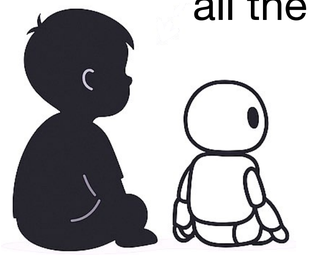
1. Everything must be tokenized — the world reduced to discrete symbols.
2. Models reason over static snapshots, not continuous change.
3. Learning remains passive and data-hungry.

**Humans learn differently:**

through *perception*, *interaction*, and *prediction*;

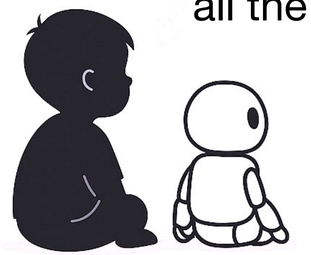
→ processing continuous sensory streams and simulating how the world evolves!

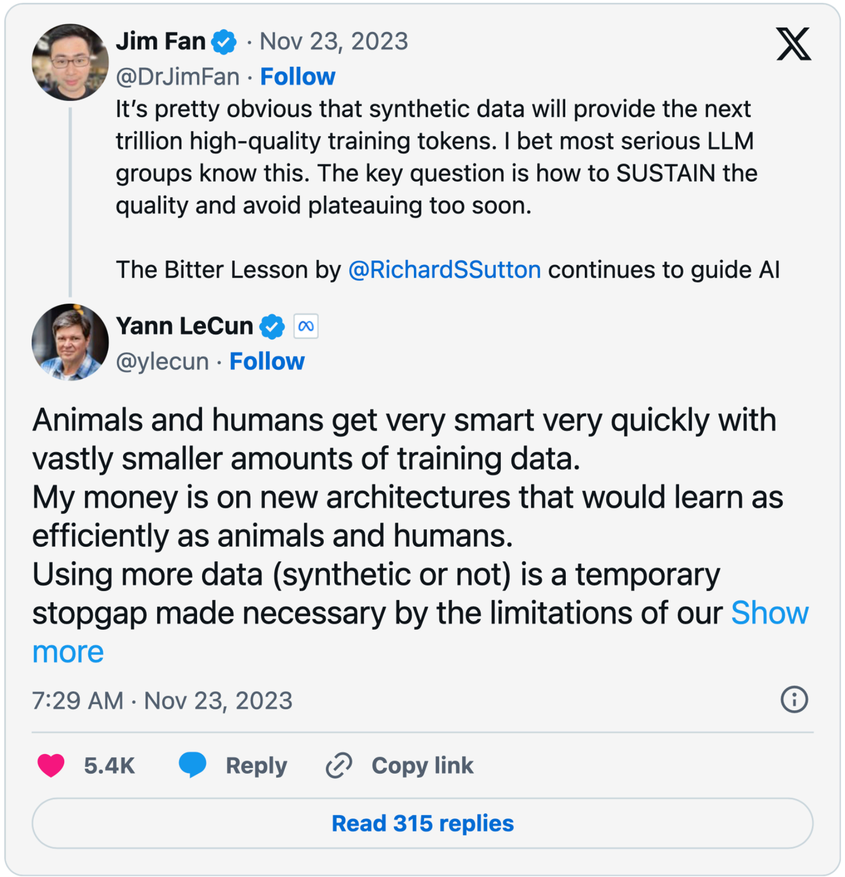
A 4-year-old child processes $10^{14}$ bytes through vision alone equal to all the text used to train GPT-4.



# What comes after the Transformer?

A 4-year-old child processes $10^{14}$ bytes through vision alone equal to all the text used to train GPT-4.






Ultimately, to model complexity of the world, we need to scale **multimodality** and **incorporate dynamics**.

Token universality built general representation, but real understanding requires:

1. **Perception**: learning from continuous sensory streams, not discrete tokens.
2. **Prediction**: modeling how states evolve, not just what co-occurs.
3. **Interaction**: learning through acting and feedback, not passive observation.

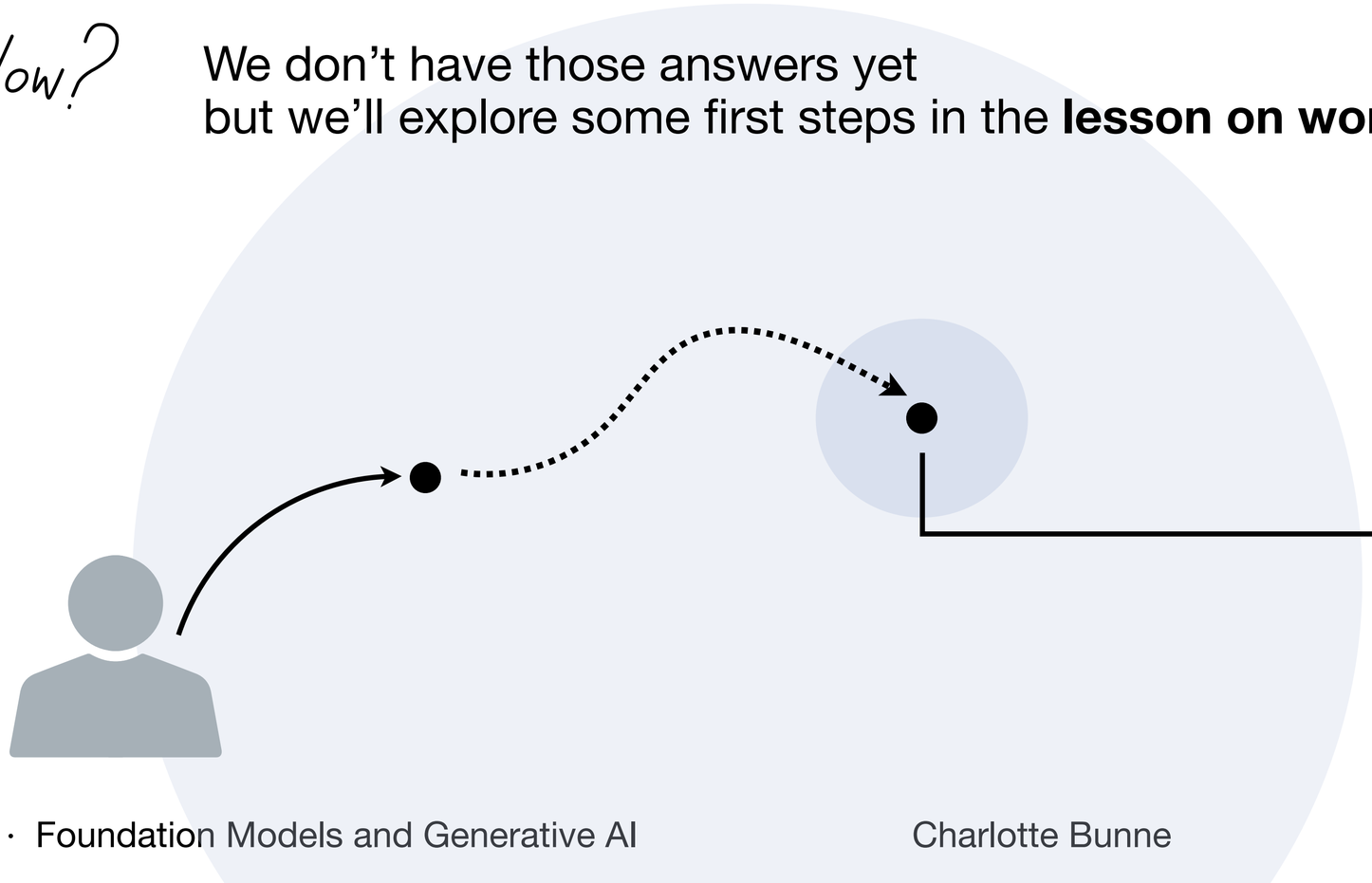
# Ideas for the Future?

BUNNE LAB

Join Us!

The **next generation of models** must move from static correlation to dynamic understanding — from *describing* the world to *simulating* it.

*How?* We don't have those answers yet
but we'll explore some first steps in the **lesson on world models.**



Lecture 12: World Models

mechanisms

Lecture 15: Outlook and Summary

# The Joint Embedding Predictive Architecture

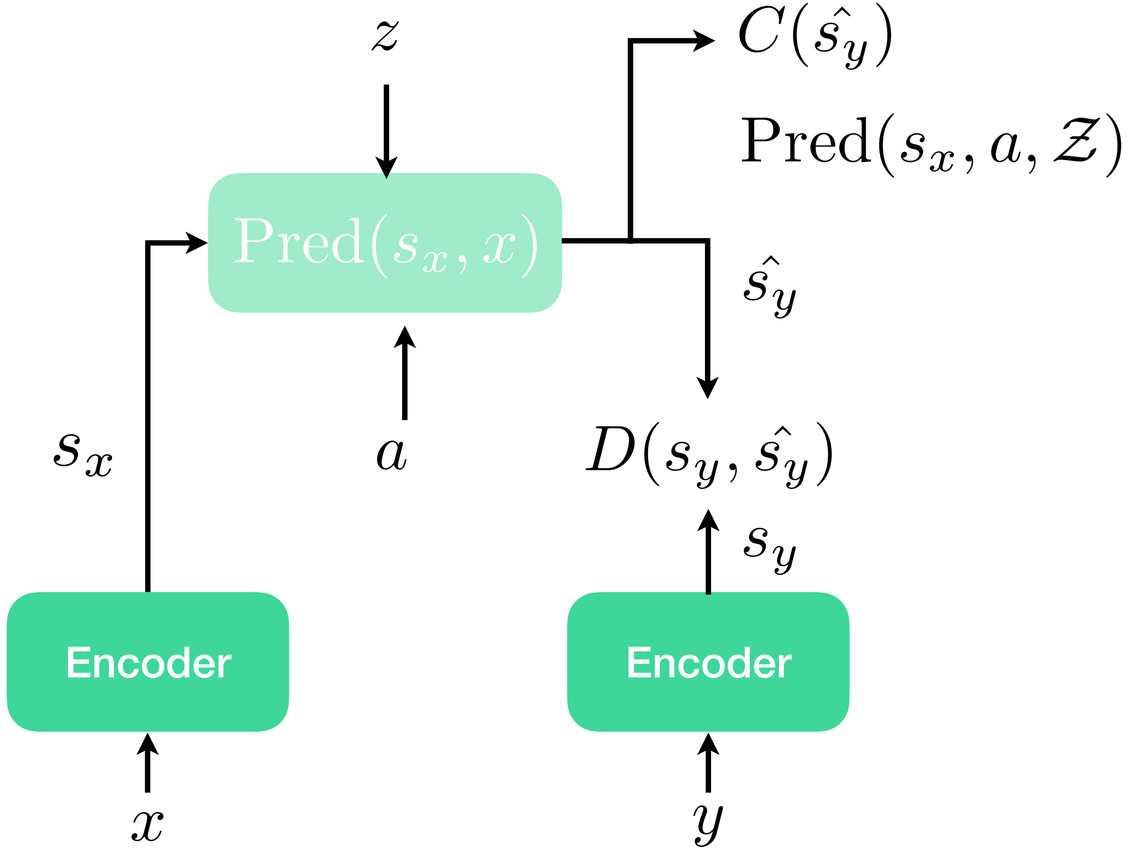
*Preview!*

Lecture 12: World Models

A counterproposal to Transformers! Architectures for world models.

- $x$ : observed past and present
- $y$ : future
- $a$ : action
- $z$: latent variable (*unknown*)
- $D(\cdot)$ : prediction cost
- $C(\cdot)$ : surrogate cost

→ JEPA predicts a representation of the future $s_y$ from a representation of the past $s_x$.



# Week 5's Papers

Papers are linked in Moodle.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Lukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* † illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Vaswani et al. "Attention Is All You Need."
Advances in Neural Information Processing Systems 30 (2017).

# Week 6's Exercise Sheet

Exercise 5 · Task 1

**Computational Cost of Multi-Head Self-Attention:**

Analyzes the *parameter count* and *computational cost* of a **single multi-head self-attention block**. It shows that the block contains four times the square of the model dimension in learnable weights—independent of the number of heads—and that the forward-pass cost scales roughly with both the sequence length and model dimension, dominated by the projection and attention matrix multiplications.

# Week 6's Code Demonstration

**Autoregressive MNIST with Transformers:**

Code Notebook 5 · Task 1

Implement and train a small **Decoder-only Transformer** for class-conditioned MNIST generation by **tokenizing images** into 2×2 binary patches, building key Transformer components (MLP, attention, residual blocks), and sampling new digits from the trained model.

→ Jupyter notebook exercise

CS-461

# Foundation Models and Generative AI

*Have a great week!*