

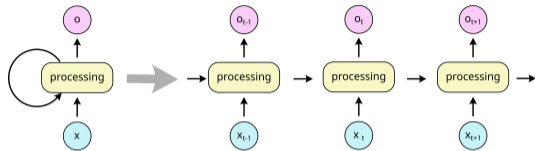
# Transformer Architectures

Paul J. Atzberger

260J: Machine Learning  
University of California Santa Barbara

# Introduction

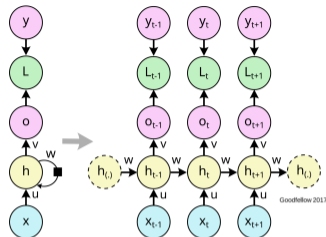
## Sequence Data Processing



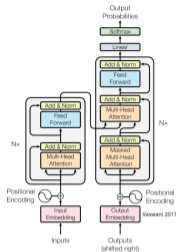
## Motivations and History

- **How can neural networks be used to process variable length sequential data**  $X = \{x_1, x_2, \dots, x_N\}$ ?
- **Tasks:** Natural Language Processing (NLP), Translation, Time-Series Prediction, and many other problems.
- **Recurrent Neural Networks (RNNs)** introduced in 1980's which feed output values back into the network when processing the sequence  $X$ .

## Recurrent Neural Networks (RNNs)



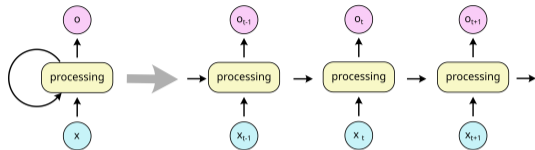
## Transformers



- **Trained with Back-Propagation-Through-Time (BPTT).**
- **However, BPTT exhibits exploding and vanishing gradients** as sequences become long.
- **Gating Units** introduced, like LSTMs and GRUs in 1990-2000's.
- **However, crucial information often dropped** for dependencies over long distances within sequences.
- **Recurrence is sequential** raising issues for parallelization.

# Introduction

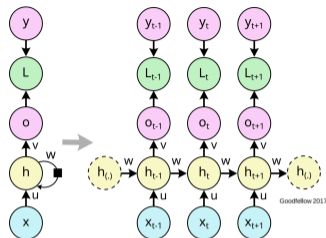
## Sequence Data Processing



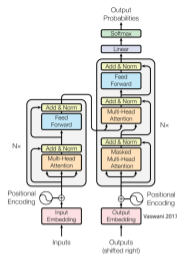
## Motivations and History

- **Path length** in back-propagation through the network processing steps impacts the dependencies and quality of gradients for learning.
- **Attention mechanisms** reduce path length and parallelize processing by providing soft weights over the previous sequence elements (introduced in 2010's).
- **'Attention is All You Need'** paper (2017) showed can drop recurrence and replace with attention mechanisms.

## Recurrent Neural Networks (RNNs)



## Transformers



- **Transformer** architecture introduced in (Vaswani 2017).
- **Big impact on scalability**, resulting in current era of Large Language Models (LLMs) and Agentic AI.
- **Applications:** Components crucial in Gemini, ChatGTP, Claude, Llama, and other LLMs.
- Active area of research and development.
- Many other potential applications.

# Transformers

**Transformer** neural network architectures process sequences  $X = \{x_1, x_2, \dots, x_N\}$  with attention mechanisms.

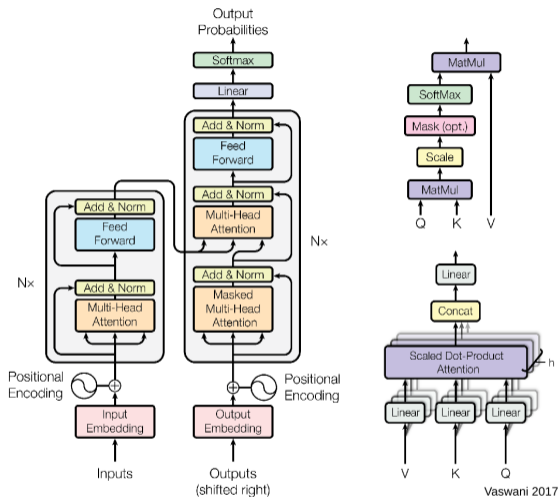
## Key Components

- Tokenization
- Embedding
- Attention mechanisms
- Positional encodings
- Encoder-Decoder transformer blocks

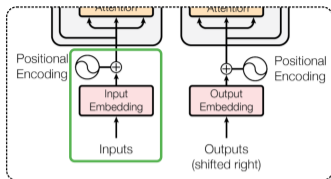
More details will be discussed on each of these parts.

**Many variants** are possible depending on choices.

**Applications:** NLP, genomics, computer vision, and many other tasks.

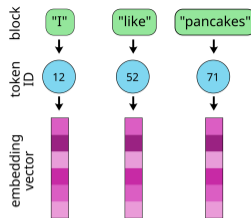


# Tokenization and Embeddings



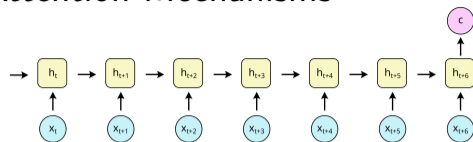
**Input sequence**  $X = \{x_1, x_2, \dots, x_N\}$  is treated as a string of symbols  $x_1 x_2 \dots x_N$  and must be broken up into parts.

- **Tokenization** breaks a string into blocks  $b_i$  mapping them to **tokens**  $\tau_i \in \mathcal{V}$  for **vocabulary**  $\mathcal{V}$ .
- **Example:** For natural language, individual words can be mapped to tokens.
- "I like pancakes"  $\rightarrow$  ["I", "like", "pancakes"].
- **Remark:** We see already it might be semantically useful to map "pancakes" to two tokens "pan" and "cakes."
- Many tokenizations use sub-words or patterns.
- Each token corresponds to an ID tag:  
 $\tau_i \in [0, 1, \dots, |V|]$ .

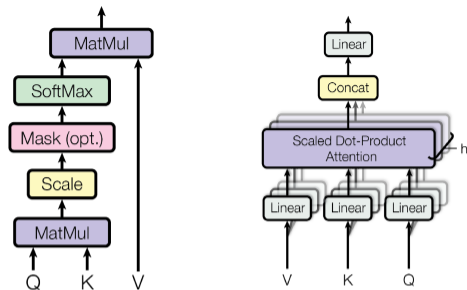


- **Tokens**  $\tau_i$  are then **embedded in a vector space**  $\tau_i \rightarrow v_i \in \mathbb{R}^{d_{\text{model}}}$ .
- Think of  $v_i$  as similar to a feature vector describing the meaning of token  $\tau_i$ .
- **Embeddings** are given by a list of row vectors  $E \in \mathbb{R}^{|V| \times d_{\text{model}}}$ .
- **Training used to obtain embeddings** from self-supervised learning (masking, next token prediction, and other tasks).
- **Example:** Masking task: "I \_\_\_ pancakes."

# Attention Mechanisms



- **Early approaches used fixed-length context vectors**  $\mathbf{c} \in \mathbb{R}^d$  to encode entire sequence  
 $X = \{x_1, x_2, \dots, x_{N_x}\}$ ,  $x_i \rightarrow \mathbf{h}_i$  with  
 $\mathbf{c} = f(\mathbf{h}_1, \dots, \mathbf{h}_{N_x})$ , (Cho 2014, Sutskever 2014).
- **However,  $\mathbf{c}$  has limited capacity** and drops important information, especially as the sequence length grows.
- **Alternative**, at each decoder step  $i$  compute a different context vector  $\mathbf{c}_i$  by attending to all the encoder steps  
 $\mathbf{c}_i = \sum_{j=1}^{N_x} \alpha_{ij} \mathbf{h}_j$  (Bahdanau 2014).
- **Uses learned  $\alpha_{ij}$**  with  $\alpha_i = (\alpha_1, \alpha_2, \dots, \alpha_{N_x}) \in \Delta^{N_x}$ .
- **Provides weights** for "soft-queries" over the sequence, now referred as attention.
- Weights aim to learn alignments with learned feature vectors.



- **Dot-Product Attention** introduced in (Vaswani 2017).
- **Similar to database look-up** with a **Q**: query vector, **K**: key vector, and **V**: value vector.
- **Multi-headed Attention** uses multiple Dot-Product Attention mechanisms.
- **Learnable linear operators** that can combine information from different dot-product attention queries.

# Attention Mechanisms

## Dot-Product Attention:

**Q**: query vector, **K**: key vector, and **V**: value vector.

Compute how well query **Q** matches with keys **K** and return averaging of the matching values **V**

$$\mathbf{H} = \text{attention}(\mathbf{Q}, \mathbf{V}, \mathbf{K}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}.$$

Normalize by  $1/\sqrt{d_k}$  the dot-product similarity to ensure sensitivity of softmax.

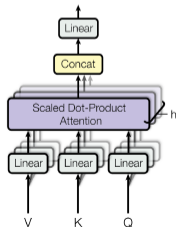
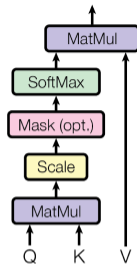
The  $d_k$  is dimension of rows of **K**.

## Multi-headed Attention:

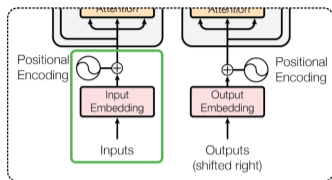
Perform  $n$  queries and combine the results. Generate queries and results by linearly mapping queries **Q** keys **K**, and values **V**.

$$\begin{aligned} \text{multihead}(\mathbf{Q}, \mathbf{V}, \mathbf{K}) &= \text{concat}(\mathbf{H}_1, \mathbf{H}_2, \dots, \mathbf{H}_n) \mathbf{W}^O \\ \mathbf{H}_i &= \text{attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{V}\mathbf{W}_i^V, \mathbf{K}\mathbf{W}_i^K\right). \end{aligned}$$

Learned weights are  $\mathbf{W}_i^Q \in \mathbb{R}^{d_m \times d_k}$ ,  $\mathbf{W}_i^K \in \mathbb{R}^{d_m \times d_k}$ ,  $\mathbf{W}_i^V \in \mathbb{R}^{d_m \times d_v}$ , and  $\mathbf{W}_i^O \in \mathbb{R}^{nd_v \times d_m}$ .



# Positional Encoding (PE)



- **Attention mechanisms** such as dot-product are permutation equivariant.

- **Word order matters:**

"Cats eat mice" vs "Mice eat cats."

- **Approach:** modify the embedding vectors  $\mathbf{v}_i$ .

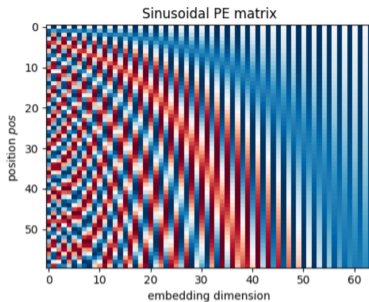
- **Sinusoidal Positional Encoding (PE)**

$\mathbf{w} = (w_1, w_2, \dots, w_{d_{\text{model}}})$  with  $w_j = PE(\text{pos}, j)$   
where

$$PE(\text{pos}, 2i) = \sin(\omega_i \text{pos})$$

$$PE(\text{pos}, 2i + 1) = \cos(\omega_i \text{pos})$$

$$\omega_i = 1/10000^{2i/d_{\text{model}}}.$$



- **Motivation:**

(large  $\omega_i$ ,  $i \ll d_{\text{model}}/2$ ): fine-grained sensitivity to pos.

(small  $\omega_i$ ,  $i \sim d_{\text{model}}/2$ ): coarse-grained less sensitive pos.

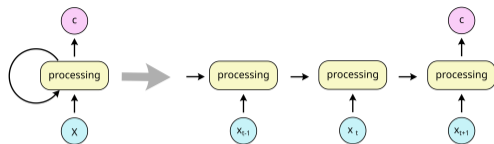
- **Provides multi-scale** characterization of a token's position.

- **Additive encoding** by modifying token embedding to  $\tilde{\mathbf{v}}_k = \mathbf{v}_k + \mathbf{w}_k$ .

- **Embedding vectors** typically have  $d_{\text{model}} = 512$ . Large enough to extract info from original vector and position.

- **Linearly extractable relative positions**  $j - i$  using  $PE_j - PE_i$ .

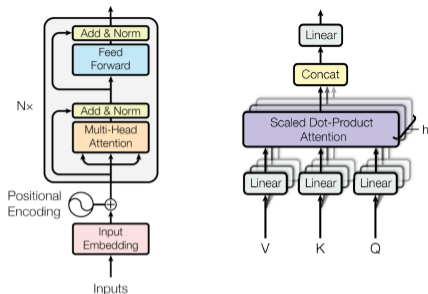
# Encoder Transformers



An **encoder** is used to process the input sequence  $X$  to obtain context information  $c$ .

## Encoder Transformer Steps:

1. **Tokenization and embedding** of the input sequence including positional encoding.
2. **Multi-headed Attention** processes in parallel the embedding vector sequence.
4. **Layer Normalization (LN)** is applied component-wise to the output to obtain  $\text{LN}(\mathbf{h}) = \frac{\mathbf{h} - \mu}{\sigma + \epsilon} \odot \gamma + \beta$  where  $\mu = \frac{1}{d} \sum_j h_j$ ,  $\sigma^2 = \frac{1}{d} \sum_j (h_j - \mu)^2$ , and  $\gamma, \beta \in \mathbb{R}^d$  are a learned scaling and shift.
3. **Skip connection** is used to also pass a copy of the embedding sequence to add to the output  $\mathbf{h}' = \mathbf{h} + \text{LN}(\text{layer}(\mathbf{h}))$ .
5. **Repeat  $n$ -times**: The output is fed to the next multi-headed attention layer repeating at step 2.

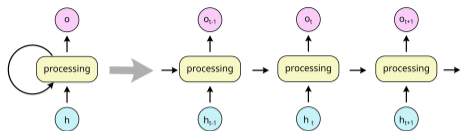


6. **Feed-Forward Neural Network (FNN)** processes the results component-wise.
7. **Skip connection** passes a copy of the input to add to the output of the FNN.
8. **Layer Normalization (LN)** is again applied to obtain the **final output  $c$** .

## Encoder Transformers

- **Provides rich context aware information** for the sequence.
- **Applications**: BERT uses only an encoder to perform classification, sentiment analysis, and tagging. Also, used in encoder-decoder transformers.

# Decoder Transformers



**Decoder** uses the context information  $c$  and attention over a target sequence to generate an output sequence.

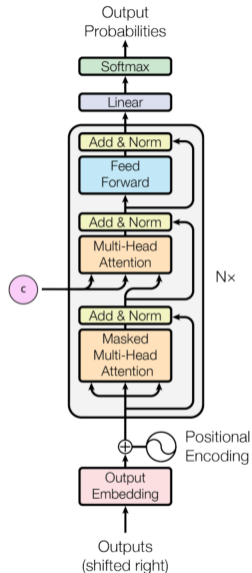
- **During Training** uses teacher-forcing protocols with a specified target sequence.
- **During Inference** the target sequence can be generated dynamically in stages.
- **Rightward shift** used for context window of the target sequence (remove the first token).

## Decoder Transformer Steps:

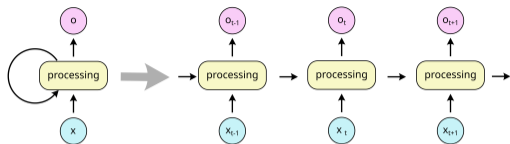
1. **Tokenization and embedding** of the input sequence including positional encoding.
2. **Multi-headed Attention (initial)** processes the target sequence.
3. **Layer Normalization (LN)** is applied component-wise.
4. **Multi-headed Attention (context info)** passes both the context vector  $c$  and target sequence encoded output to MHA layer.

## Decoder Transform Steps:

5. **Layer Normalization (LN)** is applied component-wise.
  6. **Skip connection** passes copy of inputs to the output.
  7. **Repeat  $n$ -times** processing steps from 4.
  8. **Results are processed component-wise by a Feed-Forward Neural Network (FNN).**
  9. **Layer Normalization (LN)** is applied component-wise.
  10. **Skip connection** passes a copy of the input to add to the output of the FNN.
  11. **Linear Layer** is used to process the output, modulate effective temperature, and other behaviors.
  12. **Softmax** gives final predicted probabilities over the symbols of the output vocabulary.
- **Generates** sequences from context information.
  - **Applications:** Autoregressive tasks, GPTs, conversational AI, Co-Pilot, code generation.



# Encoder-Decoder Transformers

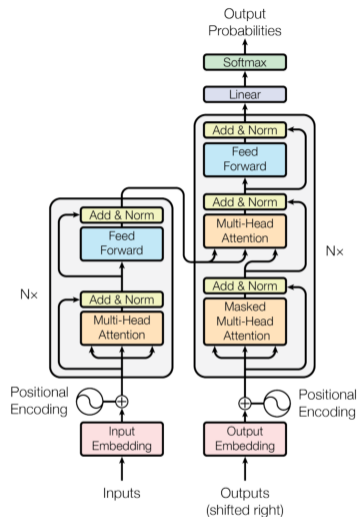


**Encoder-Decoder Transformer** architectures process sequences  $X = \{x_1, x_2, \dots, x_{N_x}\}$  using both types of transformers. Useful for generative and autoregressive tasks.

## Transformer Steps:

1. **Initialization** with a starting sequence or symbol  $X$  that is tokenized and embedded.
2. **Encoder Transformer** extracts context information  $c$  from the input.
3. **Decoder Transformer** generates next token which is added to the target sequence on the right.
4. **Repeat:** The process can be iterated repeatedly from step 2. Generates sequence of tokens until a terminal symbol is generated.

- **Applications:** Fundamental components of current LLMs and Agentic AI. Question & Answer AI, Code Generation, Gemini, ChatGPT, Claude.



# Example: Sequence Classification

## Sequence Classes

**class A:** {5, 5, 5, 1, 1}, {5, 5, 6, 3, 2},  
{7, 3, 6, 5}

**class B:** {2, 3, 5, 1, 1}, {3, 1, 1, 1, 4},  
{1, 3, 4, 5}

**Task:** Classify a sequence into class A or class B.

Basic self-contained demonstration showing steps in practice.

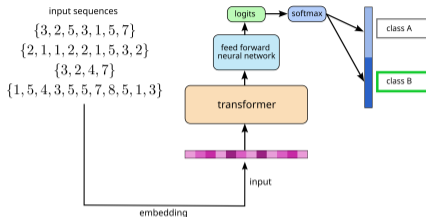
**Class A:** Majority of entries  $\geq 5$

**Class B:** Majority of entries  $< 5$

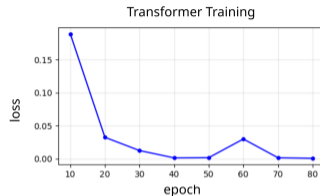
## Processing Steps

- **Tokenization of the sequence** by breaking it up into elements.
- **Tokens are mapped to embedding vectors** to obtain a sequence.
- **Transformer** processes sequence to determine logits.

## Classifier



## Training



- **Logits** used in softmax to determine probabilistic predictions over the classes.

**Results:** Test accuracy 99%+ after  $\sim 80$  epochs of training.

# Example: Time-Series Forecasting

Dataset

Target time-series are sinusoidal sums of the form

$$f(t) = \sum_i a_i \sin(\omega_i x).$$

**Task:** Given a window of past observations predict the future values

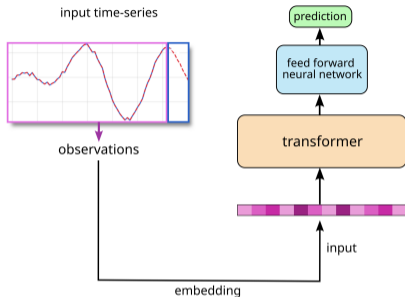
Basic self-contained demonstration showing steps in practice.

## Processing Steps

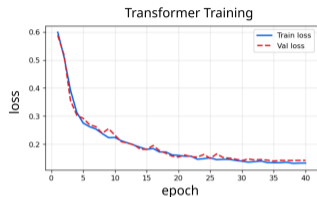
- **Tokens are mapped to embedding vectors** to obtain a sequence.
- **Encoder Transformer** used for the past observation values.
- **Transformer** predicts all at once the future values.

**Results:** Shown are forecasts for a few input windows. Initial results, only 40 epochs.

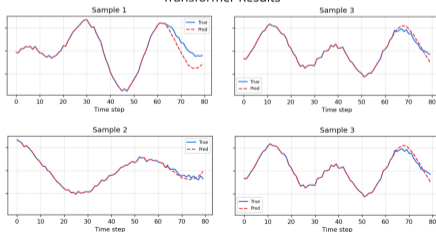
Predictor



Training and Results



Transformer Results



# Visual Transformers (ViTs)

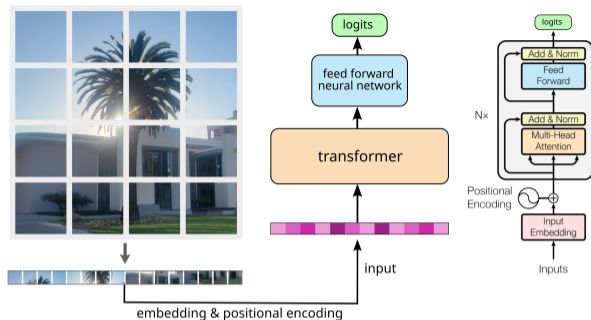
**Visual Transformers (ViTs)** process images in blocks mapped to a sequence of embedding vectors (Ramachandran 2019).

**Motivated** by human and animal perception that uses eye movement saccades to process surrounding visual stimuli sequentially.

**ViT processes** image embedded token sequence using transformers to produce the logits for probabilistic prediction  $\mathbf{p} = \text{softmax}(\text{logits})$  with  $p_i = [\mathbf{p}]_i$  for class  $i$  or output token  $\tau_i$ .

**ViTs provide scalable alternatives to CNNs** through use of attention mechanisms, but may require more data.

- **Attention Mechanisms** provide different inductive biases that are less image-centric than CNNs.

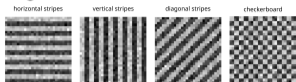


- **ViTs and CNNs can be combined** for smaller training datasets:
  - CNNs better capture fine-grained information.
  - Transformers can better integrate global information.
- **ViT-only architectures show state-of-the-art performance** on sufficiently large datasets (14M+ images) (Dosovitskiy 2021).

**Applications:** Image Classification, Object Identification, Caption Generation, and other tasks.

# Example: ViT Image Classifier

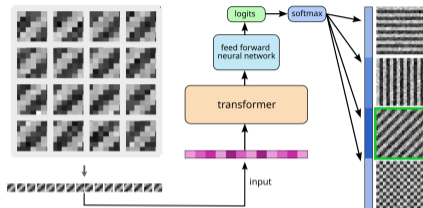
## Image Classes



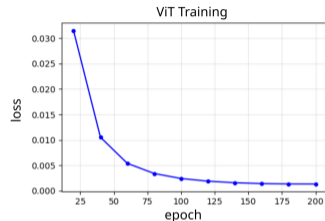
**Task:** Classify an input image into one of four classes.

Basic demonstration for using ViTs in practice.

## Classifier



## Training



## ViT Processing Steps

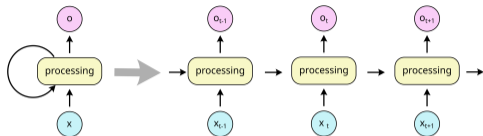
- **Tokenization of the image** by breaking them up into small patches.
- **Tokens are mapped to embedding vectors** to obtain a sequence.
- **Transformer** processes sequence to determine logits.

- **Logits** used in softmax to determine probabilistic predictions over the image classes.

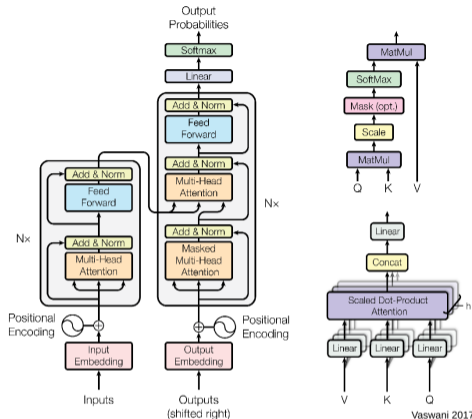
**Results:** Test accuracy 99%+ after ~ 200 epochs of training.

# Conclusions

## Sequential Data Processing



- **Transformers** provide neural network architectures for processing variable length sequential data  $X = \{x_1, x_2, \dots, x_{N_x}\}$ .
- **Provides scalable alternatives** to Recurrent Neural Networks (RNNs).
- **ViTs for processing image data** provide alternatives to Convolutional Neural Networks (CNNs).
- **Many other variants** are also possible by making different choices for architecture components or combined with other methods.
- **Active on-going research and development.**



- **Big impact on** current era of Large Language Models (LLMs) and Agentic AI.
- **Applications:** Fundamental components in Gemini, ChatGTP, Claude, Llama, and other LLMs.