Transformer and Large Language Models

Instructor: Lei Wu [1]

Mathematical Introduction to Machine Learning

Peking University, Fall 2025

[1]School of Mathematical Sciences; Center for Machine Learning Research

Outline

1. Transformer Architecture
2. Large Language Models (LLMs)
3. Vision Transformers (ViT)

Outline

Transformer

Transformers

- were introduced in *Attention is all you need* (Vaswani et al., NeurIPS 2017);
- have revolutionized NLP, CV, robotics and many applications;
- have enabled the creation of powerful LLMs such as GPT-4;
- hold the promise of unlocking the potential for AGI (artificial general intelligence).

Sequence Modeling

Consider a simple block for sequence modeling:

$$(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \xmapsto{\mathcal{T}} (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type blocks for many times (aka layers).

Sequence Modeling

Consider a simple block for sequence modeling:

$$(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \overset{\mathcal{T}}{\longmapsto} (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type blocks for many times (aka layers).

- Recurrence

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$

Sequence Modeling

Consider a simple block for sequence modeling:

$$(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) \stackrel{\mathcal{T}}{\longmapsto} (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type blocks for many times (aka layers).

- Recurrence

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$

- Convolution (local connection)

$$\mathbf{y}_i = f(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}).$$

Sequence Modeling

Consider a simple block for sequence modeling:

$$(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \xmapsto{\mathcal{T}} (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type blocks for many times (aka layers).

- Recurrence

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$

- Convolution (local connection)

$$\mathbf{y}_i = f(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}).$$

- Attention (**selective** weighted average):

$$\mathbf{y}_i = f\left(\sum_{j=1}^{n} w_{i,j}(X)\mathbf{x}_j\right),$$

where $W(X) = (w_{i,j}(X)) \in \mathbb{R}^{n\times n}$ satisfies

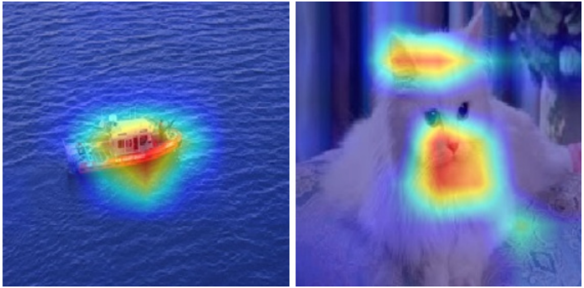
$$\sum_{j=1}^{n} w_{i,j}(X) = 1, \qquad W_{i,j}(X) \geq 0 \ \forall i, j \in [n].$$

Attention Mechanism (Cont'd)

We often call $w_{i,j}(X)$'s the **attention score** and we want

the attention scores $(w_{i,1}(X), w_{i,2}(X), \cdots, w_{i,n}(X))$ to be **sparse** (i.e., **selective**).

- Attention in vision modeling:



Attention Mechanism (Cont'd)

Attention in machine translation (**cross attention**) [2]:

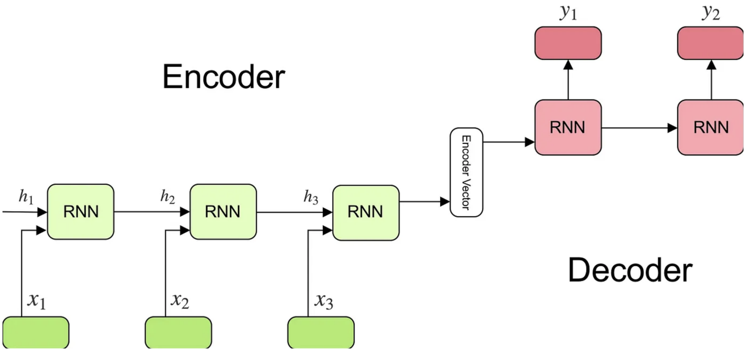


Figure 1: See a better animation in this link.

[2] Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015.

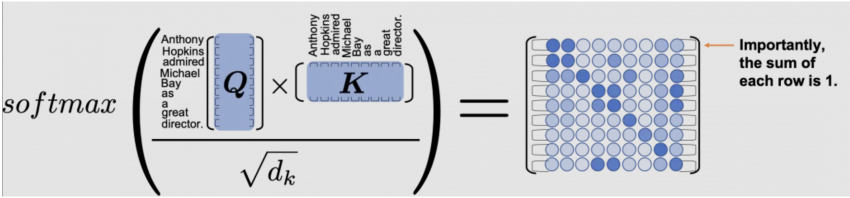
Self-Attention via Dot-Product

- Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$ be our input sequence. We often call $\{\mathbf{x}_i\}$ **tokens**.
- A self-attention $\mathbb{A} : \mathbb{R}^{d \times n} \mapsto \mathbb{R}^{n \times n}$ outputs an attention-score map $P = \mathbb{A}(X)$. The most popular choice is

$$\mathbb{A}_{W_K, W_Q}(X) = \sigma\left(\frac{1}{\sqrt{d_{\text{key}}}} (W_K X)^\top (W_Q X) \right) \in \mathbb{R}^{n \times n},$$

where

- $W_K, W_Q \in \mathbb{R}^{d_{\text{key}} \times d}$ are the key and query weight matrices, which are learned from data.
- σ denotes the softmax normalization performed in a column-wise manner, ensuring the column represent a selective average.



Self-Attention via Dot-Product (Cont'd)

- The dot-products are implemented in a **token-wise manner** (can be naively paralleled):

$$\mathbf{k}_i = W_K \mathbf{x}_i, \quad \mathbf{q}_j = W_Q \mathbf{x}_j \text{ for } i, j \in [n]$$

$$(\mathbb{A}_{W_K, W_Q}(X))_{i,j} = \frac{e^{\mathbf{k}_i^\top \mathbf{q}_j}}{\sum_{i'=1}^{n} e^{\mathbf{k}_{i'}^\top \mathbf{q}_j}}$$

- The attention scores are determined by the **dot-product correlation** among tokens.
- A single-head attention layer $\mathtt{SA} : \mathbb{R}^{d \times n} \mapsto \mathbb{R}^{d \times n}$ is given as follows

$$\mathtt{SA}_{W_K, W_Q, W_V}(X) = V\sigma(K^\top Q),$$

where Q, K, V are called the query, key, value matrices, respectively and given by

$$Q = W_Q X \in \mathbb{R}^{d \times n}, \quad K = W_K X \in \mathbb{R}^{d \times n}, \quad V = W_V X \in \mathbb{R}^{d \times n}.$$

A Transformer Block

- A transformer block defines a sequence-to-sequence map

$$X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^{d\times n} \mapsto Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n) \in \mathbb{R}^{d\times n}.$$

- This maps consists of two blocks:

$$Y = \mathbb{F}(X + \mathtt{MHA}(X)),$$

  where

  - **Multi-head attention** (MHA)

$$\mathtt{MHA}(X) := \sum_{h=1}^{H} W_O^h \mathtt{SA}^h(X).$$

  - **Tokenwise feed-forward networks** (FFN):

$$\mathbb{F}(Z) := (h(\mathbf{z}_1), h(\mathbf{z}_2), \ldots, h(\mathbf{z}_n)) \in \mathbb{R}^{d\times n}.$$

    In practice, $h : \mathbb{R}^d \mapsto \mathbb{R}^d$ is often chosen to be a two-layer MLP with hidden size d_{FF}.

$$h(\mathbf{z}) = W_1^\top \mathrm{ReLU}(W_2\mathbf{z} + \mathbf{b}),$$

    where $W_1, W_2 \in \mathbb{R}^{d_{\mathrm{FF}}\times d}$ and $\mathbf{b} \in \mathbb{R}^d$.

Transformer

- **Input:** Linear embedding to change the dimension of each token.

$$X^{(0)} = VX \text{ with } V \in \mathbb{R}^{d_{\mathrm{model}} \times d}.$$

- **Main block**:

$$X^{\ell} = \mathbb{F}^{(\ell)}(X^{(\ell-1)} + \mathtt{MHA}^{(\ell)}(X^{(\ell-1)})), \quad 1 \le \ell \le L.$$

- **Ouput:** The output format depends on the tasks. In classification, we may

$$f(X) = p(\mathbf{x}_1^{(L)}),$$

where p can be either a linear layer or small MLP.

- Architecure hyperparameters: d_{model}, H, L, d_{key} , d_{FF}. In practice, a common choice $d_{\mathrm{FF}} = 4d_{\mathrm{model}}, d_{\mathrm{key}} = d_{\mathrm{model}}/H$.

Cost Analysis of Transformer

In practice, it is common to choose

$$d_{\text{key}} = d_{\text{model}}/H, \qquad d_{\text{FF}} = 4d_{\text{model}}.$$

- **Storage cost** (per Transformer layer):

$$4d_{\text{model}}^2 \text{ (MHA)} + 8d_{\text{model}}^2 \text{ (FFN)}.$$

- **Computation cost:**
  - **MHA:**

$$\underbrace{4nd_{\text{model}}^2}_{\text{Q/K/V projections}} + \underbrace{\color{red}{d_{\text{model}}\, n^2}}_{\textbf{attention scores}}$$

  - **FFN:**

$$8d_{\text{model}}^2 n.$$

- **Parallelism:** Tokenwise operations are fully parallelizable.
- **Critical bottleneck:** The total cost grows **quadratically in the sequence length** n.
  This quadratic attention cost becomes the dominant bottleneck—especially for inference.

Absolute Positional Embedding (APE)

Transformers without positional information are **permutation-equivariant**: reordering the input tokens simply reorders the outputs. To encode order, we need to inject positional information.

The most natural way of injecting position information is using **absolute positional embedding** (APE): let $\mathbf{r}_i \in \mathbb{R}^d$ denote the **position information** for token i:

$$\mathbf{x}_i \to \mathbf{x}_i + \mathbf{r}_i,$$

- Learnable APE: $\mathbf{r}_i$ are parameters to be learned.
- One-hot APE: $\mathbf{r}_i = \mathbf{e}_i$ where $\mathbf{e}_i$ is the one-hot label with 1 in the i-th coordinates and zero else.
- Sinusoidal APE:

$$\mathbf{r}_i = \Big(\sin(i), \cos(i), \sin(i/c), \cos(i/c), \dots, \sin(i/c^{2k/d}), \cos(i/c^{2k/d})\Big) \in \mathbb{R}^d,$$

  where c is constant, e.g. 1000.

Absolute Positional Embedding (APE)

Transformers without positional information are **permutation-equivariant**: reordering the input tokens simply reorders the outputs. To encode order, we need to inject positional information.

The most natural way of injecting position information is using **absolute positional embedding** (APE): let $\mathbf{r}_i \in \mathbb{R}^d$ denote the **position information** for token i:

$$\mathbf{x}_i \to \mathbf{x}_i + \mathbf{r}_i,$$

- Learnable APE: $\mathbf{r}_i$ are parameters to be learned.
- One-hot APE: $\mathbf{r}_i = \mathbf{e}_i$ where $\mathbf{e}_i$ is the one-hot label with 1 in the i-th coordinates and zero else.
- Sinusoidal APE:

  $$\mathbf{r}_i = \Big(\sin(i), \cos(i), \sin(i/c), \cos(i/c), \ldots, \sin(i/c^{2k/d}), \cos(i/c^{2k/d})\Big) \in \mathbb{R}^d,$$

  where c is constant, e.g. 1000.

However, APE is rarely used in practice anymore due to:

- APE can not handle input sequence longer than that used in training.
- In many real problems, it is "relative distance" that matters.

Relative Positional Embedding (RPE)

- **Additive RPE**: Let $E = (W_K X)^\top (W_Q X) \in \mathbb{R}^{n\times n}$ be the pre-softmax attention logits. Then, we inject relative position information by

$$\mathbb{A}(X) = \sigma(E - P),$$

where $P = (h(j-i))_{i,j} \in \mathbb{R}^{n\times n}$.

Relative Positional Embedding (RPE)

- **Additive RPE**: Let $E = (W_K X)^\top (W_Q X) \in \mathbb{R}^{n \times n}$ be the pre-softmax attention logits. Then, we inject relative position information by

$$\mathbb{A}(X) = \sigma(E - P),$$

where $P = (h(j-i))_{i,j} \in \mathbb{R}^{n \times n}$.

  - **T5 RPE:**

$$h(t) = \begin{cases} |t| & \text{if } |t| \le B/2 \\ \frac{B}{2} + \frac{B}{2}\left\lfloor \frac{\log(\frac{|t|}{B/2})}{\log(\frac{D}{B/2})} \right\rfloor & \text{if } \frac{B}{2} \le |t| \le D \\ B-1 & \text{if } |t| \ge D \end{cases},$$

    where where B is the total number of relative position buckets and D is the maximum distance we explicitly bucketize.
  - **Alibi RPE:** $h(t) = -\alpha|t| + \beta$, where the α and β can be either learnable or fixed.

Relative Positional Embedding (RPE)

- **Additive RPE**: Let $E = (W_K X)^\top (W_Q X) \in \mathbb{R}^{n\times n}$ be the pre-softmax attention logits. Then, we inject relative position information by

$$\mathbb{A}(X) = \sigma(E - P),$$

  where $P = (h(j-i))_{i,j} \in \mathbb{R}^{n\times n}$.
  - **T5 RPE:**

$$h(t) = \begin{cases} |t| & \text{if } |t| \le B/2 \\ \frac{B}{2} + \frac{B}{2}\left\lfloor \frac{\log(\frac{|t|}{B/2})}{\log(\frac{D}{B/2})} \right\rfloor & \text{if } \frac{B}{2} \le |t| \le D \\ B-1 & \text{if } |t| \ge D \end{cases},$$

    where where B is the total number of relative position buckets and D is the maximum distance we explicitly bucketize.
  - **Alibi RPE:** $h(t) = -\alpha|t| + \beta$, where the α and β can be either learnable or fixed.
- Currently, the most popular RPE is the rotary positional embedding (**RoPE**), which has been adopted in nearly all large language models.

Readings

- The original paper https://arxiv.org/abs/1706.03762
- Annotated Transformer https://jalammar.github.io/illustrated-transformer/
- Illustrated Transformer https://poloclub.github.io/transformer-explainer/

Outline

1. Transformer Architecture
2. Large Language Models (LLMs)
3. Vision Transformers (ViT)

Tokenization

Before applying embeddings, natural language text must first be **tokenized**.

- **Definition:** Tokenization is the process of converting raw text into a sequence of discrete units called *tokens*.
- **Purpose:** It transforms unstructured text into a form that models can process, enabling downstream operations such as embedding, attention, and sequence modeling.

Tokenization

Before applying embeddings, natural language text must first be **tokenized**.

- **Definition:** Tokenization is the process of converting raw text into a sequence of discrete units called *tokens*.
- **Purpose:** It transforms unstructured text into a form that models can process, enabling downstream operations such as embedding, attention, and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words and punctuation.

  `"Transformers are amazing!" -> ["Transformers", "are", "amazing", "!"]`

Tokenization

Before applying embeddings, natural language text must first be **tokenized**.

- **Definition:** Tokenization is the process of converting raw text into a sequence of discrete units called *tokens*.
- **Purpose:** It transforms unstructured text into a form that models can process, enabling downstream operations such as embedding, attention, and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words and punctuation.

  ```
  "Transformers are amazing!" -> ["Transformers", "are", "amazing", "!"]
  ```

- **Subword-level Tokenization:** Decomposes words into smaller units (subwords), allowing robust **handling of rare words, morphology, and open vocabularies**.

  ```
  "Transformers are amazing!" ->
            ["Trans", "##form", "##ers", "are", "amaz", "##ing", "!"]
  ```

Tokenization

Before applying embeddings, natural language text must first be **tokenized**.

- **Definition:** Tokenization is the process of converting raw text into a sequence of discrete units called *tokens*.
- **Purpose:** It transforms unstructured text into a form that models can process, enabling downstream operations such as embedding, attention, and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words and punctuation.

  ```
  "Transformers are amazing!" -> ["Transformers", "are", "amazing", "!"]
  ```

- **Subword-level Tokenization:** Decomposes words into smaller units (subwords), allowing robust **handling of rare words, morphology, and open vocabularies**.

  ```
  "Transformers are amazing!" ->
            ["Trans", "##form", "##ers", "are", "amaz", "##ing", "!"]
  ```

- **In practice:** Modern LLMs rely on standardized subword tokenizers, for example those implemented in Hugging Face's `transformers` library.

Training LLMs Requires Many Stabilization Tricks

- **Scaled dot-product attention** Scaling by $1/\sqrt{d_k}$ controls the variance of attention scores:

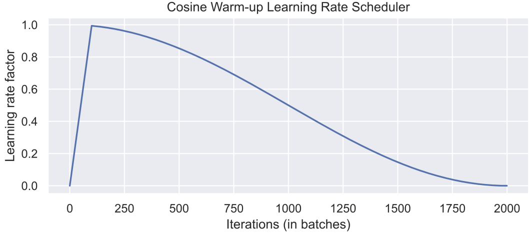
$$\mathbb{A}_{W_K,W_Q}(X) = \sigma\left(\frac{1}{\sqrt{d_k}}(W_K X)^\top (W_Q X)\right) \in \mathbb{R}^{n\times n}.$$

- **Normalization** (LayerNorm or RMSNorm) with residual connections: These are essential for stable signal propagation across depth.

$$\tilde{X}^{(\ell-1)} = \mathrm{Norm}\left(X^{(\ell-1)}\right),$$

$$X^{(\ell)} = \tilde{X}^{(\ell-1)} + \mathbb{F}\left(\mathtt{MHA}(\tilde{X}^{(\ell-1)})\right).$$

- **AdamW** (Adam+decoupled weight decay) optimizer with $\beta_1 = 0.9$ and $\beta_2 \in \{0.95,\ 0.98\}$, together with gradient clipping.
- **Learning-rate scheduling:** warmup followed by cosine decay.



Bidirectional Encoder Representations from Transformers (BERT)

- Developed by Google (Devlin et al., 2018).
- **Bidirectional encoder**: Unlike traditional left-to-right or right-to-left language models, BERT uses *deep bidirectional attention*, allowing it to incorporate context from both sides simultaneously.

Bidirectional Encoder Representations from Transformers (BERT)

- Developed by Google (Devlin et al., 2018).
- **Bidirectional encoder**: Unlike traditional left-to-right or right-to-left language models, BERT uses *deep bidirectional attention*, allowing it to incorporate context from both sides simultaneously.

Pre-training Tasks

- Masked Language Modeling (MLM): Randomly masks a subset of tokens and predicts the missing ones.
- Next Sentence Prediction (NSP): Predicts whether two sentences appear consecutively in the original text.

(Later work shows that NSP provides little benefit and can often be removed.)

Bidirectional Encoder Representations from Transformers (BERT)

- Developed by Google (Devlin et al., 2018).
- **Bidirectional encoder**: Unlike traditional left-to-right or right-to-left language models, BERT uses *deep bidirectional attention*, allowing it to incorporate context from both sides simultaneously.

Pre-training Tasks

- Masked Language Modeling (MLM): Randomly masks a subset of tokens and predicts the missing ones.
- Next Sentence Prediction (NSP): Predicts whether two sentences appear consecutively in the original text.

(Later work shows that NSP provides little benefit and can often be removed.)

Fine-tuning

- The pre-trained encoder can be adapted to downstream tasks such as question answering, sentiment analysis, text classification, etc.

Generative Pre-trained Transformer (GPT)

- **Next-token prediction** (autoregressive model):

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log P_{\theta}(x_i | x_1, \ldots, x_{i-1}).$$

Generative Pre-trained Transformer (GPT)

- **Next-token prediction** (autoregressive model):

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log P_\theta(x_i | x_1, \ldots, x_{i-1}).$$

- Text Generation:

text $= [\langle bos \rangle]$ or [some context]

while True:

  logit $=$ decoder(embed(text))

  index $=$ top(logit$[-1]$)

  token $=$ vocabulary(index)

  if token $== \langle eos \rangle$:

    break

  text.append(token)

return text

Emergent Abilities: In-Context Learning and Prompting

- Traditional pre-train $\rightarrow$ fine-tune pipelines require updating model parameters.
- GPT's autoregressive training objective leads to **emergent abilities** that BERT-style models largely lack.
- **In-Context Learning (ICL)**: The model can learn from examples provided in the input.

```
In the following lines, the symbol -> represents a simple mathematical operation.
100 + 200 -> 301
838 + 520 -> 1359
343 + 128 -> 472
647 + 471 -> 1119
64 + 138 -> 203
498 + 592 ->
```

Answer:

```
1091
```

- **Prompting**: Solve tasks by giving instructions or demonstrations without parameter updates.

Next-Token Prediction Training Yields Emergent Abilities

- **Training objective:** GPT is trained to model

$$P(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

which requires predicting the next token conditioned on *any* prefix.

Next-Token Prediction Training Yields Emergent Abilities

- **Training objective:** GPT is trained to model

  $$P(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

  which requires predicting the next token conditioned on *any* prefix.
- **Prefixes = implicit tasks:** During training, prefixes may contain:
  - instructions,
  - demonstrations,
  - question–answer pairs,
  - multi-step reasoning patterns.

  The model must interpret these patterns to predict the next token correctly.

Next-Token Prediction Training Yields Emergent Abilities

- **Training objective:** GPT is trained to model

$$P(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

  which requires predicting the next token conditioned on *any* prefix.
- **Prefixes = implicit tasks:** During training, prefixes may contain:
  - instructions,
  - demonstrations,
  - question–answer pairs,
  - multi-step reasoning patterns.

  The model must interpret these patterns to predict the next token correctly.
- **Task inference emerges:** To minimize training loss, the model implicitly learns to:
  - infer the task from the prefix,
  - adapt behavior based on examples in the input,
  - follow natural-language instructions.

Next-Token Prediction Training Yields Emergent Abilities

- **Training objective:** GPT is trained to model

$$P(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

  which requires predicting the next token conditioned on *any* prefix.
- **Prefixes = implicit tasks:** During training, prefixes may contain:
  - instructions,
  - demonstrations,
  - question–answer pairs,
  - multi-step reasoning patterns.

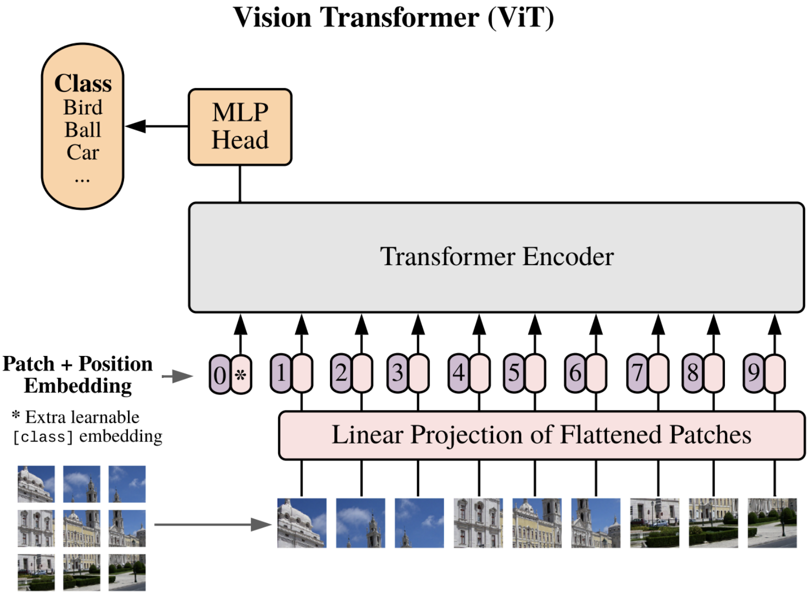
  The model must interpret these patterns to predict the next token correctly.
- **Task inference emerges:** To minimize training loss, the model implicitly learns to:
  - infer the task from the prefix,
  - adapt behavior based on examples in the input,
  - follow natural-language instructions.
- **Resulting abilities:**
  - **In-Context Learning (ICL)**: learning from examples in the prompt;
  - **Prompting**: solving diverse tasks without parameter updates;
  - **Generalization beyond language**: symbolic reasoning, coding, planning, etc.

Remark

GPT and its focus on next-token prediction have fundamentally transformed how pre-trained models are utilized, marking a significant step toward AGI. The transition from BERT to GPT represents a **major breakthrough** in this evolution.

Outline

Vision Transformer (ViT)



Summary

- **Transformer Architecture** Self-attention + MHA + FFN form a fully parallel sequence model; positional encodings inject order information.

Summary

- **Transformer Architecture** Self-attention + MHA + FFN form a fully parallel sequence model; positional encodings inject order information.
- **Large Language Models** GPT-style autoregressive training + modern optimization lead to in-context learning and prompting.

Summary

- **Transformer Architecture** Self-attention + MHA + FFN form a fully parallel sequence model; positional encodings inject order information.
- **Large Language Models** GPT-style autoregressive training + modern optimization lead to in-context learning and prompting.
- **Vision Transformer (ViT)** Applying Transformer to image patches shows the architecture's universality.