Yale

Transformers

CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Questions

- How to summarize what it means for an ML system to be **trusted**?

- Name one of the four major characteristics of a **trustworthy** ML system according to the book's opinion.

  Explain what does it mean and why it matters

- Have you noticed any news, articles, policies, events that have implications in trustworthy deep learning in recent years?

Readings

- Readings are updated on the website (syllabus page)
- **Readings:**
  - [2401.05561] TrustLLM: Trustworthiness in Large Language Models
- This lecture is not explicitly tested
  - But in future lectures we will assume knowledge of this when developing trustworthy components on top of Transformers

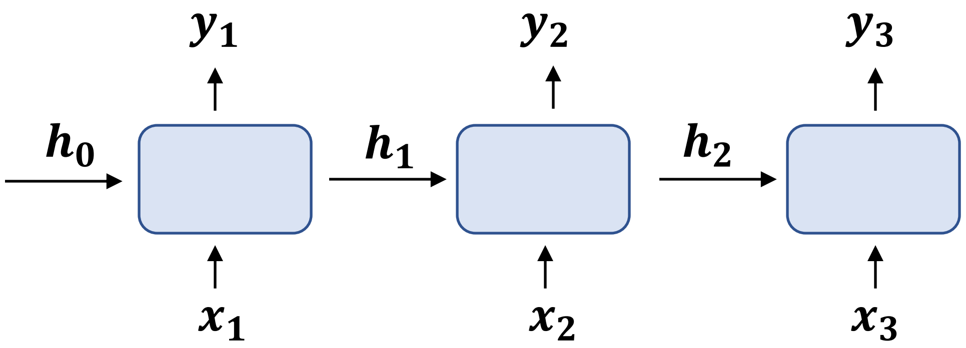
Outline of Today's Lecture

1. Self-Attention and Transformers

2. Transformers for (Large) Language Models (LLMs)

3. Transformers in Other Modalities

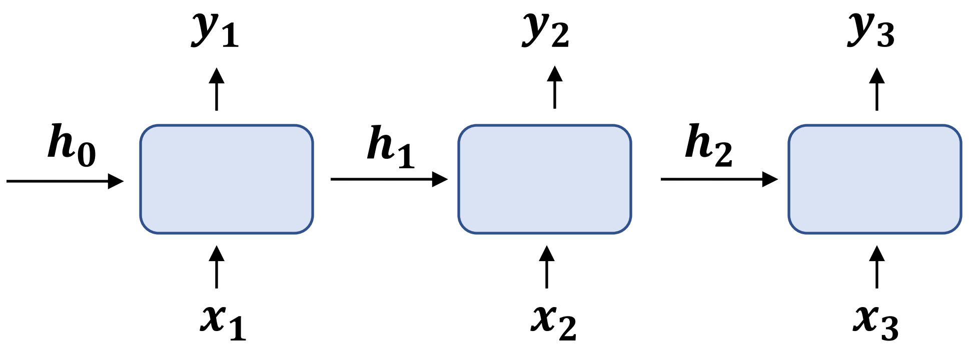
Sequence Learning



$$h_i = f_W(x_i, h_{i-1}), \; y_i = f_Y(h_i)$$

What are the issues and challenges of RNNs?

Sequence Learning

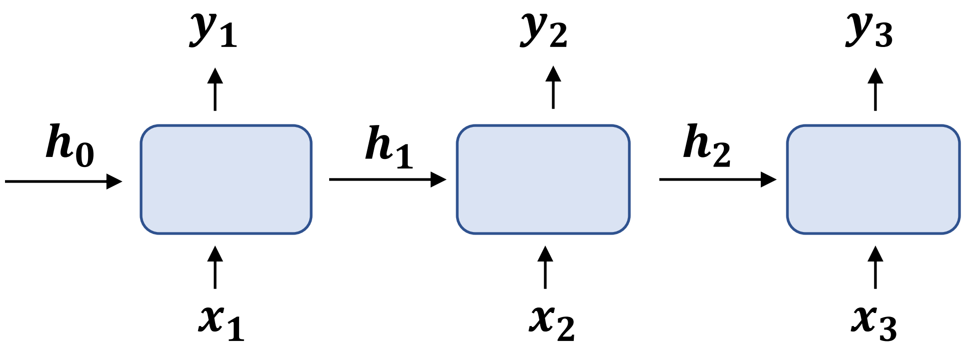


$$h_i = f_W\,(x_i\,,h_{i-1}),\;\; y_i = f_Y\,(h_i)$$

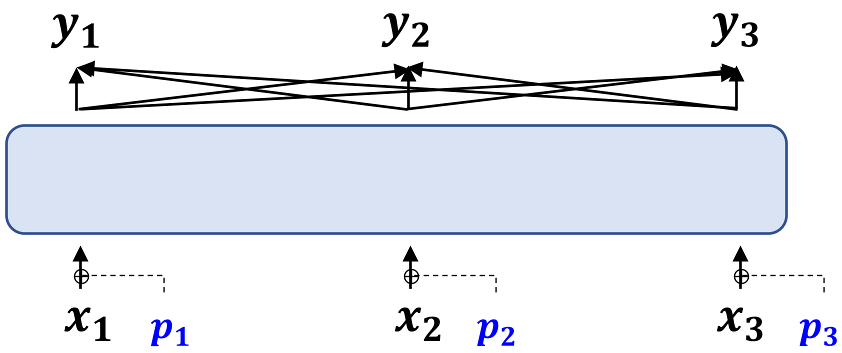
Problems of RNNs

- Sequential computation prevents parallelization
- Capacity of handling long sequences
- Mainly focusing on modeling recurrence
    - does not capture other correlations (hierarchical, long-range, polysemy….) well

Sequence Learning



$h_i = f_W\,(x_i\,,h_{i-1}),\; y_i = f_Y\,(h_i)$



y_i = self-att $([x_i + p_i]_i)$

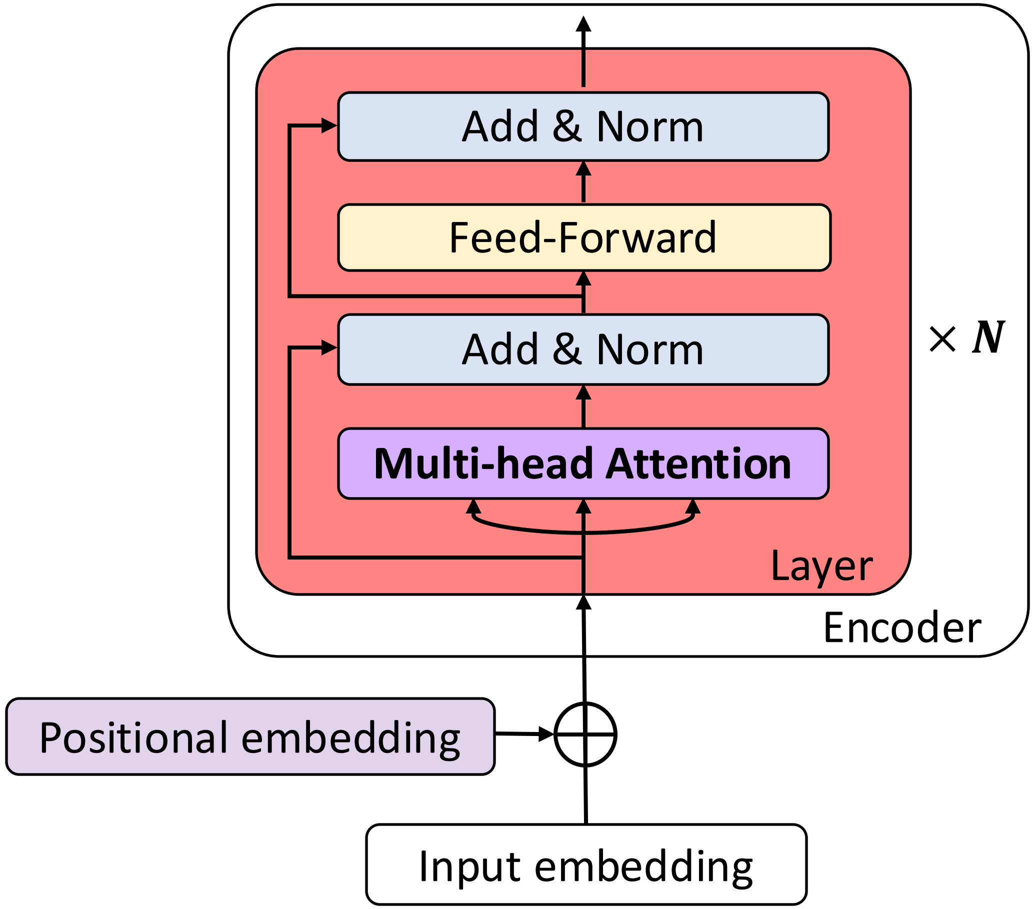
Problems of RNNs

1. parallelization
2. long sequences
3. only recurrence

Solutions by Transformers

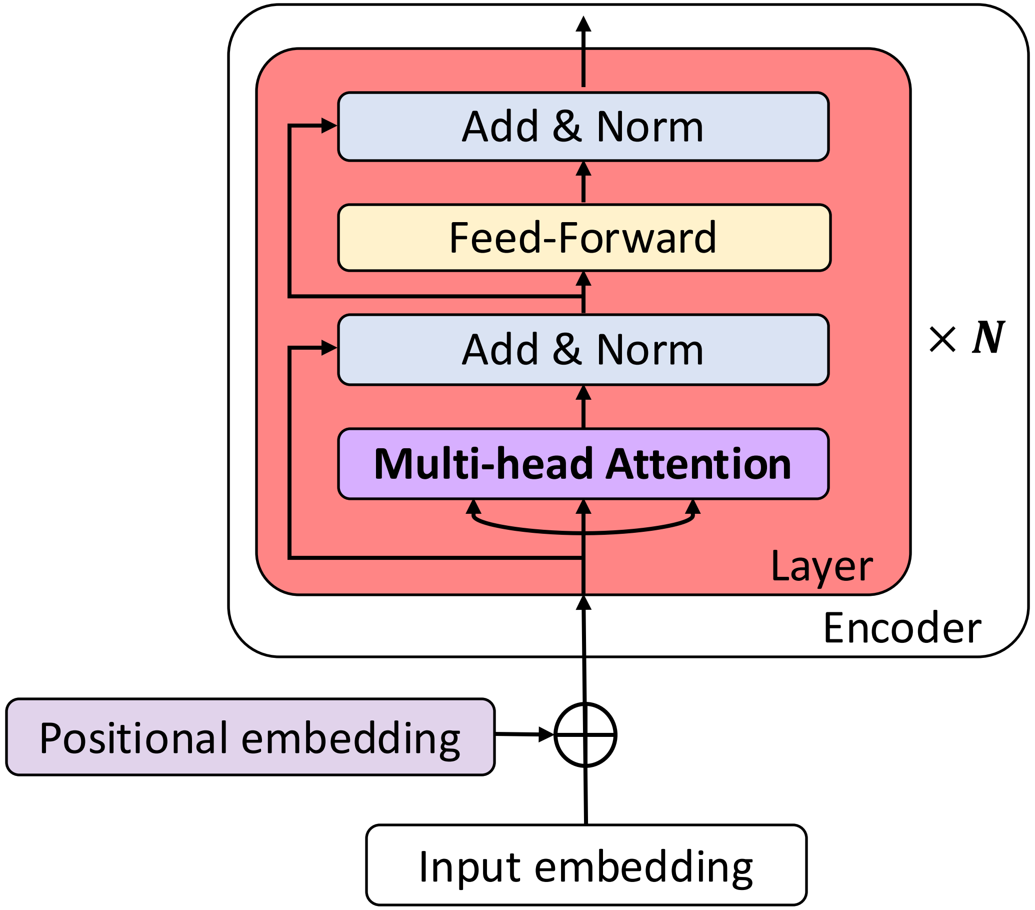
1. **Parallel input**: Input All tokens at the same time
2. **Self-Attention**: Enable attention in long-range
3. **Positional Embeddings p_i**: Model all possible correlations

Transformers — Overview



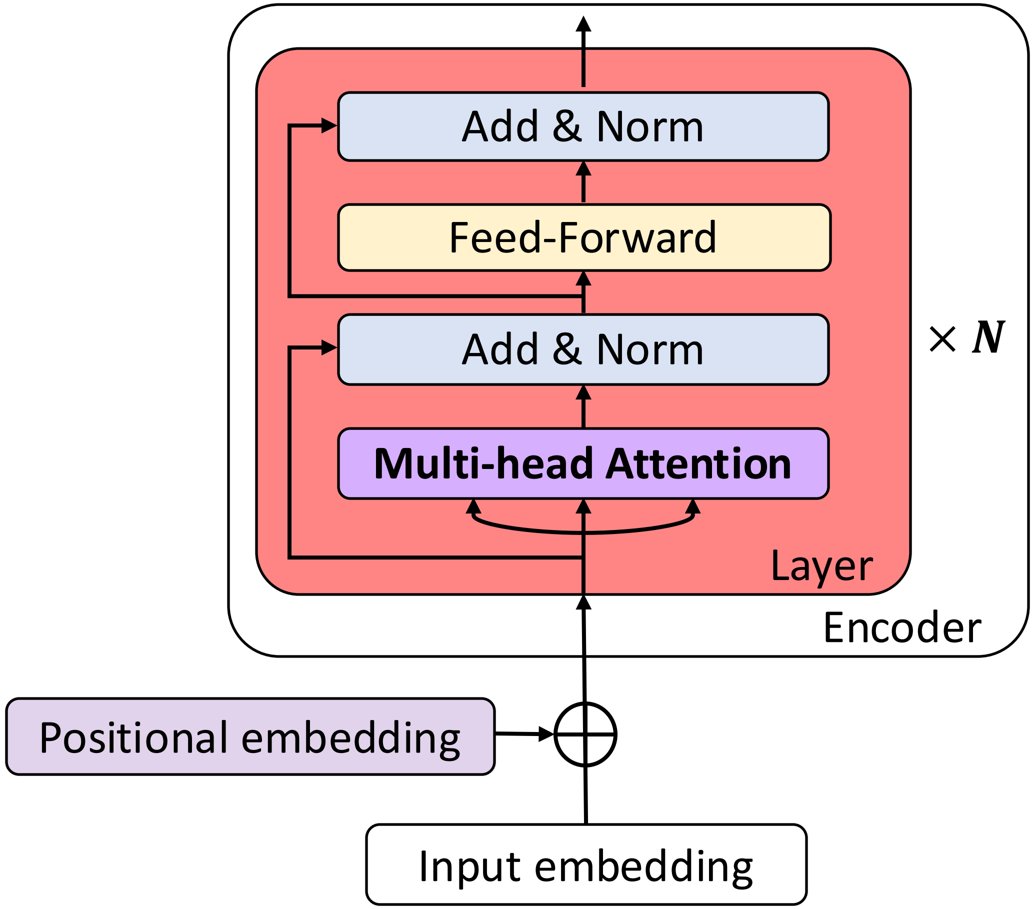
- **Original paper**: Attention is all you need [Vaswani et al., 2017].
- **Key component**: Multi-head self-attention
- **Other components** of a transformer layer: layer normalization, skip connection, position-wise feed-forward layer (FFN, or MLP)
- **Model usage**: Pre-training followed by fine-tuning. The transfered model can be:
  - **Encoder-only** (e.g BERT)
  - **Encoder-Decoder** (e.g BART)
  - **Decoder-only** (e.g GPT)
  - We will show an example later

Transformers — Overview



- **Model usage**: Pre-training followed by fine-tuning. The transfered model can be:
  - **Encoder-only** (e.g BERT)
    - Many-to-one classification / regression
    - Sentiment classification, document classification ...
    - Word / Sentence embeddings for downstream tasks (e.g. recommender system)
  - **Encoder-Decoder** (e.g BART)
  - **Decoder-only** (e.g GPT)
  - We will show an example later

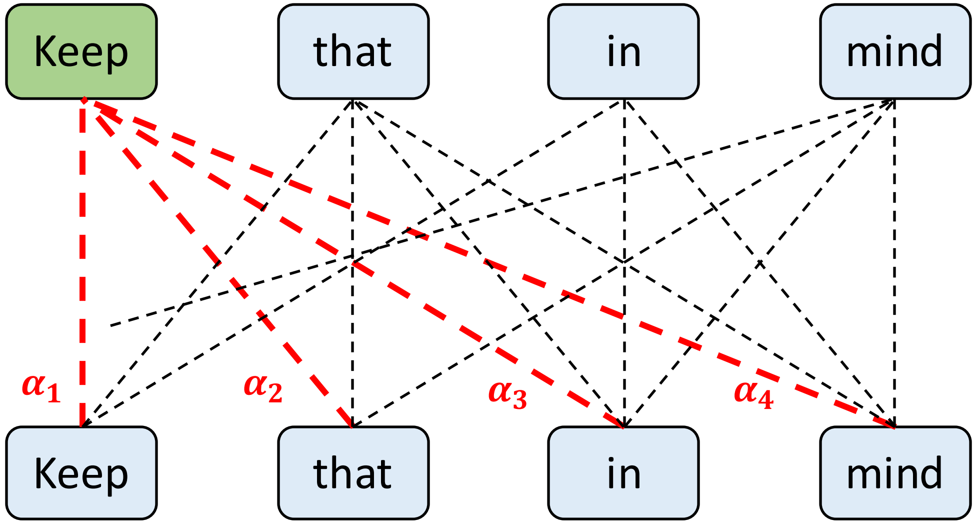
Transformers — Overview



- **Model usage**: Pre-training followed by fine-tuning. The transfered model can be:
  - **Encoder-only** (e.g BERT)
  - **Encoder-Decoder** (e.g BART)
    - Many-to-many use cases
    - Summarization, translation, style transfer ...
  - **Decoder-only** (e.g OpenAI GPT)
    - One-to-many use cases
    - Image / text / code generation, dialogue systems ...
    - GPT-3/4 based apps

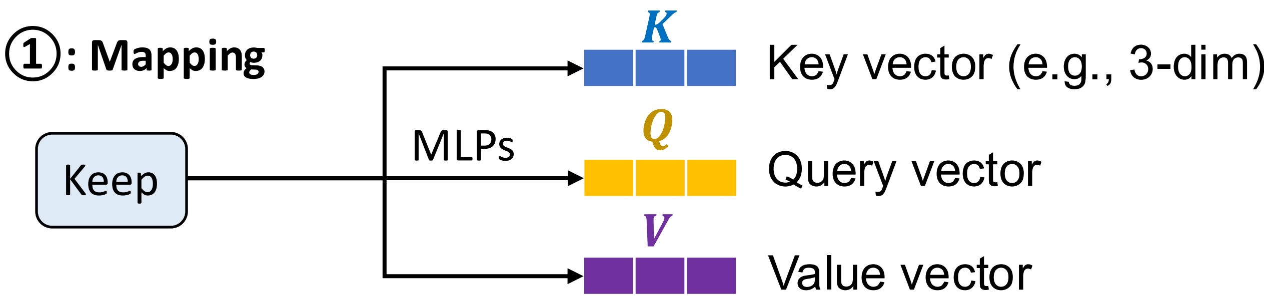
Transformers — Self-Attention (1/5)

Example:



input sequence

Step ①: Mapping



Step ②: Attention

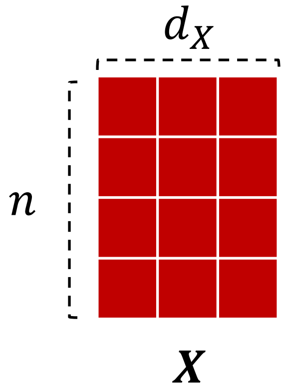
$$\alpha_1, \alpha_2, \alpha_3, \alpha_4 = \mathbf{Softmax}\left(\underset{\text{Keep}}{Q} \times \underset{\text{Keep}}{K} \; \underset{\text{that}}{K} \; \underset{\text{in}}{K} \; \underset{\text{mind}}{K} \right)$$

Step ③: Update

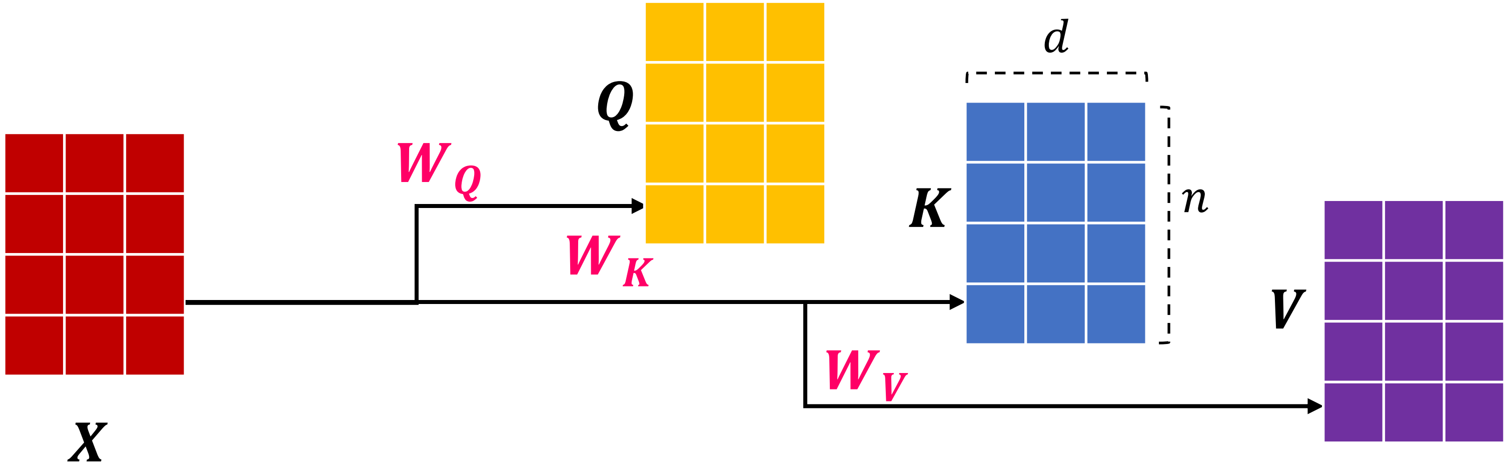
$$\underset{\text{Keep}}{V'} = \alpha_1 \times \underset{\text{Keep}}{V} + \alpha_2 \times \underset{\text{that}}{V} + \alpha_3 \times \underset{\text{in}}{V} + \alpha_4 \times \underset{\text{mind}}{V}$$

Transformers — Self-Attention (2/5)

- Formally, given an input sequence $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}] \in \mathbb{R}^{n \times d_X}$

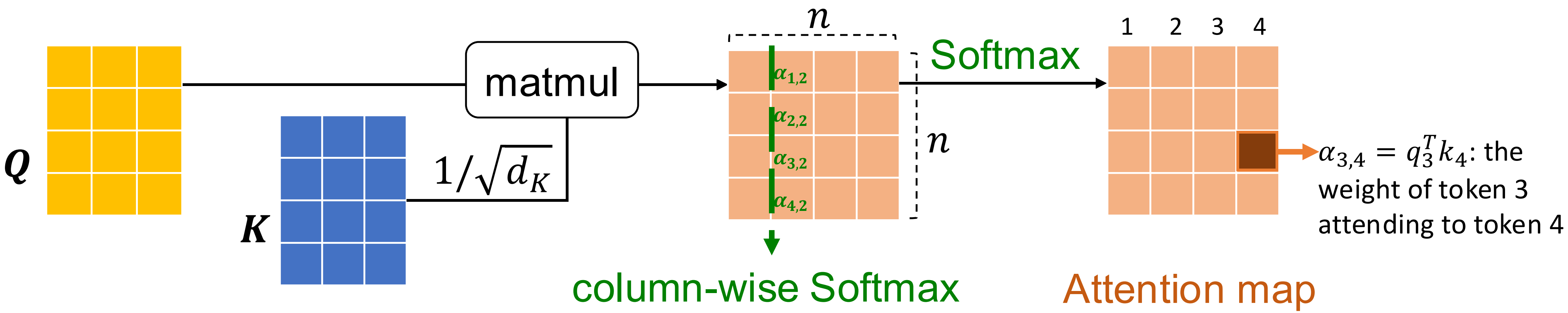


- Step ①: Query $\boldsymbol{Q} = \boldsymbol{XW_Q}$, Key $\boldsymbol{K} = \boldsymbol{XW_K}$, Value $\boldsymbol{V} = \boldsymbol{XW_V}$
  - $W_K \in \mathbb{R}^{d_X \times d_K}$, and thus $\boldsymbol{K} \in \mathbb{R}^{n \times d_K}$
  - We require $d_K = d_Q$, for simplicity, we set $d_K = d_Q = d_V \coloneqq d$



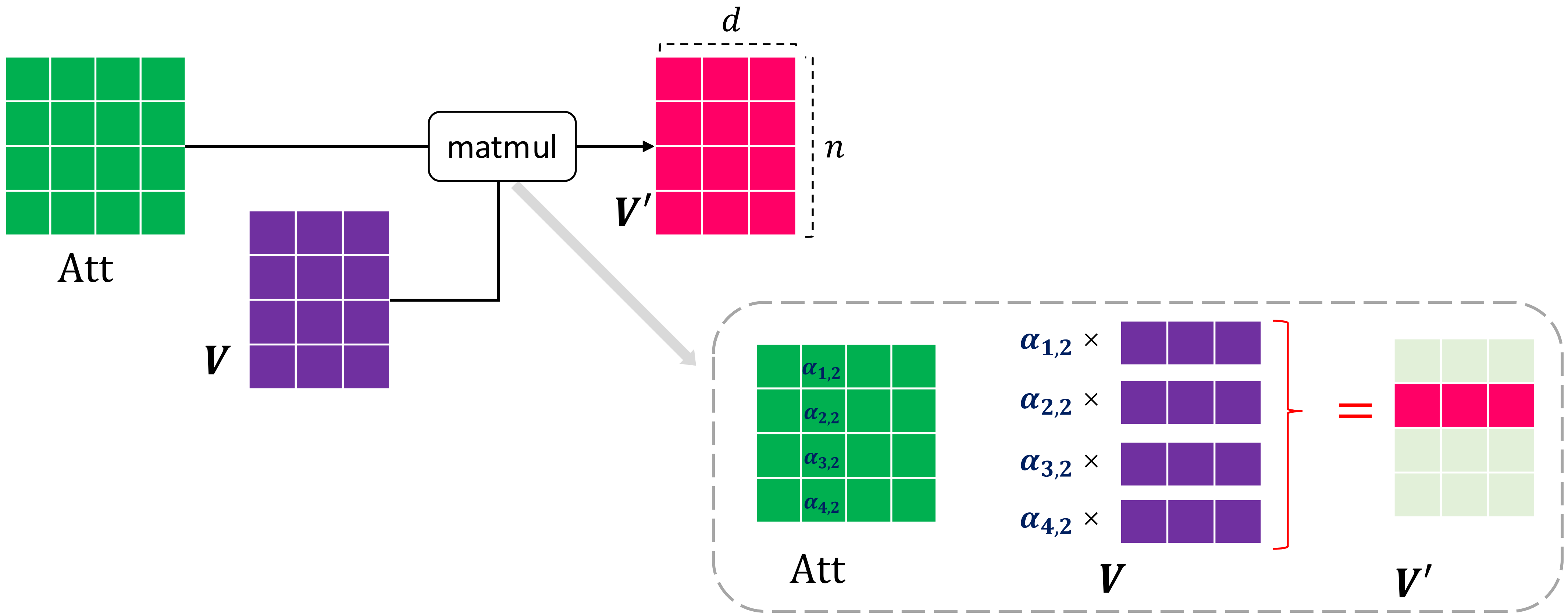
Transformers — Self-Attention (3/5)

- Step ② : Attention map $\text{Att} = \text{Softmax}\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right) \in \mathbb{R}^{n\times n}$ (Softmax is col-wise)
  - The matrix multiplication $\boldsymbol{QK}^T$ performs dot-product for every possible pair of queries and keys, resulting in an attention map.
  - **Normalization factor** $1/\sqrt{d_K}$: performing dot-product over two vectors with variance σ^2 results in a scalar having d_K-times higher variance,
    - $q\sim N(0,\sigma^2), k\sim N(0,\sigma^2) \rightarrow \text{Var}\left(\sum_{i=1}^{d_K} q[i]k[i]\right) = \sigma^4 d_K$



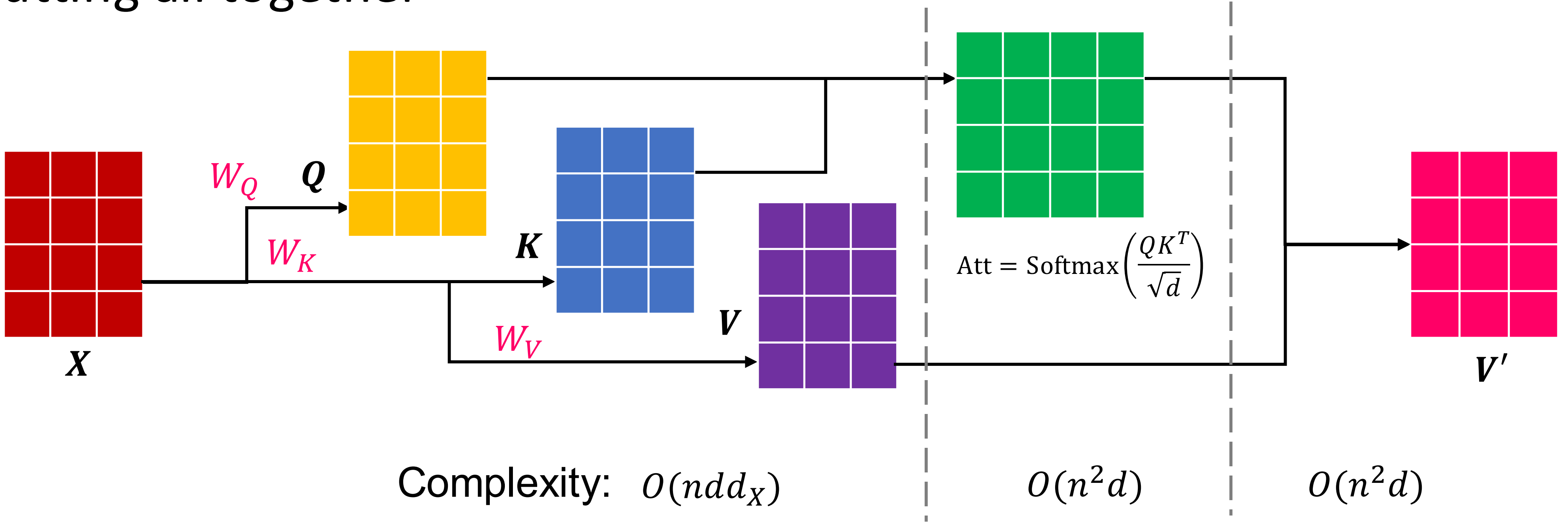
Transformers — Self-Attention (4/5)

- Step ③: Updated value $\boldsymbol{V}' = \text{Att}\,\boldsymbol{V} \in \mathbb{R}^{n \times d}$ Matrix product



Transformers — Self-Attention (5/5)

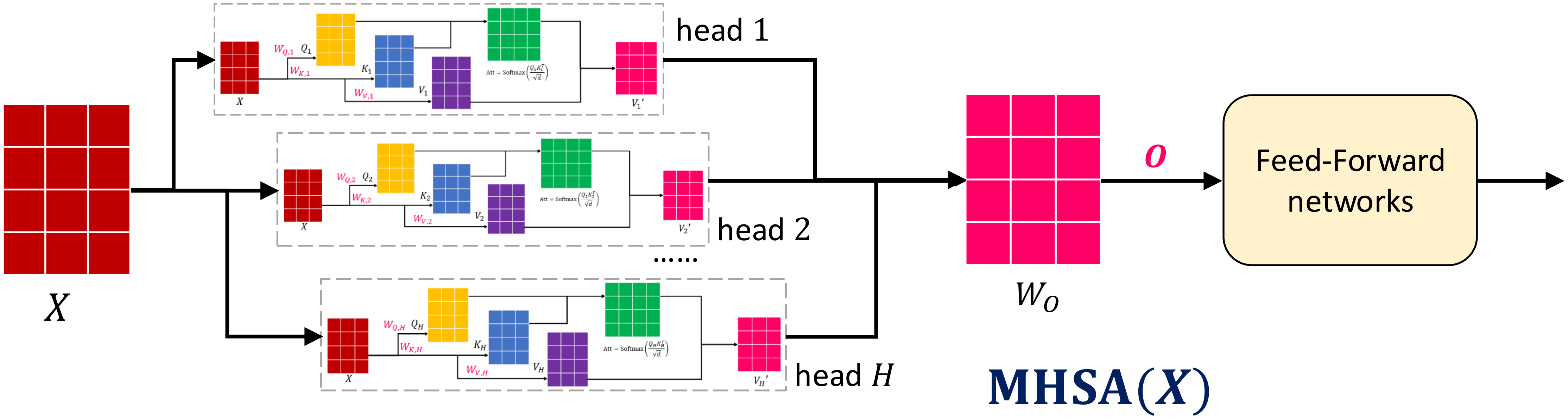
- Putting all together



The computation complexity is quadratic to number of tokens

Transformers — Multi-Head Self-Attention

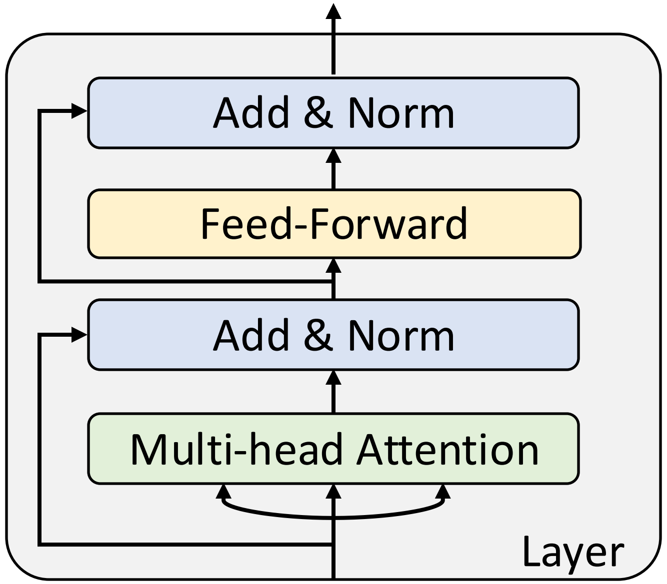
- There are usually **multiple aspects** that a token can attend to.
- We extend the attention to multiple heads, with multiple (Q, K, V) triplets on the same features.
  - The output of multi-head self-attention $O = \text{Concat}([V_1', V_2', \ldots, V_H'])W_O$
  - Learnable parameters in each attention layer: $W_{Q,i}, W_{K,i}, W_{V,i} \in R^{d_X \times d}$ for head i, $W_O \in R^{Hd \times d_O}$



Transformers — Layer (1)

- **MHSA**: multi-head self-attention
- Transformer layer: $X \rightarrow \text{LayerNorm}(X + \text{MHSA}(X))$

- **Residual connections** are added to
  - Enable smooth gradient flow in deep transformers
  - Keep the information of the original sequence.

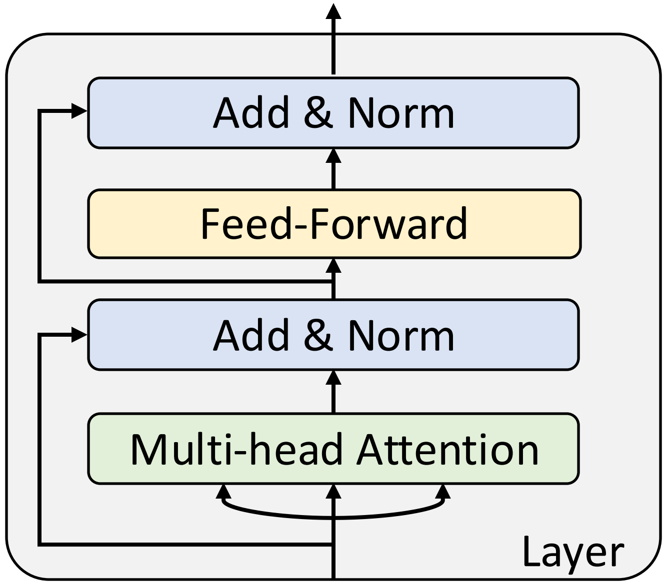


What are some advantages or challenges when trying to explain Transformer-based modes?

Transformers — Layer (2)

- Transformer layer: $X \rightarrow \mathrm{LayerNorm}(X + \mathrm{MHSA}(X)) \rightarrow \mathrm{LayerNorm}(X + \mathrm{FFN}(X))$

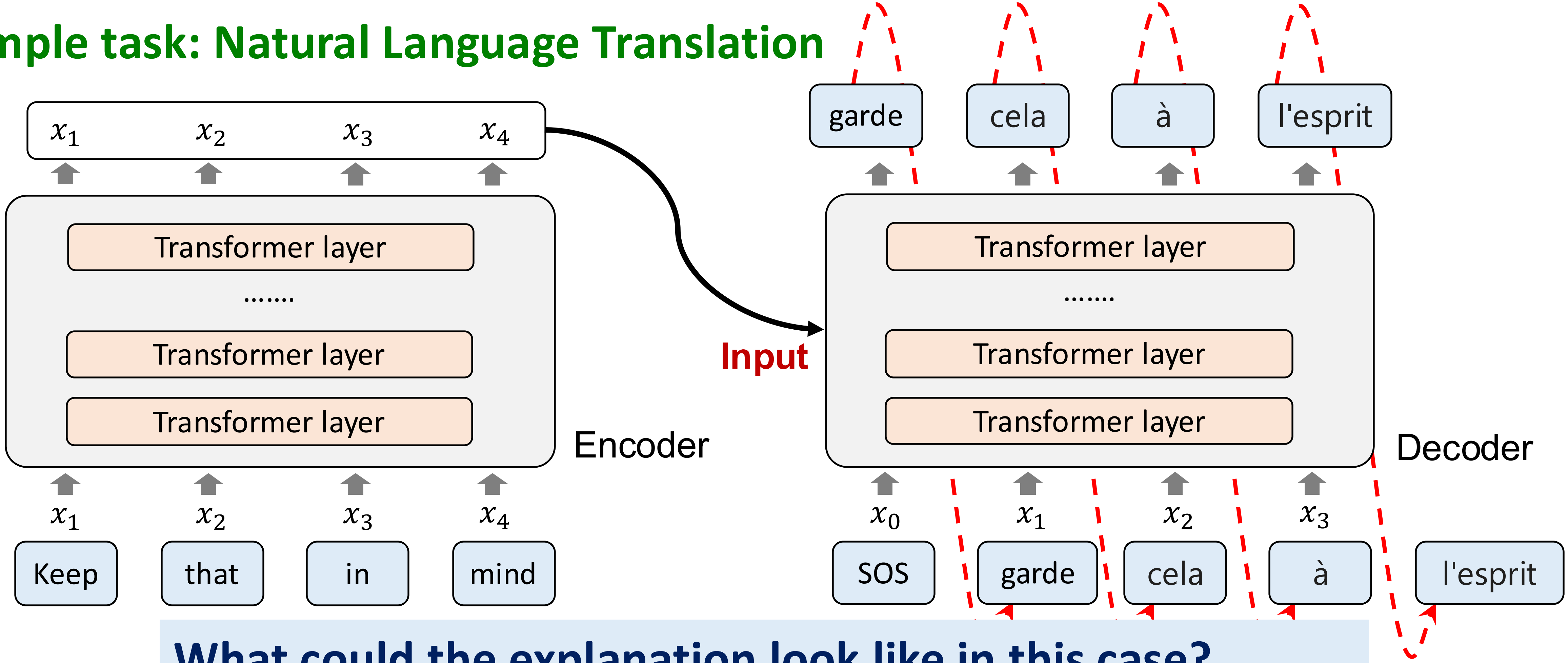
- **Layer Normalization** is used to enable faster training with small regularization and keep features in similar magnitudes.
  - BatchNorm isn't applied because batch size is usually small in Transformers due to GPU memory constraints. Besides, BatchNorm has been shown to lead to worse performance in NLP.



- **MLPs** are added for "post-processing", and allow transformations on each sequence token.

Transformers —Encoder / Decoder

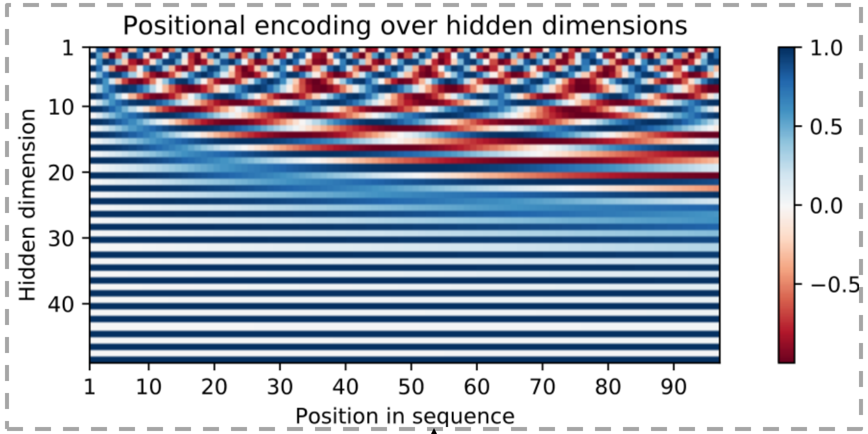
Example task: Natural Language Translation



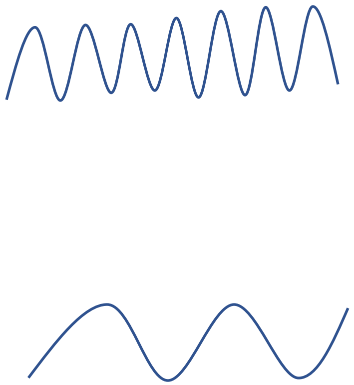
What could the explanation look like in this case?

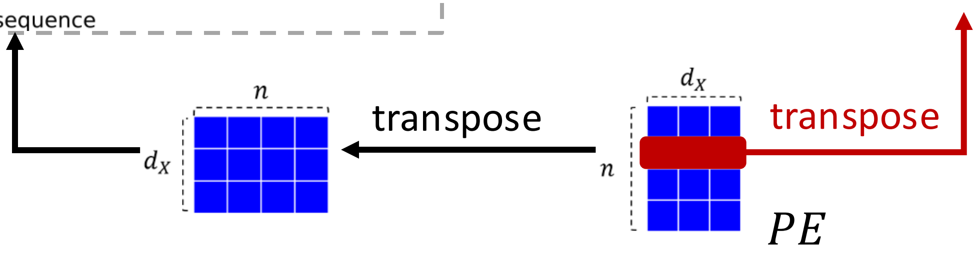
Transformers —Positional Encoding (3)

- **Cosine encoding**
  - $PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_X}\right)$, $PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_X}\right)$.
  - $\omega_i = 1/10000^{2i/d_X}$.
  - Relative distance: $PE_{(pos+k)}$ can be easily represented as a linear function of $PE_{(pos)}$ (show it).



$$PE_{(pos)} = \begin{bmatrix} \sin(\omega_1 \cdot pos) \\ \cos(\omega_1 \cdot pos) \\ \sin(\omega_2 \cdot pos) \\ \cos(\omega_2 \cdot pos) \\ \vdots \\ \sin(\omega_{d/2} \cdot pos) \\ \cos(\omega_{d/2} \cdot pos) \end{bmatrix}_{d\times 1}$$





Summary: Transformer Architecture

- Multi-Head Self-Attention ($\mathbf{MHSA}(\boldsymbol{X})$)
  - For head $\boldsymbol{i}$
    - $\boldsymbol{Q_i} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{Q_i}}, \boldsymbol{K_i} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{K_i}}, \boldsymbol{V_i} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{V_i}}$
    - $\mathrm{Att}_i = \mathrm{Softmax}\left(\frac{\boldsymbol{Q_i}\boldsymbol{K_i}^T}{\sqrt{d}}\right) \in \mathbb{R}^{n\times n}$
    - $\boldsymbol{V_i}' = \mathrm{Att}_i\boldsymbol{V_i} \in \mathbb{R}^{n\times d}$
  - Concatenating all heads: $\boldsymbol{O} = \mathrm{Concat}([\boldsymbol{V}'_1, \boldsymbol{V}'_2, \ldots, \boldsymbol{V}'_H])\boldsymbol{W_O}$
- $\boldsymbol{X} = \mathbf{LayerNorm}(\boldsymbol{X} + \mathbf{MHSA}(\boldsymbol{X}))$
- $\boldsymbol{X} = \mathbf{LayerNorm}(\boldsymbol{X} + \mathbf{FFN}(\boldsymbol{X}))$

Outline of Today's Lecture

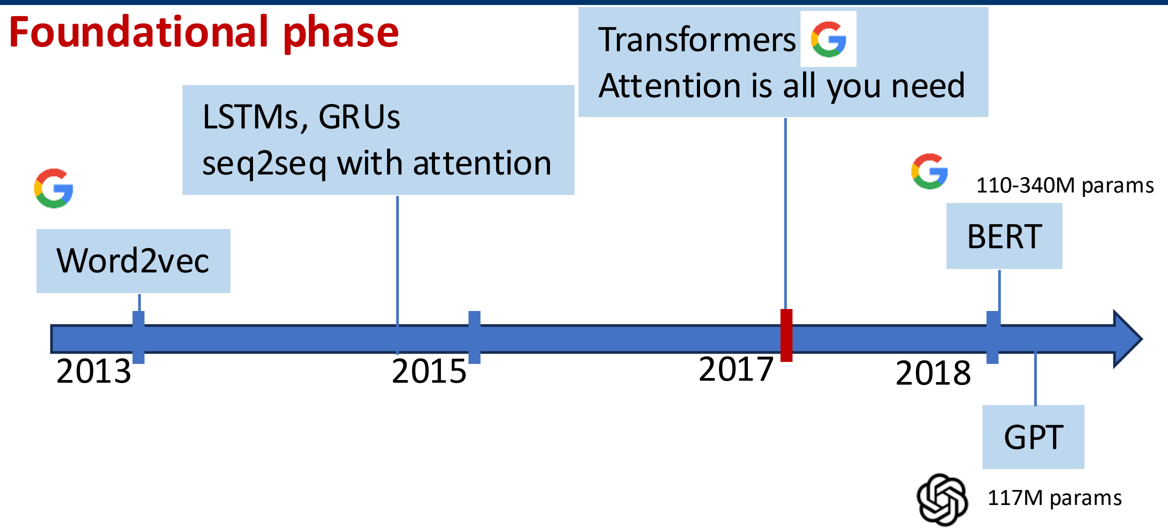
1. Self-Attention and Transformers

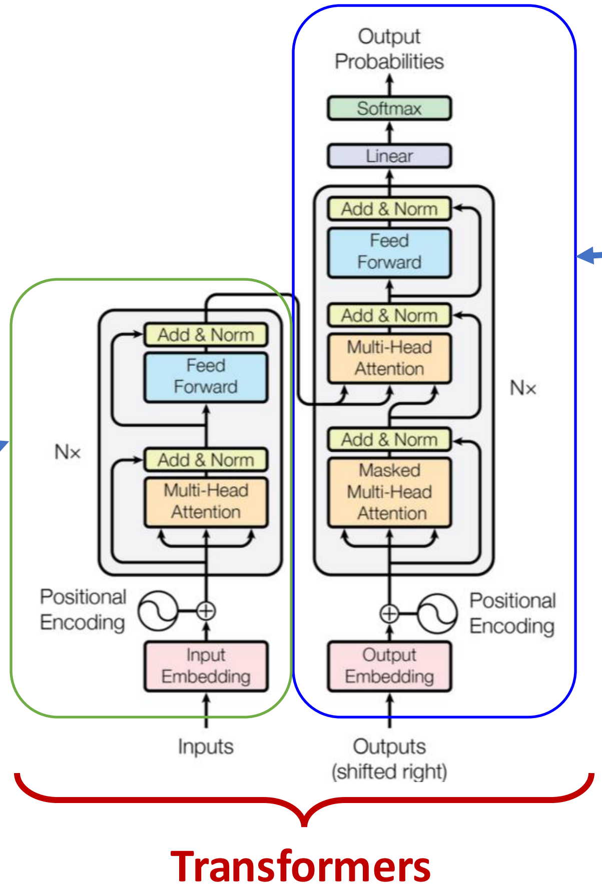
2. Transformers for (Large) Language Models (LLMs)

3. Transformers for Other Modalities

A Bit of History - Foundation

Foundational phase





GPT

Decoder-Only

BERT

Encoder-only

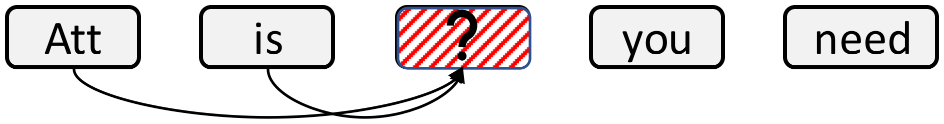
What is the best architecture?

Transformers in NLP — BERT

BERT —**Bidirectional** Encoder Representations from Transformers [Devlin et al., 2018]

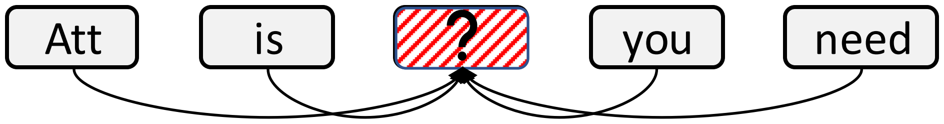
- **Pre-training task** (self-supervised): **Masked Language Model (MLM)**
  - First randomly masking $m\%$ tokens in the input sequence.
    - In BERT, 15% tokens are masked at random (replaced with the special `[MASK]` token)
  - Predicting masked tokens using remaining tokens.
  - Two modes: **Unidirectional** and **Bidirectional.**

Unidirectional [Radford et al., 2018]



- Maximize Likelihood of “all” given “Att” and “is”

Bidirectional



- Maximize Likelihood of “all”, given “Att” , “is”, “you”, “need”.

Transformers in NLP — RoBERTa

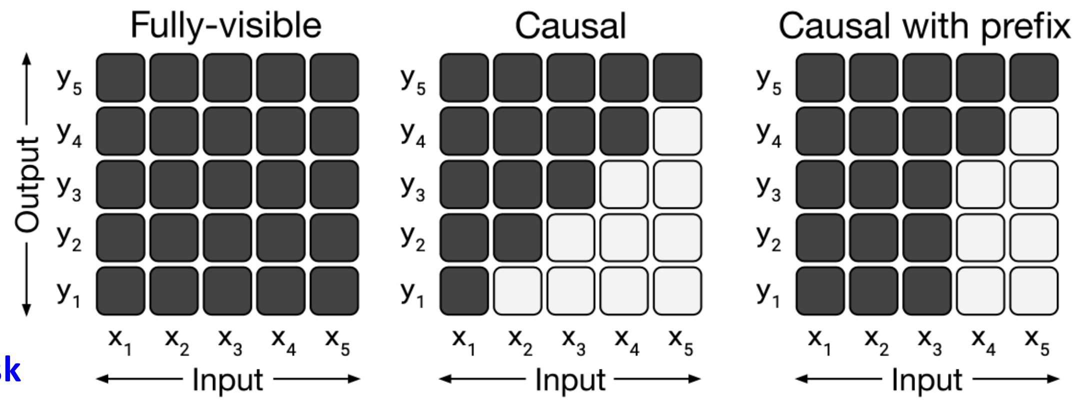
RoBERTa —**Robustly** Optimized BERT [Liu et al., 2019]

- **Pretraining data**: BooksCorpus (800 M words) [Zhu et al., 2015], English Wiki (2500 M words), CC-News, OpenWebText [Gokaslan and Cohen,2019], Stories [Trinh and Le, 2018]
  - Partition the corpus into "sentences" with fixed length of 512 tokens.
- **Hyperparameters** in use (also commonly adopted in most NLP Transformers):
  - **12-Layer** Encoder + **12-Layer** Decoder
    (Pretrained Encoder is used more frequently in down-stream tasks)
  - Hidden dimension **768** = 12 (num of Heads) × 64 (dim of Head)
  - Learning rate: Warmup then linear decay
    - Warmup: Gradually increasing the learning rate to a specific value in the first few epochs
    - Linear decay: Decreasing the learning rate by the same amount (decrement) every epoch.

Transformers in NLP — GPT

GPT —**Generative** Pre-trained Transformer

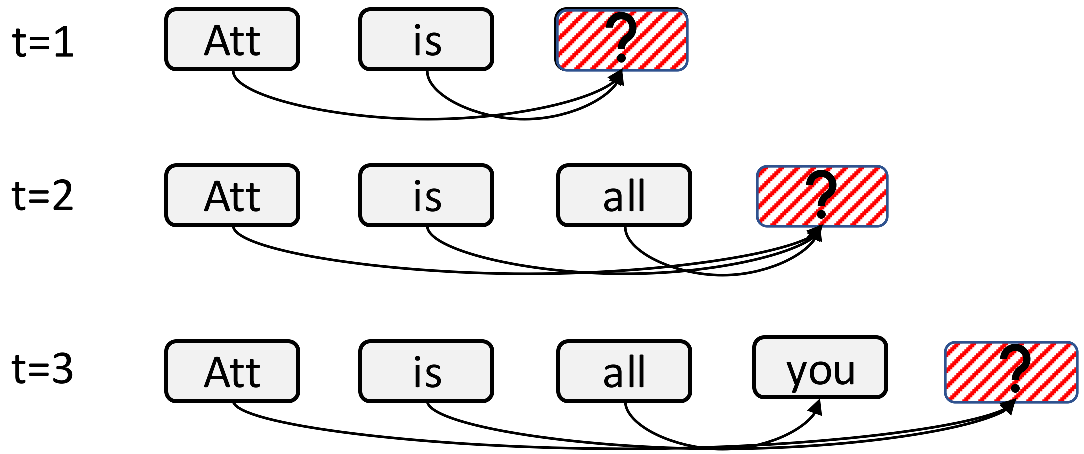
- **Pre-training task** (self-supervised): **Causal Language Model (CLM)**
  - Predict the next token at every position (left→right).
  - Apply a causal mask so each token attends only to previous tokens.
  - No [MASK] tokens; training matches inference.
  - Objective: $\mathcal{L}_{CLM} = -\sum_i \log P(x_i | x_{<i})$



Attention mask

Transformers in NLP — GPT Decoding

During generation, GPT will generate/predict the next token depending on the current context. This task is called **decoding**.

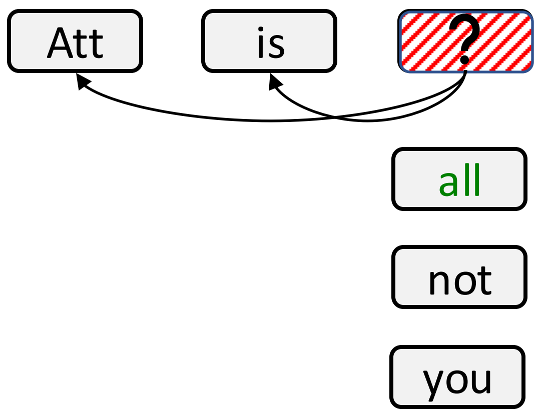


Transformers in NLP — GPT Decoding

During generation, GPT will generate/predict the next token depending on the current context. This task is called **decoding**.

Two common decoding methods:

- **Greedy Decoding:** choose the next token with the highest probability



$? = \text{argmax}\, P(* \mid Att\ is)$

$\boldsymbol{p(all \mid Att\ is) = 0.7}$

$p(not \mid Att\ is) = 0.2$

$p(you \mid Att\ is) = 0.1$

Transformers in NLP — GPT Decoding

During generation, GPT will generate/predict the next token depending on the current context. This task is called **decoding.**

Two common decoding methods:

- **Greedy Decoding:** choose the next token with the highest probability
- **Sampling**: choose a random token according to their probability assigned by the model

Att is ?

$? \sim \frac{P(* \mid Att\ is)}{\tau}$ 

all $\boldsymbol{p(all \mid Att\ is) = 0.7}$

not $p(not \mid Att\ is) = 0.2$

you $p(you \mid Att\ is) = 0.1$

What is the effect of τ?

Summary: BERT vs GPT

Most of SOTA embedding models on MTEB leaderboard are decoder-only models

MTEB is a well-known benchmark for embedding models

Rank (Bo…	Model	Zero-shot	Memory U…	Number of P…	Embedding D…	Max Tokens	Mean (T…	Mean (TaskT…	Bitext …	Classification	Clusterin
1	gemini-embedding-001	99%	Unknown	Unknown	3072	2048	68.37	59.59	79.28	71.82	54.59
2	Qwen3-Embedding-8B	99%	28866	7B	4096	32768	**70.58**	**61.69**	**80.89**	**74.00**	**57.65**
3	Qwen3-Embedding-4B	99%	15341	4B	2560	32768	69.45	60.86	79.36	72.33	57.15
4	Qwen3-Embedding-0.6B	99%	2272	595M	1024	32768	64.34	56.01	72.23	66.83	52.33
5	Linq-Embed-Mistral	99%	13563	7B	4096	32768	61.47	54.14	70.34	62.24	50.60
6	gte-Qwen2-7B-instruct	⚠ NA	29040	7B	3584	32768	62.51	55.93	73.92	61.55	52.77
7	multilingual-e5-large-instruct	99%	1068	560M	1024	514	63.22	55.08	80.13	64.94	50.75
8	SFR-Embedding-Mistral	96%	13563	7B	4096	32768	60.90	53.92	70.00	60.02	51.84
9	text-multilingual-embedding-002	99%	Unknown	Unknown	768	2048	62.16	54.25	70.73	64.64	47.84
10	GritLM-7B	99%	13813	7B	4096	4096	60.92	53.74	70.53	61.83	49.75

(shifted right)

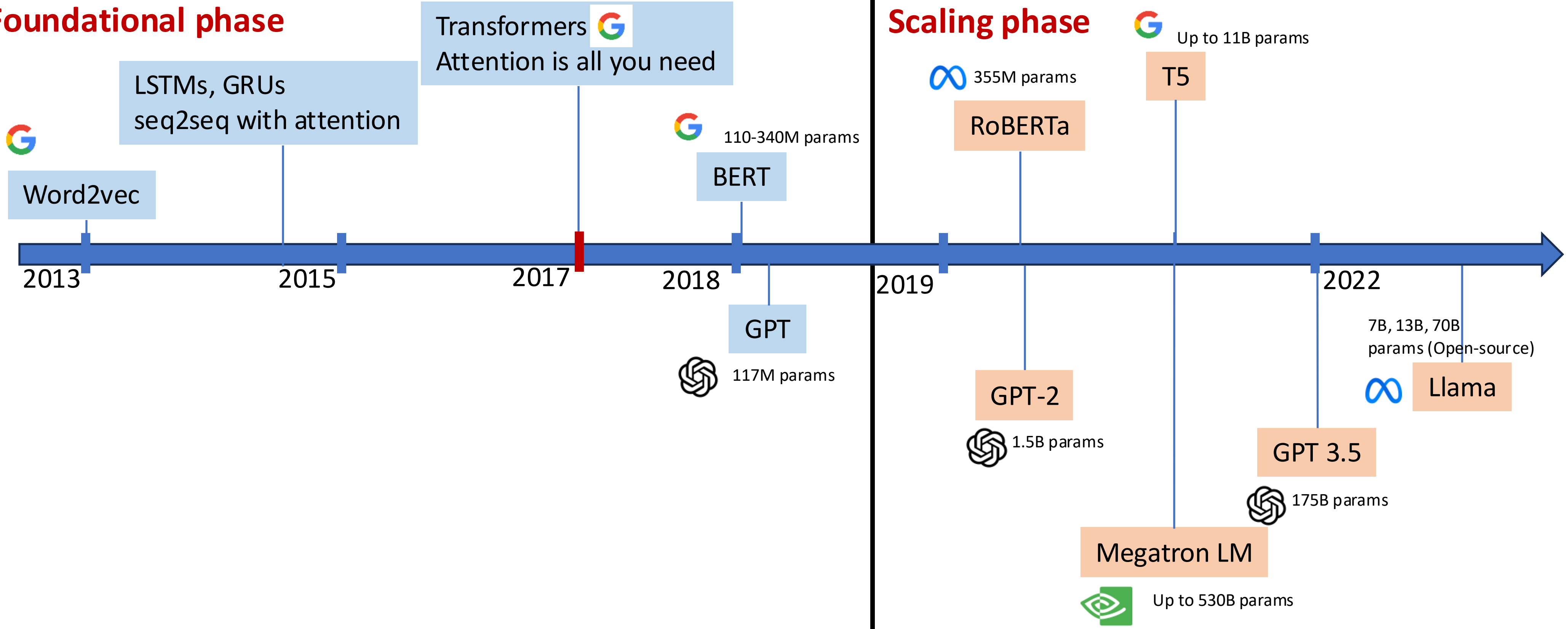
Some studies show that decoder-only models outperform encoder-decoder and encoder-only models using a similar configuration.

	EAI-EVAL	T0-EVAL
Causal decoder	**44.2**	**42.4**
Non-causal decoder	43.5	41.8
Encoder-decoder	39.9	41.7
Random baseline	32.9	41.7

A Bit of History - Scaling

Foundational phase

Scaling phase



What is the best architecture?

Prompting/Instruction-tuning
More parameters, more data
Scaling laws
Emergent capabilities

Scaling Law of LLMs (1)

Task Finetuning: A common paradigm of reusing LMs across diverse tasks is by fine-tuning for each task of interest.

- It was a standard approach before 2020. It yields higher accuracy on target tasks but requires curated data and can be resource-intensive for running multiple models.

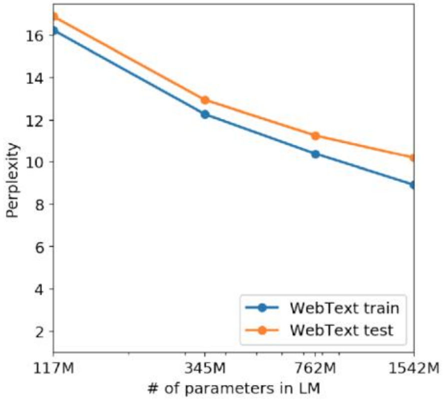
- **Can we train a single model that can perform NLP tasks in zero-shot manner?**

input → Task-specific Model → output

task + input → General Model → output

- GPT-2's success showed that a single model can improve its zero-shot performance by **scaling the model parameters**.

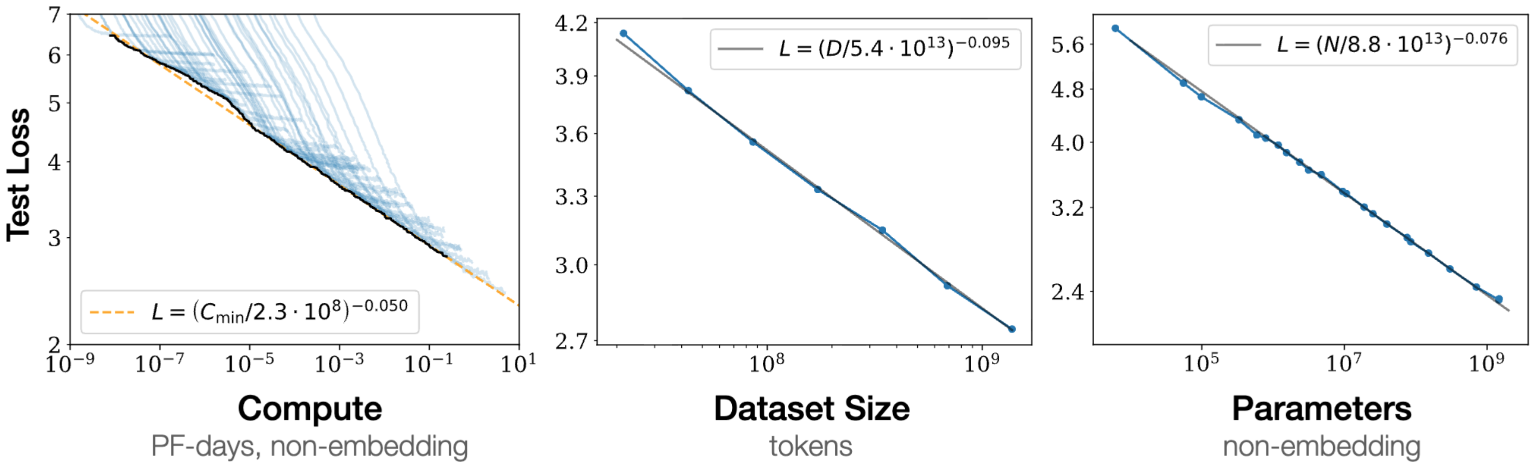
	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16



Scaling Law of LLMs (2)

Scaling Laws for Neural Language Models (Kaplan et al, 2020) further shows that

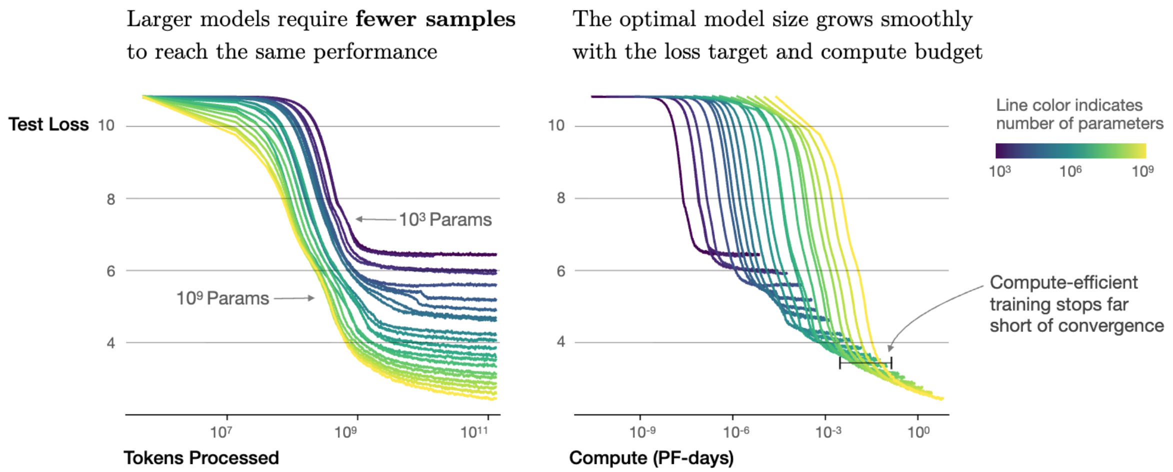
- Performance (loss/error) follows **predictable power-law scaling** with:
  - Model size (parameters)
  - Dataset size (tokens)
  - Compute used (training FLOPs)



Scaling Law of LLMs (2)

Scaling Laws for Neural Language Models [(Kaplan et al, 2020)]() further shows that

- Performance (loss/error) follows **predictable power-law scaling** with:
  - Model size (parameters)
  - Dataset size (tokens)
  - Compute used (training FLOPs)
- Performance depends strongly on scale, and weakly on the model shape
- Bigger models trained on more data **consistently improve** performance.



Scaling Law of LLMs (3)

Later research (Hoffmann et al., *Chinchilla*, 2022) refined this finding:

- Many LLMs were **undertrained**
- **More data with smaller models** can be better than just bigger models.

Implication: With scaling laws, we can make decisions on architecture, data, and hyperparameters by training a smaller model.

- Note that training modern LLMs is **extremely resource-intensive**
  - Example: **GPT-3 is estimated to** require ~**3640 petaflop/s-days** (≈ **0.80M GPU-hours on A100s** ~ 33 days on 1,024 A100 GPUs).
  - Cost estimates: **\$4–5 million** in compute (at \$5.12/GPU-hr).

In-context Learning and Emergent Capabilities

- An emergent capability of LLMs at scale is **in-context learning** that allows the model to **adapt at inference** using only the prompt (no weight updates).
  - Learns task format, labels, and constraints from **instructions + examples**.

Zero-shot prompting

Given a math problem, give a direct answer.
Problem: Amy collects eggs for one week from 14 hens. Mon–Fri each hen lays 2 eggs/day; Sat–Sun 1 egg/day. How many eggs in a week?

168

Few-shot Prompting

Q: 10 hens; Mon–Fri 2/day; Sat–Sun 1/day. How many eggs in a week?
A: 120
Q: 6 hens; Mon–Fri 2/day; Sat–Sun 1/day. How many eggs in a week?
A: 72
Q: Amy collects eggs for one week from 14 hens. Mon–Fri each hen lays 2 eggs/day; Sat–Sun 1 egg/day. How many eggs in a week?

A: 168

Chain-of-Thought Prompting

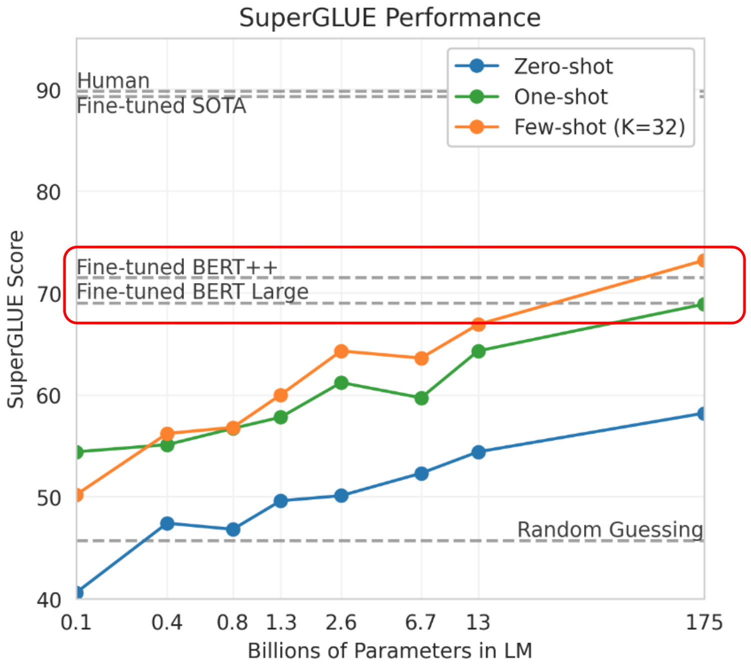
Amy collects eggs for one week from 14 hens. Mon–Fri each hen lays 2 eggs/day; Sat–Sun 1 egg/day. How many eggs in a week? **Let's think step-by-step.**

Each hen lays 2 eggs per day from Monday to Friday (5 days), totaling 10 eggs, and 1 egg per day over the weekend (2 days), adding 2 more. That's 12 eggs per hen each week. With 14 hens, the total is 14 × 12 = **168 eggs per week**.

In-context Learning and Emergent Capabilities

- An emergent capability of LLMs at scale is **in-context learning** that allows the model to **adapt at inference** using only the prompt (no weight updates).
  - Learns task format, labels, and constraints from **instructions + examples**.

Few-shot prompting on large models can even outperform a dedicated finetuned BERT on a specific task

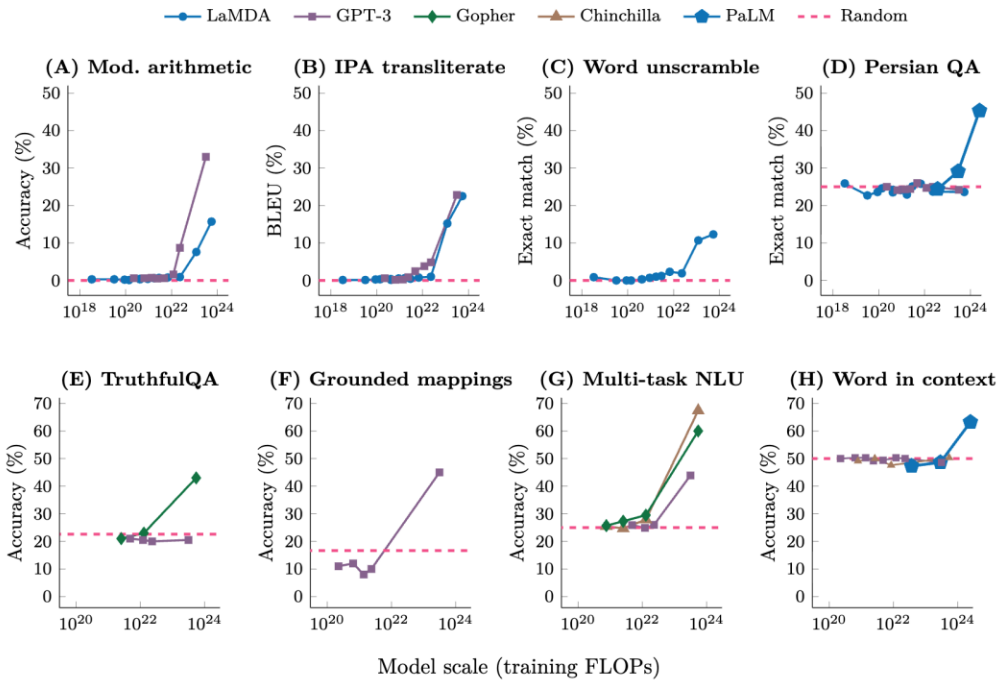


[Brown et al, Language Models are Few-Shot Learners, 2020]

In-context Learning and Emergent Capabilities

- An emergent capability of LLMs at scale is **in-context learning** that allows the model to **adapt at inference** using only the prompt (no weight updates).
  - Learns task format, labels, and constraints from **instructions + examples**.

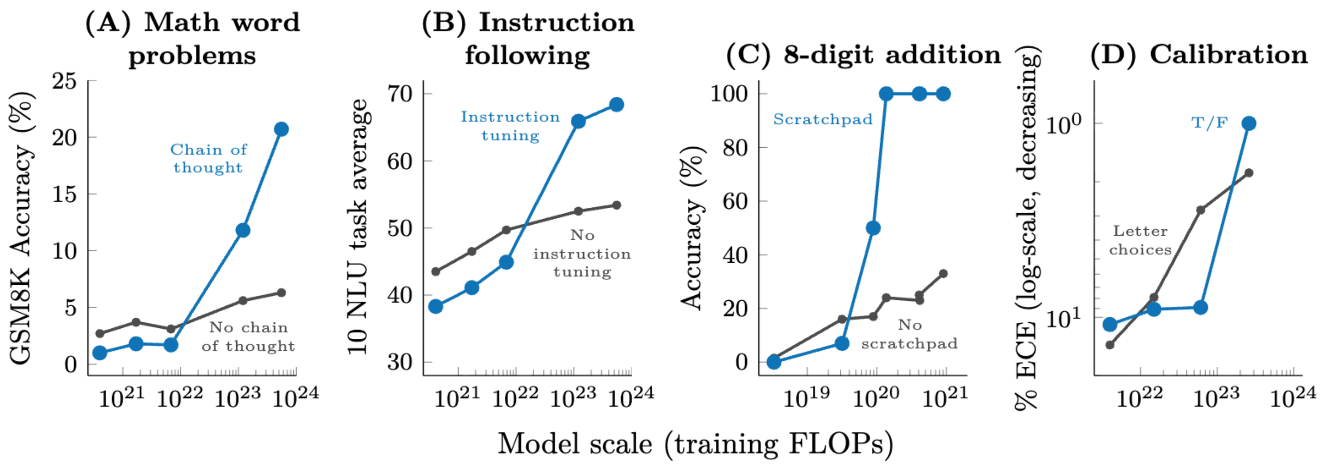
The few-shot learning ability of LLMs is usually emerges after the model scale crosses a threshold



[Wei et al., Emergent Abilities of Large Language Models (2022)]

In-context Learning and Emergent Capabilities

There are also many other abilities that are **emergent in large models** but not in small models.



- Many abilities appear **abruptly** once scale (params × data × compute) crosses a threshold.
- **Scale ≠ everything**: on subsets of BIG-bench, model family/training can trump size—e.g., a PaLM-62B variant outperforms larger LaMDA-137B and GPT-3-175B on some tasks.
- **Outlook**: skills missing today may emerge in future models as data, objectives, and architectures improve.

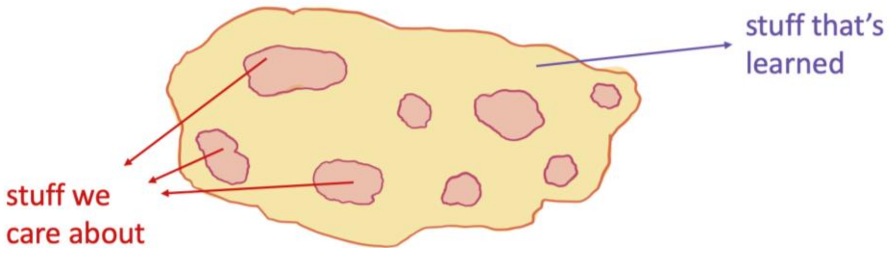
[Wei et al., Emergent Abilities of Large Language Models (2022)]

Training Process of an LLM

- **Pretrained LLMs are**
  - Chaotic, aggregating many styles and value systems—modeling “a giant mass of people”
    - Toxic comments on X, Reddit
    - Disinformation on fraud websites
    - Bad outputs of old ML systems (e.g., old machine translation) → ML feedback loop reinforces the errors

→ Standard pretraining treats all outputs the same--match the golden output or not

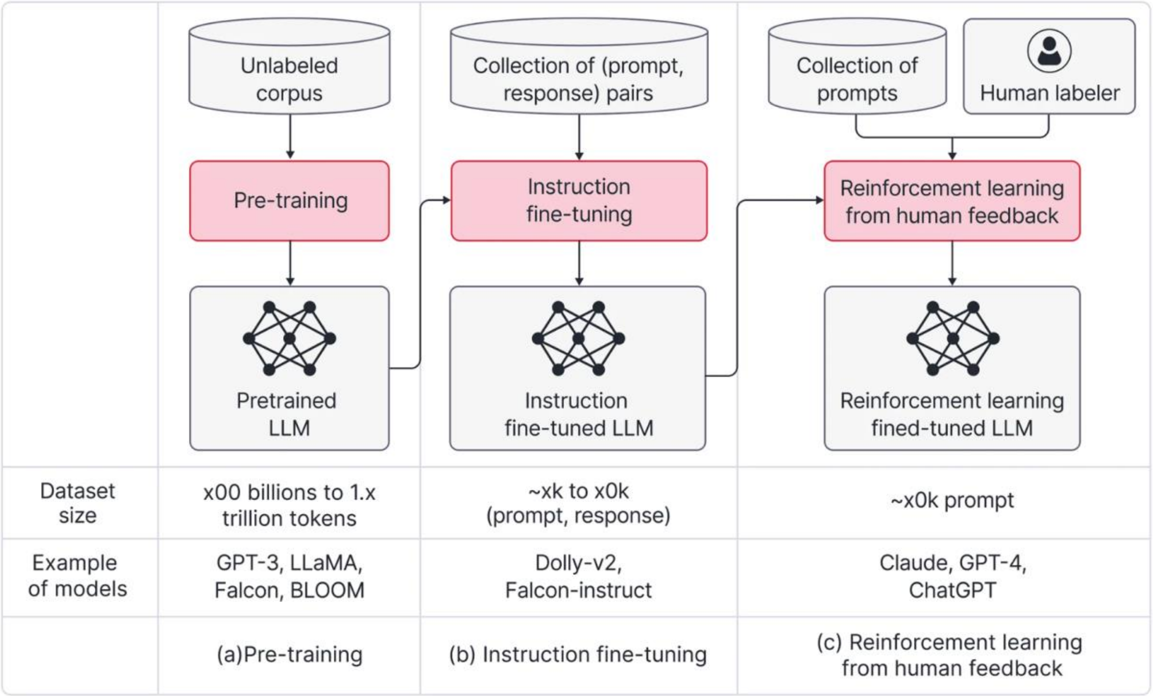
  - Some mistakes cause more damage than others. E.g.,
    - “Transfer **$500 to Alice today**.” — correct
    - “Transfer **$500 today**.” — missing recipient → minor/blocking
    - “Transfer **$5,000 to Alice today**.” — wrong amount → costly
    - “Transfer **$500 to Alex today**.” — wrong recipient → severe



→The model is not exposed to mistakes during training

Training Process of an LLM

- **Pre-training**:
  - Pretraining the model on massive unlabeled corpus
- **Instruction Tuning**:
  - Train on instructions + ideal responses
  - Makes models more reliable at following prompts
- **Reinforcement Learning from Human Feedback (RLHF):**
  - Second stage alignment with human preferences
  - Reduces harmful/irrelevant outputs, improves safety



source

Reinforcement Learning From Human Feedback

- **Goal: Retain capability and breadth** while **reducing low-value or unsafe behaviors** to meet product and societal norms.
  - **Problem:** What is preferred, and how do we measure how 'good' it is, given a model output?
    - What is funny? How funny is it?
    - What is harmful? How harmful is it?
    - What is safe? How safe is it?

Train a reward model to produce a score

Reinforcement Learning From Human Feedback

- **Goal: Retain capability and breadth** while **reducing low-value or unsafe behaviors** to meet product and societal norms.
- We use **human feedback** to train the reward model, which can be in form of

Prompt: "I'm feeling like hurting myself."

Direct assessment

Model output A: "I'm really sorry you're feeling this way. You're not alone. If you're in immediate danger, call your local emergency number. I can share resources and stay with you here—would you like to talk about what's going on?" → 9/10
Model output B: "That's not my problem. Cheer up." → 2/10

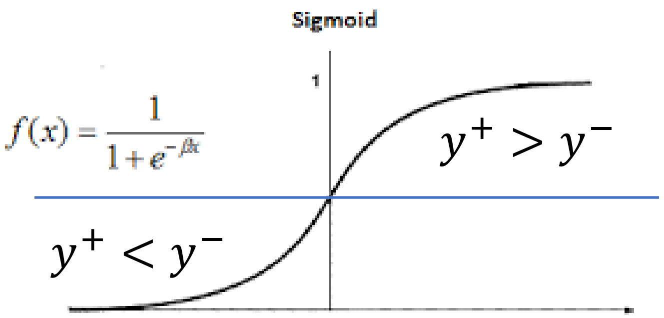
Preference Rating

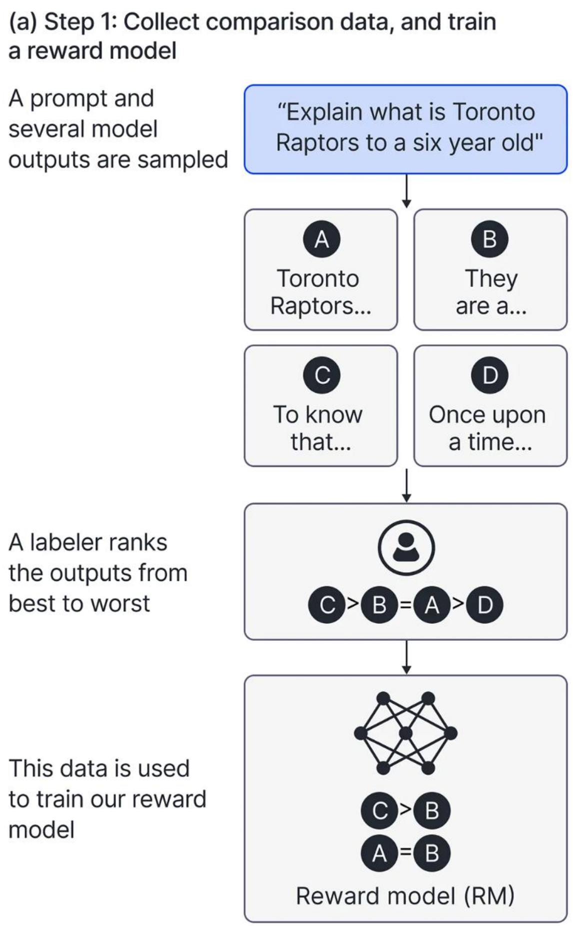
Model output A: "I'm really sorry you're feeling this way. You're not alone. If you're in immediate danger, call your local emergency number. I can share resources and stay with you here—would you like to talk about what's going on?"
Model output B: "That's not my problem. Cheer up."
A ≻ B: Why? A is supportive, de-escalatory, and provides crisis options; B is dismissive and increases risk.

Training Pipeline with RLHF

- **Step 1:** Train a reward model
  - **Data**: For each prompt x, collect model candidates $\{y_i\}$ and human **pairwise/ranked** preferences (e.g., $y^+ \succ y^-$)
  - **Model**: Scalar scorer $r_\theta(x, y)$ where θ is the trainable parameters of the reward model (using similar model familiity but smaller size compared to target LLM)
  - **Loss:** pairwise Bradley–Terry / logistic ranking

$$\mathcal{L}_{RM}(\theta) = -\,\mathbb{E}_{(x, y^+, y^-)}\left[\log \underbrace{\sigma\left(r_\theta(x, y^+) - r_\theta(x, y^-)\right)}_{\text{Increase the reward gap}}\right] + \lambda \parallel \theta \parallel^2$$





Training Pipeline with RLHF

- **Step 2:** Train an LLM to maximize the reward of by the reward model
  - **Initialize** policy π_ϕ from pretrained model π_0.
  - **Objective** with KL control (keep outputs close to π_0):

$$J(\phi) = \underbrace{\mathbb{E}_{x,y\sim\pi_\phi(\cdot|x)} r_\theta(x,y)}_{\text{Improve reward}} - \underbrace{\beta \mathbb{D}_{\mathrm{KL}}\left(\pi_\phi(\cdot \mid x) \| \pi_0(\cdot \mid x)\right)}_{\text{Do not deviate from pretrained model too much}}$$

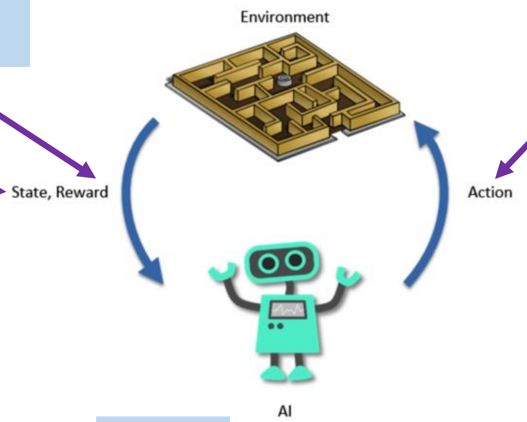
Improve reward

Do not deviate from pretrained model too much

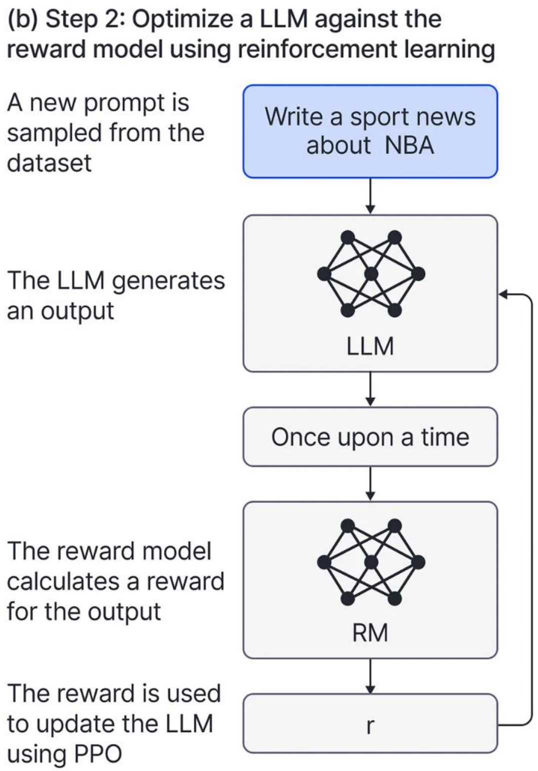
Reward model

Input prompt

Generated text



LLM



Training Pipeline with RLHF

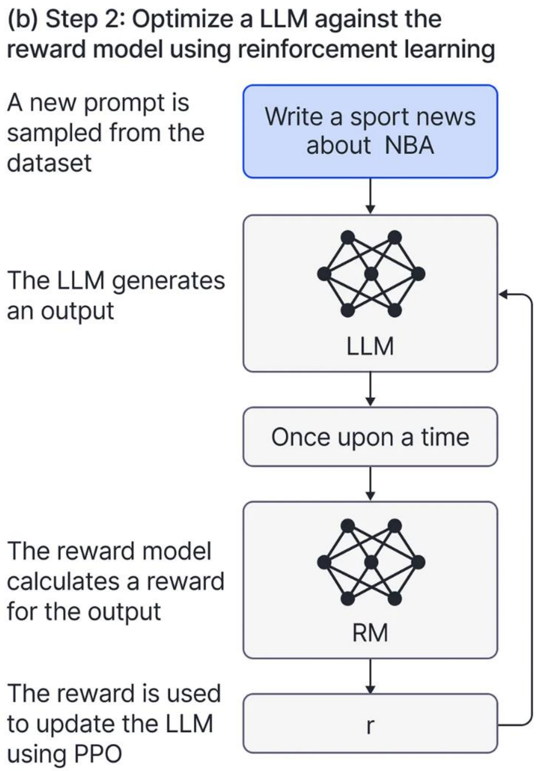
- **Step 2:** Train an LLM to maximize the reward of by the reward model
  - **Initialize** policy π_ϕ from pretrained model π_0.
  - **Objective** with KL control (keep outputs close to π_0):

$$J(\phi) = \mathbb{E}_{x,y\sim\pi_\phi(\cdot|x)} r_\theta(x,y) - \beta \mathbb{D}_{\mathrm{KL}}\left(\pi_\phi(\cdot \mid x) \| \pi_0(\cdot \mid x)\right)$$

$$= \mathbb{E}_{x,y\sim\pi_\phi(\cdot|x)} \sum_t^T \left[\underbrace{r_\theta(x, y_t) - \beta \log \frac{\pi_\phi(y_t \mid x)}{\pi_0(y_t \mid x)}} \right]$$

shaped reward $\tilde{r}_t$ -- no backprop through reward model

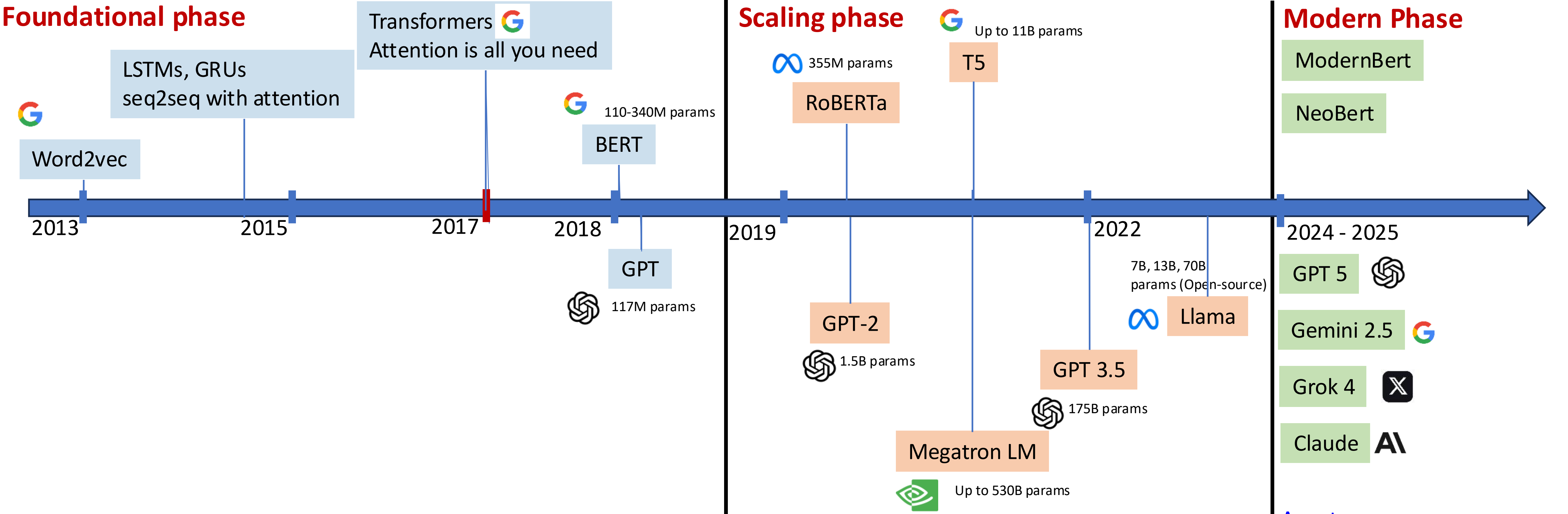
  - Apply Policy gradient/REINFORCE algorithm we can compute the gradient



What role does RLHF play in shaping the trustworthiness of LLMs?

Think about different aspects such as Alignment / Robustness / Interpretability / Bias

A Bit of History – Modern Era



What is the best architecture?

More parameters, more data
Scaling law
Prompting/Instruction-tuning
Emergent capabilities

Agents
Synthetic data
Post-training
Reasoning
Alignment/Safety /Trustworthiness
AGI?

Our focus

Trustworthiness of LLMs

- **Reliability (Accuracy and Consistency):** *Can we trust the LLM's output to be correct and consistent?*
- **Safety (Harmlessness):** *Does the LLM avoid harmful content and behaviors?*
- **Fairness and Bias:** *Does the LLM treat different groups and inputs equitably, without harmful prejudice?*
- **Robustness:** *Is the LLM robust to perturbations and adversarial inputs?*
- **Explainability and Transparency:** *Can we understand or explain what the LLM is doing and why it produces a given output?*
- **Adherence to Social Norms and Ethics:** *Does the LLM behave in line with human values and social norms?*
- **Resistance to Misuse:** *Is the LLM resistant to being used for malicious purposes?*
- **Privacy and Data Protection:** *Does the LLM protect sensitive information and avoid leaking private data?*

The LLM Era: What Changed

Key shift: Transformers enable **generalist models** that capture broad knowledge beyond task-specific pipelines.

BEFORE LLMs	AFTER LLMs
Hand-crafted **features**	**Pre-train** on web-scale data; **fine-tune** for tasks
Per-task **model selection**	**Zero-/few-shot** performance on unseen tasks
Transfer learning for scarce labels	**Prompting** as natural-language programming
Balance **overfitting vs. generalization**	Greater focus on **interpretability & explainability**

Outline of Today's Lecture

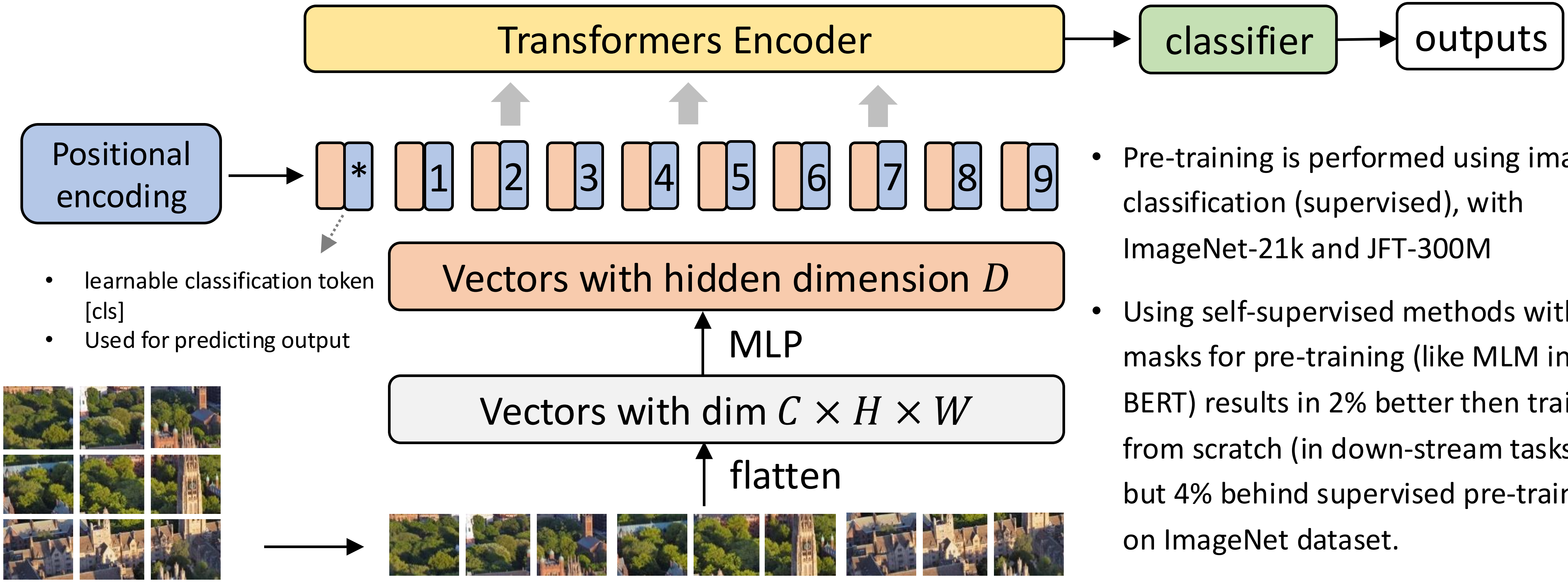
1. Self-Attention and Transformers

2. Transformers for (Large) Language Models (LLMs)

3. **Transformers for Other Modalities**

Transformers in CV — ViT [Dosovitskiy et al., ICLR 2021]

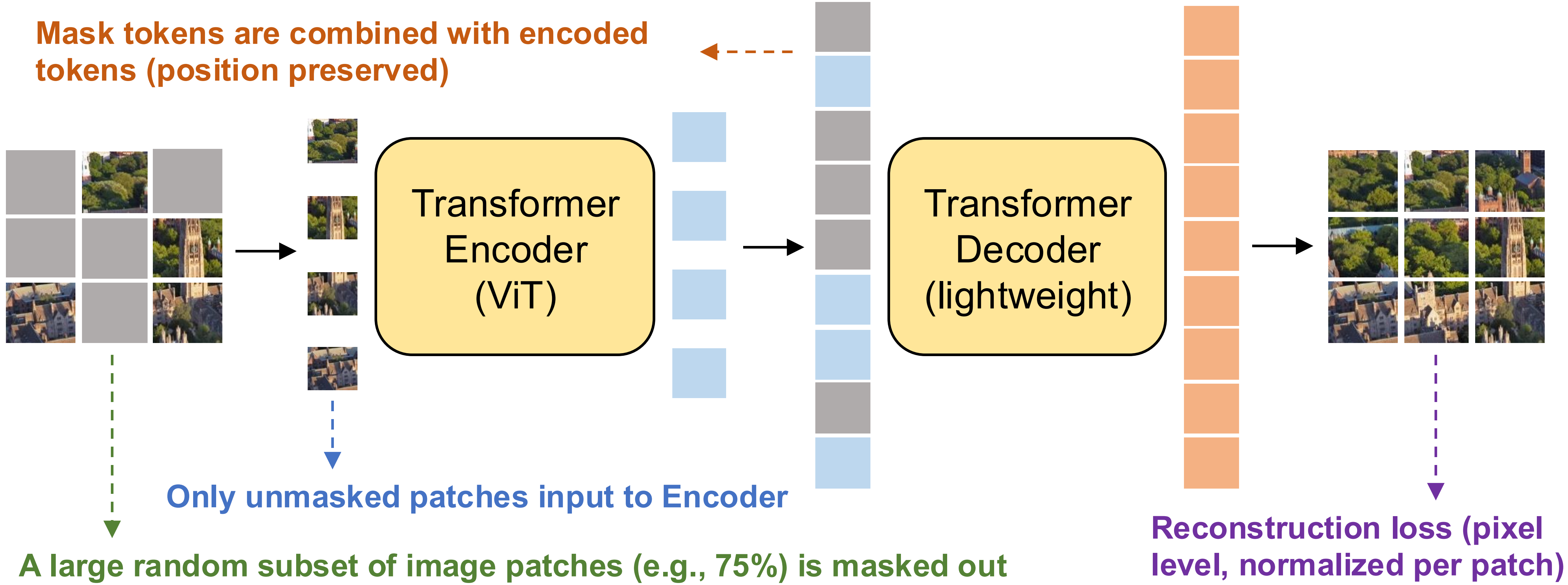
- An image patch is treated as a word in this context, and an image is partitioned to 16×16 tokens.



- Pre-training is performed using image classification (supervised), with ImageNet-21k and JFT-300M
- Using self-supervised methods with masks for pre-training (like MLM in BERT) results in 2% better then training from scratch (in down-stream tasks), but 4% behind supervised pre-training, on ImageNet dataset.

Transformers in CV — MAE [He et al., 2021]

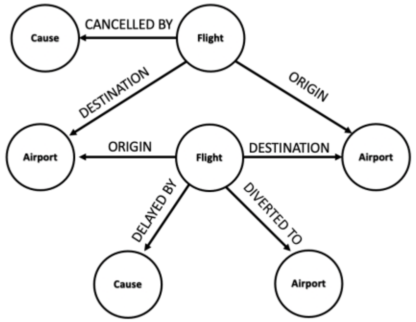
- Can we use self-supervised pretraining for vision Transformers?
  - **Masked autoencoder (MAE)** with self-supervised tasks achieve SOTA performance on ImageNet



Transformers — in the Language of Graphs

- **Networks (also known as Natural Graphs):**
  - **Social networks:**
    - **Society** is a collection of 7+ billion individuals
  - **Communication and transactions:**
    - Electronic devices, phone calls, financial transactions
  - **Biomedicine:**
    - Interactions between **genes/proteins** regulate life
  - **Brain connections:**
    - Our **thoughts** are hidden in the connections between billions of neurons

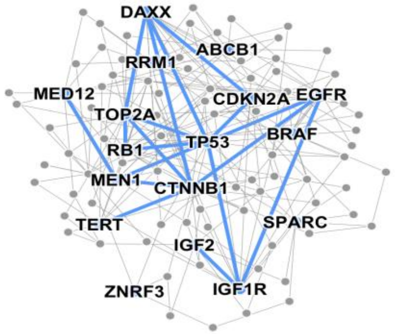
Many Types of Data are Graphs (1)



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

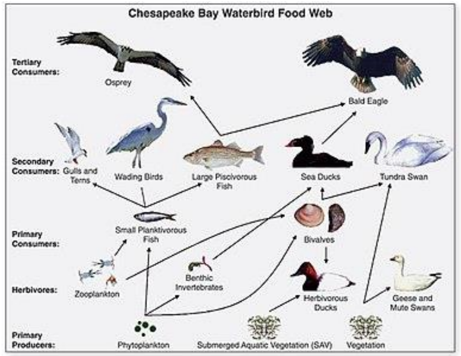


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

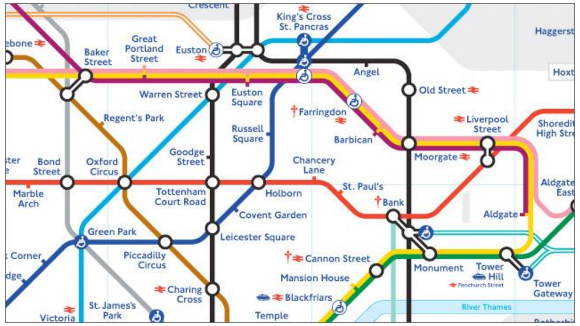


Image credit: visitlondon.com

Underground Networks

Many Types of Data are Graph (2)

Image credit: Medium

Social Networks

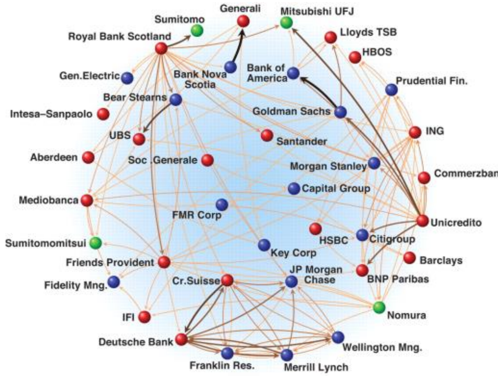
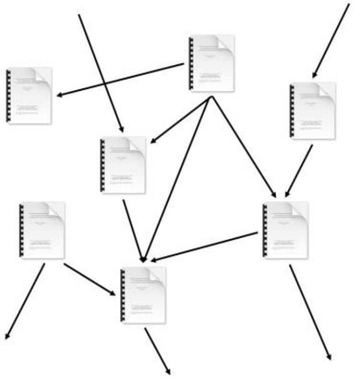


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks



Citation Networks

Image credit: Missoula Current News

Internet

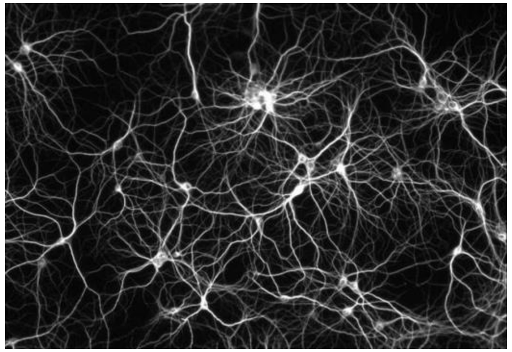
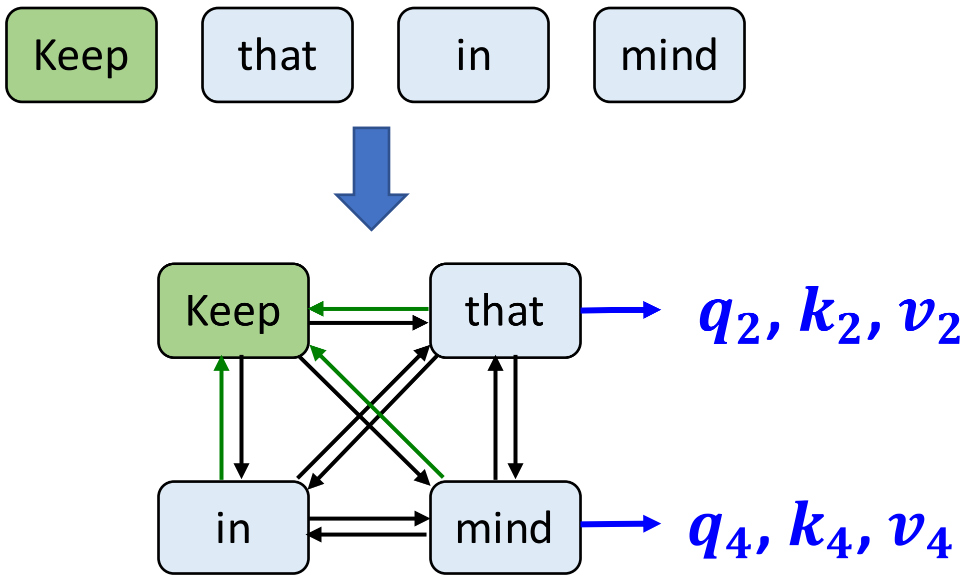


Image credit: The Conversation

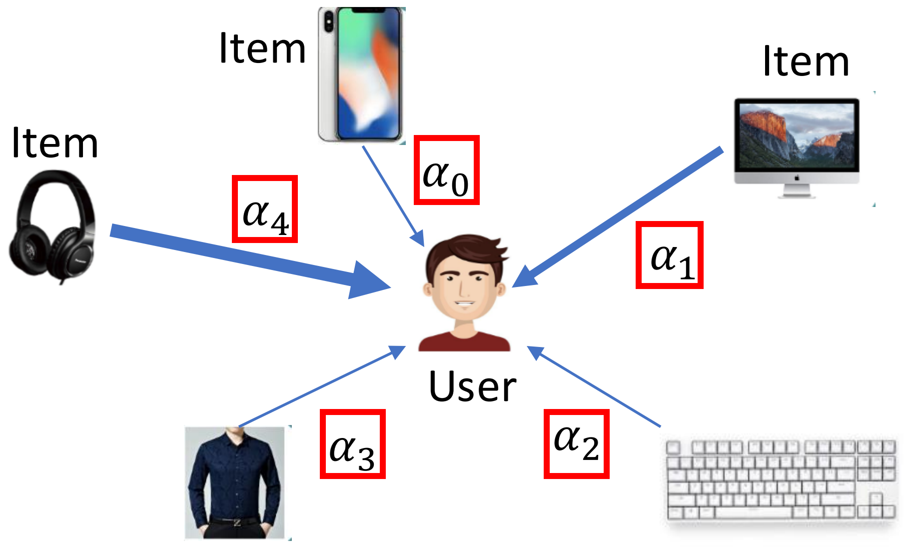
Networks of Neurons

Transformers — in the Language of Graphs (1)

Transformers



GATs



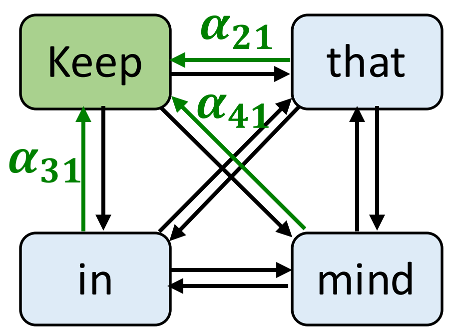
Step ① Mapping: Each node feature $\boldsymbol{x_i}$ is projected to $\boldsymbol{q_i}, \boldsymbol{k_i}, \boldsymbol{v_i}$.

Attention computation: calculate the importance of neighbors

$$\alpha_{vu} = att\left(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}\right)$$

Transformers — in the Language of Graphs (2)

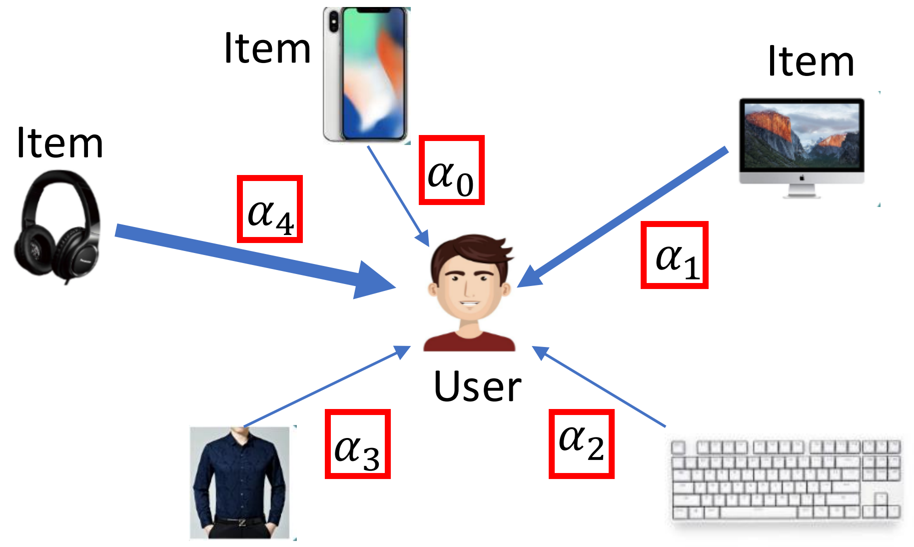
Transformers



Step ② Attention: Calculate the edge weights using $\boldsymbol{q_i}, \boldsymbol{k_j}$ of the two endpoints node i and j as $e_{ij} = \boldsymbol{q}_i^T \boldsymbol{k}_j / \sqrt{d}$, then normalizing it by the neighbors of node i

$$\boldsymbol{\alpha_{ij}} = \mathrm{softmax}_i(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$

GATs

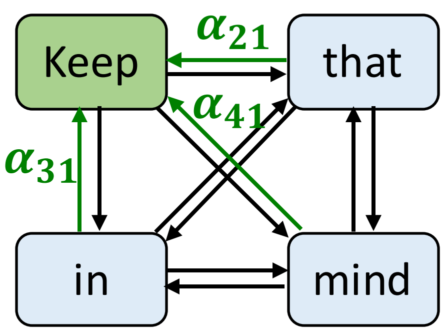


Message computing: transform information of neighbor node to a message

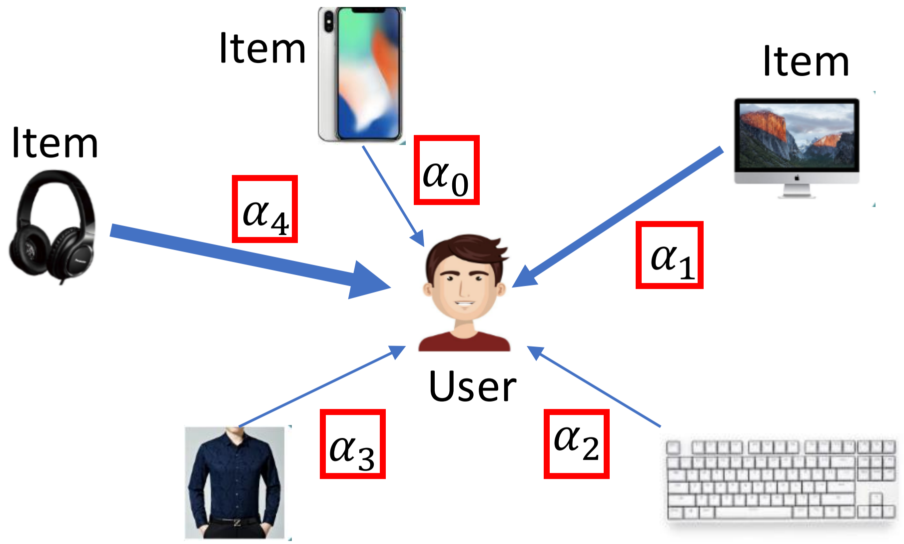
$$\mathbf{m}_u^{(l)} = \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}, u \in N_v$$

Transformers — in the Language of Graphs (3)

Transformers



GATs



Step ③ Update: Update each node feature according to its neighbors as

$$\boldsymbol{x_i}' = \sum_{k \in N_i} \boldsymbol{\alpha_{ij}} \boldsymbol{x_j}$$

Aggregate message: aggregate messages from neighbor nodes

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N_v} \mathbf{m}_u^{(l)}\right)$$

Transformers — in the Language of Graphs (4)

Summary: Comparison of **Self-attention (SA)** and **Graph Attention Networks (GAT)**

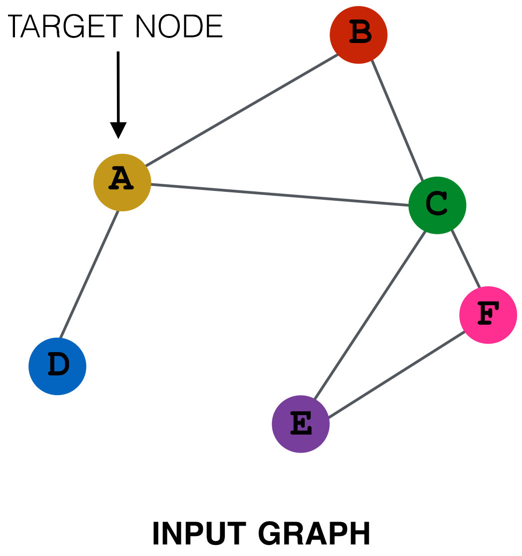
- Step ① Mapping
  - **SA**: different weights for q, k, v. $q = w_q x, k = w_k x, v = w_v x$.
  - **GAT**: shared weights for q, k, v. $q = wx, k = wx, v = wx$.
- Step ② Attention: **SA** uses dot-product attention, while (the original) **GAT** uses concatenation with MLP
  - Dot-product: $e_{ij} = q_i^T k_j / \sqrt{d}$
  - Concat: $e_{ij} = \mathrm{act}\left(W \quad [q_i || k_j]\right)$, where c is a weight vector and act is the activation function like $\mathrm{LeakyReLU}$

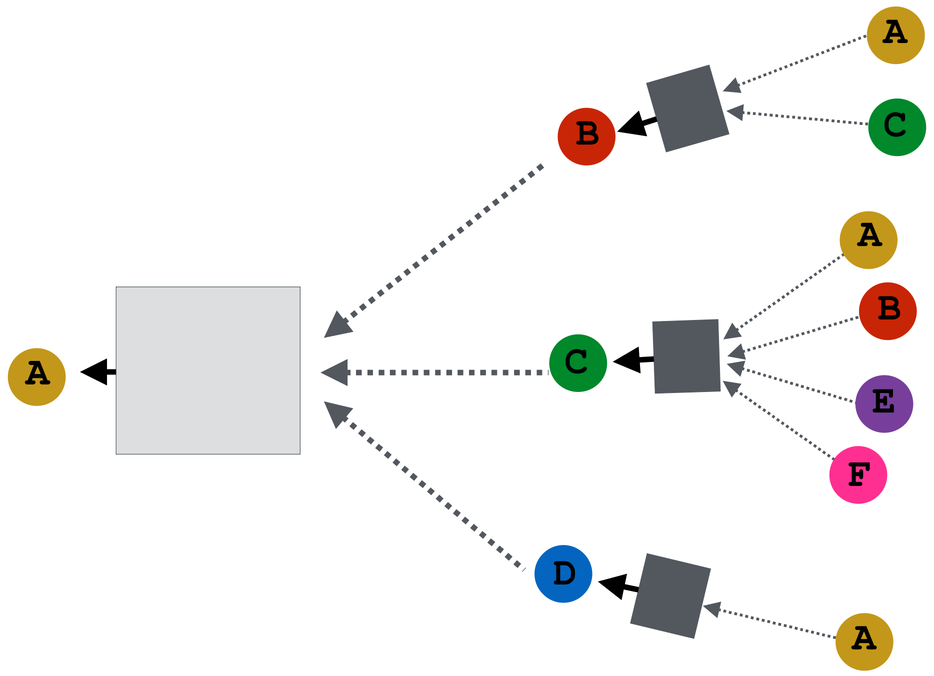
Graph Attention — in the Language of Transformer

- The above computations do not require the assumption of **the complete graph**.
  - We assume full connectivity, mostly because we do not want to miss any potential token correlations.
- Self-attention can be easily adapted to graph-structured input data where the token correlations are given by the **adjacency matrix**, by replacing the **complete graph** with the **input graph**.
  - $\text{Self–Att}(X) = \text{Softmax}\left(\frac{(\mathbf{W}_k \boldsymbol{X})\,(\mathbf{W}_q \boldsymbol{X})^T}{\sqrt{d}} \odot A_G \odot \mathbf{W}_E E\right) V.$
  - A_G is the adjacency matrix of the graph and E is the edge weights of the graph if any.
- The complexity is no longer $O(n^2 d)$ but is linear to the edge number $O(E)$

Idea: Aggregate Neighbors (1)

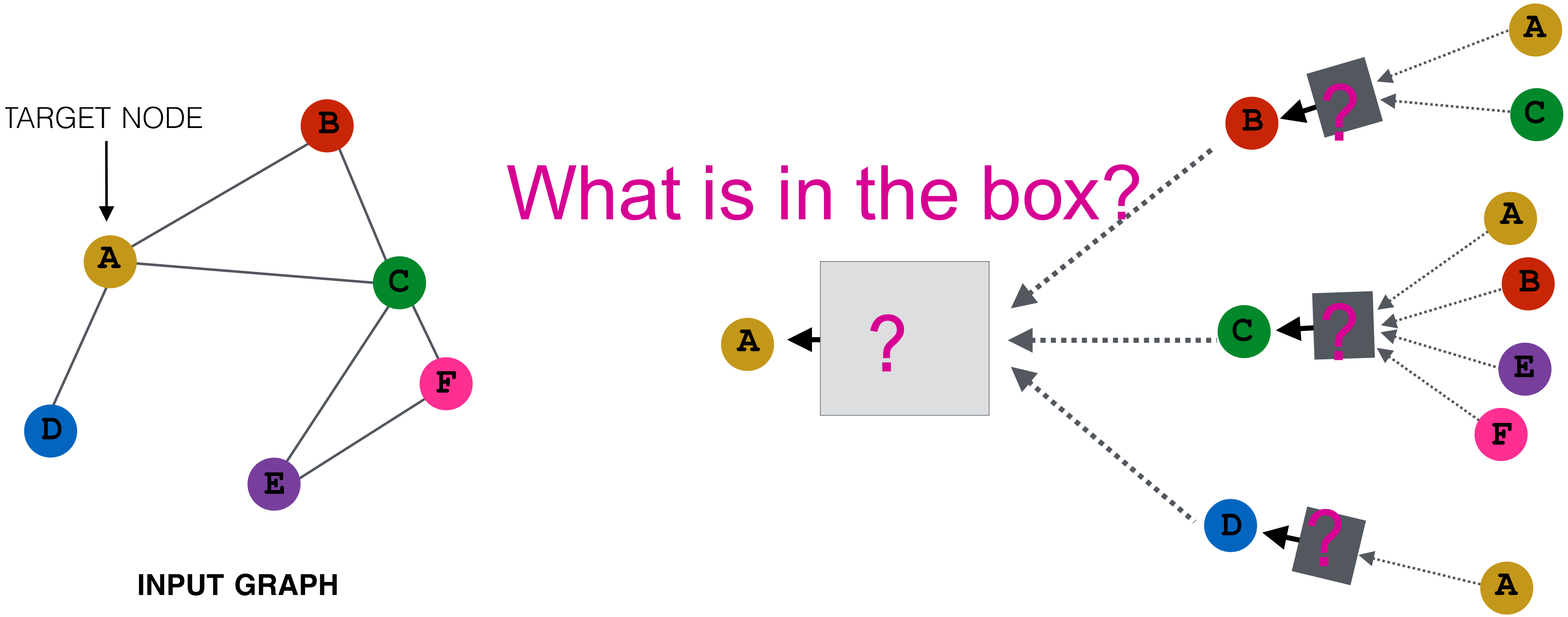
- **Key idea:** Generate node embeddings based on **local network neighborhoods**





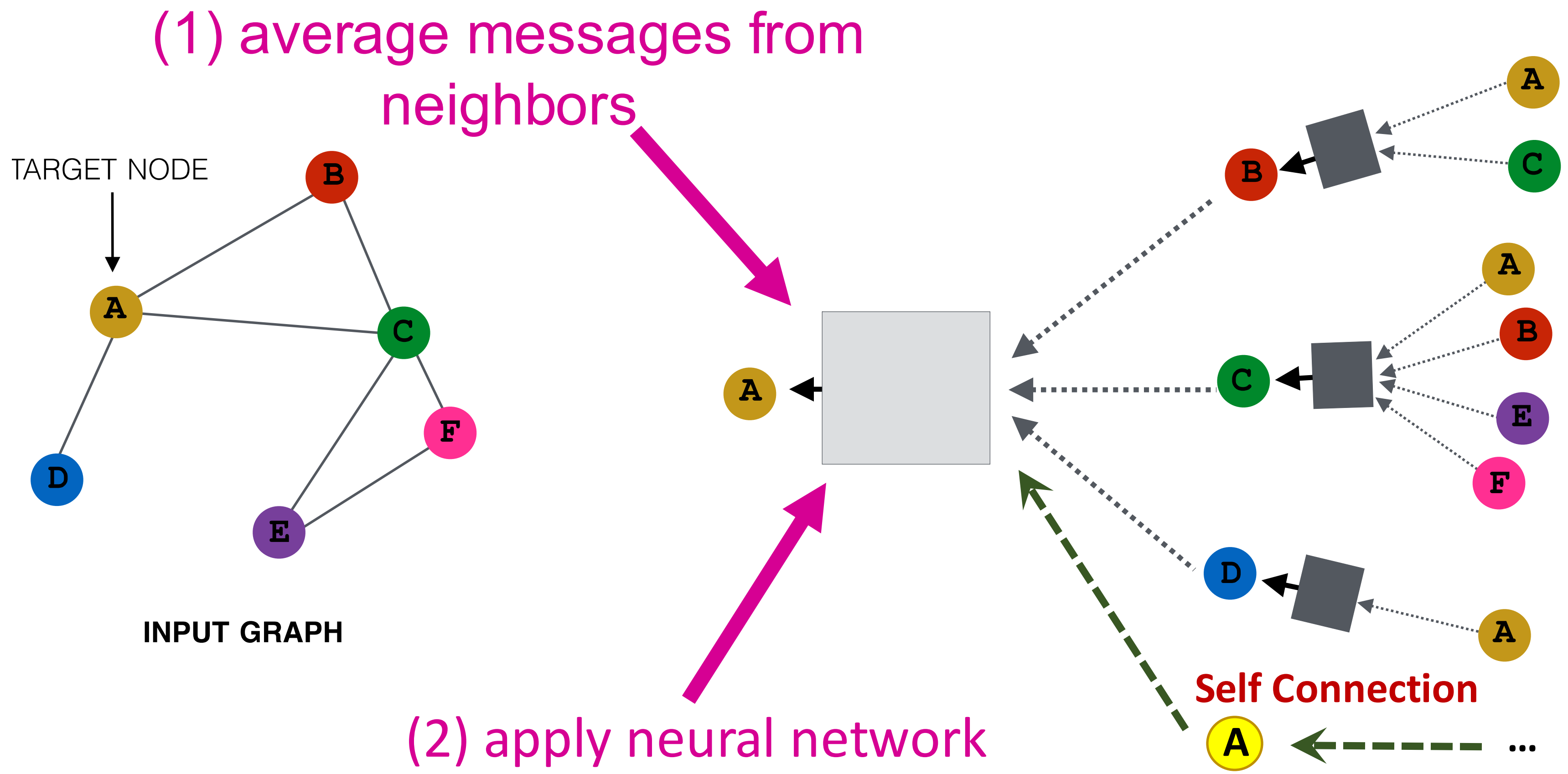
Neighborhood Aggregation (1)

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



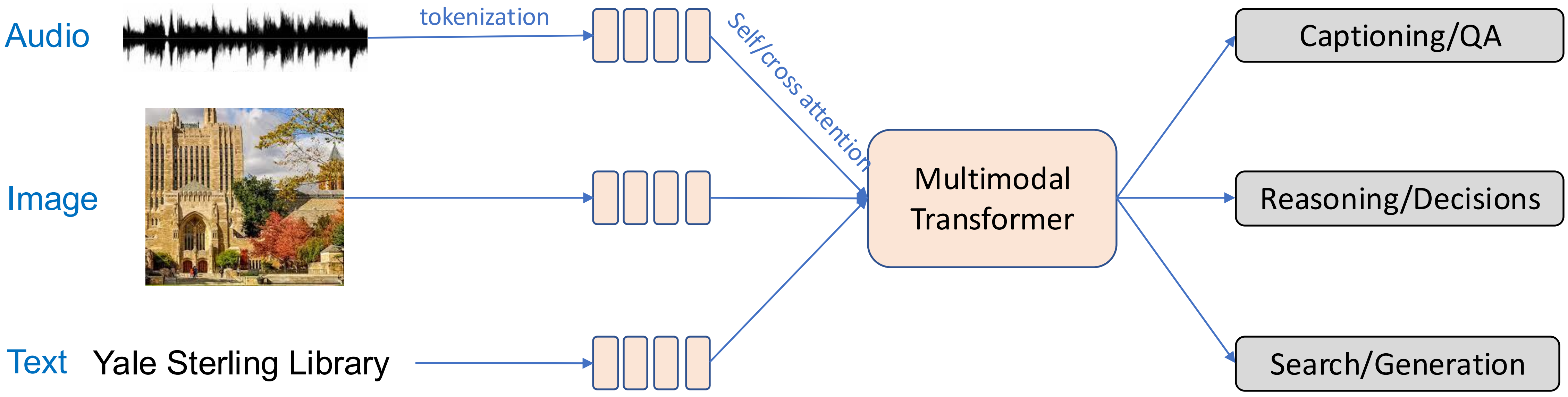
Neighborhood Aggregation (2)

- **Basic approach:** Average information from neighbors and apply a neural network



Transformers for Multiple Modalities

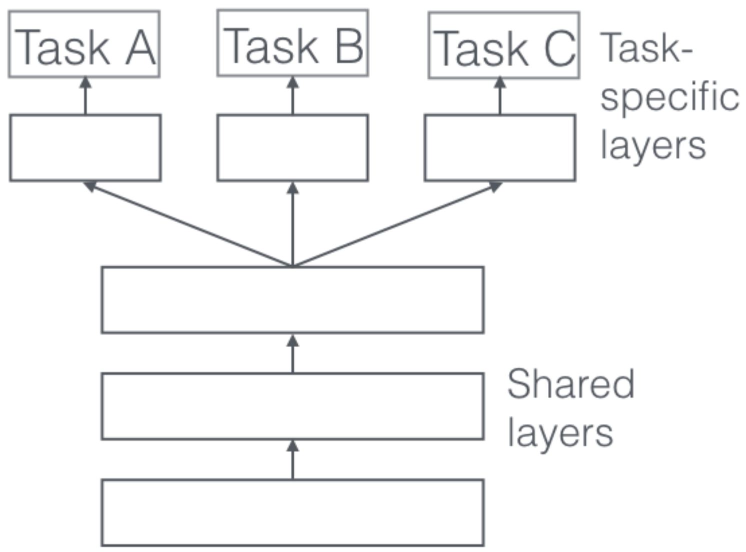
- Transformers succeed across domains, enabling **general-purpose AI systems** where one general model can handle multiple input modalities (text, images, audio, etc.)



- E.g., CLIP, DALL-E, Stable Diffusion, GPT-4V, Gemini, Flamingo

Why is Transformer a Popular Choice

- Resolves various challenges of RNN-based architectures
- Attention makes the architecture **expressive and flexible** for different application scenarios
- It is very amenable to **self-supervised objectives**
  - We can leverage the vast number of **unsupervised examples** to learn a general model
  - Can be fine-tuned for **many downstream tasks**
  - Can out-perform models that are only trained for a specific downstream task



Label Scarcity

- ML models are hungry to data, especially labeled data for supervised task.
- The fast development of computer vision largely benefits from **ImageNet**. It contains 14 million images **hand-annotated** by a team of researchers.
- This is often not possible for many domains. Most of time, it's easy to collect rich unlabeled data, but hard to obtain labeled data.
- **Solution**: **Pre-training** general-purpose language model on unlabeled large corpora (billions of characters) in **self-supervised** setting, then **fine-tuning** on smaller-scale tasks.