C. Attention and Transformers

C.1 Origins of attention

C.2 Attention as a standalone operator

C.3 Transformer architectures

C.4 Transformers across different data structures

C.5 Training transformers

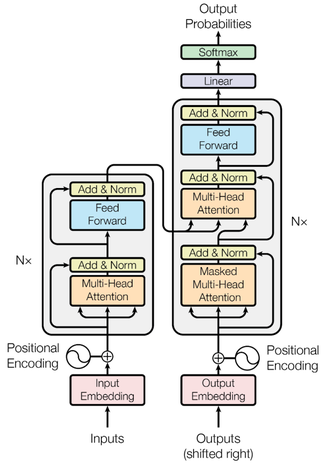
We will see that:

1) Transformers naturally fit *sets*
2) For *sequences* / *grids*, we add positional encoding
3) For *sequences and graphs* we inject graph structure

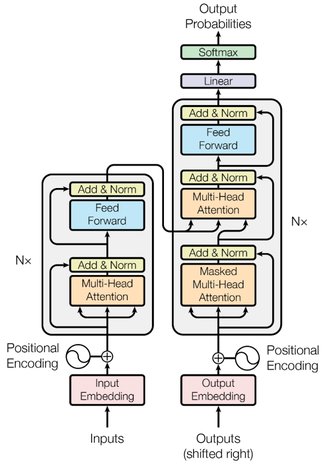
Tiam Heydari, PhD
Postdoctoral Fellow, UBC

Where Transformers are being used in 2026:

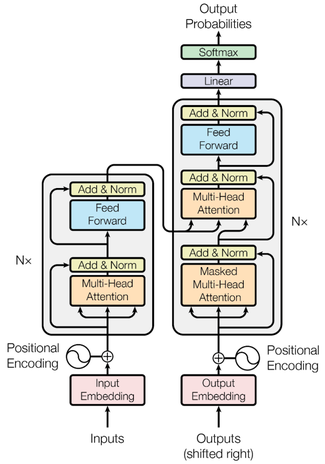
Language models



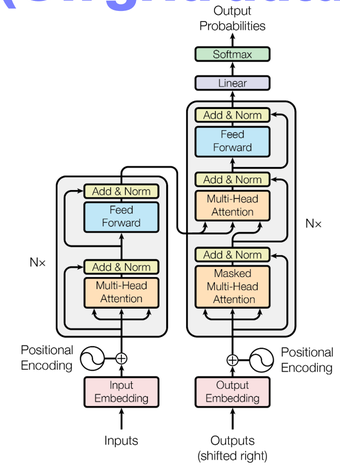
Translation



Speech

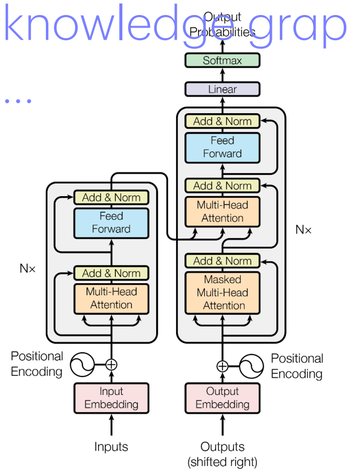


Computer vision (On grid data)



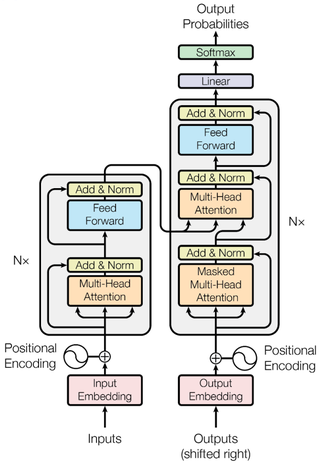
Graph-structured data

molecules
interaction networks,
knowledge graphs
...



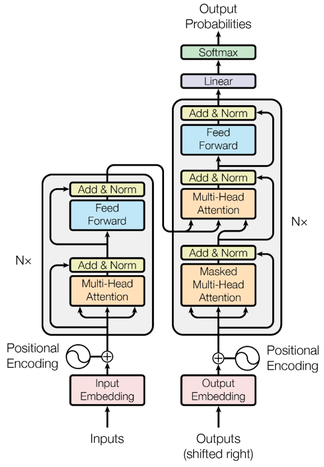
Genomics sequences

regulatory prediction
genome language models
...



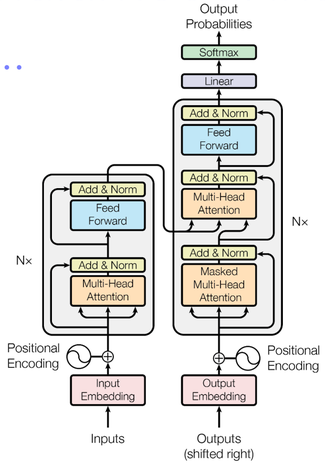
Protein sequences

structure prediction
protein design
...



Single cell omics

cell representation
annotation,
multimodal integration
....

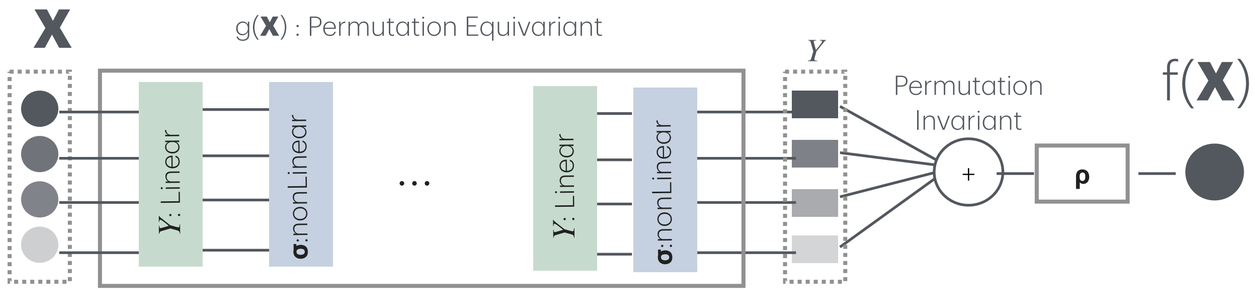


...

1) Transformers naturally fit ***sets***

Transformers are naturally fit to process sets

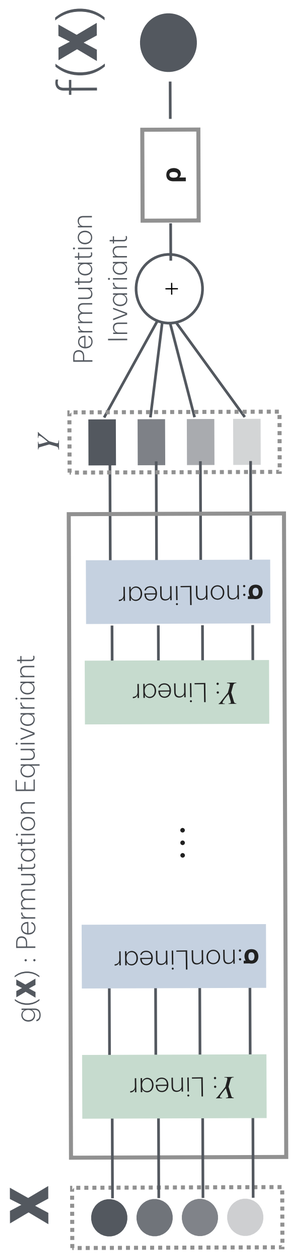
- Why is that? First remember our deep set architecture:



Transformers naturally process set-structured data

- Why is that? First remember our deep set architecture:

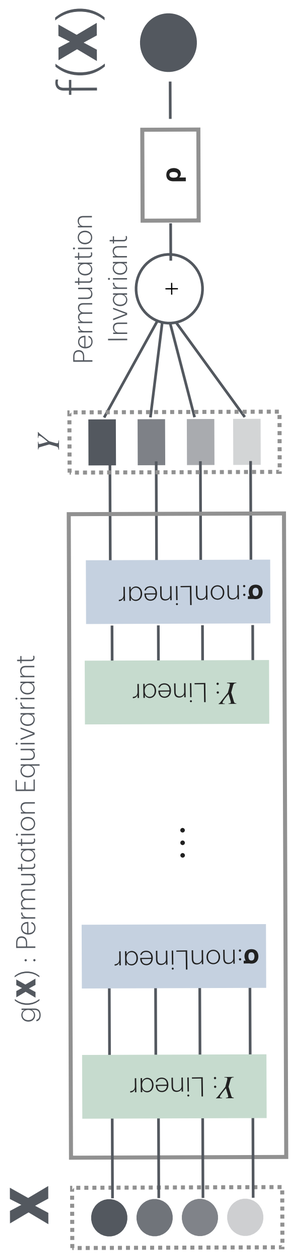
$$f(X) = \rho\left(\sum_{x \in X} g(x)\right)$$

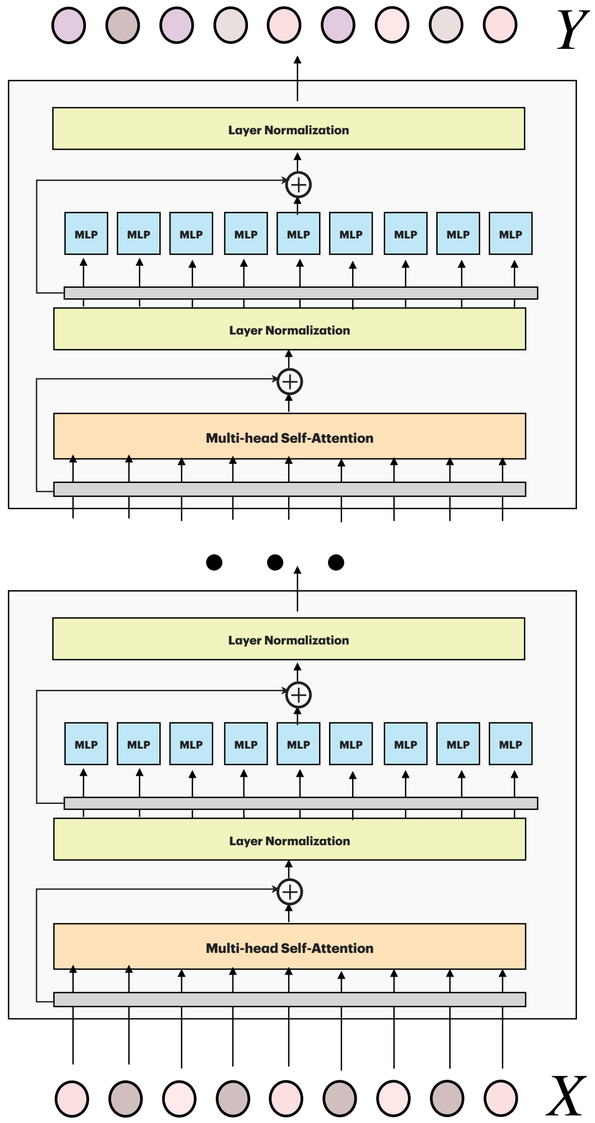


Transformers naturally process set-structured data

- Why is that? First remember our deep set architecture:

$$f(X) = \rho\left(\sum_{x \in X} g(x)\right)$$





Transformers naturally process set-structured data

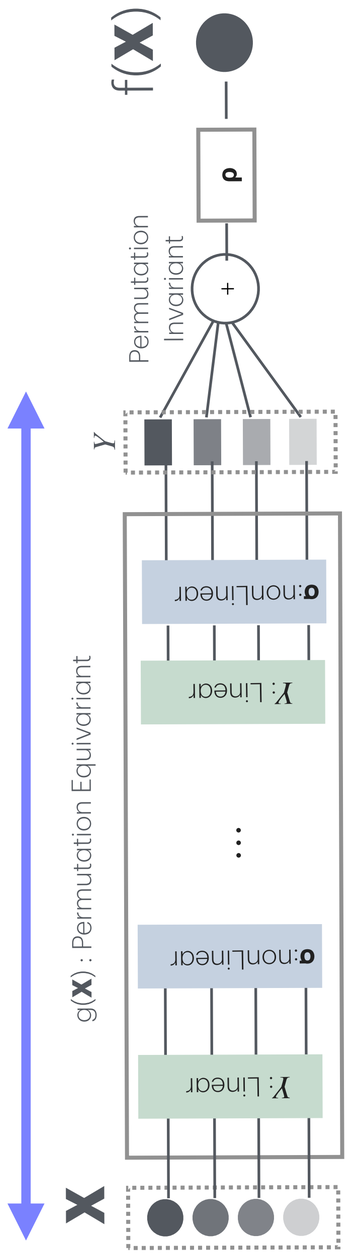
- Why is that? First remember our deep set architecture:

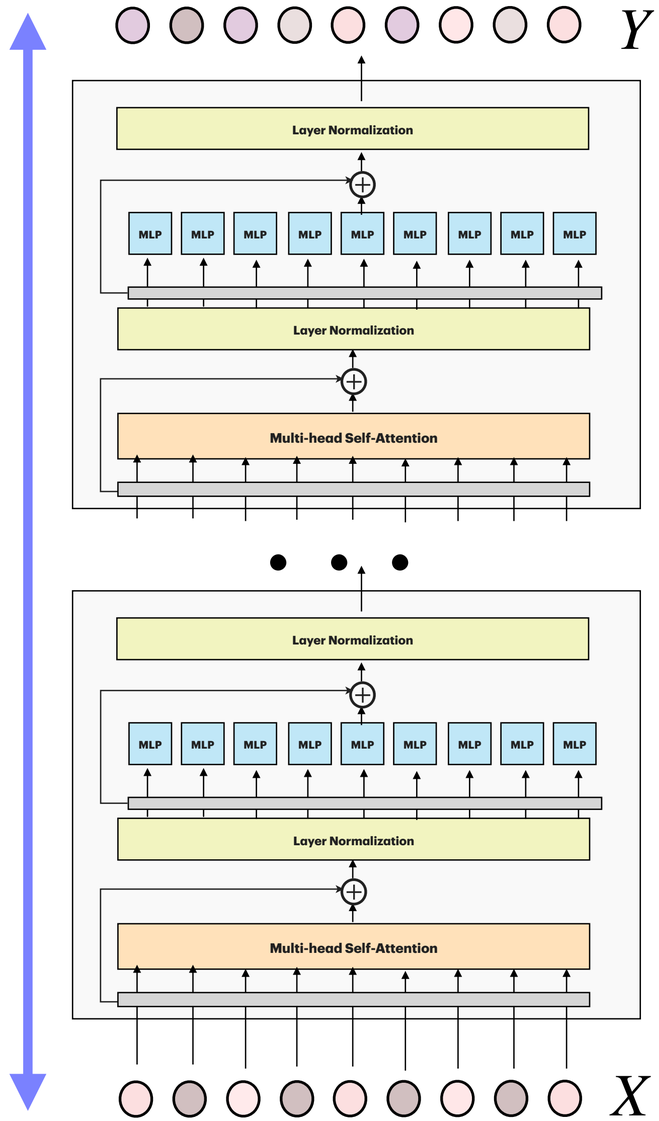
$$f(X) = \rho\left(\sum_{x \in X} g(x)\right)$$

- This means Transformer blocks can be used naturally as $g(x)$ in our deep sets!

Both Permutation Equivaraint

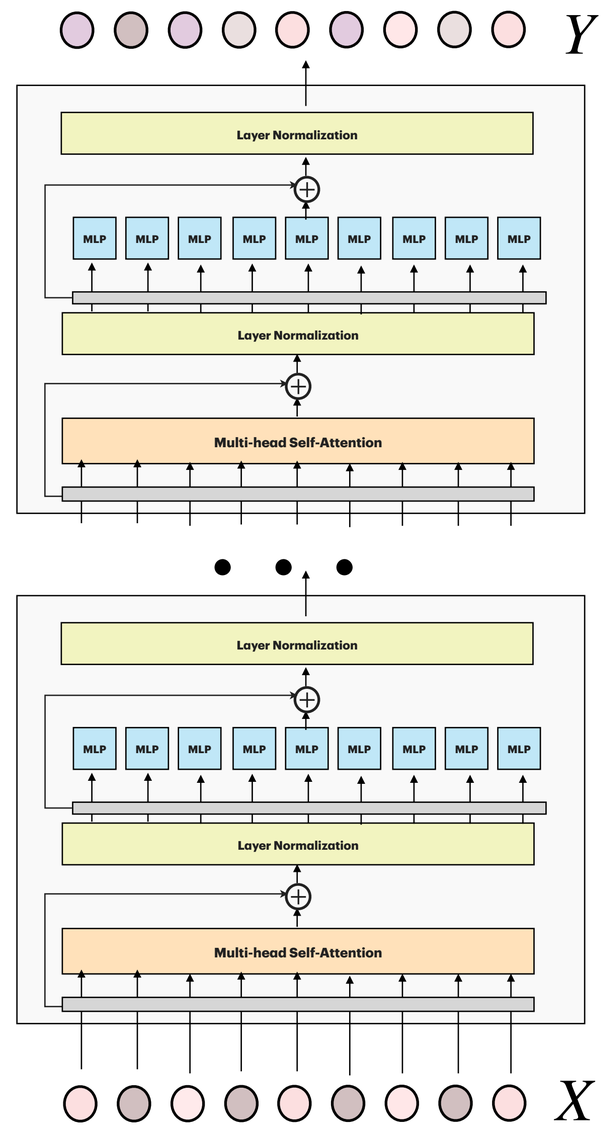
- Deep Sets need a permutation-equivariant set-to-set map; Transformer blocks provide exactly that.





Transformers naturally process set-structured data

- Attention (the core mechanism of Transformers) is naturally a **set operator:**
  - It takes a collection of vector elements in, and returns a collection of vectors out.
  - By itself, it does not know order, grid structure, or graph structure.



Computation phase: Each updated vector element goes through an individual transformation via MLP →

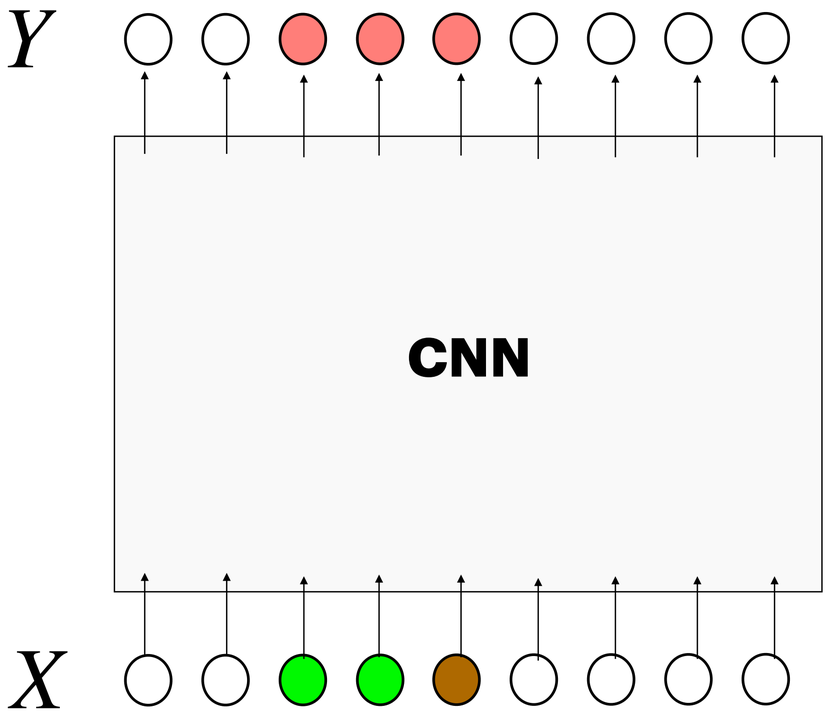
Communication Phase: Each vector element communicates with all elements and update itself →

Gets the input set of vector elements →

2) For ***sequences*** / ***grids***, we add positional encoding

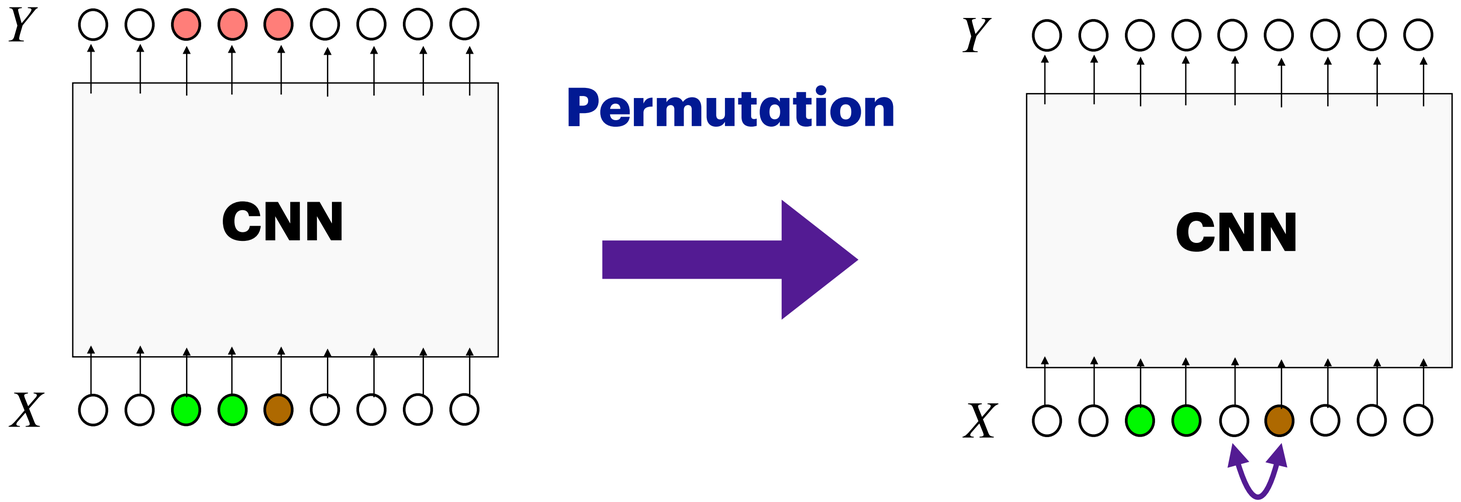
- In this section, our goal is to make some changes and tweak to the vanilla Transformer and make it more suitable to process grids (e.g. images) and sequences (e.g text, DNA, etc).
- But before doing those tweaks, we want to take few steps back, sand have a deeper view of the situation!

- Let's remind ourselves about what was the essence of **CNNs**:
  - Consider a CNN that does segmentation
    (for example for segmentation tasks, where we want to find **where a tree is**)



- Consider a CNN that does segmentation
  (for example for segmentation tasks, where we want to find **where a tree is**)
- The CNN is **Translation (T) Equivaraint:** $CNN(T\,.\,X) = T\,.\,CNN(X)$

- But, CNN is <u>not</u> **permutation (P) equivariant** in general: $CNN(P\,.\,X) \neq P\,.\,CNN(X)$



- Now we can ask the same question about **Transformers.**
  - Consider a Transformer on a similar input and on a similar segmentation task, where we want to find **where a tree is**.

  - The Transformer is **permutation (P) equivariant**: $Transformer(P\,.\,X) = P\,.\,Transformer(X)$

This is not good! Because there is no three where it predicted to be, but transformer has no way to understand local interactions, it sees the input as just an unstructured set!

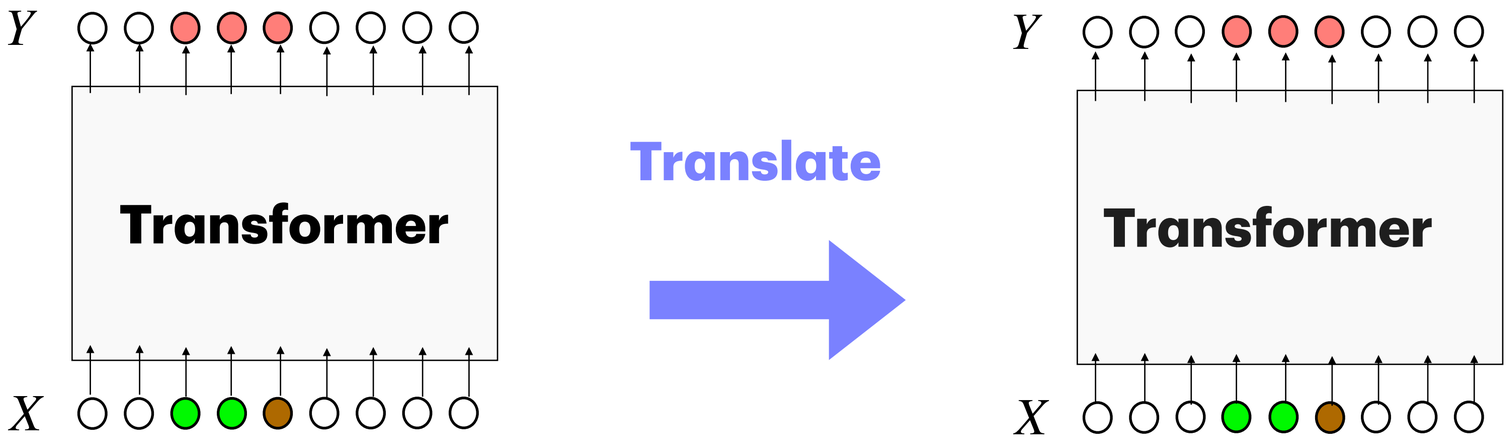
"Transformers: Distance-Blind Global Attention"

What about **translation**?

The key to investigate the effect of Translation (T) on transformers is:

“A circular Translation can be represented as a permutation of positions”

- Because (circular) Translation is basically a specific permutation:
  - **Translation (T) Equivaraint:** $Transformer(T\,.\,X) = T\,.\,Transformer(X)$



To summarize, for grid data:

We want Translation (T) Equivaraince	We don't want permutation (P) equivariance
👍 *CNN allows translation equivaraince*	👍 *CNN does not allow Permutation equivaraince*
👍 ***Transformer*** *allows has translation equivaraince*	👎 ***Transformer*** *allows Permutation equivaraince*

- Plain self-attention is permutation equivariant, that means it is naturally a set operator
- For sequences and grids, this symmetry is **too strong.**
- Transformer model is **geometry-blind**: it does not know before/after near/far row/column and local neighborhoods. It just assumes a bag of vectors.
- More data cannot fix this, because exact architectural symmetry cannot be broken by learning alone

Positional Encoding (PE)

- How can we solve this to be able to use Transformers in grids and sequences?
  - A nice trick: Positional encoding:
    - We inject location into each element by adding a positional vector:

$$\tilde{x}_i = x_i + p_i$$

    - Now two identical elements at different positions are no longer identical to the model

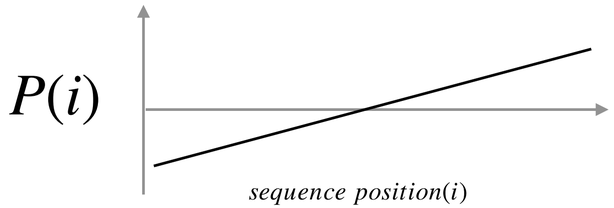
$$\underbrace{\tilde{X}}_{T \times d_x} = \underbrace{X}_{T \times d_x} + \underbrace{P}_{T \times d_x}$$

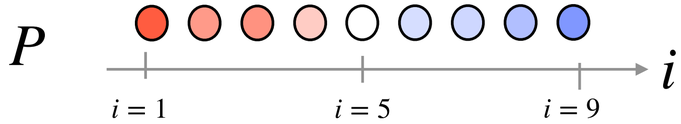
- **This is the most common way of adding positional encoding to Transformers**

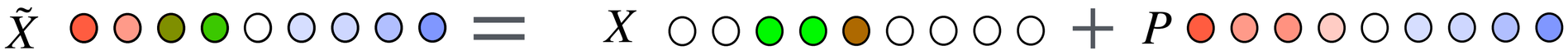
https://medium.com/@manishnegi101/understanding-positional-encoding-in-transformers-5fa6378878fc
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding (PE)

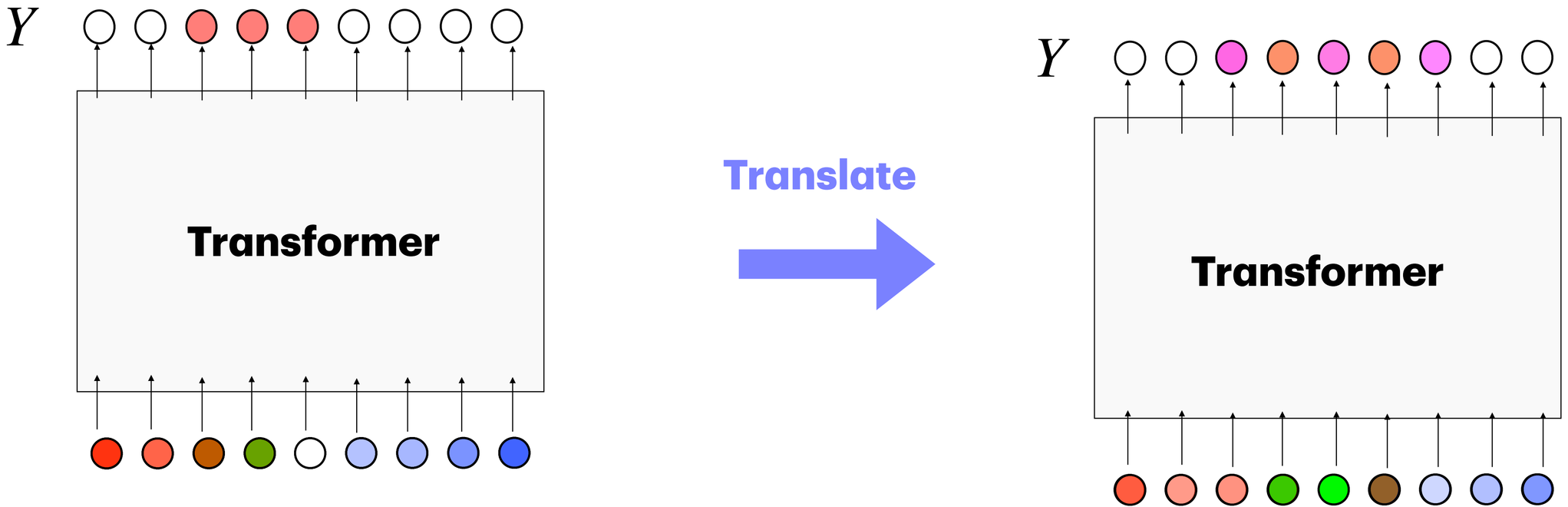
- An illustrative Example:







- But note that this type of positional encoding **breaks** both **permutation** and **translational** symmetry!
- It means it will takes more data for Transformers to learn relationships.
- But it works because Transformers are very expressive



To summarize, so far we saw:

- Plain self-attention is permutation equivariant so it is geometry-blind.
- For sequences/grids, that symmetry is too strong.
- Positional encoding (PE) fixes this by injecting location.
- But PE breaks translation equivariance as well, so we need more data to train the model

You might ask is there a more efficient way to inject PE? Is there a way to inject positional encoding without braking translation equivariance?

- Yes! **Relative Positional Encoding** (RPE) - used in models like minGPT, T5, and modern efficient transformers.

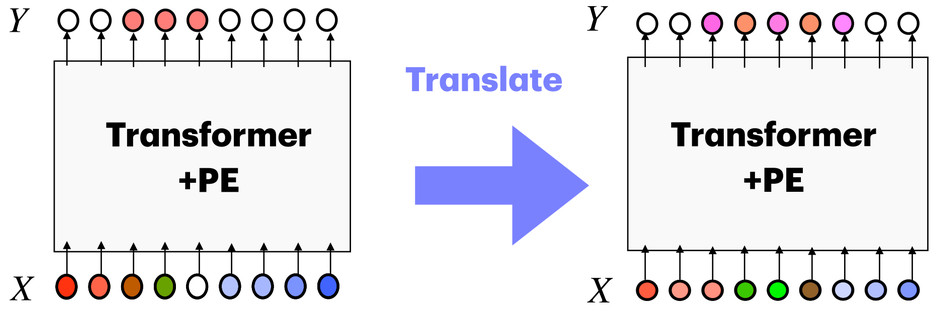
Relative Positional Encoding (RPE)

In **PE**: $\tilde{X} = X + P(i)$ ← Adds position to embeddings

Inside the transformer the Attention layer states intact:

$$Q = \tilde{X}W_Q, \qquad K = \tilde{X}W_K, \qquad V = \tilde{X}W_V$$

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



RE based models:

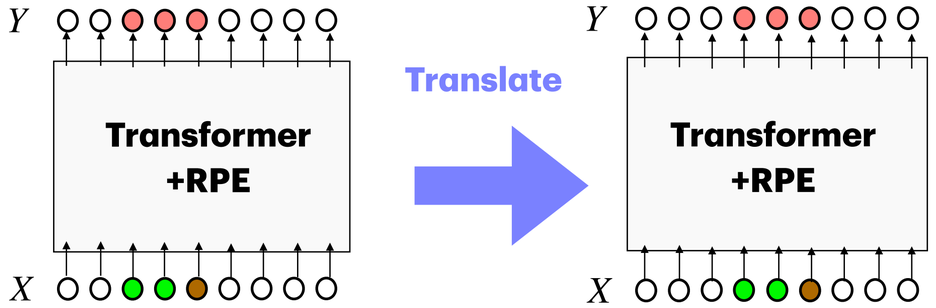
- Break translation equivariance (unlike CNNs)
- Less data-efficient for grids/sequences (must learn same pattern at each position)
- More expressive (can learn position-specific features)
- Simpler to implement and computationally cheaper

In **RPE**: $\tilde{X} = X$ (unchanged), but we modify attention layer.

One possibility is to use this:

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V$$

$$Y = \text{softmax}_{\text{row}}\left(\frac{QK^\top}{\sqrt{d_k}} + B^{\text{rel}}\right)V, \qquad B^{\text{rel}}_{ij} = b_{i-j}$$



PRE based models are

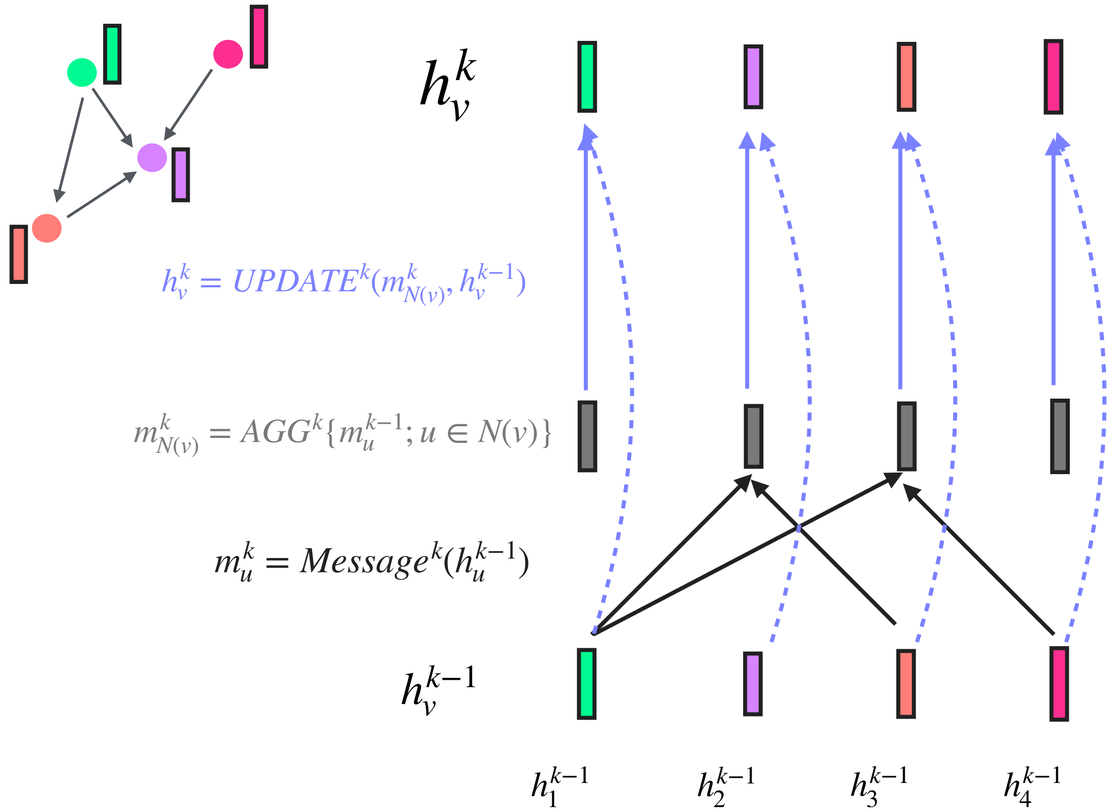
- Translation equivariant (like CNNs)
- More data-efficient for grids/sequences
- Less expressive than PE (can't learn position-specific features)
- Still expressive enough for complex tasks

Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani. "Self-attention with relative position representations." *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 2018.

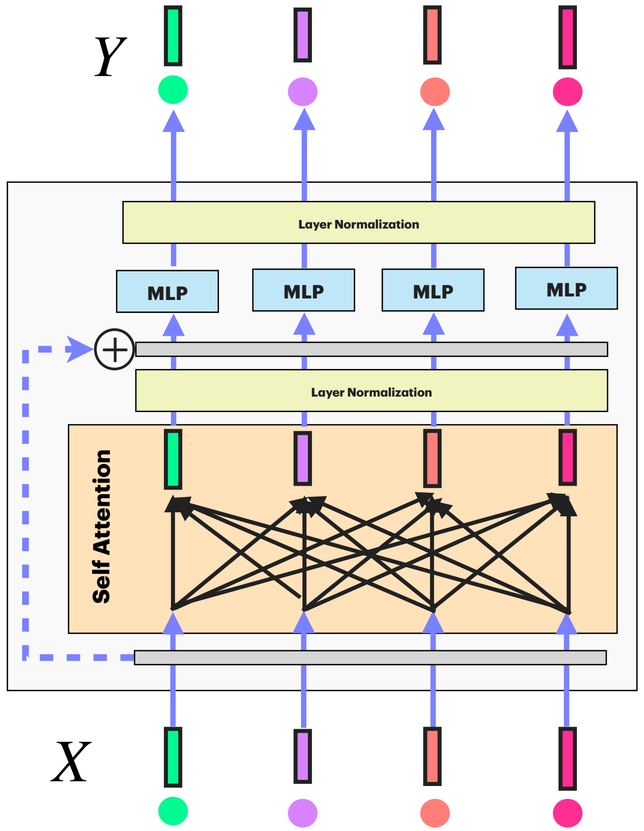
3) For ***sequences and graphs*** we inject graph structure

- Lets remind ourselves about computation graphs of GNNs and Transformers.
- Just looking at it, will be very revealing about what we should focus on to be able to process graphs via Transformers

GNN computation graph

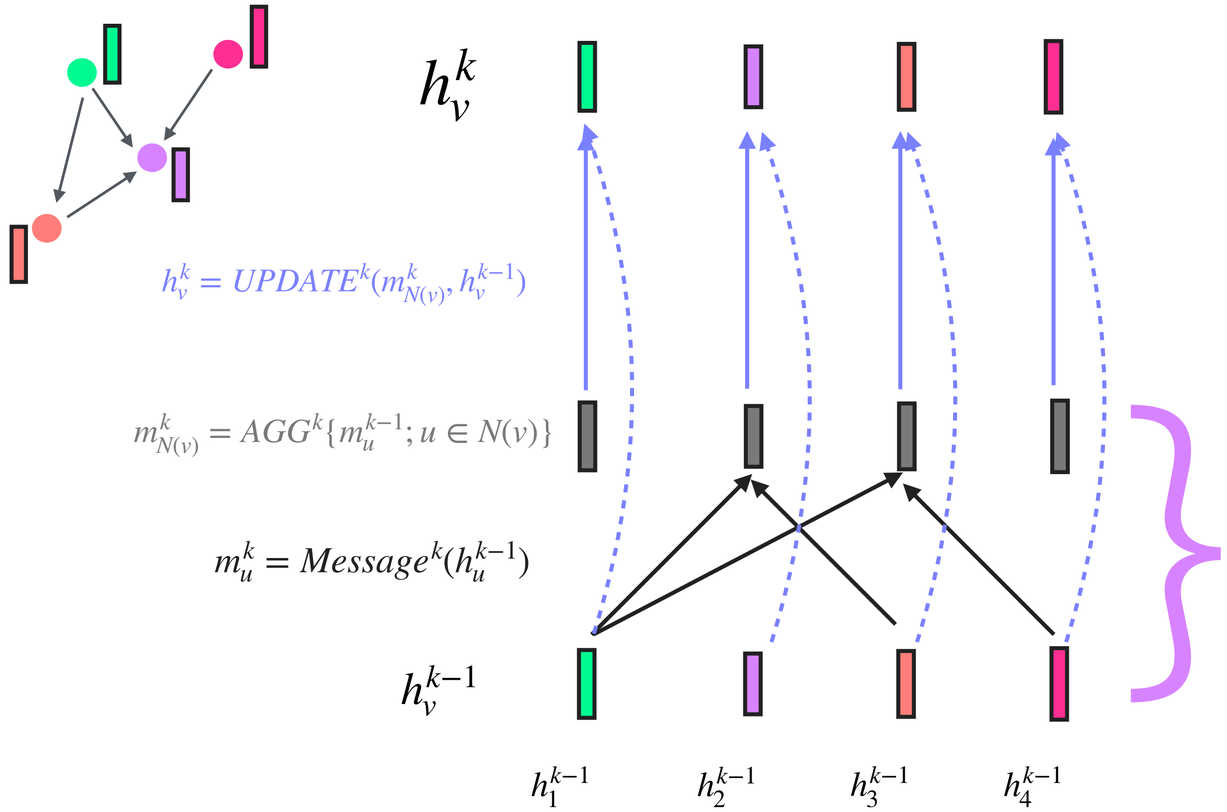


Transformer computation graph



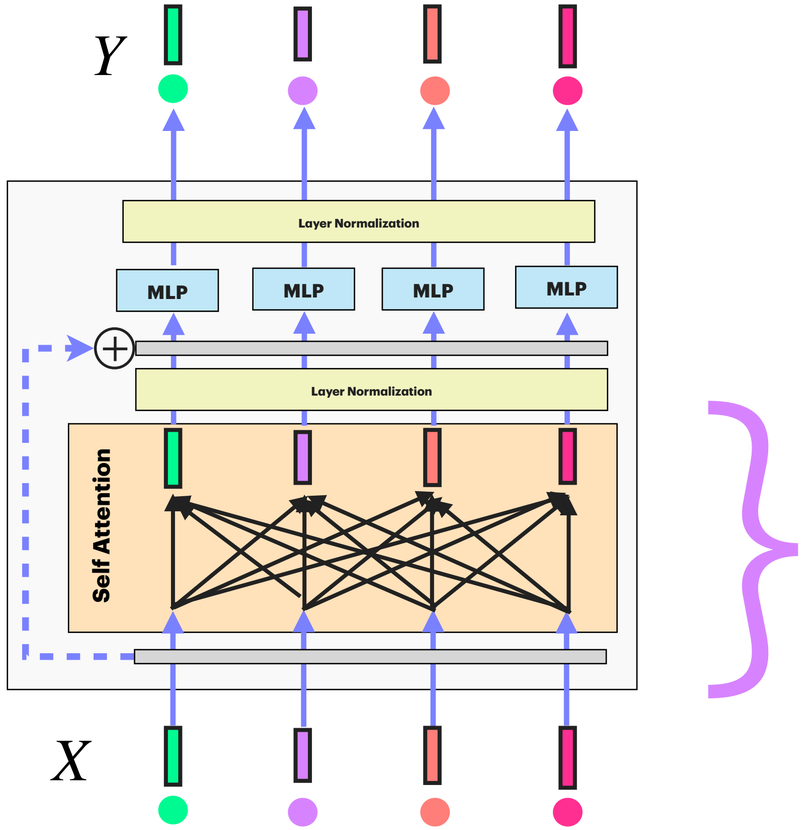
- Lets remind ourselves about computation graphs of GNNs and Transformers.
- Just looking at it, will be very revealing about what we should focus on to be able to process graphs via Transformers

GNN computation graph



Message passing follows graph edges only

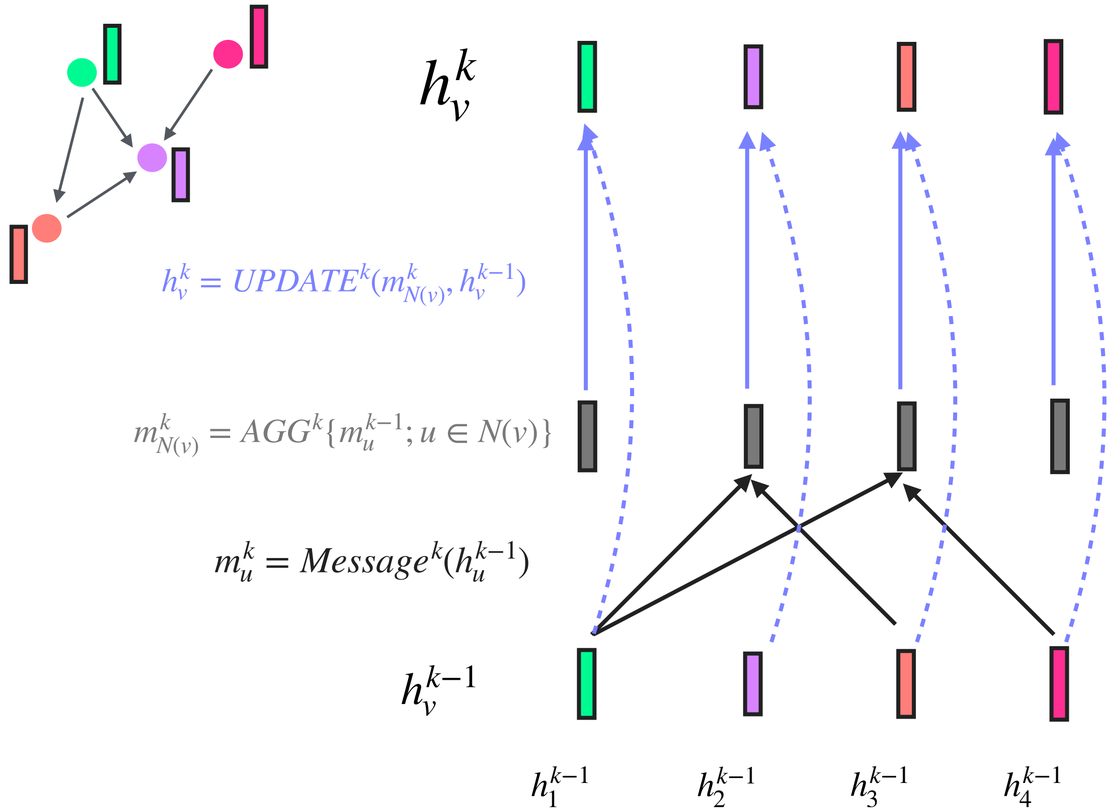
Transformer computation graph



Self-attention is fully connected

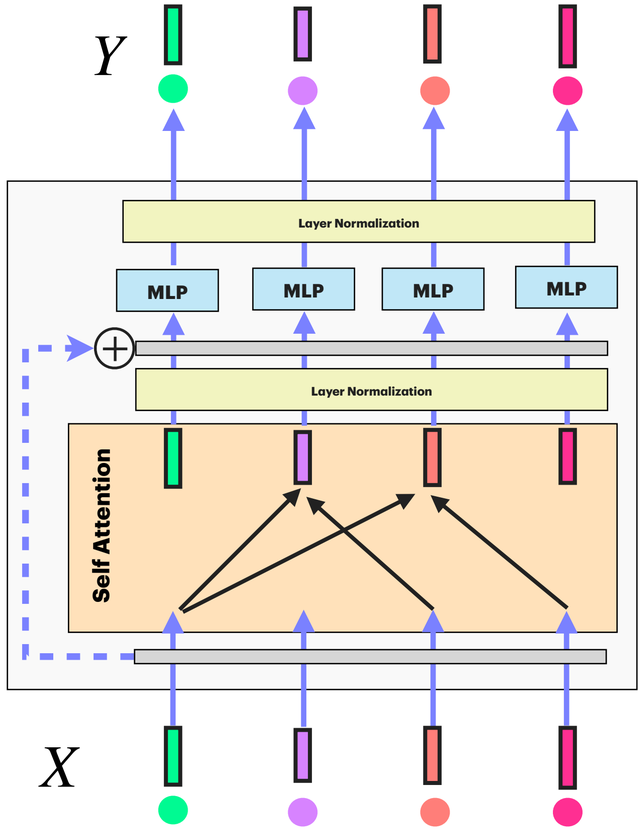
- **The Solution**: Making Transformers Self Attention to only attend to the relevant elements.
- This is called **Masked Attention**

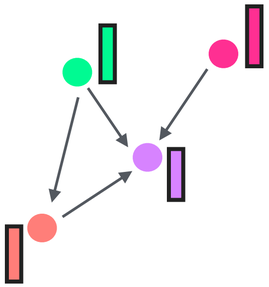
GNN computation graph



Message passing follows graph edges only

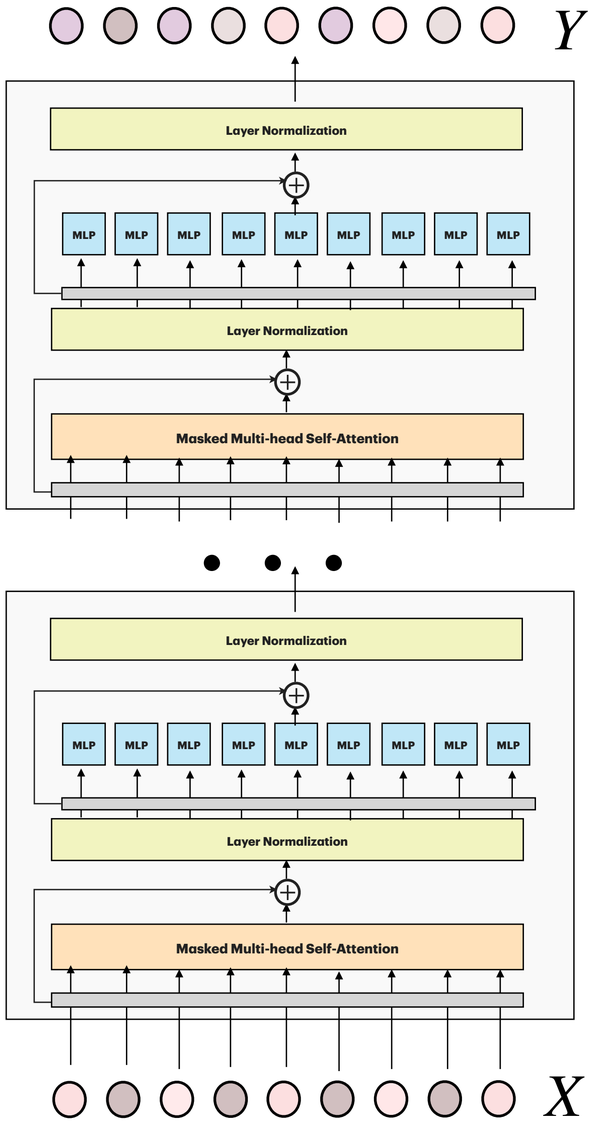
Suggested **Graph Transformer** computation graph



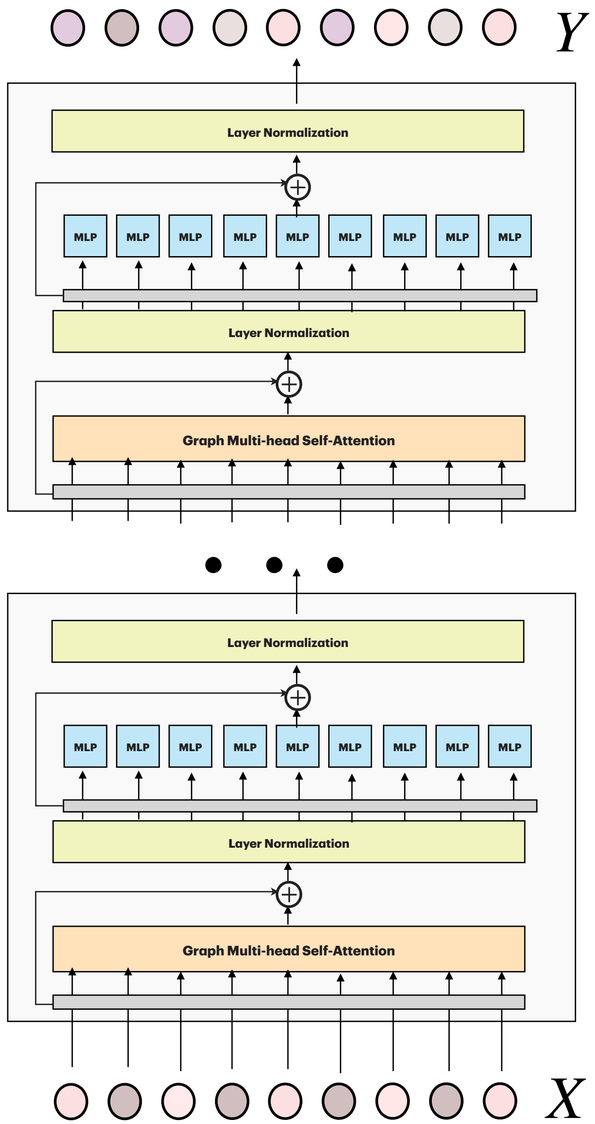


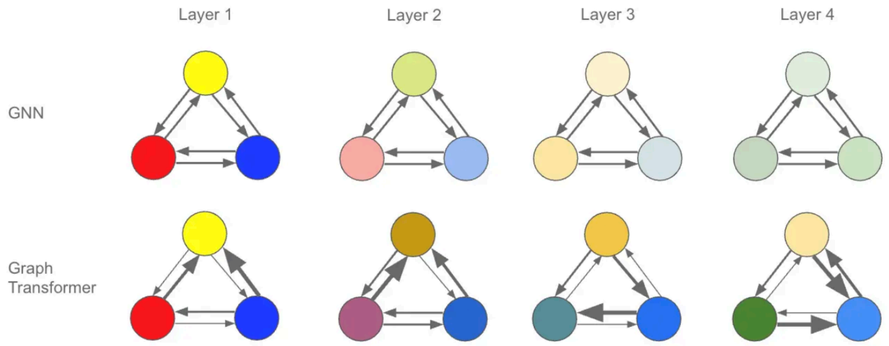
Masked Self-attention follows graph edges only

Graph Transformers: Graph Transformer introduces an inductive bias by incorporating the graph's topology into the attention mechanism. Instead of allowing every node to attend to all others

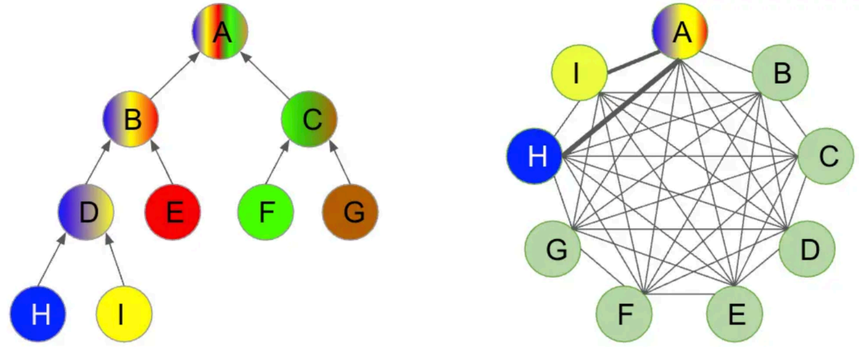


Graph Transformers V2: Another variant will just add biases to the relevant attention scores, offering greater flexibility and long-range modeling at the cost of higher computational complexity and potential loss of inductive bias.





Top: Deep GNNs cause node features to blur, leading to over-smoothing. Bottom: Graph Transformers use varied attention to preserve node uniqueness and relevance.



Left: GNNs compress distant signals through bottlenecks like node A, losing detail. Right: Graph Transformers use global attention, letting A directly access H and I, avoiding over-squashing.

Aspect	Graph Neural Networks	Graph Transformers
Information Flow	Sequential message passing between connected nodes.	Self-attention allows direct aggregation from all nodes.
Long-Range Dependencies	Struggle with distant relationships due to limited hops.	Capture long-range dependencies more effectively.
Over-Smoothing	Node embeddings become too similar with deep layers.	Less prone, as attention allows flexible information mixing.
Over-Squashing	Information gets compressed when aggregating many neighbors.	Mitigated by attention, distributing information more effectively.
Heterogeneous Models	RGCN, HAN, HGT	HGT, Graphormer, HEAT

PE implementations also might be different in GTs.

https://kumo.ai/research/introduction-to-graph-transformers/

Graph Transformers V2: Another variant will just add biases to the relevant attention scores, offering greater flexibility and long-range modeling at the cost of higher computational complexity and potential loss of inductive bias.

Aspect	Pros	Cons
Information Flow	Decouples node updates from graph structure, enabling flexible learning.	Loses the strong locality bias that makes GNNs effective on many structured graphs.
Long-Range Dependencies	Can model distant node interactions naturally through self-attention.	Fully connected attention can lead to unnecessary noise in learning.
Graph Structure Awareness	Can incorporate edge features and relational information.	Requires careful design to ensure graph structure is not lost.
Computational Complexity	No sequential message passing bottleneck, allows parallel computation.	Attention mechanism has $O(N^2)$ complexity vs. O(E) for GNNs, making scalability a challenge for large graphs.
Positional Encoding	Can use graph-aware positional encodings to differentiate nodes.	Designing effective positional encodings for graphs remains an active research area.
Heterogeneous Graphs	Handles multiple node and edge types effectively via attention.	More complex to implement than standard GNN architectures for heterogeneous data.
Flexibility	Allows for learned adaptive neighborhood aggregation instead of fixed message passing.	Without strong inductive biases, requires more data to generalize well.

Excellent blog post:

https://kumo.ai/research/introduction-to-graph-transformers/

Refrences:

A Generalization of Transformer Networks to Graphs Vijay Prakash Dwivedi, Xavier Bresson arXiv:2012.09699

Recipe for a General, Powerful, Scalable Graph Transformer Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, Dominique Beaini arXiv:2205.12454