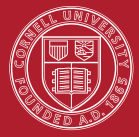


Cornell University

Lecture 23 – Graph Transformers and Foundation Models

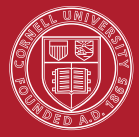
ECE 5260 – ORIE 5735

Anna Scaglione

Cornell Tech

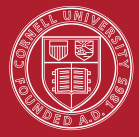
April 27, 2026

Content



Cornell University

Outline



Cornell University

Where we left off

- Lectures 21 and 22 built a GNN as a stack of MIMO graph perceptrons

$$\mathbf{X}_\ell = \sigma[\mathbf{Z}_\ell] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right], \qquad \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H}),\ \mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L].$$

- The graph shift operator $\mathbf{S}$ was *fixed*: an adjacency, a normalized Laplacian, or a weighted distance kernel.
- Two structural consequences:
  - At layer ℓ, node i can only look $K-1$ hops away $\Rightarrow$ bounded receptive field per layer.
  - Every neighbor contributes through $\mathbf{S}$ with the *same* weight for every signal $\Rightarrow$ weights do not adapt to the content of $\mathbf{X}$.
- Two empirical pains:
  - **Over-smoothing**: deep GCNs collapse node embeddings to eigenvectors of $\mathbf{S}$.
  - **Bottlenecks**: long-range interactions must be through $\mathbf{S}^k$.

Where we left off

- Lectures 21 and 22 built a GNN as a stack of MIMO graph perceptrons

$$\mathbf{X}_\ell = \sigma[\mathbf{Z}_\ell] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right], \qquad \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H}),\ \mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L].$$

- The graph shift operator $\mathbf{S}$ was *fixed*: an adjacency, a normalized Laplacian, or a weighted distance kernel.
- Two structural consequences:
  - At layer ℓ, node i can only look $K-1$ hops away $\Rightarrow$ bounded receptive field per layer.
  - Every neighbor contributes through $\mathbf{S}$ with the *same* weight for every signal $\Rightarrow$ weights do not adapt to the content of $\mathbf{X}$.
- Two empirical pains:
  - **Over-smoothing**: deep GCNs collapse node embeddings to eigenvectors of $\mathbf{S}$.
  - **Bottlenecks**: long-range interactions must be through $\mathbf{S}^k$.

Where we left off

- Lectures 21 and 22 built a GNN as a stack of MIMO graph perceptrons

$$\mathbf{X}_\ell = \sigma[\mathbf{Z}_\ell] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right], \qquad \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H}),\ \mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L].$$

- The graph shift operator $\mathbf{S}$ was *fixed*: an adjacency, a normalized Laplacian, or a weighted distance kernel.
- Two structural consequences:
  - At layer ℓ, node i can only look $K-1$ hops away $\Rightarrow$ bounded receptive field per layer.
  - Every neighbor contributes through $\mathbf{S}$ with the *same* weight for every signal $\Rightarrow$ weights do not adapt to the content of $\mathbf{X}$.
- Two empirical pains:
  - **Over-smoothing**: deep GCNs collapse node embeddings to eigenvectors of $\mathbf{S}$.
  - **Bottlenecks**: long-range interactions must be through $\mathbf{S}^k$.

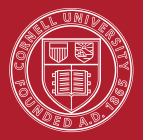
Where we left off

- Lectures 21 and 22 built a GNN as a stack of MIMO graph perceptrons

$$\mathbf{X}_\ell = \sigma[\mathbf{Z}_\ell] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right], \qquad \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H}),\ \mathcal{H} = [\mathbf{H}_1, \ldots, \mathbf{H}_L].$$

- The graph shift operator $\mathbf{S}$ was *fixed*: an adjacency, a normalized Laplacian, or a weighted distance kernel.
- Two structural consequences:
  - At layer ℓ, node i can only look $K-1$ hops away $\Rightarrow$ bounded receptive field per layer.
  - Every neighbor contributes through $\mathbf{S}$ with the *same* weight for every signal $\Rightarrow$ weights do not adapt to the content of $\mathbf{X}$.
- Two empirical pains:
  - **Over-smoothing**: deep GCNs collapse node embeddings to eigenvectors of $\mathbf{S}$.
  - **Bottlenecks**: long-range interactions must be through $\mathbf{S}^k$.

Can we learn the graph shift?

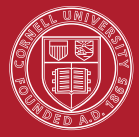


- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

Can we learn the graph shift?



- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

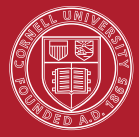
Can we learn the graph shift?

- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

Can we learn the graph shift?

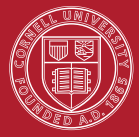


- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

Can we learn the graph shift?

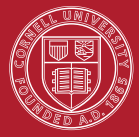


- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

Can we learn the graph shift?

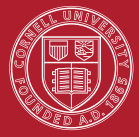


- What if $\mathbf{S}$ itself were a function of the data $\mathbf{X}$, learned end-to-end?

$$\mathbf{S} \longrightarrow \mathbf{S}(\mathbf{X}; \boldsymbol{\theta}).$$

- The *attention mechanism* is exactly that: a parametrized, data-dependent shift operator.
- Preview of the lecture:
  - define *self-attention*, the cornerstone of "Transformers";
  - read it as a $K = 1$ graph filter with a learned GSO;
  - extend it to graphs with positional/structural encodings (Graphormer, GraphGPS);
  - extend to *heterogeneous* graphs and knowledge graphs (RGCN, HGT);
  - end on *foundation models* – language and graph foundation models.

Outline



Cornell University

Self-attention on a set of tokens

- Input: an $N \times F$ matrix $\mathbf{X}$ whose rows $\boldsymbol{x}_i^\top$ are token embeddings (same $\mathbf{X}$ we used for graph signals, now viewed as a sequence).
- Three learned linear maps – the query, key, value projections:

$$\mathbf{Q} = \mathbf{X}\,\mathbf{W}^Q, \qquad \mathbf{K} = \mathbf{X}\,\mathbf{W}^K, \qquad \mathbf{V} = \mathbf{X}\,\mathbf{W}^V,$$

  with $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{F \times d_k}$ and $\mathbf{W}^V \in \mathbb{R}^{F \times d_v}$.
- The attention matrix is

$$\mathbf{A}(\mathbf{X}) = \mathrm{softmax}\left(\tfrac{1}{\sqrt{d_k}}\,\mathbf{Q}\mathbf{K}^\top\right) \in \mathbb{R}^{N \times N},$$

  with row-wise softmax: $[\mathbf{A}]_{ij} = \dfrac{\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d_k})}{\sum_{j'} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_{j'} / \sqrt{d_k})}$.
- The layer output is

$$\mathrm{Attn}(\mathbf{X}) = \mathbf{A}(\mathbf{X})\,\mathbf{V} \;=\; \mathbf{A}(\mathbf{X})\,\mathbf{X}\,\mathbf{W}^V.$$

Multi-head attention and the transformer block

- C heads in parallel, each with its own $(\mathbf{W}^{Q,c}, \mathbf{W}^{K,c}, \mathbf{W}^{V,c})$:

$$\mathbf{head}^c = \mathrm{softmax}\Big(\tfrac{1}{\sqrt{d_k}}\mathbf{X}\mathbf{W}^{Q,c}(\mathbf{X}\mathbf{W}^{K,c})^\top\Big)\,\mathbf{X}\,\mathbf{W}^{V,c}.$$

- Concatenate and project: $\mathrm{MHA}(\mathbf{X}) = [\mathbf{head}^1, \ldots, \mathbf{head}^C]\,\mathbf{W}^O$.
- One transformer block applies MHA, a position-wise feed-forward network, and uses residual connections and (Post-)LayerNorm (makes the features dimensions zero mean and with unit variance):

$$\begin{aligned}
\mathbf{X}' &= \mathrm{LN}\big(\mathbf{X} + \mathrm{MHA}(\mathbf{X})\big),\\
\mathbf{X}_\ell &= \mathrm{LN}\big(\mathbf{X}' + \mathrm{FFN}(\mathbf{X}')\big),\\
\mathrm{FFN}(\boldsymbol{x}_i) &= \mathrm{ReLU}(\boldsymbol{x}_i\mathbf{W}_1 + \boldsymbol{b}_1)\mathbf{W}_2 + \boldsymbol{b}_2.
\end{aligned}$$

- The residual stream $\mathbf{X} \to \mathbf{X}' \to \mathbf{X}_\ell$ is where all information accumulates across blocks.

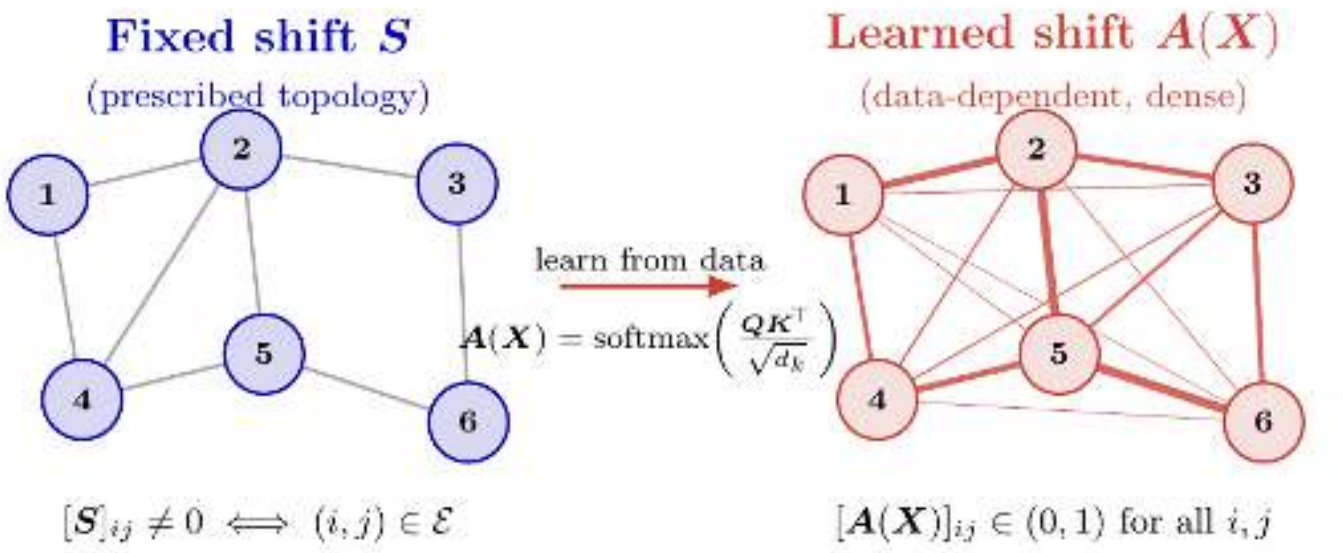
Attention as a data-dependent graph shift

- Compare the MIMO graph perceptron and a single attention head:

  Graph filter ($K = 1$): $\mathbf{Z}_\ell = \mathbf{S}\,\mathbf{X}_{\ell-1}\,\mathbf{H}_{\ell 0}$.

  Attention head: $\mathrm{Attn}(\mathbf{X}_{\ell-1}) = \mathbf{A}(\mathbf{X}_{\ell-1})\,\mathbf{X}_{\ell-1}\,\mathbf{W}^V$.

- Structurally identical: $\mathbf{A}(\mathbf{X})$ plays the role of $\mathbf{S}$; $\mathbf{W}^V$ plays the role of $\mathbf{H}_{\ell 0}$.



Same functional form, different S: $\boldsymbol{X}_\ell = \sigma\big[\boldsymbol{S}\,\boldsymbol{X}_{\ell-1}\boldsymbol{H}_{\ell 0}\big]$ vs. $\boldsymbol{X}_\ell = \sigma\big[\boldsymbol{A}(\boldsymbol{X}_{\ell-1})\,\boldsymbol{X}_{\ell-1}\boldsymbol{W}^V\big]$

Attention as a data-dependent graph shift

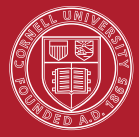
- Compare the MIMO graph perceptron and a single attention head:

  Graph filter ($K = 1$): $\mathbf{Z}_\ell = \mathbf{S}\,\mathbf{X}_{\ell-1}\,\mathbf{H}_{\ell 0}$.

  Attention head: $\mathrm{Attn}(\mathbf{X}_{\ell-1}) = \mathbf{A}(\mathbf{X}_{\ell-1})\,\mathbf{X}_{\ell-1}\,\mathbf{W}^V$.

- Structurally identical: $\mathbf{A}(\mathbf{X})$ plays the role of $\mathbf{S}$; $\mathbf{W}^V$ plays the role of $\mathbf{H}_{\ell 0}$.
- Key differences:
  - $\mathbf{A}(\mathbf{X})$ is *learned* and *content-dependent*; $\mathbf{S}$ was fixed prior information.
  - $\mathbf{A}(\mathbf{X})$ is dense ($N \times N$): every token attends to every other token.
  - No $\mathbf{S}^k$ powers: one head = one hop of a *learned* graph.
- This is the core reason transformers dominate for unstructured data (text, images as patches): when you have no good prior graph, learning $\mathbf{S}$ from data wins over fixing it.

Outline



Cornell University

Why a *graph* transformer?

- On graphs we do have a meaningful structure $\mathbf{S}$, and throwing it away costs us:
  - quadratic memory $\mathcal{O}(N^2)$ per layer,
  - permutation-invariant but structure-blind: two nodes far apart in the graph look identical to two nodes that are adjacent.
- A *graph transformer* re-introduces $\mathbf{S}$ into attention in two complementary ways:
  1. **Structural bias**: condition the attention logits on graph structure

$$[\mathbf{A}(\mathbf{X})]_{ij} \propto \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}} + b_{ij}(\mathbf{S})\right),$$

     where $b_{ij}(\mathbf{S})$ encodes e.g. shortest-path distance or $[\mathbf{S}^k]_{ij}$.
  2. **Positional encoding**: augment $\mathbf{X}$ with a node descriptor $\mathbf{P}(\mathbf{S})$ so that attention *sees* the graph, $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{P}(\mathbf{S})]$.
- We now survey both, and how they combine with Lecture 21.

Why a *graph* transformer?

Cornell University

- On graphs we do have a meaningful structure $\mathbf{S}$, and throwing it away costs us:
  - quadratic memory $\mathcal{O}(N^2)$ per layer,
  - permutation-invariant but structure-blind: two nodes far apart in the graph look identical to two nodes that are adjacent.
- A *graph transformer* re-introduces $\mathbf{S}$ into attention in two complementary ways:
  1. **Structural bias**: condition the attention logits on graph structure

$$[\mathbf{A}(\mathbf{X})]_{ij} \propto \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}} + b_{ij}(\mathbf{S})\right),$$

     where $b_{ij}(\mathbf{S})$ encodes e.g. shortest-path distance or $[\mathbf{S}^k]_{ij}$.
  2. **Positional encoding**: augment $\mathbf{X}$ with a node descriptor $\mathbf{P}(\mathbf{S})$ so that attention *sees* the graph, $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{P}(\mathbf{S})]$.
- We now survey both, and how they combine with Lecture 21.

Why a *graph* transformer?

- On graphs we do have a meaningful structure $\mathbf{S}$, and throwing it away costs us:
  - quadratic memory $\mathcal{O}(N^2)$ per layer,
  - permutation-invariant but structure-blind: two nodes far apart in the graph look identical to two nodes that are adjacent.
- A *graph transformer* re-introduces $\mathbf{S}$ into attention in two complementary ways:
  1. **Structural bias**: condition the attention logits on graph structure

$$[\mathbf{A}(\mathbf{X})]_{ij} \propto \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}} + b_{ij}(\mathbf{S})\right),$$

     where $b_{ij}(\mathbf{S})$ encodes e.g. shortest-path distance or $[\mathbf{S}^k]_{ij}$.
  2. **Positional encoding**: augment $\mathbf{X}$ with a node descriptor $\mathbf{P}(\mathbf{S})$ so that attention *sees* the graph, $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{P}(\mathbf{S})]$.
- We now survey both, and how they combine with Lecture 21.

Why a *graph* transformer?

- On graphs we do have a meaningful structure $\mathbf{S}$, and throwing it away costs us:
  - quadratic memory $\mathcal{O}(N^2)$ per layer,
  - permutation-invariant but structure-blind: two nodes far apart in the graph look identical to two nodes that are adjacent.
- A *graph transformer* re-introduces $\mathbf{S}$ into attention in two complementary ways:
  1. **Structural bias**: condition the attention logits on graph structure

$$[\mathbf{A}(\mathbf{X})]_{ij} \propto \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}} + b_{ij}(\mathbf{S})\right),$$

     where $b_{ij}(\mathbf{S})$ encodes e.g. shortest-path distance or $[\mathbf{S}^k]_{ij}$.
  2. **Positional encoding**: augment $\mathbf{X}$ with a node descriptor $\mathbf{P}(\mathbf{S})$ so that attention *sees* the graph, $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{P}(\mathbf{S})]$.
- We now survey both, and how they combine with Lecture 21.

Positional encodings: from sequences to graphs

- One option for the positional encoding for time sequences (line graph) is to add a sinusoidal position embedding to $\mathbf{X}$ because self-attention is permutation-invariant.
- On a graph we do not have a linear order – we need a structural encoding instead.
- **Laplacian PE**. Let $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ with $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ (Lecture 20, graph Fourier basis). Take the first k non-trivial eigenvectors as coordinates:

$$\mathbf{P}_{\text{LPE}} = [\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{k+1}] \in \mathbb{R}^{N \times k}.$$

  Rows of $\mathbf{P}_{\text{LPE}}$ are the spectral coordinates of the nodes.
- **Random-walk PE.** For $\mathbf{S} = \mathbf{D}^{-1}\mathbf{A}$, take

$$[\mathbf{P}_{\text{RW}}]_{i,k} = [\mathbf{S}^k]_{ii}, \qquad k = 1, \ldots, K.$$

  These are the k-step return probabilities – a node's local "fingerprint".
- The transformer sees $[\mathbf{X}, \mathbf{P}(\mathbf{S})]$: attention can now differentiate graph-distant nodes.

Graphormer

- Graphormer (Ying *et al.*, NeurIPS 2021) built three structural biases into the attention logits:
  1. Centrality encoding: each node i is additionally embedded by its degree d_i (two trainable tables for in/out degree) and the embedding is added to $\boldsymbol{x}_i$.
  2. Spatial encoding: a scalar bias $b_{\phi(i,j)}$ depending on the shortest-path distance $\phi(i,j)$,

$$[\mathbf{A}]_{ij} \propto \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_k}} + b_{\phi(i,j)}\right).$$

  3. Edge encoding: if (i,j) has an edge feature $\boldsymbol{e}_{ij}$, its learnt embedding is added along the shortest $i \to j$ path.
- With these three biases, a *plain* transformer becomes state-of-the-art on molecular property prediction (OGB-LSC, PCQM4M).
- Connection with our graph signal: Graphormer keeps $\mathbf{A}(\mathbf{X})$ dense but re-injects $\mathbf{S}$ through b_{ij} and the edge term.

GraphGPS: MPNN + Transformer

Cornell University

- Rampasek *et al.* (NeurIPS 2022) argued that *both* views of $\mathbf{S}$ are useful:
  - a *local* branch, which is a standard MPNN/GCN (Lecture 21);
  - a *global* branch, which is a transformer block.
- One GraphGPS layer:

$$\begin{aligned}
\mathbf{M}_\ell &= \mathrm{MPNN}(\mathbf{X}_{\ell-1}, \mathbf{S}) && \text{(local; uses } \mathbf{S}\text{)} \\
\mathbf{T}_\ell &= \mathrm{Attn}(\mathbf{X}_{\ell-1} + \mathbf{P}(\mathbf{S})) && \text{(global; all-pairs)} \\
\mathbf{X}_\ell &= \mathrm{FFN}\big(\,\mathrm{LN}(\mathbf{M}_\ell + \mathbf{T}_\ell + \mathbf{X}_{\ell-1})\big).
\end{aligned}$$

- GraphGPS is modular: any MPNN (our $\sum_k \mathbf{S}^k \mathbf{X} \mathbf{H}_{\ell k}$, GIN, GatedGCN) + any transformer + any PE slot in.
- Empirically, the hybrid beats pure transformers on sparse graphs and pure MPNNs on graphs with long-range interactions.

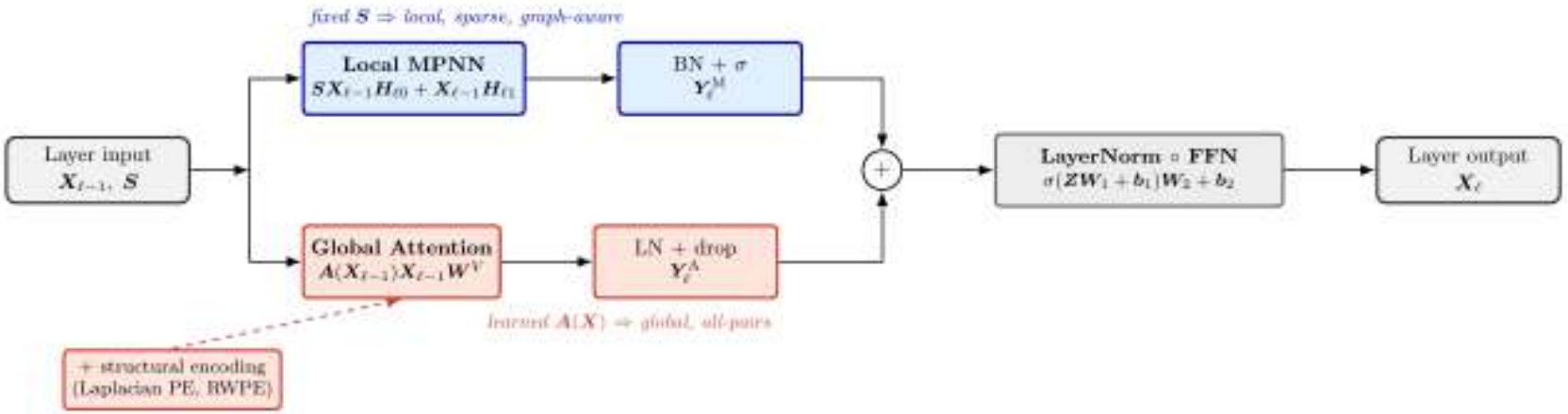
GraphGPS: MPNN + Transformer

- Let us refer to GCN with $K = 1$ as Message Passing Neural Networks (MPNN) which is the first generation of GCN, i.e. $\mathbf{X}_\ell = \mathbf{S}\,\mathbf{X}_{\ell-1}\,\mathbf{H}_{\ell 0}$.
- Rampasek *et al.* (NeurIPS 2022) argued that *both* views of $\mathbf{S}$ are useful:
  - a *local* branch, which is a standard MPNN/GCN;
  - a *global* branch, which is a transformer block.
- One GraphGPS layer:

$$\begin{aligned}
\mathbf{M}_\ell &= \mathrm{MPNN}(\mathbf{X}_{\ell-1}, \mathbf{S}) && \text{(local; uses } \mathbf{S}\text{)} \\
\mathbf{T}_\ell &= \mathrm{Attn}(\mathbf{X}_{\ell-1} + \mathbf{P}(\mathbf{S})) && \text{(global; all-pairs)} \\
\mathbf{X}_\ell &= \mathrm{FFN}\big(\,\mathrm{LN}(\mathbf{M}_\ell + \mathbf{T}_\ell + \mathbf{X}_{\ell-1})\big).
\end{aligned}$$

- GraphGPS is modular: any (our MPNN, a GCN $\sum_k \mathbf{S}^k \mathbf{X}\mathbf{H}_{\ell k}$, GIN, GatedGCN) + any transformer + any PE slot in.

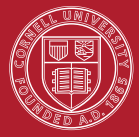
GraphGPS architecture



GraphGPS layer: run a GNN/MPNN and a Transformer *in parallel* on the same input, sum the outputs, then pass through LayerNorm + FFN. Structural / positional encodings are added to the features so attention can still "see" the graph.

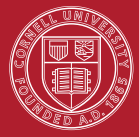
- Empirically, the hybrid beats pure transformers on sparse graphs and pure MPNNs on graphs with long-range interactions.

Outline



Cornell University

Heterogeneous graphs



- So far every node and every edge played the same role. Many real graphs are *typed* $\rightarrow$:
  - recommenders: users, items, categories; "bought", "rated", "viewed";
  - biomedical graphs: diseases, genes, drugs; "treats", "interacts-with", "is-a";
  - academic graphs: papers, authors, venues, topics.
- Formally: a heterogeneous graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \phi)$ with a node-type map $\tau : \mathcal{V} \rightarrow \mathcal{T}_v$ and an edge-type map $\phi : \mathcal{E} \rightarrow \mathcal{T}_e$.
- A *knowledge graph* (KG) is the special case where edges are labelled relations. Facts are triples

$$(h, r, t) \in \mathcal{V} \times \mathcal{R} \times \mathcal{V},$$

with h the head entity, t the tail, and r the relation. E.g. `(Aspirin, treats, Headache)`. It is an **example of multi-graph**, where each relation $\mathcal{R}$ is a layer

Relational GCN (R-GCN)

- A clean way to extend our layer $\mathbf{X}_\ell = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right]$ to typed edges: one GSO per relation.
- Let $\mathbf{S}_r$ be the (normalized) adjacency restricted to relation $r \in \mathcal{R}$. The R-GCN update is

$$\mathbf{X}_\ell = \sigma\left[\sum_{r\in\mathcal{R}} \sum_{k=0}^{K-1} \mathbf{S}_r^k \, \mathbf{X}_{\ell-1} \, \mathbf{H}_{\ell k}^{(r)}\right],$$

  with a separate filter tensor $\mathcal{H}^{(r)}$ for each relation.
- Parameters grow linearly in $|\mathcal{R}| \Rightarrow$ typical remedies:
  - *basis decomposition*: $\mathbf{H}_{\ell k}^{(r)} = \sum_{b=1}^{B} \alpha_{rb} \, \mathbf{B}_{\ell k}^{(b)}$;
  - *block-diagonal* $\mathbf{H}^{(r)}$ to reduce parameters and overfitting.

Heterogeneous Graph Transformer (HGT)

- HGT (Hu *et al.*, WWW 2020) turns R-GCN into an attention model: parameters are indexed by the types of source, edge, and target.
- For an edge $i \to j$ of type $r = \phi(i, j)$ with $\tau(i) = s$ and $\tau(j) = t$:

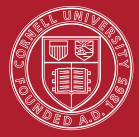
$$\boldsymbol{q}_j = \mathbf{W}_t^Q \boldsymbol{x}_j, \qquad \boldsymbol{k}_i = \mathbf{W}_s^K \boldsymbol{x}_i, \qquad \boldsymbol{v}_i = \mathbf{W}_s^V \boldsymbol{x}_i,$$

$$\alpha_{ij} \propto \exp\left(\frac{\boldsymbol{q}_j^\top \mathbf{W}_r^{\text{rel}} \boldsymbol{k}_i}{\sqrt{d_k}} \right),$$

$$\boldsymbol{x}_j^{\text{out}} = \sum_{i \in \mathcal{N}(j)} \alpha_{ij} \, \mathbf{W}_r^{\text{msg}} \, \boldsymbol{v}_i.$$

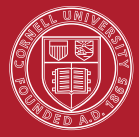
- One $\mathbf{W}$ per node/edge type: attention weights become type-aware, and message passing respects the heterogeneous schema.
- HGT is strictly more expressive than R-GCN and attention-based, at a modest parameter cost.

Outline



Cornell University

What is a foundation model?



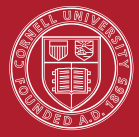
Cornell University

Definition (Bommasani *et al.*, 2021)

A *foundation model* is a model *trained on broad data at scale* and designed to be *adapted* (via fine-tuning, prompting, in-context learning, . . .) to a wide range of downstream tasks.

- Two defining characteristics:
  - Emergence – capabilities not explicitly programmed nor present at smaller scale appear when you scale data × compute × parameters.
  - Homogenisation – one backbone model serves many downstream tasks, replacing a zoo of task-specific systems.
- Enablers behind the scenes:
  - massive data (web-scale),
  - hardware (GPU/TPU clusters),
  - self-supervised learning (no labels needed),
  - the transformer architecture we just described.

What is a foundation model?



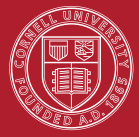
Cornell University

Definition (Bommasani *et al.*, 2021)

A *foundation model* is a model *trained on broad data at scale* and designed to be *adapted* (via fine-tuning, prompting, in-context learning, . . .) to a wide range of downstream tasks.

- Two defining characteristics:
  - Emergence – capabilities not explicitly programmed nor present at smaller scale appear when you scale data × compute × parameters.
  - Homogenisation – one backbone model serves many downstream tasks, replacing a zoo of task-specific systems.
- Enablers behind the scenes:
  - massive data (web-scale),
  - hardware (GPU/TPU clusters),
  - self-supervised learning (no labels needed),
  - the transformer architecture we just described.

What is a foundation model?



Cornell University

Definition (Bommasani *et al.*, 2021)

A *foundation model* is a model *trained on broad data at scale* and designed to be *adapted* (via fine-tuning, prompting, in-context learning, . . .) to a wide range of downstream tasks.

- Two defining characteristics:
  - Emergence – capabilities not explicitly programmed nor present at smaller scale appear when you scale data $\times$ compute $\times$ parameters.
  - Homogenisation – one backbone model serves many downstream tasks, replacing a zoo of task-specific systems.
- Enablers behind the scenes:
  - massive data (web-scale),
  - hardware (GPU/TPU clusters),
  - self-supervised learning (no labels needed),
  - the transformer architecture we just described.

What is a foundation model?

Cornell University

Definition (Bommasani *et al.*, 2021)

A *foundation model* is a model *trained on broad data at scale* and designed to be *adapted* (via fine-tuning, prompting, in-context learning, . . .) to a wide range of downstream tasks.

- Two defining characteristics:
  - Emergence – capabilities not explicitly programmed nor present at smaller scale appear when you scale data $\times$ compute $\times$ parameters.
  - Homogenisation – one backbone model serves many downstream tasks, replacing a zoo of task-specific systems.
- Enablers behind the scenes:
  - massive data (web-scale),
  - hardware (GPU/TPU clusters),
  - self-supervised learning (no labels needed),
  - the transformer architecture we just described.

Three transformer shapes for language

- All three are stacks of the transformer block but differ in the attention *mask* and the training objective.
- **Encoders** (BERT, HuBERT). Bidirectional attention; trained with masked language modelling

$$\max \sum_{i \in \mathcal{M}} \log p(w_i \mid w_{\neg \mathcal{M}}).$$

  Good backbones for classification, retrieval, feature extraction.
- **Decoders** (GPT, Claude, Llama, Mixtral). Causal (lower-triangular) mask on $\mathbf{A}$; trained with next-token prediction

$$\max \sum_{i} \log p(w_i \mid w_{<i}).$$

  Good for open-ended generation; almost every task can be written as "predict the next word".
- **Encoder-decoders** (T5, Flan-T5, Whisper). Bidirectional encoder + causal decoder with cross-attention. Good when inputs and outputs are different spaces (translation, summarisation, ASR).

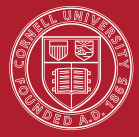
Decoder LLMs: tasks as conditional generation

- A decoder LLM models $p(w_1, \ldots, w_T) = \prod_i p(w_i \mid w_{<i})$ with a softmax head:

$$p(w_i = v \mid w_{<i}) = \mathrm{softmax}\big(\mathbf{W}_{\mathrm{LM}}\, \boldsymbol{x}_i\big)_v.$$

- An apparently narrow objective absorbs most NLP tasks by prompt engineering:
  - *Sentiment*: "The sentiment of the sentence '`I like Jackie Chan`' is:" $\Rightarrow$ `positive`.
  - *QA*: "Q: Who wrote *Hamlet*?\nA:" $\Rightarrow$ `Shakespeare`.
  - *Summarisation*: append "`tl;dr:`" to the document and keep generating.
- **Decoding**. Random sampling $w_i \sim p(w_i \mid w_{<i})$ is unbiased but can pick very unlikely tokens. Practical variants: top-k, top-p (nucleus) sampling, temperature scaling, beam search.
- This "tasks-as-text" trick is what makes emergence visible: one model, many capabilities, no architectural change.

Outline



Cornell University

Can we build a foundation model for graphs?

Graph Foundation Model (GFM)

A GFM is a model pre-trained on *large and diverse graph data*, intended to be adapted to many downstream graph tasks (node/edge/graph level) across potentially different graphs.

- The GNN/graph-transformer stack gives us a candidate architecture.
  The harder questions are about *data* and *objectives*:
  - Graphs across domains share no vocabulary (unlike text, which is always tokens).
  - A node "id" in graph $\mathcal{G}_1$ has no meaning in graph $\mathcal{G}_2$: we cannot reuse an embedding table.
  - Relation schemas differ: "treats" in Hetionet vs. "bought-together" in Amazon.

Can we build a foundation model for graphs?

Graph Foundation Model (GFM)

A GFM is a model pre-trained on *large and diverse graph data*, intended to be adapted to many downstream graph tasks (node/edge/graph level) across potentially different graphs.

- The GNN/graph-transformer stack gives us a candidate architecture. The harder questions are about *data* and *objectives*:
  - Graphs across domains share no vocabulary (unlike text, which is always tokens).
  - A node "id" in graph $\mathcal{G}_1$ has no meaning in graph $\mathcal{G}_2$: we cannot reuse an embedding table.
  - Relation schemas differ: "treats" in Hetionet vs. "bought-together" in Amazon.

Can we build a foundation model for graphs?

Graph Foundation Model (GFM)

A GFM is a model pre-trained on *large and diverse graph data*, intended to be adapted to many downstream graph tasks (node/edge/graph level) across potentially different graphs.

- The GNN/graph-transformer stack gives us a candidate architecture. The harder questions are about *data* and *objectives*:
  - Graphs across domains share no vocabulary (unlike text, which is always tokens).
  - A node "id" in graph $\mathcal{G}_1$ has no meaning in graph $\mathcal{G}_2$: we cannot reuse an embedding table.
  - Relation schemas differ: "treats" in Hetionet vs. "bought-together" in Amazon.

Can we build a foundation model for graphs?

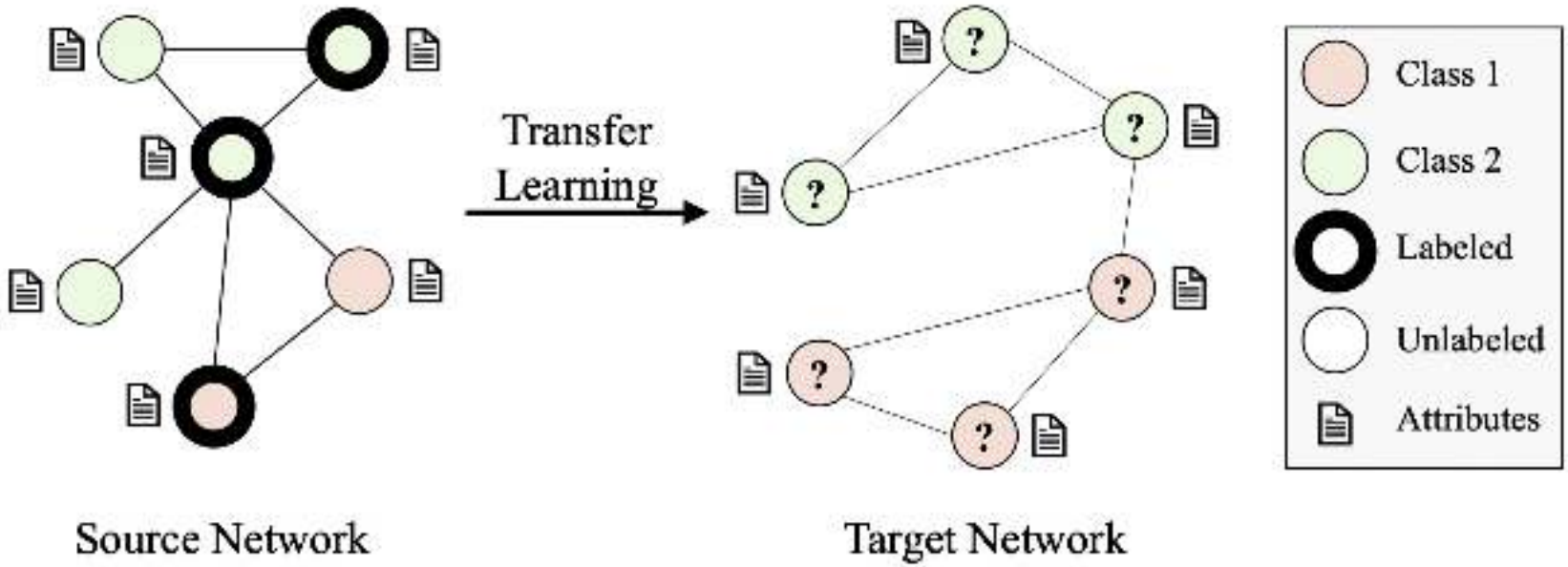
Graph Foundation Model (GFM)

A GFM is a model pre-trained on *large and diverse graph data*, intended to be adapted to many downstream graph tasks (node/edge/graph level) across potentially different graphs.

- The GNN/graph-transformer stack gives us a candidate architecture. The harder questions are about *data* and *objectives*:
  - Graphs across domains share no vocabulary (unlike text, which is always tokens).
  - A node "id" in graph $\mathcal{G}_1$ has no meaning in graph $\mathcal{G}_2$: we cannot reuse an embedding table.
  - Relation schemas differ: "treats" in Hetionet vs. "bought-together" in Amazon.

Can we build a foundation model for graphs?

- Two desiderata drive the field:
  - **Transfer** across graphs with *disjoint nodes*;
  - **Transfer** across graphs with *disjoint relations*.



Canonical task: knowledge-graph completion

- Given a knowledge graph $\mathcal{G} = \{(h, r, t)\}$, predict missing tails (or heads):

$$\text{given } (h, r, ?) \;\Rightarrow\; \text{score } t \in \mathcal{V}.$$

- A *score function* $s(h, r, t) \in \mathbb{R}$ is trained so that true triples score higher than corrupted ones.
- Shallow KG embedding models (pre-GNN era) $\Rightarrow$ learn vectors $\boldsymbol{e}_h, \boldsymbol{e}_r, \boldsymbol{e}_t$ directly:

Model	**Score** $s(h, r, t)$	**What relations it can capture**
TransE	$-\lVert \boldsymbol{e}_h + \boldsymbol{e}_r - \boldsymbol{e}_t \rVert$	translations; struggles with 1-to-N
TransR	$-\lVert \mathbf{M}_r \boldsymbol{e}_h + \boldsymbol{e}_r - \mathbf{M}_r \boldsymbol{e}_t \rVert$	relation-specific spaces
DistMult	$\langle \boldsymbol{e}_h, \boldsymbol{e}_r, \boldsymbol{e}_t \rangle$	symmetric relations
ComplEx	$\mathrm{Re}\, \langle \boldsymbol{e}_h, \boldsymbol{e}_r, \bar{\boldsymbol{e}}_t \rangle$	asymmetric via complex numbers
RotatE	$-\lVert \boldsymbol{e}_h \circ \boldsymbol{e}_r - \boldsymbol{e}_t \rVert$	symmetry, inversion, composition

- All of the above are structure-agnostic: an embedding is tied to a specific node id in a specific KG $\Rightarrow$ **no transfer** across KGs.

Transductive vs. inductive link prediction

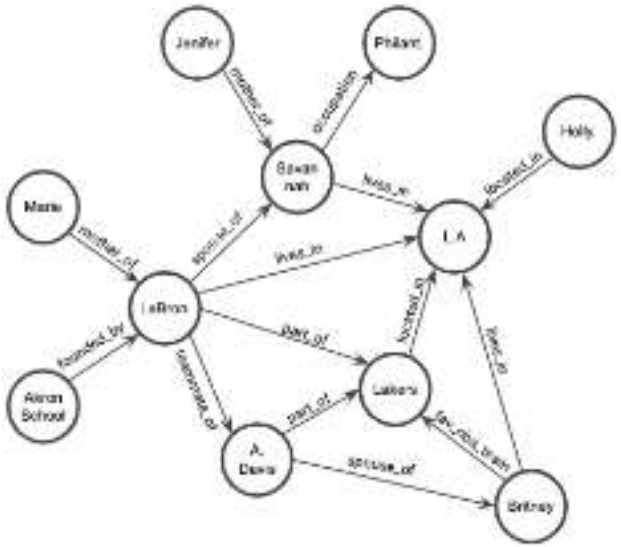
- **Transductive.** Train and test triples use the *same* entity set. Shallow embeddings are fine.
- **Inductive.** Test graph contains entities unseen at training time. A model that only knows "id $\rightarrow$ vector" cannot predict – the new ids have no embedding.
- A GNN-style score is inductive by construction:

$$\hat{s}(h, r, t) = f\big(\Phi(\mathcal{G}; \mathcal{H})_h,\ \boldsymbol{e}_r,\ \Phi(\mathcal{G}; \mathcal{H})_t\big),$$

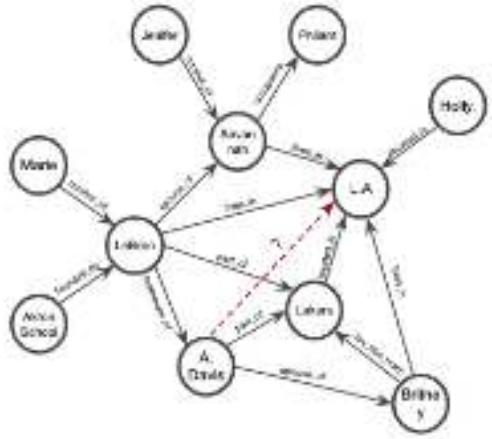
because node embeddings are *functions of the local structure* $\Phi(\mathcal{G}; \mathcal{H})$, not free parameters.

Transductive vs. inductive link prediction

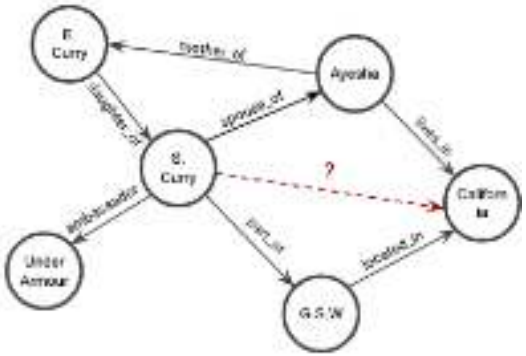
- This is precisely the GNN message-passing we developed in Lectures 21–22, applied to a relational graph via R-GCN or HGT.



a. Training graph



b. Transductive inference



c. Inductive inference

Towards double equivariance

- A GFM for KGs should be invariant to *two* relabellings:
  - permuting entities (nodes), and
  - permuting relations.
- If π_v permutes nodes and π_r permutes relations, we want

$$\Phi(\pi_v \cdot \pi_r \cdot \mathcal{G}) = \pi_v \cdot \pi_r \cdot \Phi(\mathcal{G}).$$

- A *doubly-equivariant* encoder can be pre-trained on one KG and applied on another, as long as we expose it to both node and relation neighbourhoods – no entity-id lookup.
- Representative line of work: ULTRA / NBFNet / InGram / PRODIGY. Common theme: score a triple through local substructure rather than through a global embedding table.

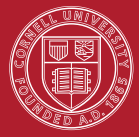
PRODIGY: in-context learning on graphs

- PRODIGY (Huang *et al.*, NeurIPS 2023) ports in-context learning from LLMs to graphs.
- Idea: at test time the user gives a handful of *demonstration triples*, and the model predicts on a *query triple* – without gradient updates.
- How:
  - pre-training creates a *prompt graph* mixing demonstrations and queries;
  - a graph transformer with double equivariance is trained to answer the query from the structural context;
  - this mirrors how GPT "reads" a few examples and generalises.
- Upshot: a single pretrained GFM can be used for node classification, link prediction, and few-shot tasks on *new* graphs with *new* label sets.

Where graph models meet LLMs

- Two complementary weaknesses invite combining them:
  - LLMs are fluent but weak on explicit graph structure (multi-hop reasoning, schema) and hallucinate facts.
  - Graph models exploit structure but lack world knowledge and natural-language interface.
- Current patterns (high level):
  - *LLM as augmenter*: LLM describes nodes/edges in natural language, GNN consumes embeddings.
  - *LLM as predictor*: linearize the graph into text prompts, let the LLM answer graph queries.
  - *Hybrid*: GFM retrieves relevant subgraph, LLM generates the answer – a graph-aware RAG.
- Evidence so far: hybrids outperform either alone on reasoning over structured knowledge. The architectural frontier is still moving.

Outline



Cornell University

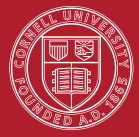
One architecture, two views

Cornell University

	Graph filter/GCN	**Self-attention/Graph transf.**
graph shift	$\mathbf{S}$ (fixed, prior)	$\mathbf{A}(\mathbf{X})$ (learned, data-dependent)
layer	$\sum_k \mathbf{S}^k \mathbf{X} \mathbf{H}_{\ell k}$	$\mathbf{A}(\mathbf{X}) \mathbf{X} \mathbf{W}^V$
receptive field	$K-1$ hops	all-pairs per block
inductive bias	locality, smoothness	flexible; needs PE $\mathbf{P}(\mathbf{S})$ on graphs
complexity	$\mathcal{O}(K \lvert\mathcal{E}\rvert F)$	$\mathcal{O}(N^2 d)$
structure reuse	directly via $\mathbf{S}$	via PE and/or attention biases

- The graph transformer is not a replacement for the GNN $\Rightarrow$ it is the same machinery with a learned, data-dependent GSO.
- Hybrids (GraphGPS, HGT+MPNN) keep both $\mathbf{S}$-based locality and attention's global view.

Takeaways



- Attention generalises the graph perceptron: it learns $\mathbf{S}$ from data.
- To apply it to graphs without losing structure, we add positional encodings $\mathbf{P}(\mathbf{S})$ and/or structural biases in the attention logits (Graphormer, GraphGPS).
- For heterogeneous data, relation-indexed weights give R-GCN and HGT.
- Foundation models in language work because self-supervision, scale, and the transformer align; the same recipe for graphs is an active research area.
- Graph Foundation Models aim for transfer across graphs with disjoint nodes *and* disjoint relations, via double equivariance and in-context learning (PRODIGY).
- With this we have closed the architectural loop of Module 3: shallow embeddings $\rightarrow$ GSP $\rightarrow$ GCN/GRNN $\rightarrow$ graph transformers $\rightarrow$ GFMs.
- **Next.** Module 3 will close with Graphical Models – a complementary, probabilistic view of structure on graphs.

References

[1] A. Vaswani *et al.*, "Attention Is All You Need," *NeurIPS*, 2017.

[2] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL*, 2019.

[3] T. Brown *et al.*, "Language Models are Few-Shot Learners," *NeurIPS*, 2020.

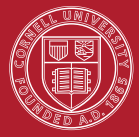
[4] R. Bommasani *et al.*, "On the Opportunities and Risks of Foundation Models," *arXiv:2108.07258*, 2021.

[5] V. P. Dwivedi *et al.*, "A Generalization of Transformer Networks to Graphs," *AAAI Workshop*, 2021.

[6] C. Ying *et al.*, "Do Transformers Really Perform Bad for Graph Representation?," *NeurIPS*, 2021.

[7] L. Rampasek *et al.*, "Recipe for a General, Powerful, Scalable Graph Transformer," *NeurIPS*, 2022.

[8] M. Schlichtkrull *et al.*, "Modeling Relational Data with Graph Convolutional Networks," *ESWC*, 2018.



[9] Z. Hu *et al.*, "Heterogeneous Graph Transformer," *WWW*, 2020.

[10] M. Galkin *et al.*, "Towards Foundation Models for Knowledge Graph Reasoning," *ICLR*, 2024.

[11] Q. Huang *et al.*, "PRODIGY: Enabling In-context Learning Over Graphs," *NeurIPS*, 2023.

[12] J. Liu *et al.*, "Towards Graph Foundation Models: A Survey and Beyond," *arXiv:2310.11829*, 2023.