

# FORGE: Foundational Optimization Representations from Graph Embeddings



Zohair Shafi  
Northeastern University



Serdar Kadioglu  
Fidelity Investments  
Brown University

# Background

## Mixed Integer Programming

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

# Background

## Mixed Integer Programming

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

Objective  
Function

# Background

## Mixed Integer Programming

Decision Variables

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

Objective  
Function

# Background

## Mixed Integer Programming

Decision Variables

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

Constraints

Objective  
Function

# Background

## Mixed Integer Programming

Decision Variables

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

Objective  
Function

Constraints

Some subset of these decision  
variables must have integer  
values

# Motivation

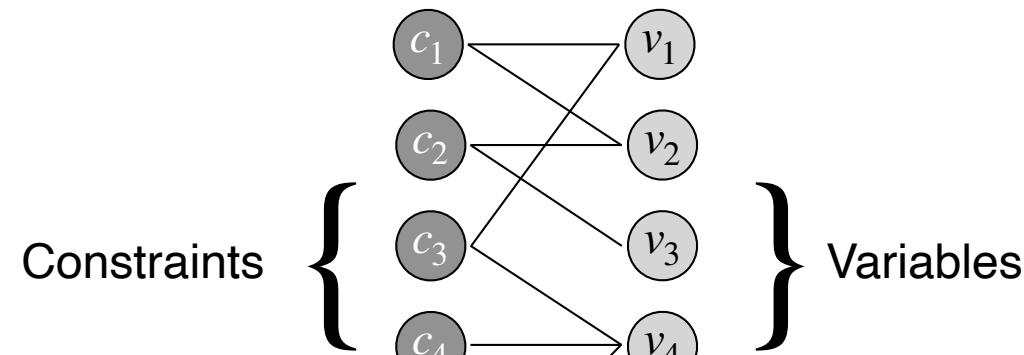
- There is an abundance of mixed integer programming (MIP) instances.
  - E.g., vehicle routing, job scheduling, flight scheduling, fibre optic network design
- Can we use these instances without solving them to create a “foundational” model?
- Why?
  - Recent advances in ML for CO problems are problem type or task specific.
  - A lot of training data is needed for current methods.
  - This training data is collected by solving instances which is extremely expensive.

# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.

# Methodology

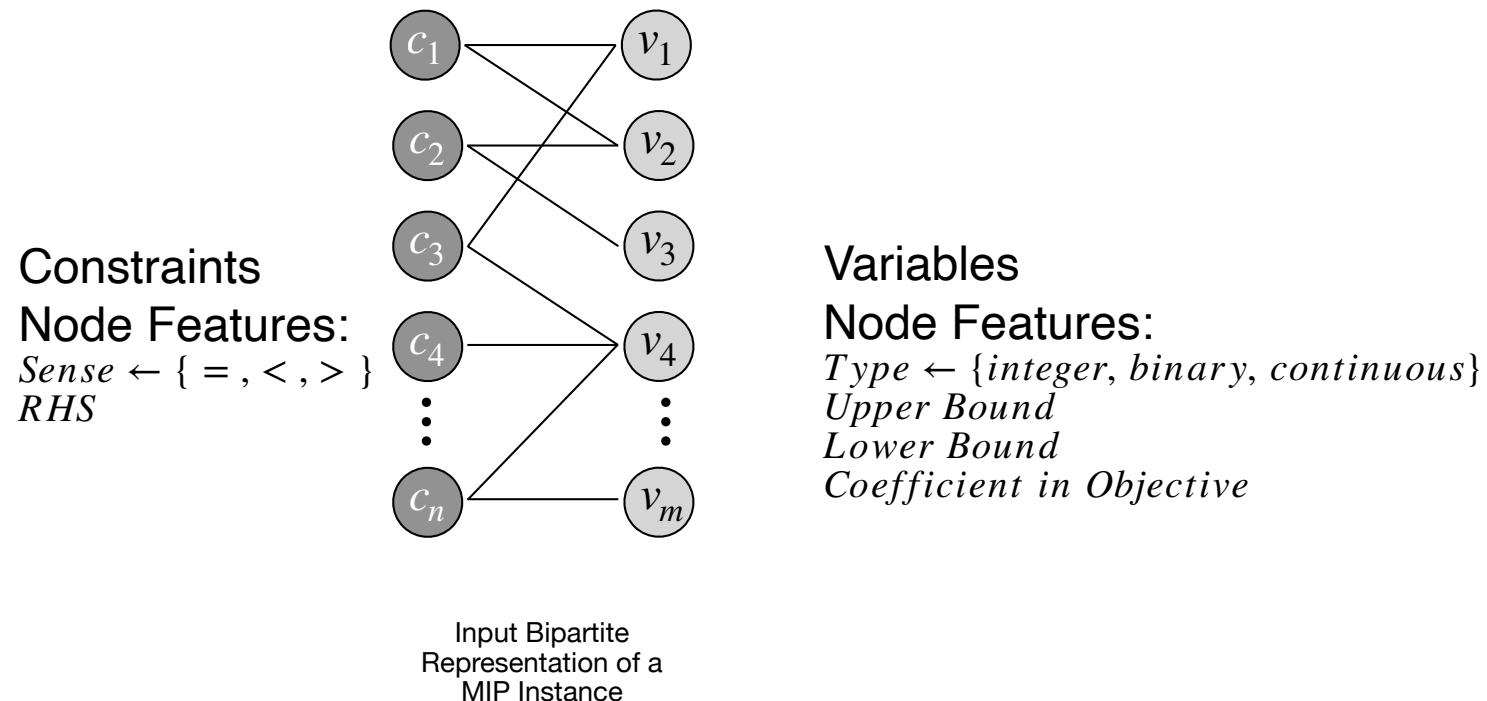
- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph



Input Bipartite  
Representation of a  
MIP Instance

# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph



# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \forall j \in I\}$$

Non-zero entries of

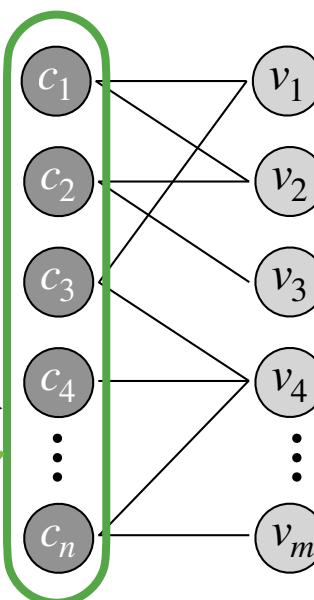
$A \in \mathbb{R}^{\#constraints \times \#variables}$

Constraints

Node Features:

$Sense \leftarrow \{ =, <, > \}$

RHS



Input Bipartite  
Representation of a  
MIP Instance

Variables

Node Features:

$Type \leftarrow \{integer, binary, continuous\}$

Upper Bound

Lower Bound

Coefficient in Objective

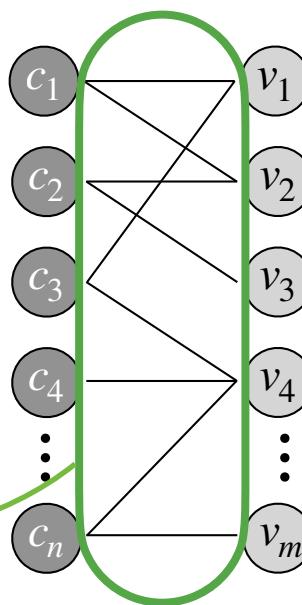
# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \forall j \in I\}$$

Edge weights are the magnitude of the non-zero entries of  
 $A \in \mathbb{R}^{\#constraints \times \#variables}$

Constraints  
Node Features:  
 $Sense \leftarrow \{ =, <, > \}$   
 $RHS$



Input Bipartite  
Representation of a  
MIP Instance

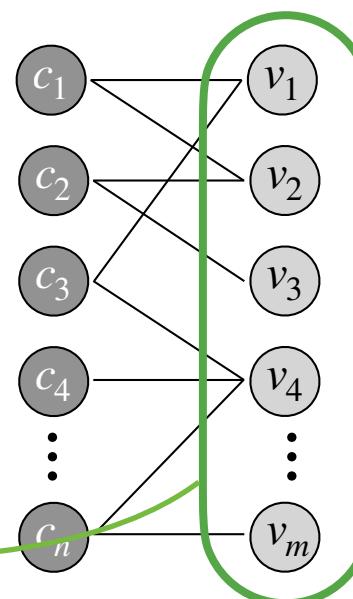
Variables  
Node Features:  
 $Type \leftarrow \{integer, binary, continuous\}$   
 $Upper Bound$   
 $Lower Bound$   
 $Coefficient in Objective$

# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph

$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \forall j \in I\}$$

Constraints  
Node Features:  
 $Sense \leftarrow \{ =, <, > \}$   
 $RHS$   
Variables



Input Bipartite  
Representation of a  
MIP Instance

Variables  
Node Features:  
 $Type \leftarrow \{integer, binary, continuous\}$   
 $Upper Bound$   
 $Lower Bound$   
 $Coefficient in Objective$

# Methodology

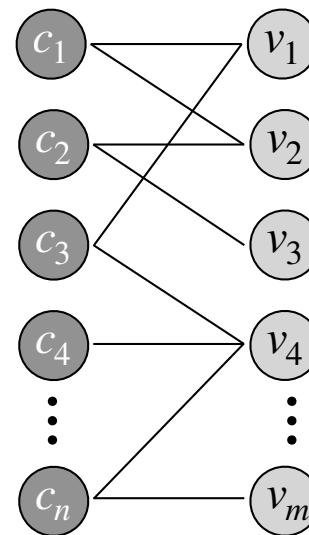
- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph

$$f(x) = \min \{ c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \forall j \in I \}$$

Constraints

Node Features:

*Sense*  $\leftarrow \{ =, <, > \}$   
*RHS*



Input Bipartite  
Representation of a  
MIP Instance

Variables

Node Features:

*Type*  $\leftarrow \{ \text{integer}, \text{binary}, \text{continuous} \}$   
*Upper Bound*  
*Lower Bound*  
*Coefficient in Objective*

# Methodology

- We aim to first learn the structure of a MIP problem in an unsupervised manner.
- How?  
As is commonly done, we first represent a MIP instance as a bipartite graph

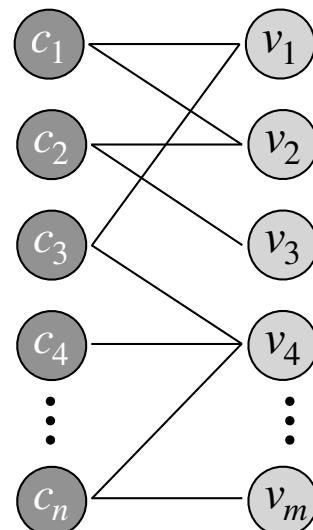
$$f(x) = \min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I\}$$

Constraints

Node Features:

$Sense \leftarrow \{ =, <, > \}$

RHS



Input Bipartite  
Representation of a  
MIP Instance

Variables

Node Features:

$Type \leftarrow \{integer, binary, continuous\}$

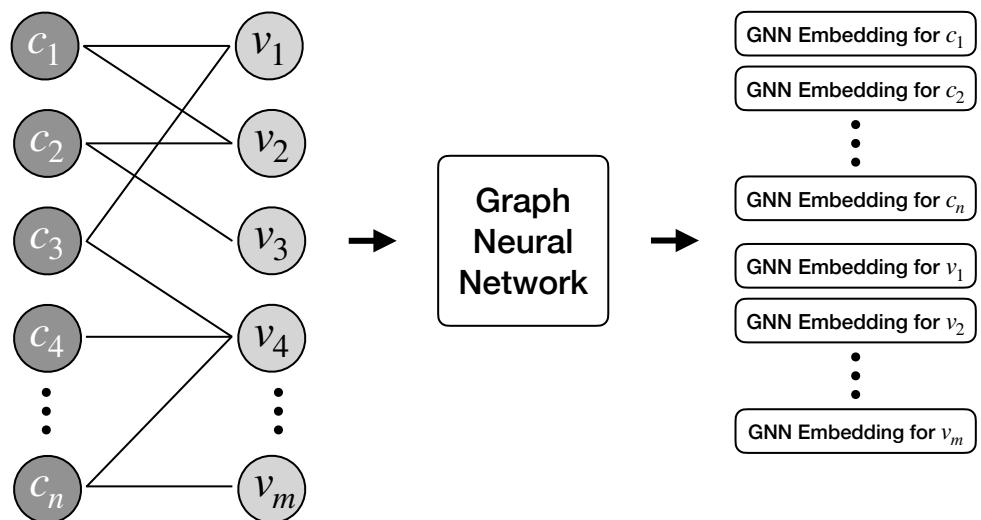
Upper Bound

Lower Bound

Coefficient in Objective

# Methodology

- This bipartite graph is then passed into a Graph Neural Network (GNN)
- But GNNs are not very good at preserving global structure due to inherent locality bias.
  - **Preserving global structure is important** in CO problems, especially to generalize across problem types.



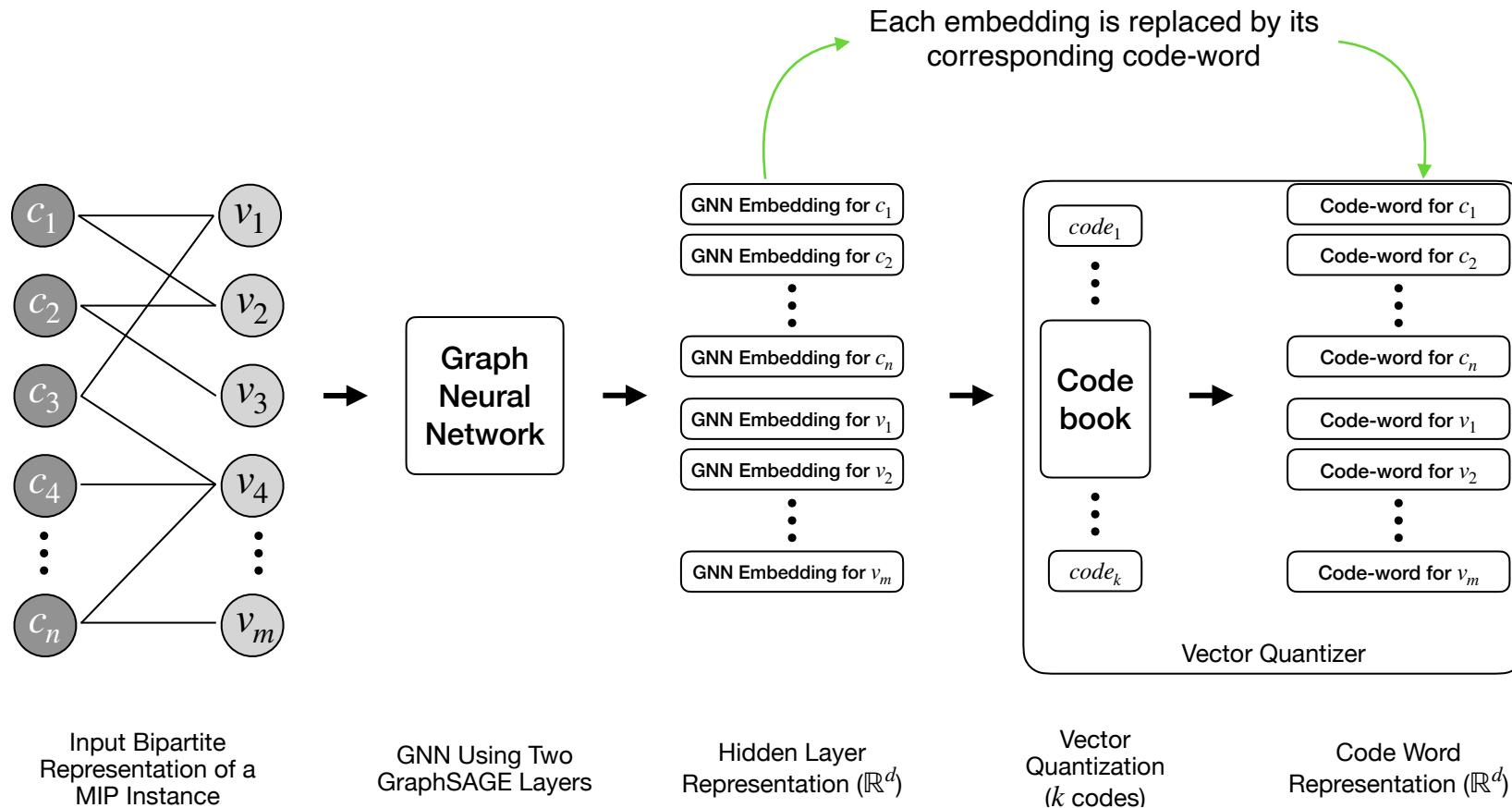
Input Bipartite  
Representation of a  
MIP Instance

GNN Using Two  
GraphSAGE Layers

Hidden Layer  
Representation ( $\mathbb{R}^d$ )

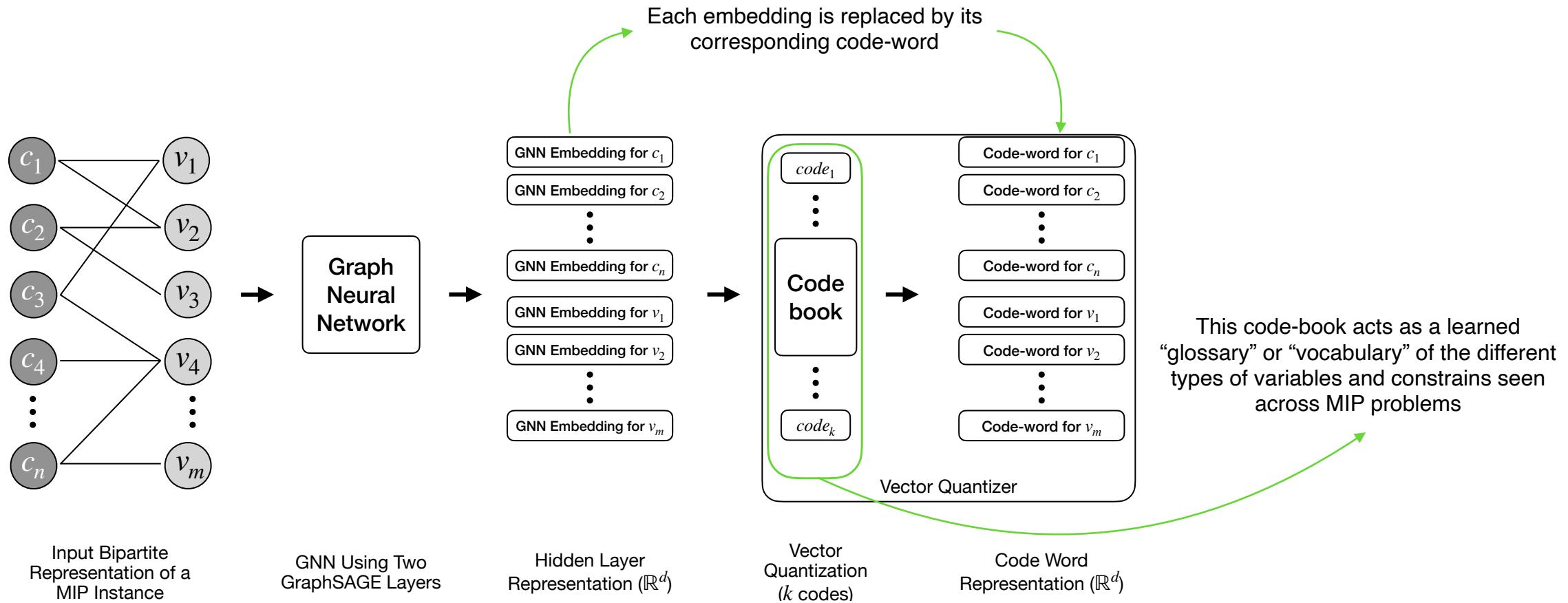
# Methodology

- This is where Vector Quantization comes in.



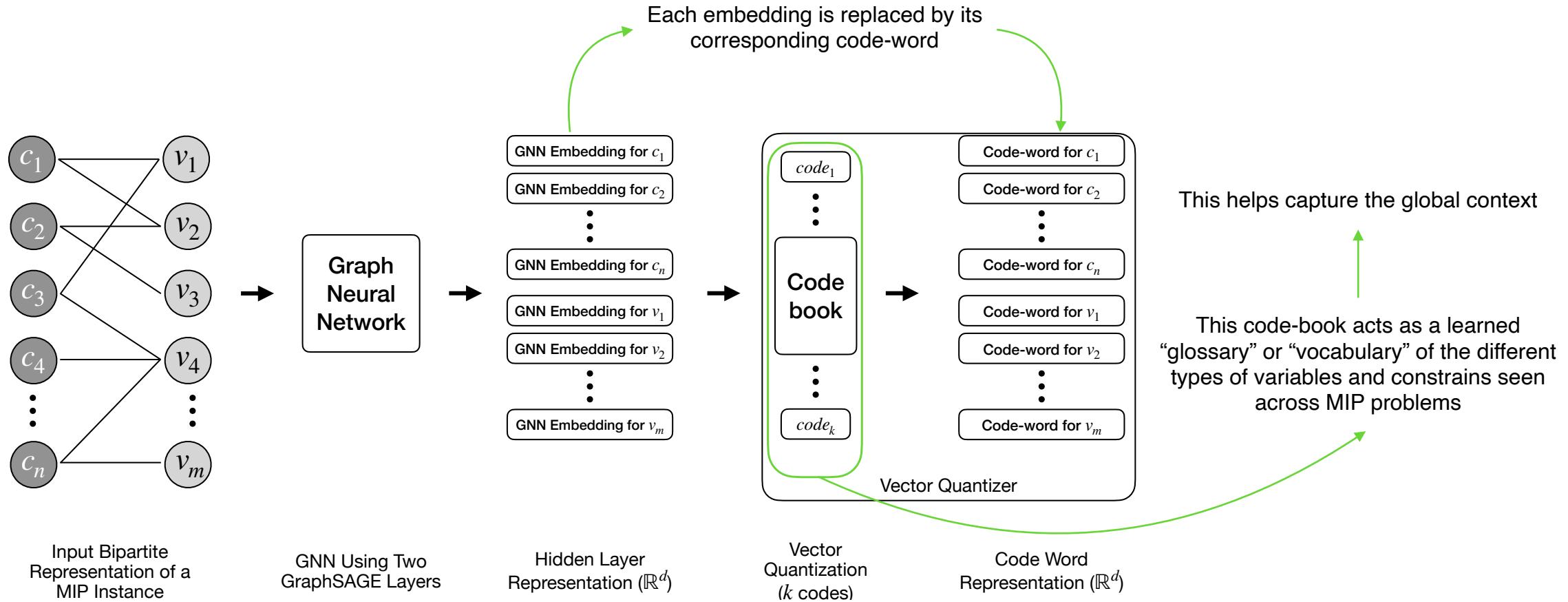
# Methodology

- This is where Vector Quantization comes in.



# Methodology

- This is where Vector Quantization comes in.



# **Methodology**

## **Vector Quantization Aside**

$\mathcal{L}_{Tokenizer} =$

# Methodology

## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} +$$

$$\mathcal{L}_{Rec} = \underbrace{\frac{1}{N} \sum_{i=1}^N \left( 1 - \frac{\mathbf{v}_i^T \hat{\mathbf{v}}_i}{\|\mathbf{v}_i\| \cdot \|\hat{\mathbf{v}}_i\|} \right)^\gamma}_{\text{node reconstruction}} + \underbrace{\left\| \mathbf{A} - \sigma(\hat{\mathbf{X}} \cdot \hat{\mathbf{X}}^T) \right\|_2^2}_{\text{edge reconstruction}},$$

$\mathbf{v}_i$  is the original node feature

$\hat{\mathbf{v}}_i$  is the regenerated node feature

$\hat{\mathbf{X}}$  is the matrix of all  $\hat{\mathbf{v}}_i$

# Methodology

## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2$$

Codebook Loss

Update codebook embeddings  $\mathbf{e}_{z_i}$   
to make them closer to encoder output  $\mathbf{h}_i$

$$\mathcal{L}_{Rec} = \underbrace{\frac{1}{N} \sum_{i=1}^N \left(1 - \frac{\mathbf{v}_i^T \hat{\mathbf{v}}_i}{\|\mathbf{v}_i\| \cdot \|\hat{\mathbf{v}}_i\|}\right)^\gamma}_{\text{node reconstruction}} + \underbrace{\left\| \mathbf{A} - \sigma(\hat{\mathbf{X}} \cdot \hat{\mathbf{X}}^T) \right\|_2^2}_{\text{edge reconstruction}},$$

$v_i$  is the original node feature

$\hat{v}_i$  is the regenerated node feature

$X$  is the matrix of all  $\hat{v}_i$

# Methodology

## Vector Quantization Aside

Codebook Loss

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2$$

Update codebook embeddings  $\mathbf{e}_{z_i}$   
to make them closer to encoder output  $\mathbf{h}_i$

This update is only applied to codebook variables.  
Gradients are **not** applied to  $\mathbf{h}_i$

$$\mathcal{L}_{Rec} = \underbrace{\frac{1}{N} \sum_{i=1}^N \left(1 - \frac{\mathbf{v}_i^T \hat{\mathbf{v}}_i}{\|\mathbf{v}_i\| \cdot \|\hat{\mathbf{v}}_i\|}\right)^\gamma}_{\text{node reconstruction}} + \underbrace{\left\| \mathbf{A} - \sigma(\hat{\mathbf{X}} \cdot \hat{\mathbf{X}}^T) \right\|_2^2}_{\text{edge reconstruction}},$$

$v_i$  is the original node feature

$\hat{v}_i$  is the regenerated node feature

$X$  is the matrix of all  $\hat{v}_i$

# Methodology

## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2 + \boxed{\frac{\eta}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{e}_{z_i}] - \mathbf{h}_i\|_2^2},$$

Update encoder weights  $h_i$  to be close to chosen code  $e_{z_i}$  to **avoid fluctuations in code assignment**

$$\mathcal{L}_{Rec} = \underbrace{\frac{1}{N} \sum_{i=1}^N \left(1 - \frac{\mathbf{v}_i^T \hat{\mathbf{v}}_i}{\|\mathbf{v}_i\| \cdot \|\hat{\mathbf{v}}_i\|}\right)^\gamma}_{\text{node reconstruction}} + \underbrace{\left\| \mathbf{A} - \sigma(\hat{\mathbf{X}} \cdot \hat{\mathbf{X}}^T) \right\|_2^2}_{\text{edge reconstruction}},$$

This update is only applied to encoder variables.

Gradients are **not** applied to  $e_z$ .

$v_i$  is the original node feature

$\hat{v}_i$  is the regenerated node feature

$X$  is the matrix of all  $\hat{v}_i$

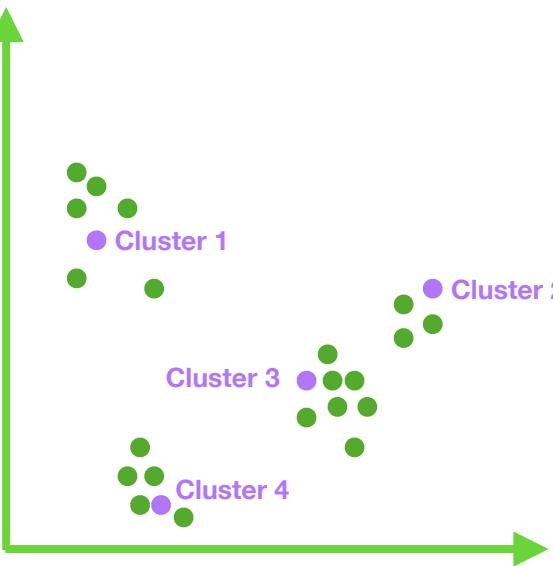
# Methodology

## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2 + \frac{\eta}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{e}_{z_i}] - \mathbf{h}_i\|_2^2,$$

Codebook Loss  
Move cluster centroids only  
(think standard k-means)

Commitment Loss  
Move data embedding only



# Methodology

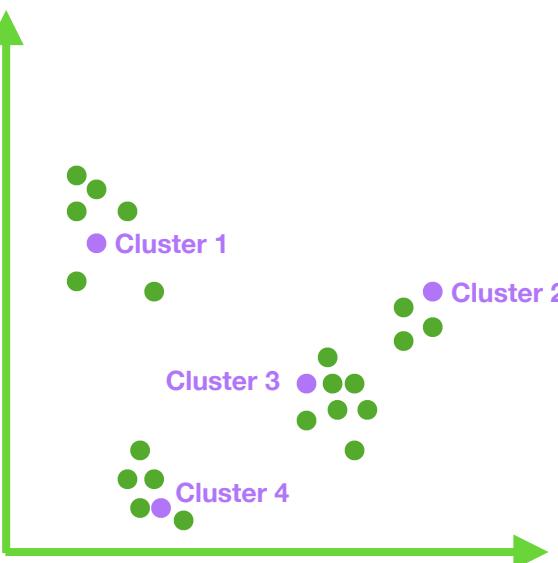
## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2 + \frac{\eta}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{e}_{z_i}] - \mathbf{h}_i\|_2^2,$$

Codebook Loss      Commitment Loss

Move cluster centroids only  
(think standard k-means)

Move data embedding only



Vector Quantization essentially replaces each **green** point with the closest **purple** point

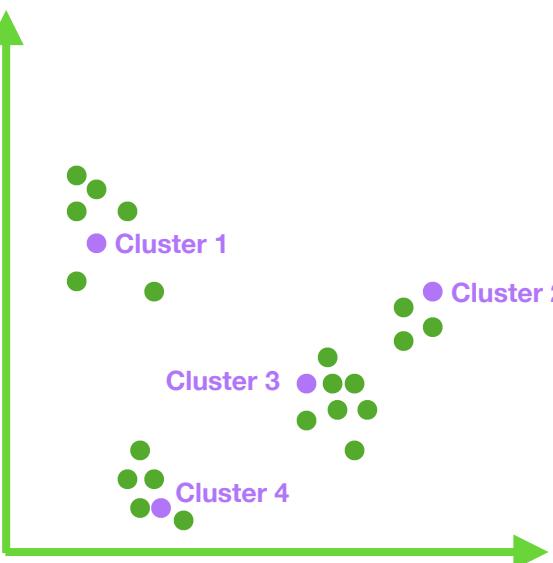
# Methodology

## Vector Quantization Aside

$$\mathcal{L}_{Tokenizer} = \mathcal{L}_{Rec} + \frac{1}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{h}_i] - \mathbf{e}_{z_i}\|_2^2 + \frac{\eta}{N} \sum_{i=1}^N \|\text{sg}[\mathbf{e}_{z_i}] - \mathbf{h}_i\|_2^2,$$

**Codebook Loss**  
Move cluster centroids only  
(think standard k-means)

**Commitment Loss**  
Move data embedding only

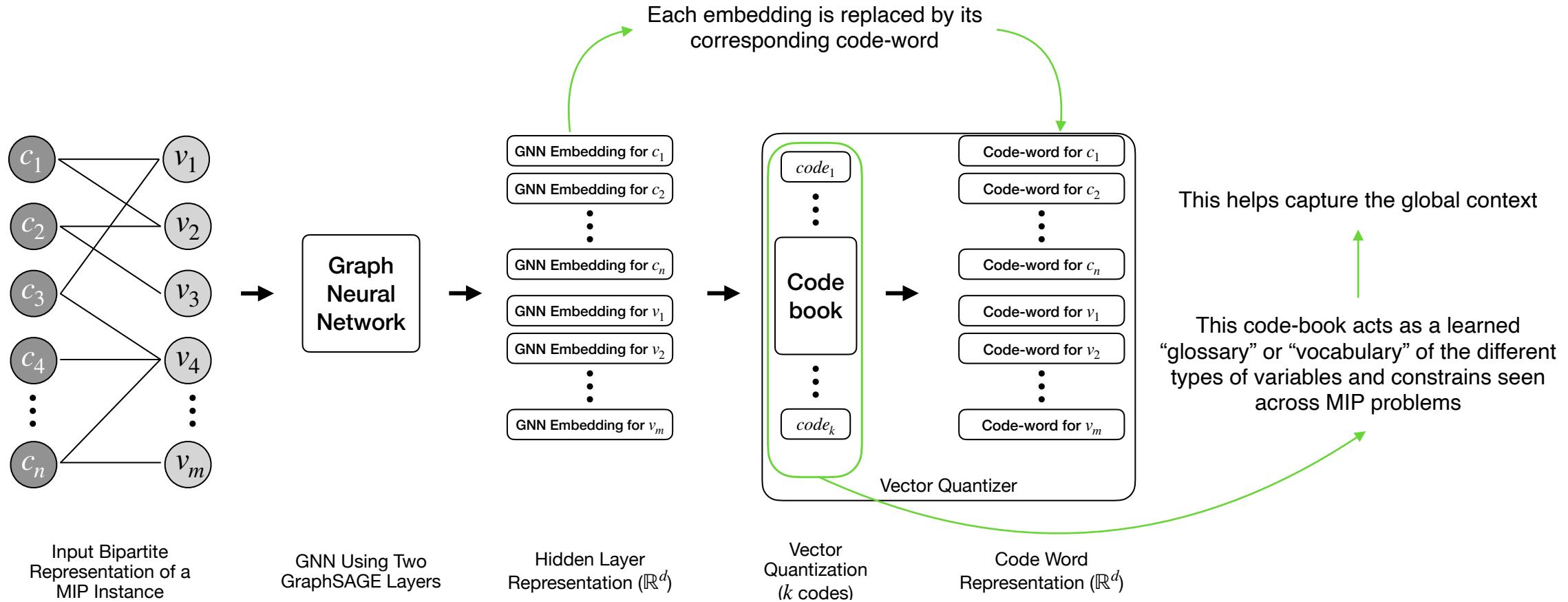


Vector Quantization essentially replaces each **green** point with the closest **purple** point

The index of the cluster each data point belongs to is the **discrete index/code** assigned to that data point

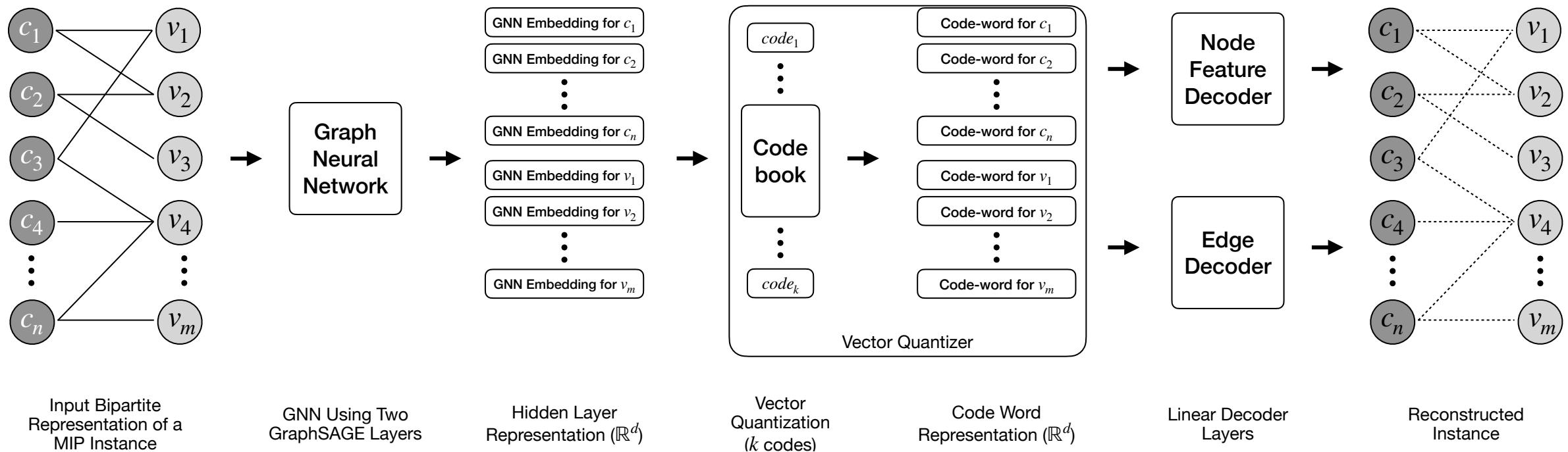
# Methodology

- Back to overview



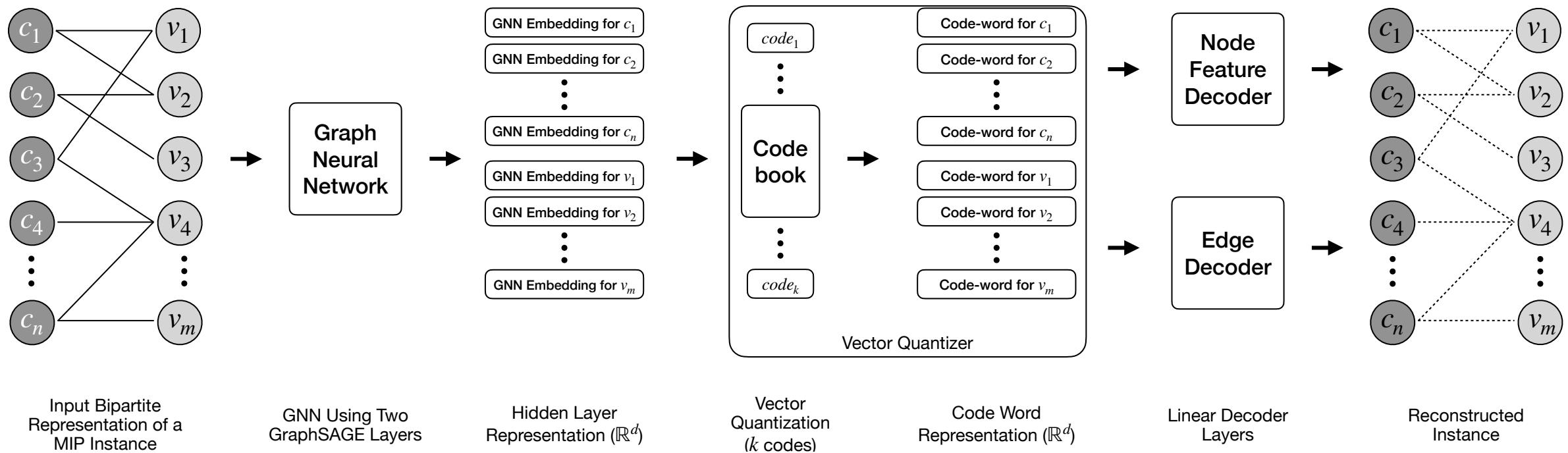
# Methodology

- The code-words are then used to reconstruct the input graph structure and node features.



# Methodology

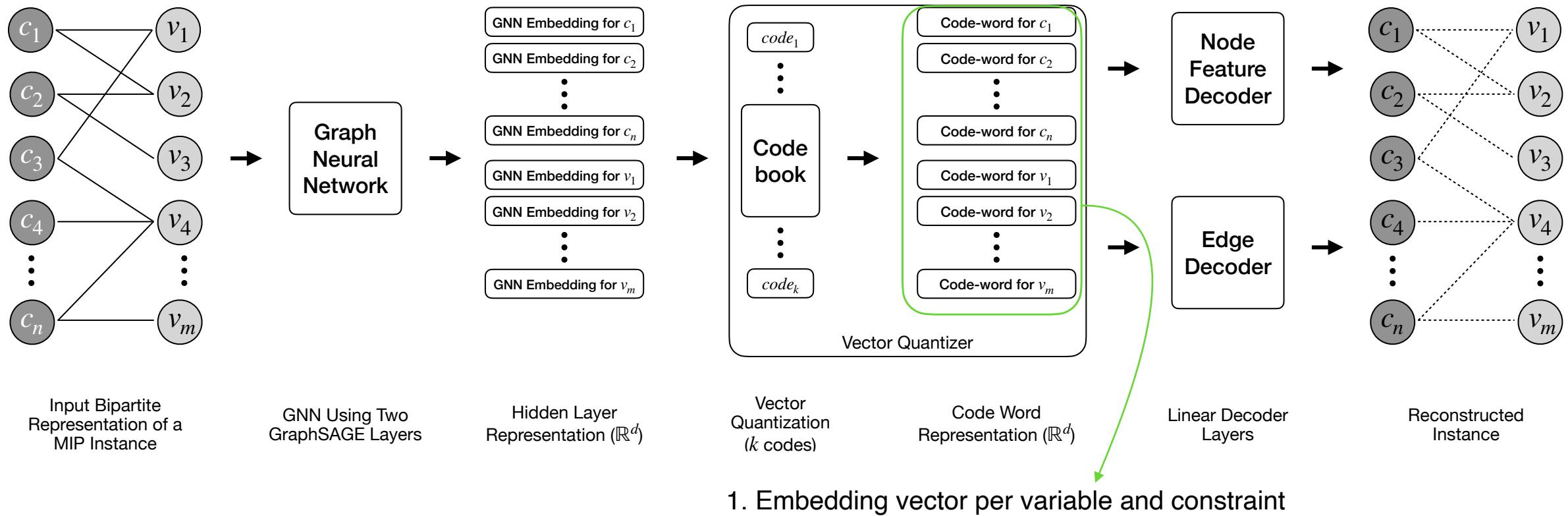
## Overall Architecture of Unsupervised Pre-training



# Methodology

## Overall Architecture of Unsupervised Pre-training

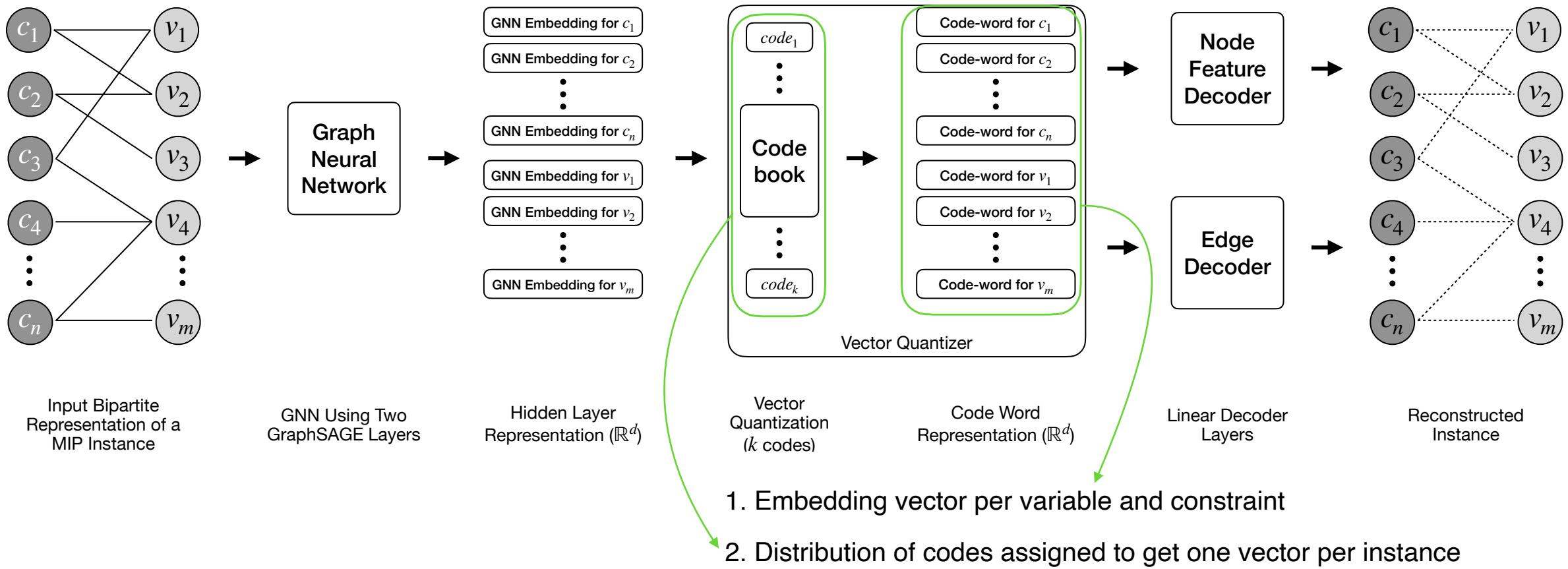
Observe that we get 2 types of embeddings



# Methodology

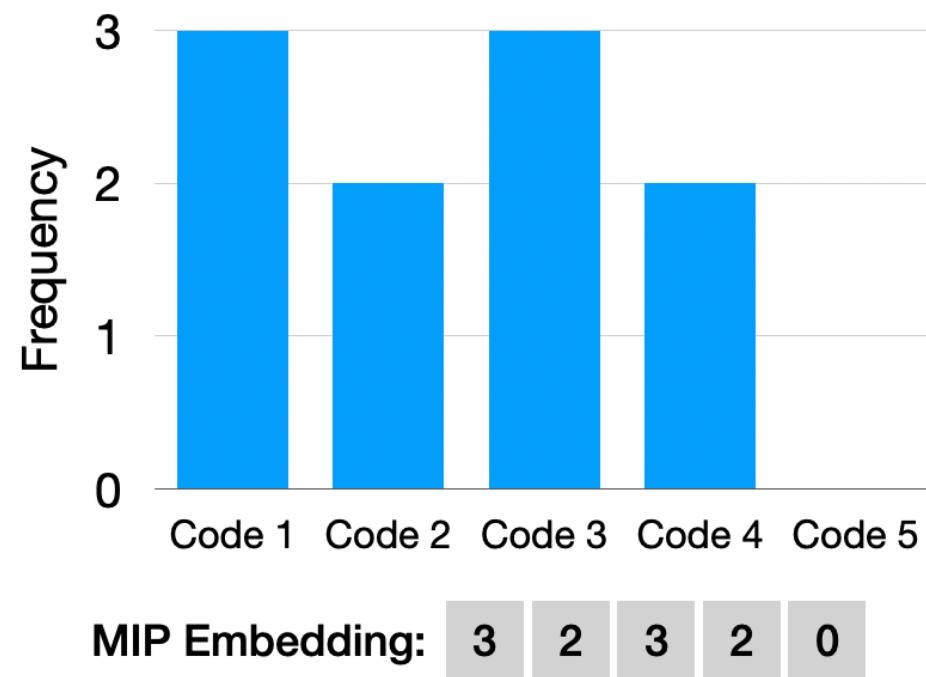
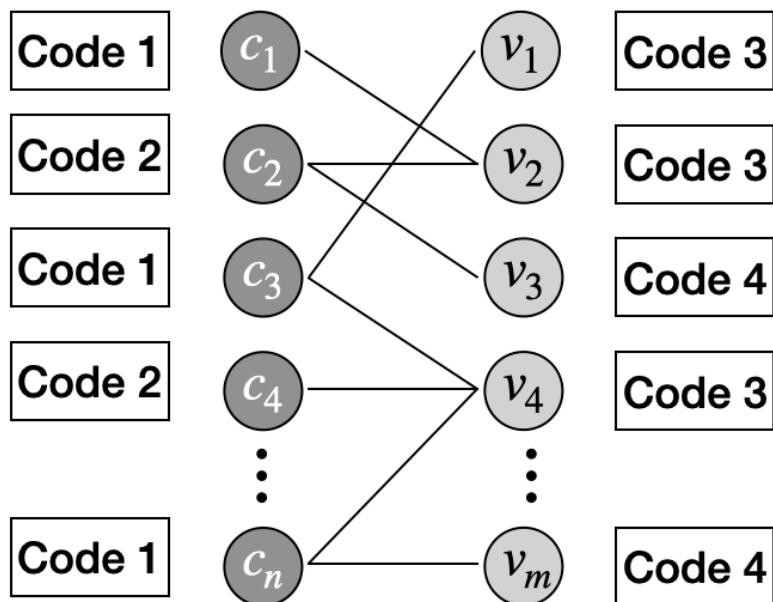
## Overall Architecture of Unsupervised Pre-training

Observe that we get 2 types of embeddings



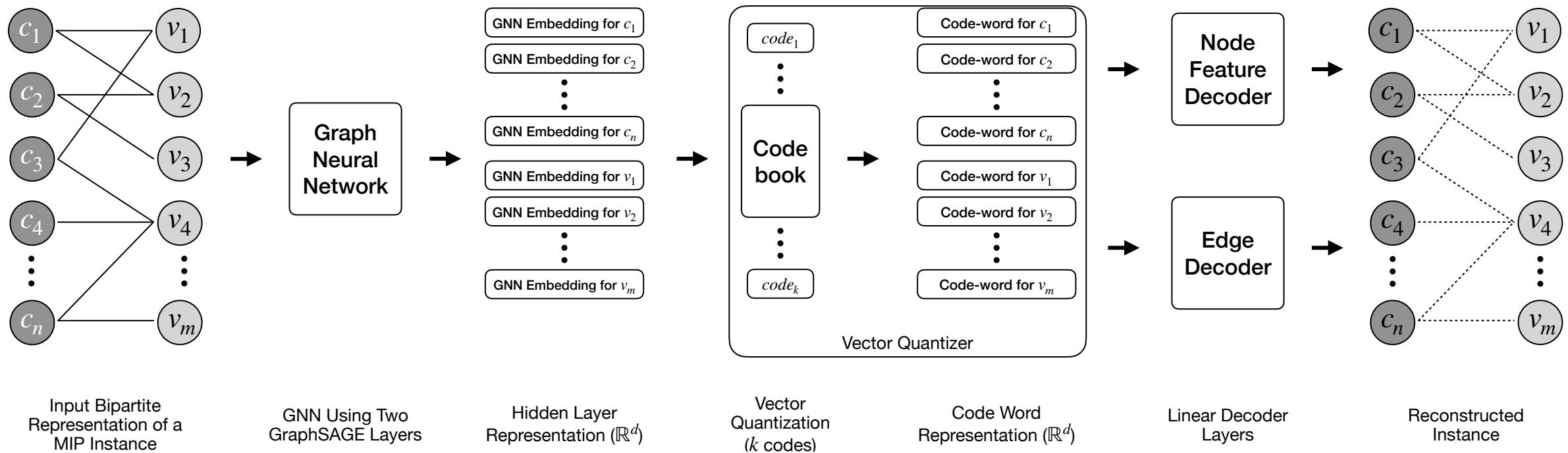
# Methodology

## MIP Embedding Aside



# Methodology

## Overall Architecture of Unsupervised Pre-training



# Datasets

- MIPLIB
  - 600 instances

# Datasets

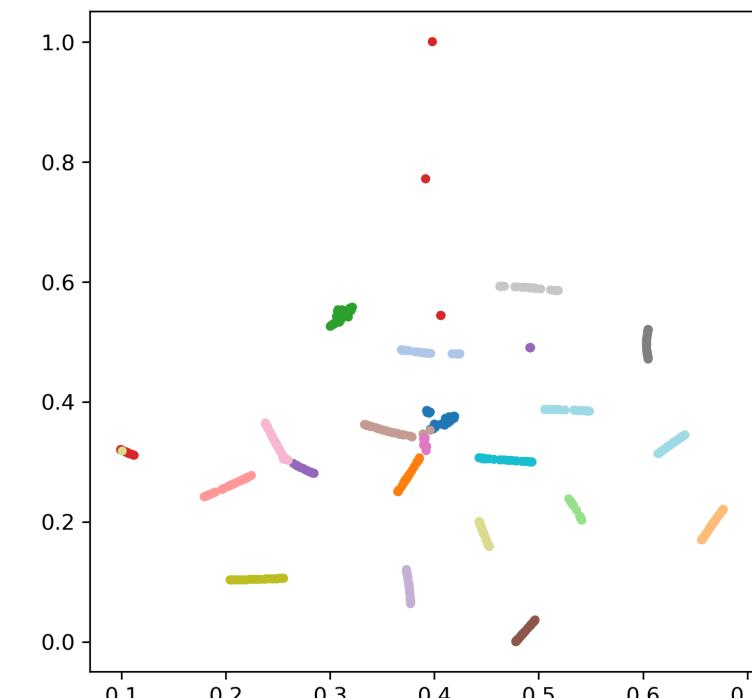
- MIPLIB
  - 600 instances
  - For each instance, create two more instances by randomly deleting 5% and 10% of constraints
  - Each instance maintains feasibility
  - These 1800 instances are used to train the unsupervised FORGE model

# Datasets

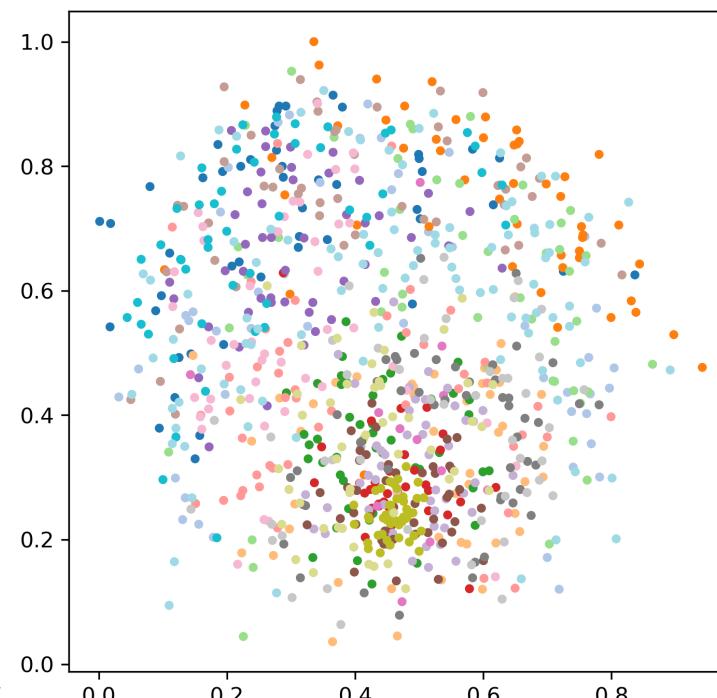
- MIPLIB
  - 600 instances
  - For each instance, create two more instances by randomly deleting 5% and 10% of constraints
  - Each instance maintains feasibility
  - These 1800 instances are used to train the unsupervised FORGE model
- Distributional MIPLIB
  - Set Cover (easy, medium, hard)
  - Maximum Independent Set (easy, medium)
  - Minimum Vertex Cover (easy, medium, hard)
  - Generalized Independent Set (easy, medium, hard, very-hard, very-hard2, ext-hard)
  - Combinatorial Auction (very-easy, easy, medium, very-hard, very-hard2)
  - Item Placement (very-hard)
  - Maritime Inventory Routing Problem (medium)
- 50 instances from each category - 1050 instances used as the test set

# Visualizing MIP Instances from Unsupervised Pre-training

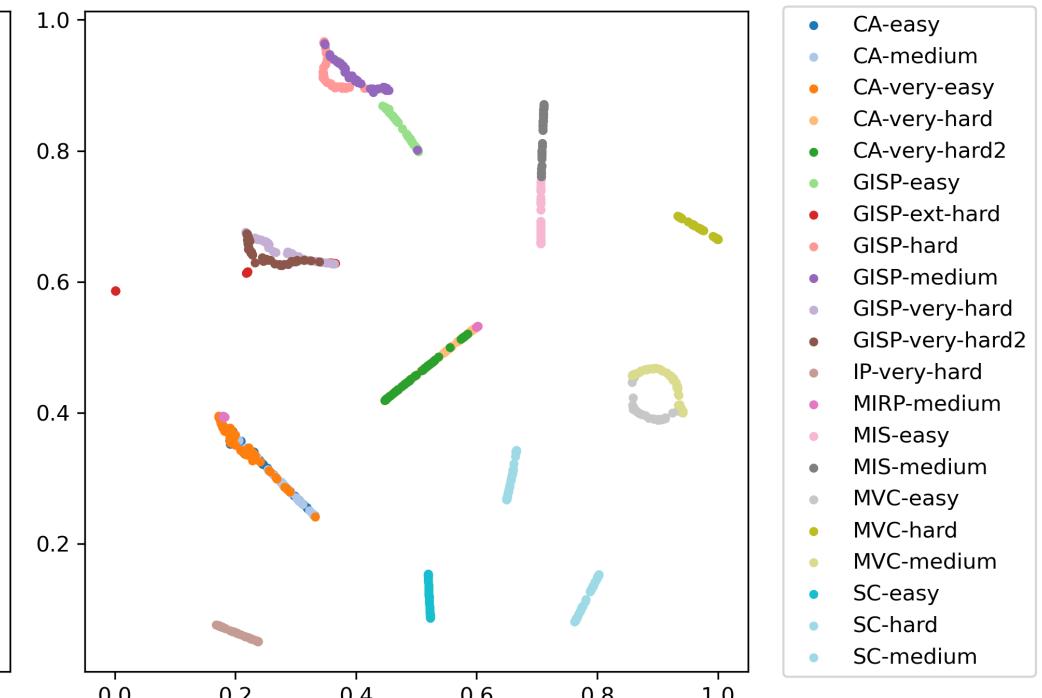
(a) FORGE Embedding  
NMI:  $0.843 \pm 0.003$



(b) Mean Readout  
NMI:  $0.087 \pm 0.035$



(c) Label Propagation  
NMI:  $0.7907 \pm 0.025$



- CA-easy
- CA-medium
- CA-very-easy
- CA-very-hard
- CA-very-hard2
- GISP-easy
- GISP-ext-hard
- GISP-hard
- GISP-medium
- GISP-very-hard
- GISP-very-hard2
- IP-very-hard
- MIRP-medium
- MIS-easy
- MIS-medium
- MVC-easy
- MVC-hard
- MVC-medium
- SC-easy
- SC-hard
- SC-medium

## Takeaway:

FORGE can cleanly cluster out previously unseen MIP instances with the highest NMI

# **Supervised Fine-tuning - Integrality Gap**

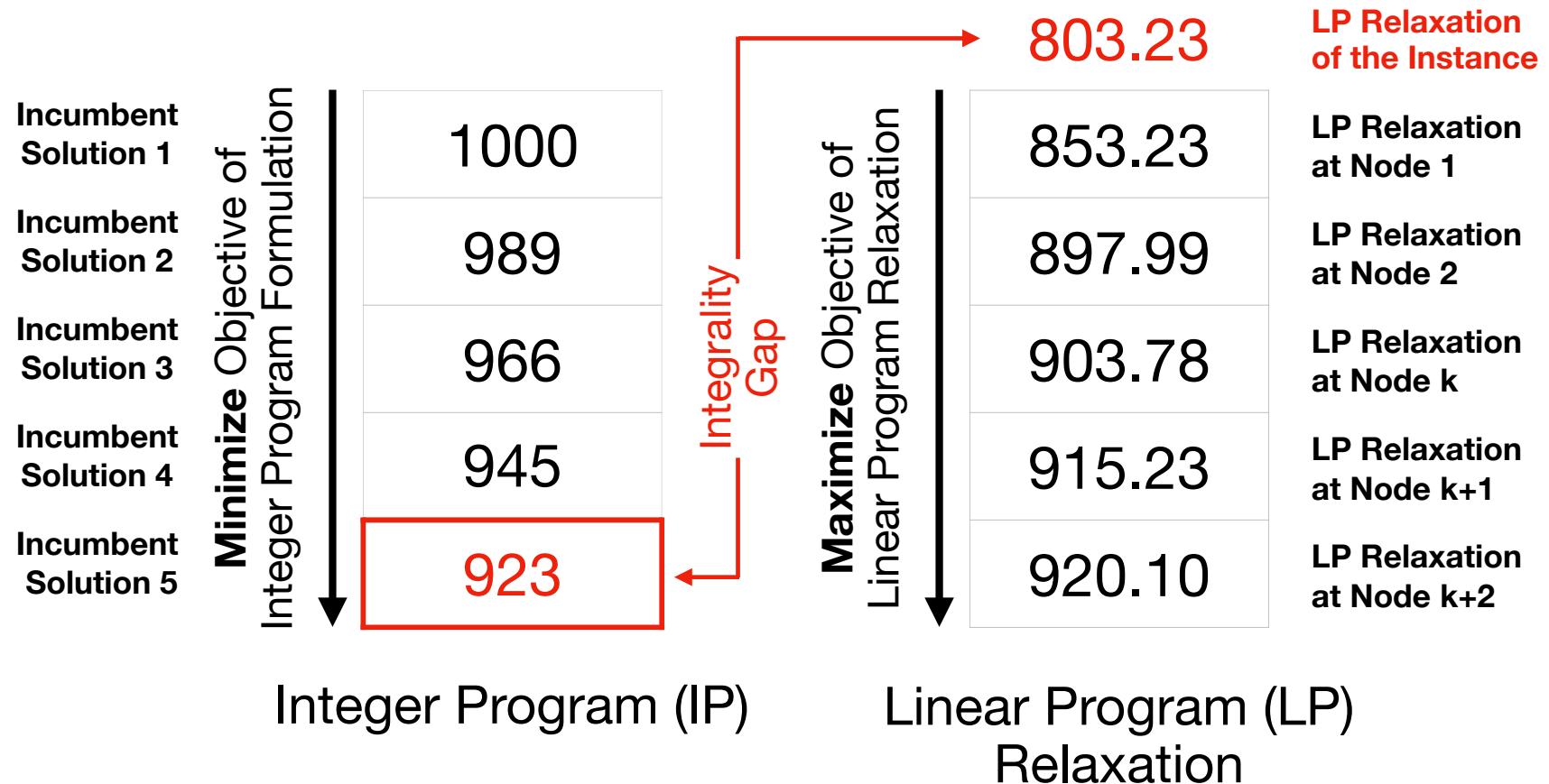
- Can we fine tune FORGE to predict the integrality gap?

# Supervised Fine-tuning - Integrality Gap

- What is an *integrality gap*?

# Supervised Fine-tuning - Integrality Gap

- What is an *integrality gap*?



# Supervised Fine-tuning - Integrality Gap

- We add a simple single layer prediction head to predict the integrality gap.
- The predicted gap is then used to compute a “pseudo-cut”.
  - This pseudo-cut is added as a constraint to a solver.
  - Note that a overestimation of the pseudo-cut would lead to a suboptimal solution.
- FORGE is pre-trained to learn the structures of all 1800 MIPLIB instances as well as 1050 Distributional MIPLIB instances.

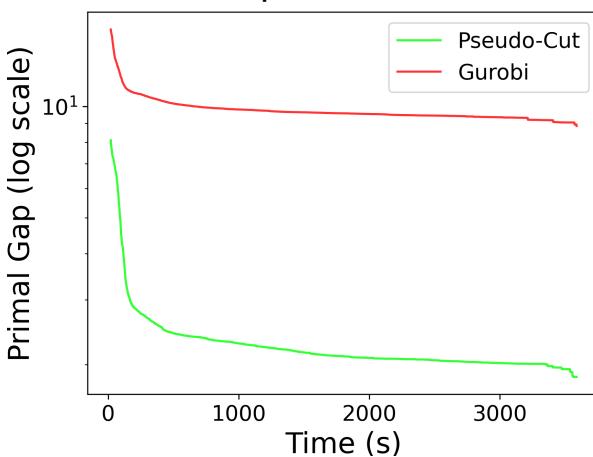
# Supervised Fine-tuning - Integrality Gap

- We add a simple single layer prediction head to predict the integrality gap.
- The predicted gap is then used to compute a “pseudo-cut”.
  - This pseudo-cut is added as a constraint to a solver.
  - Note that a overestimation of the pseudo-cut would lead to a suboptimal solution.
- FORGE is pre-trained to learn the structures of all 1800 MIPLIB instances as well as 1050 Distributional MIPLIB instances.
- **Fine Tuning Training Data:** CA (very-easy, easy, medium), SC (easy, medium, hard), and GIS (easy, medium, hard) with 50 instances for each. In total, we obtain a total of 450 training instances.

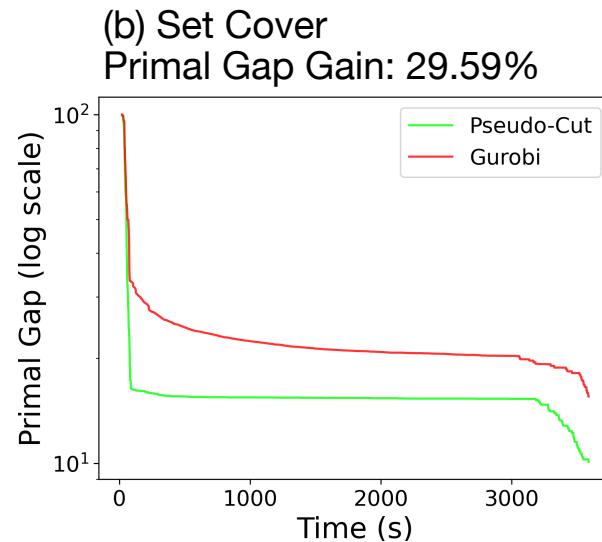
# Results - Integrality Gap

Tests are run on 50 ‘very-hard’ unseen instances from Distributional MIPLIB.

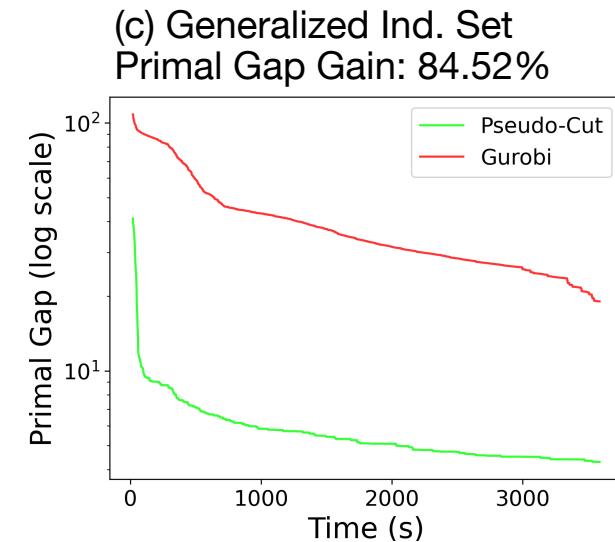
(a) Combinatorial Auction  
Primal Gap Gain: 76.77%



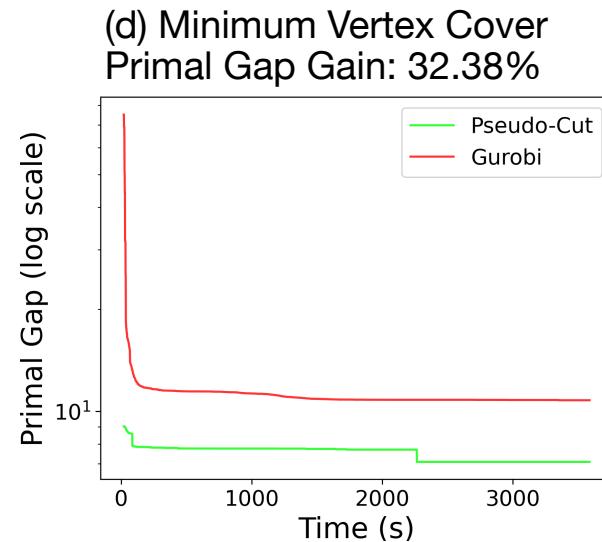
(b) Set Cover  
Primal Gap Gain: 29.59%



(c) Generalized Ind. Set  
Primal Gap Gain: 84.52%



(d) Minimum Vertex Cover  
Primal Gap Gain: 32.38%



## Takeaway:

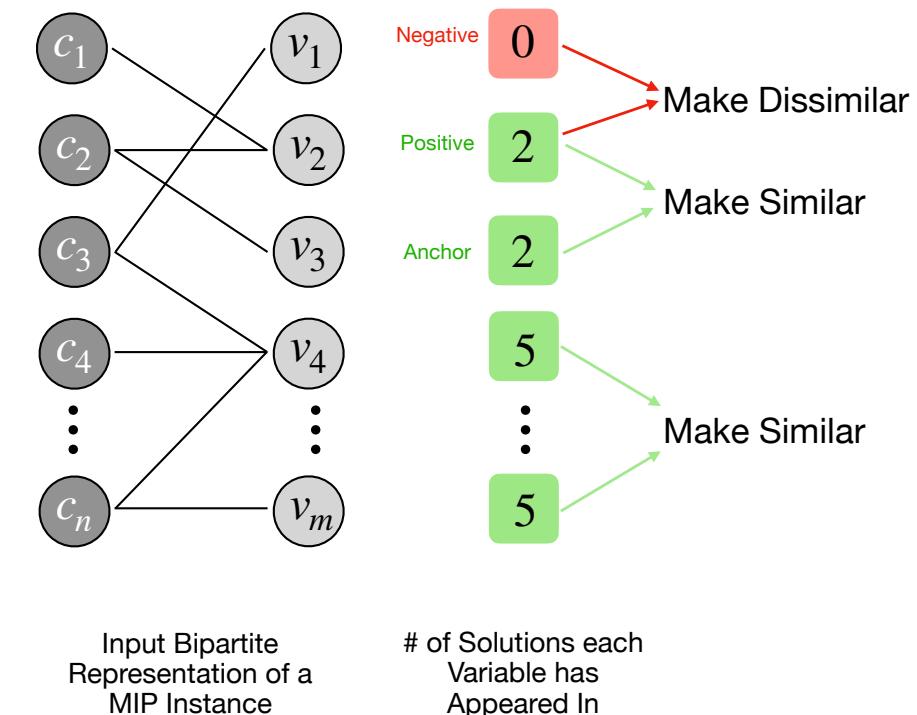
FORGE generated pseudo-cuts lead to a significant decrease in primal gaps.

# Supervised Fine-tuning - Warm Start

- Can we predict which variables will be part of the solution?
- How do we train this?
  - Binary Cross Entropy - commonly used approach but has a large class imbalance issue

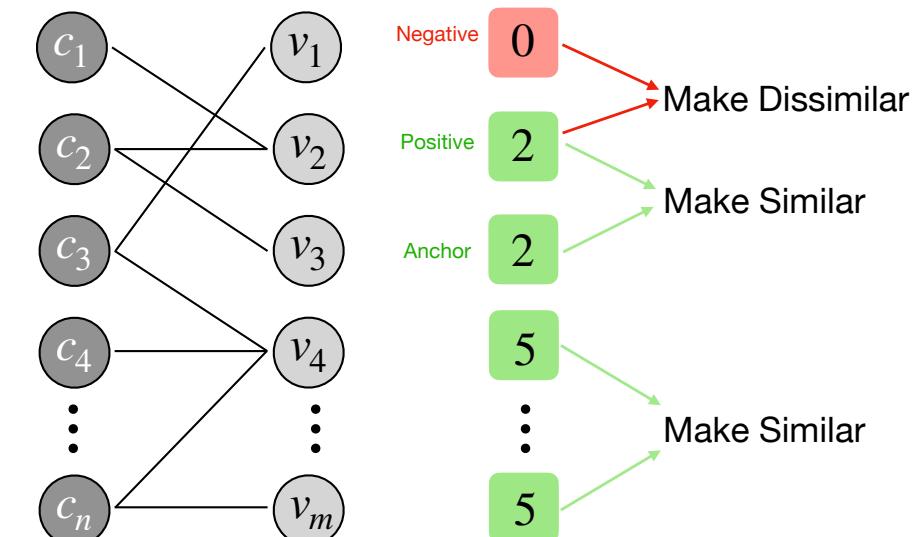
# Supervised Fine-tuning - Warm Start

- Can we predict which variables will be part of the solution?
- How do we train this?
  - Binary Cross Entropy - commonly used approach but has a large class imbalance issue
  - Triplet Loss:
    - Generate 5 solutions
    - Make variables appearing in solutions similar to each other
    - Variables appearing in **none** of the solutions are used as negative variables
    - Negative variables are further filtered as variables that don't appear in any solution but are closest to positive variables in **unsupervised embedding space**



# Supervised Fine-tuning - Warm Start

- **Fine Tuning Training Data:** 100 instances each from CA (easy, medium), SC (easy, medium, hard) and GIS (easy, medium) for a total of 700 training instances.



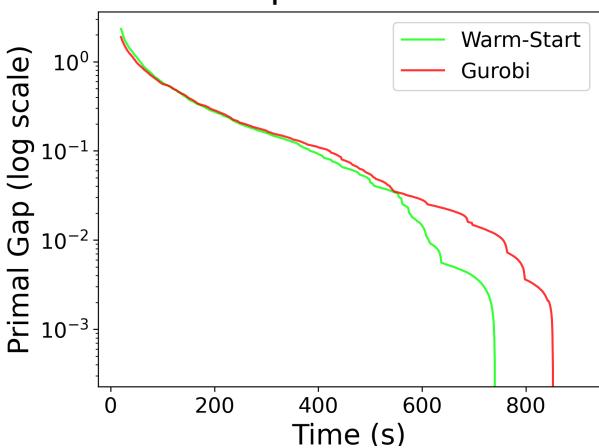
Input Bipartite  
Representation of a  
MIP Instance

# of Solutions each  
Variable has  
Appeared In

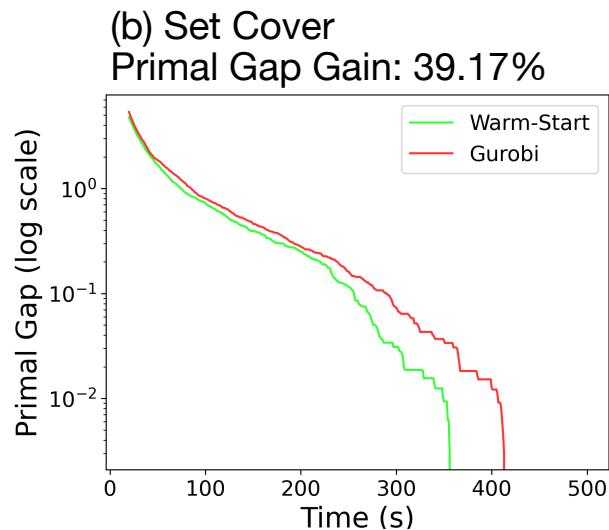
# Results - Warm Start

Tests are run on 50 ‘medium’ unseen instances from Distributional MIPLIB.

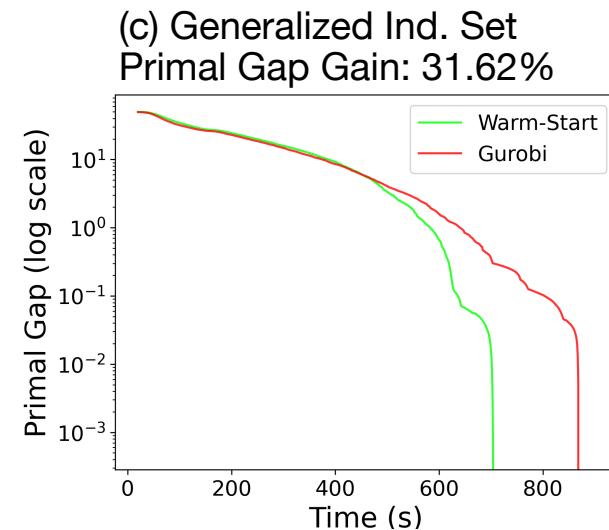
(a) Combinatorial Auction  
Primal Gap Gain: 31.16%



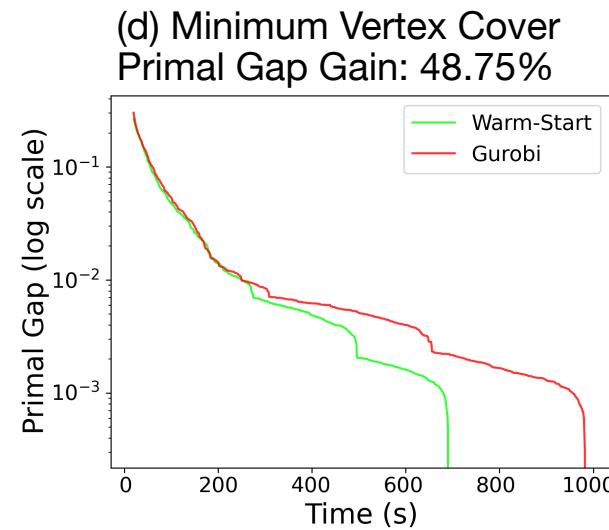
(b) Set Cover  
Primal Gap Gain: 39.17%



(c) Generalized Ind. Set  
Primal Gap Gain: 31.62%



(d) Minimum Vertex Cover  
Primal Gap Gain: 48.75%



## Takeaway:

Gurobi with FORGE generated warm starts leads to a significant decrease in primal gaps and faster run times.

# Additional Results

## Integrality Gaps

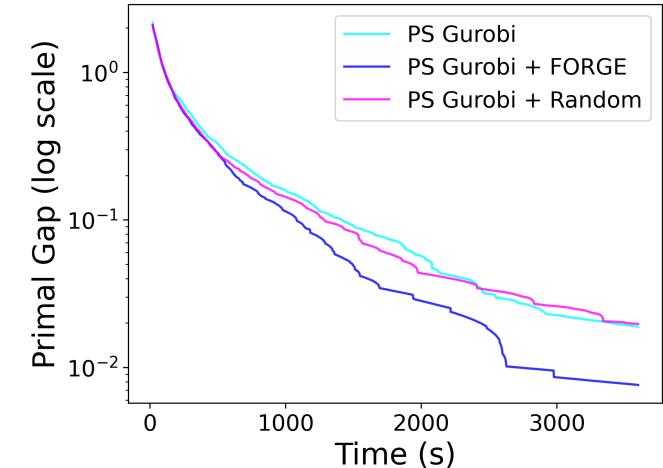
- Li et al. [1] train a GNN on 38,256 instances from 643 generated problem types and test on 11,584 instances spanning 157 problem types.
- We train on **no additional data** and test on 17,500 previously unseen instances spanning 400 problem types, from the dataset in [1].
- FORGE achieves a mean deviation of 18.63% in integrality gap prediction, outperforming the 20.14% deviation reported.

# Additional Results

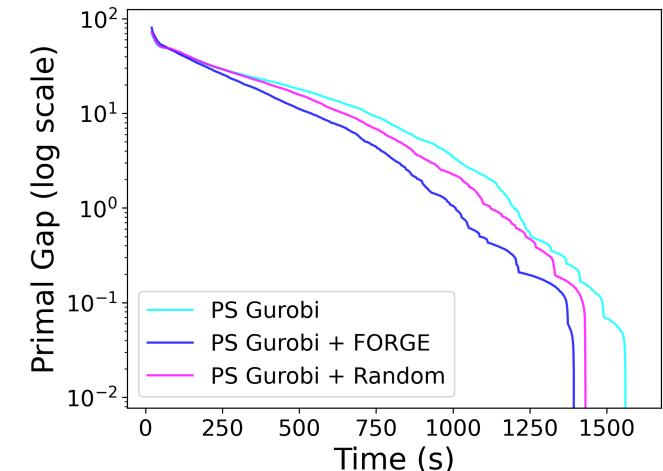
## Warm Starts

- We also compare warm starts with PS-Gurobi [2].
  - FORGE embeddings for each variable and constraint are added to PS-Gurobi.
  - Since adding these embeddings increases model complexity significantly, we also add random embeddings of the same size to ensure any gains are **not due to larger model size**.
  - FORGE + PS-Gurobi outperforms the original variant in terms of primal gap and run time.

(a) Combinatorial Auction  
Primal Gap Gain: 41.07%



(b) Generalized Ind. Set  
Primal Gap Gain: 50.51%



# FORGE in Practice

- Integrality Gap
  - Easiest to use
  - Pass in a .lp or a .mps file - get back a real number
  - Add constraint that the integer solution is greater than the real number generated
- Warm Starts
  - Pass in a .lp or a .mps file - get back a list of variables
  - Set initial values of variables - solver specific - for example, hint values in Gurobi

# Summary

- FORGE uses a **single** model with **~3.25M parameters**.
- FORGE can generate one embedding vector per MIP instance and can effectively cluster unseen instances and place them within the space of all MIP instances.
- FORGE can be fine tuned on a variety of tasks for multiple problem types.
  - A single FORGE model can be used to predict **both** warm-starts as well as integrality gaps for a variety of problem type and difficulty pairs.



**Thank you!  
Questions?**