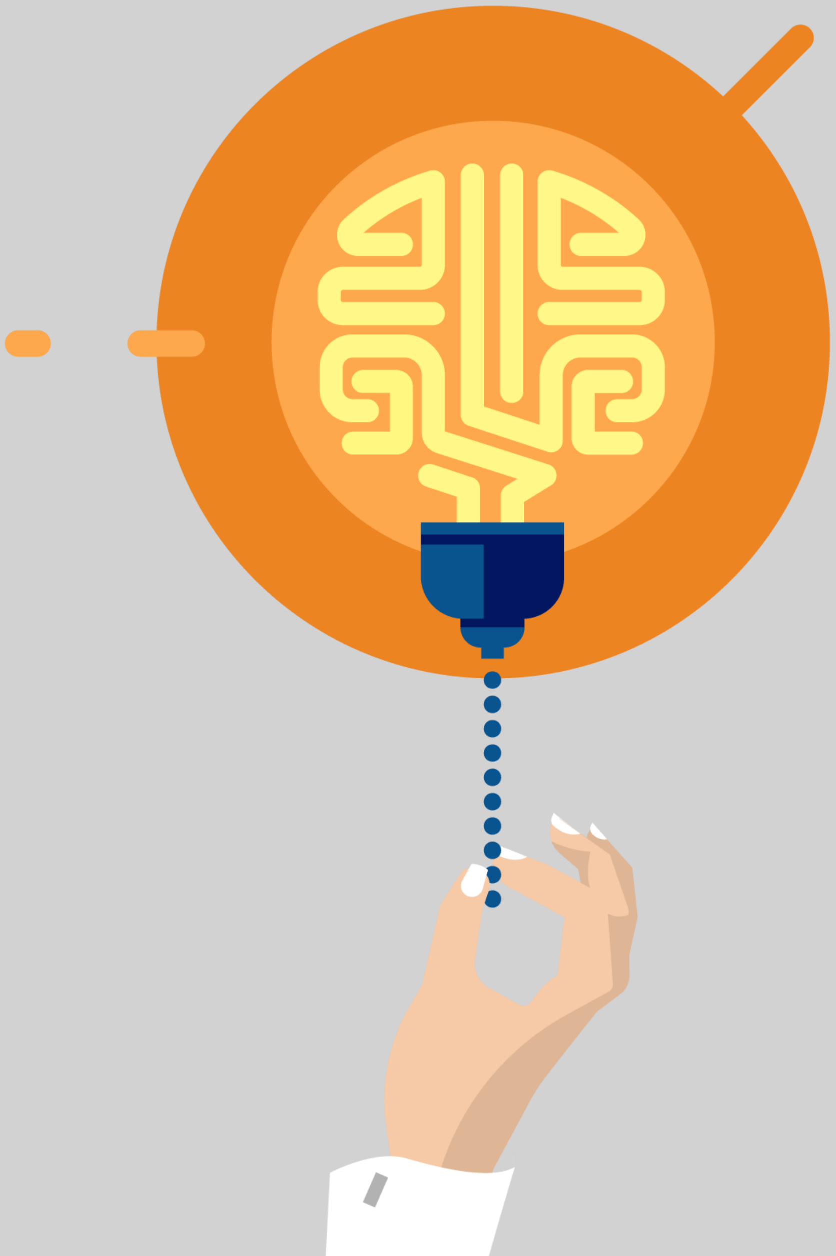
Microsoft

Graph Neural Networks in Software Engineering Research

Miltos Allamanis

TU Delft 2020 – IN4334

Background



Machine Learning

Machine Learning Model

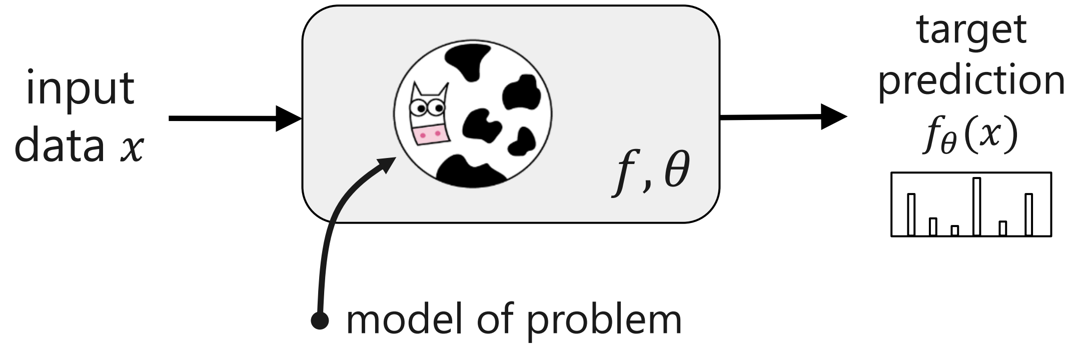
Designed by humans

Model Parameters

Learned from data

"All models are wrong, some are useful"
– George Box

Supervised Machine Learning



- Given an i.i.d. dataset $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Pick θ that minimize Loss $\mathcal{L}(\theta) = \frac{1}{N} \sum_i L(f_\theta(x_i), y_i)$

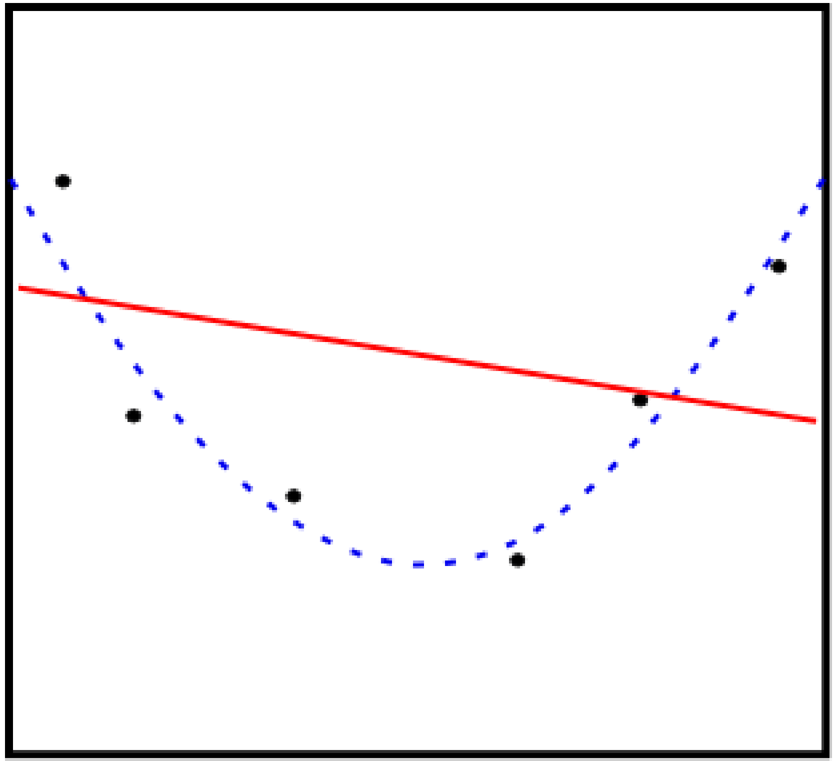
Gradient Descent: Learning Model Parameters θ

while not converged
- Compute (estimate of) derivative $g \approx \nabla_\theta \mathcal{L}(\theta)$
- Update $\theta \leftarrow \theta - q(g)$

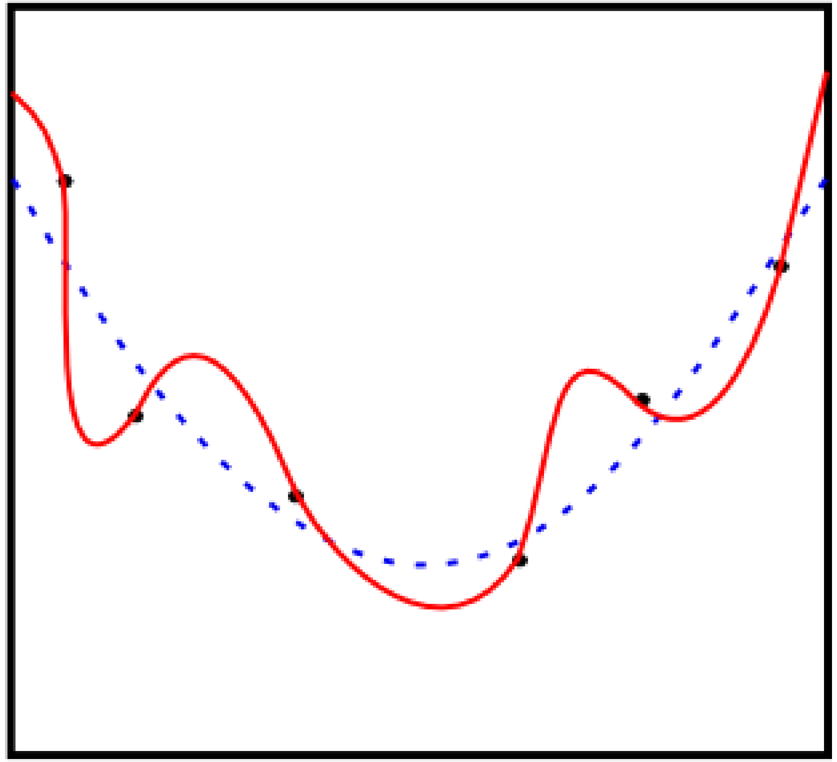
Image from marple.eeb.uconn.edu

- **Minimize** loss function in training set
- Use **computational methods** of optimization

Generalization



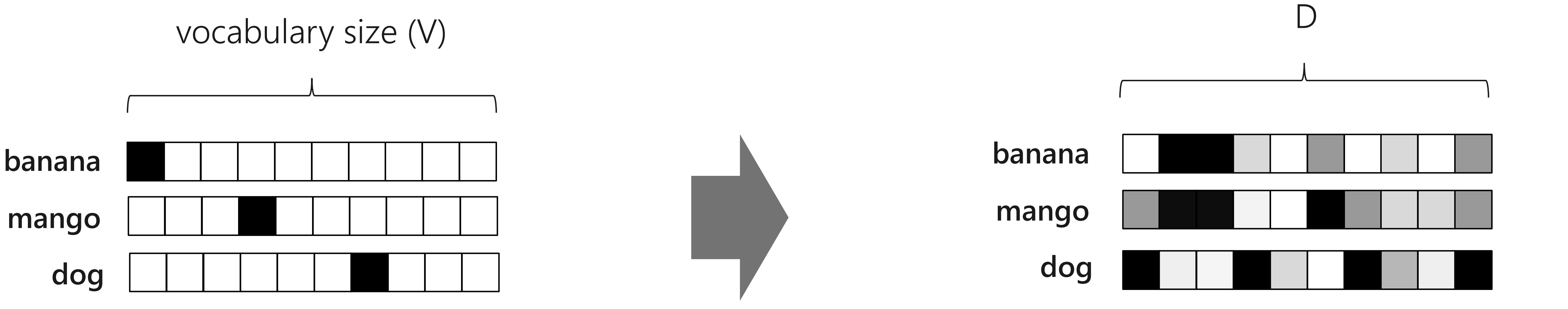
Underfitting



Overfitting

graphs from http://antianti.org/?p=175

Distributed Vector Representations



Local representation
(1-hot)

Distributed representation

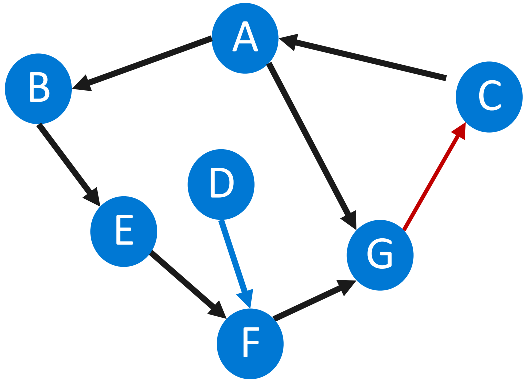
$\boldsymbol{r} = E\ \mathbb{I}_w$ with E a DxV matrix

“vocabulary”

Graph Notation

- Nodes/Vertices

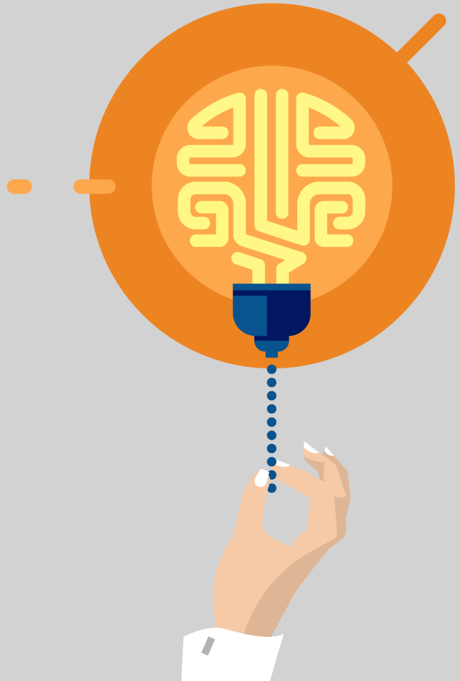
- Edges/Links



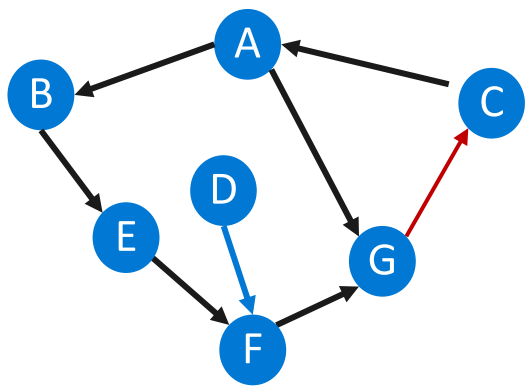
$$G = (V, E)$$

Graph Neural Networks

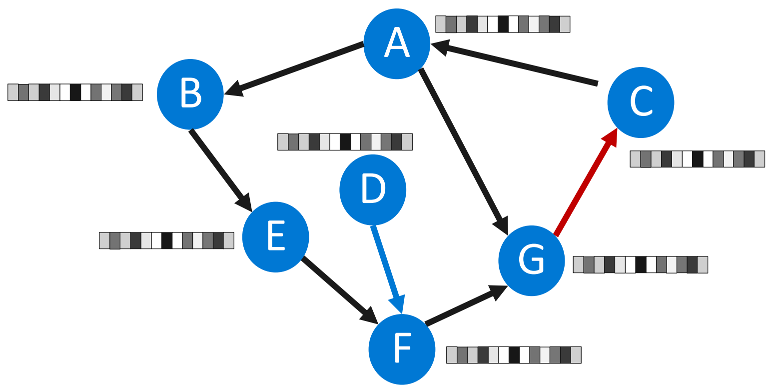
and Neural Message Passing



Graph Neural Networks

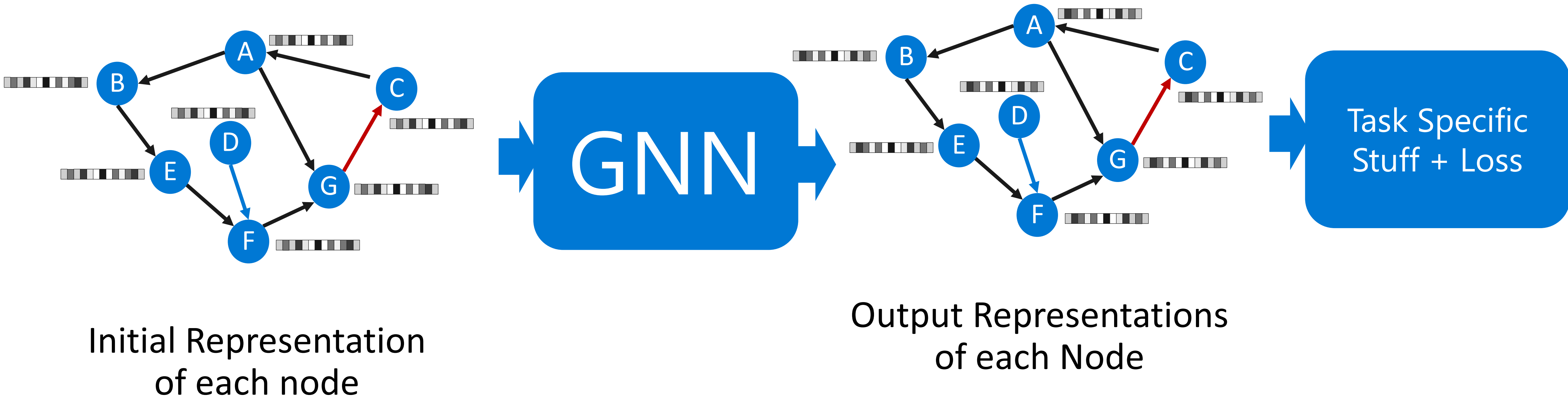


Graph Representation
of Problem

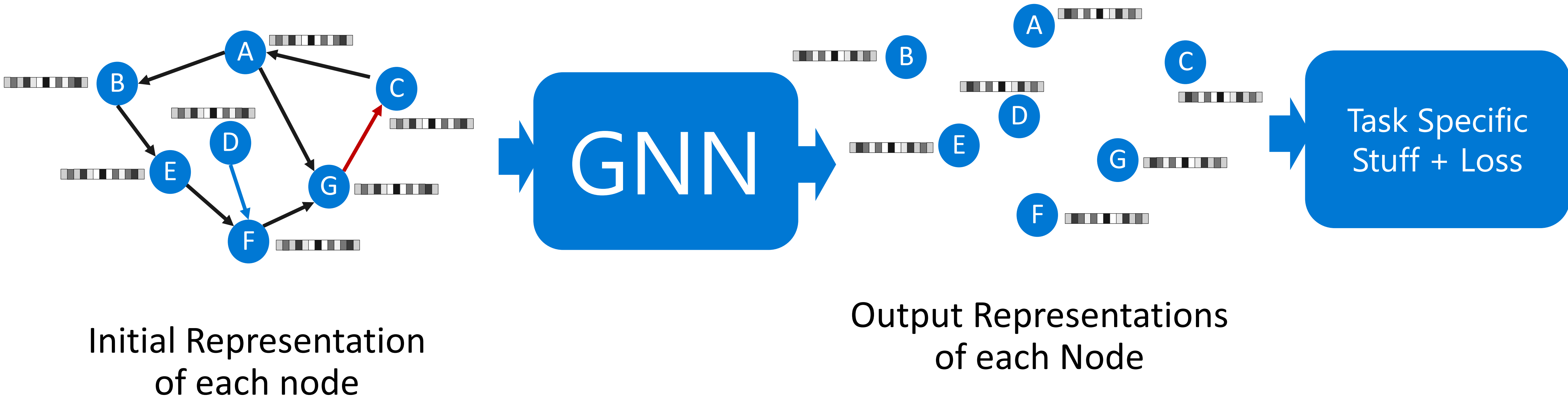


Initial Representation
of each node

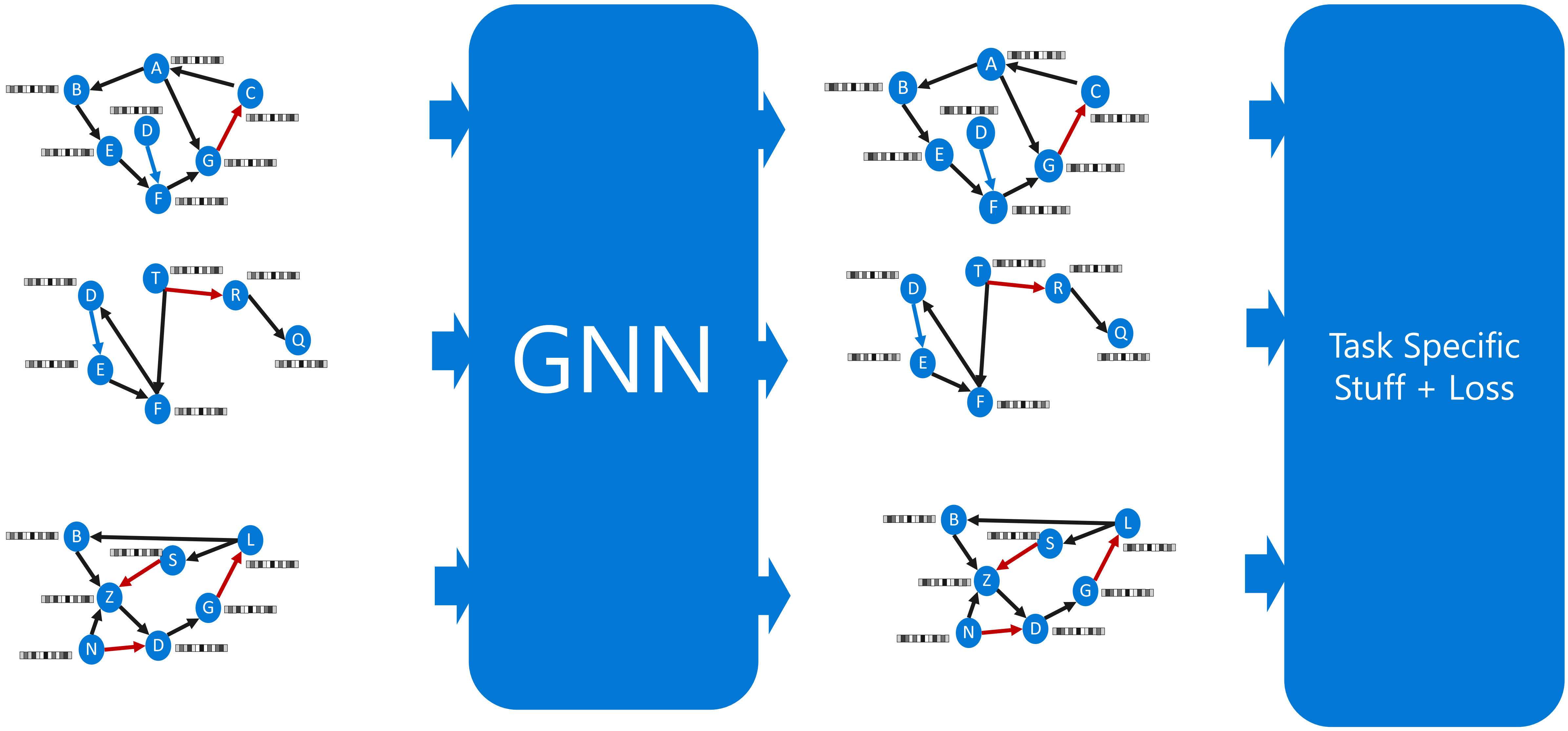
Graph Neural Networks



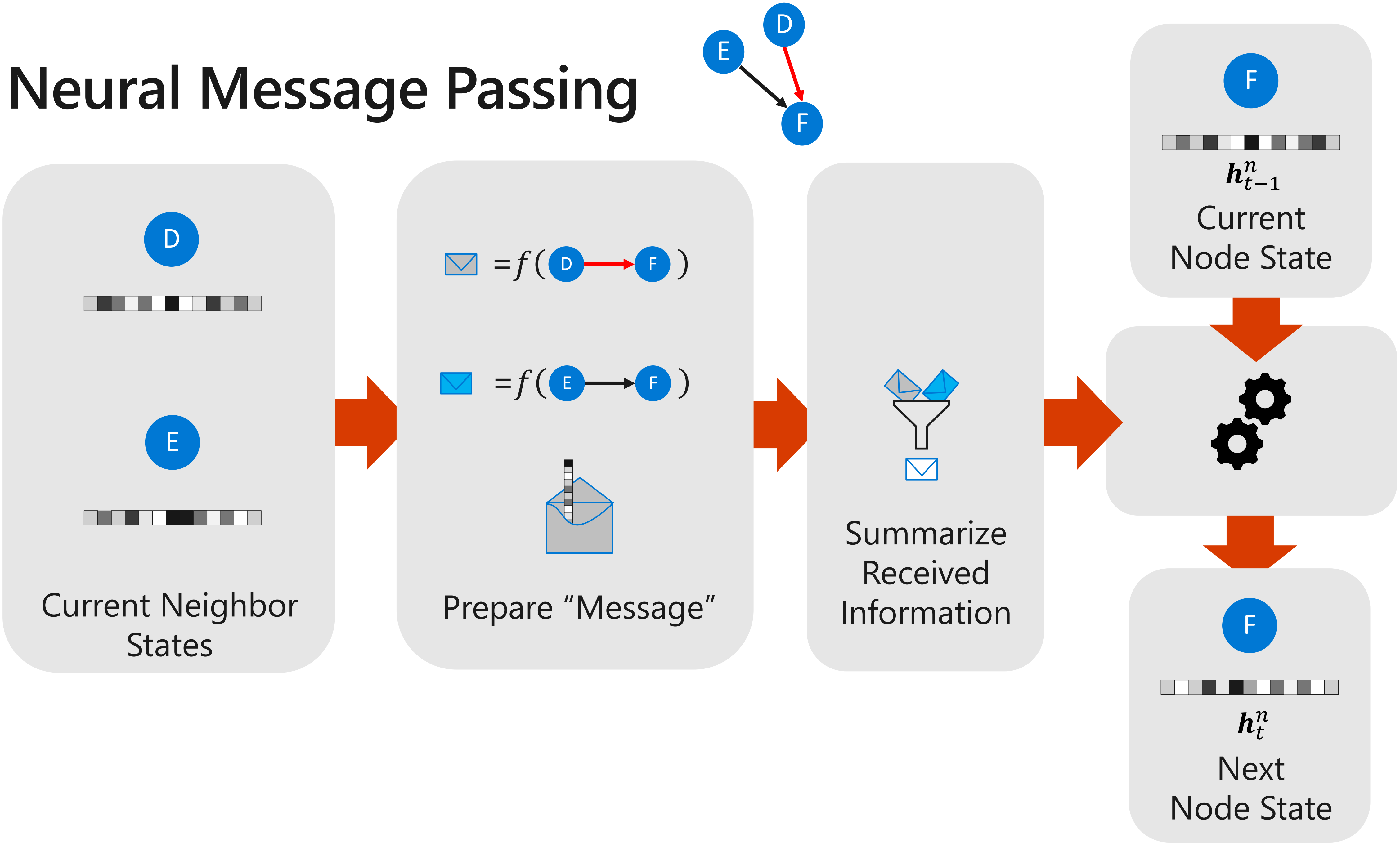
Graph Neural Networks

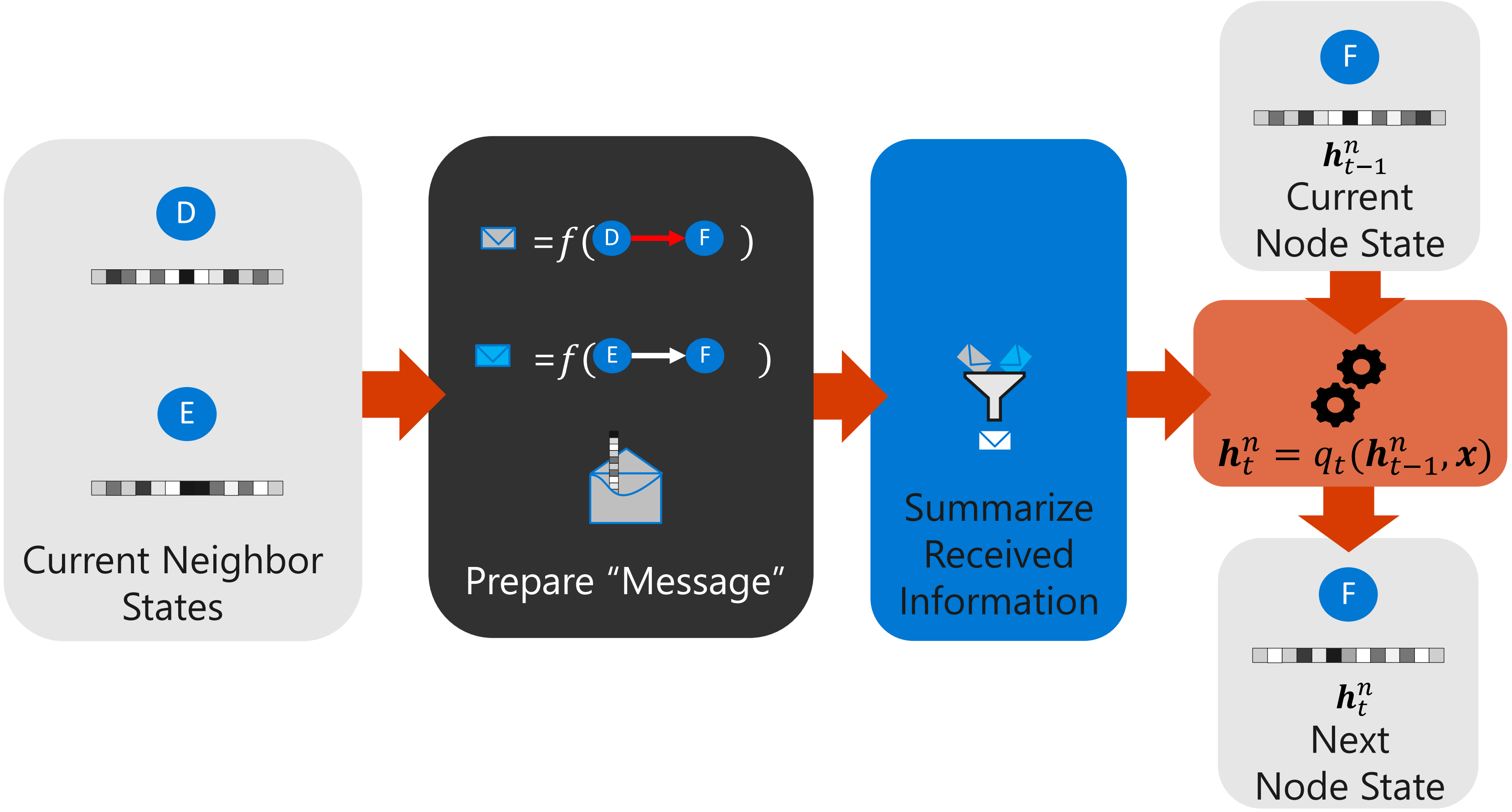


Graph Neural Networks



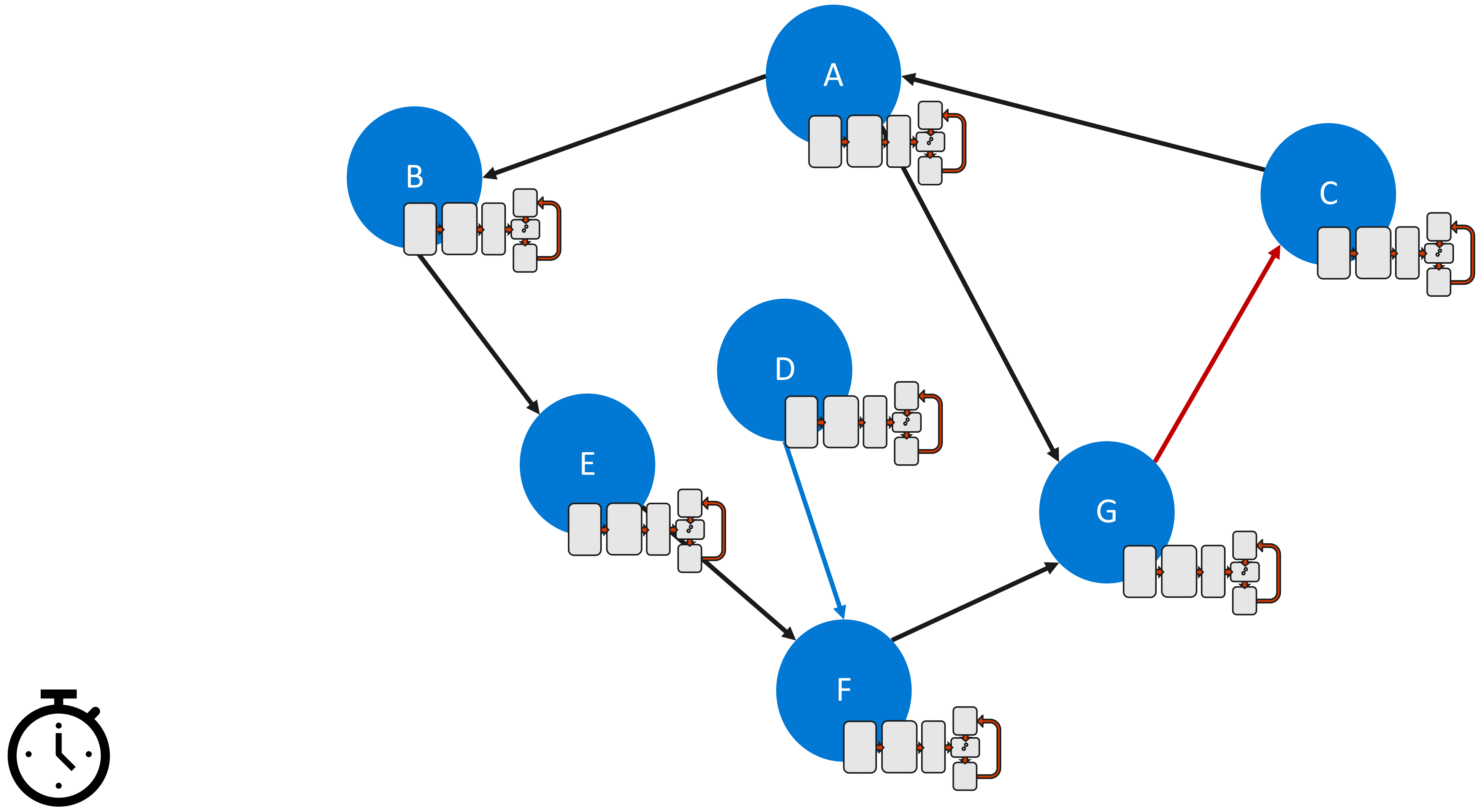
Neural Message Passing

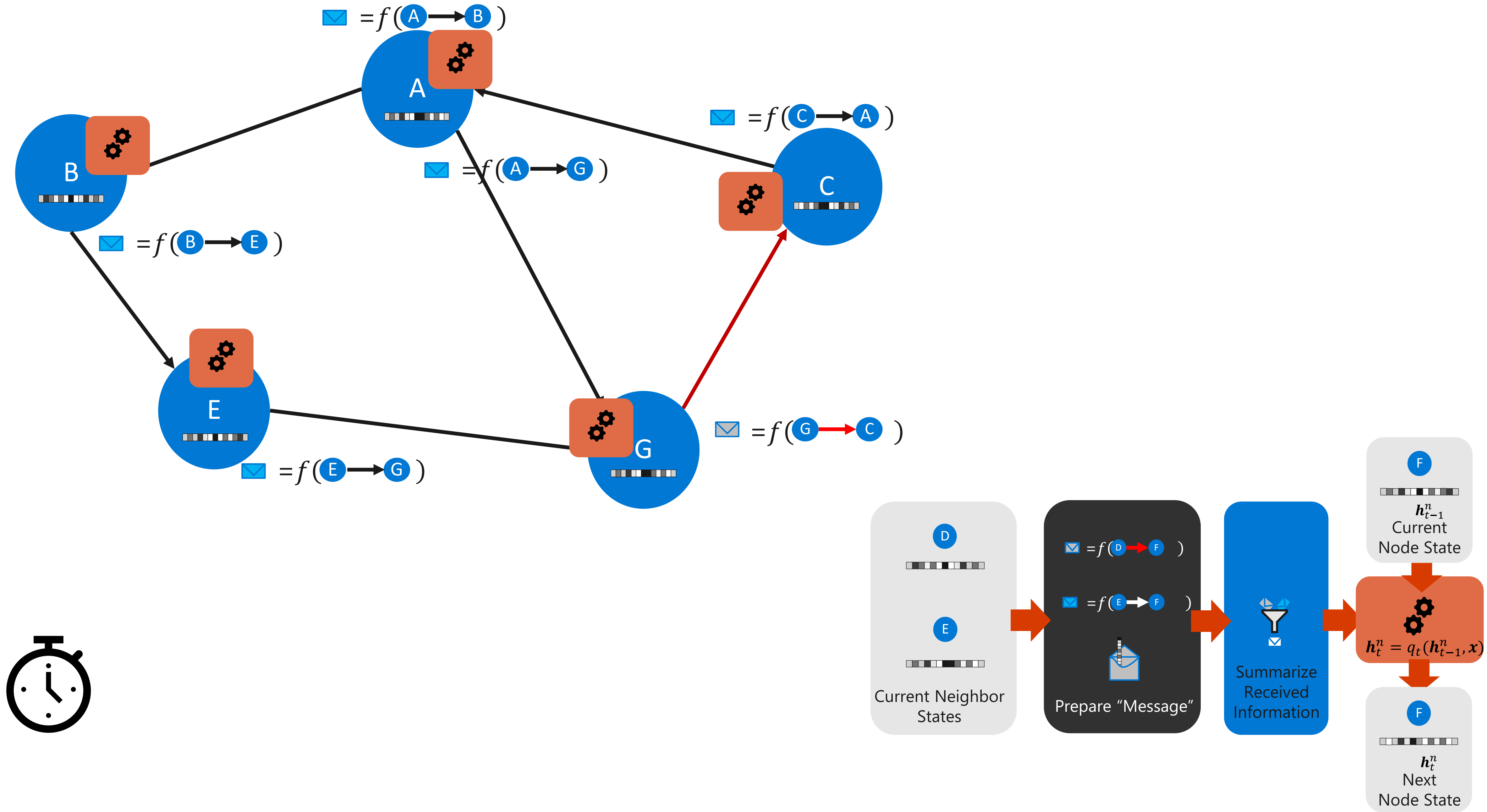





```
h_0^n = x_n, ∀n
for t = 0 ... T do
    for n ∈ V do
        a_t^n = Aggregate({h_t^v | v ∈ N(n)})
        h_{t+1}^n = Update(a_t^n, h_{t+1}^n)
return {h_T^n}
```

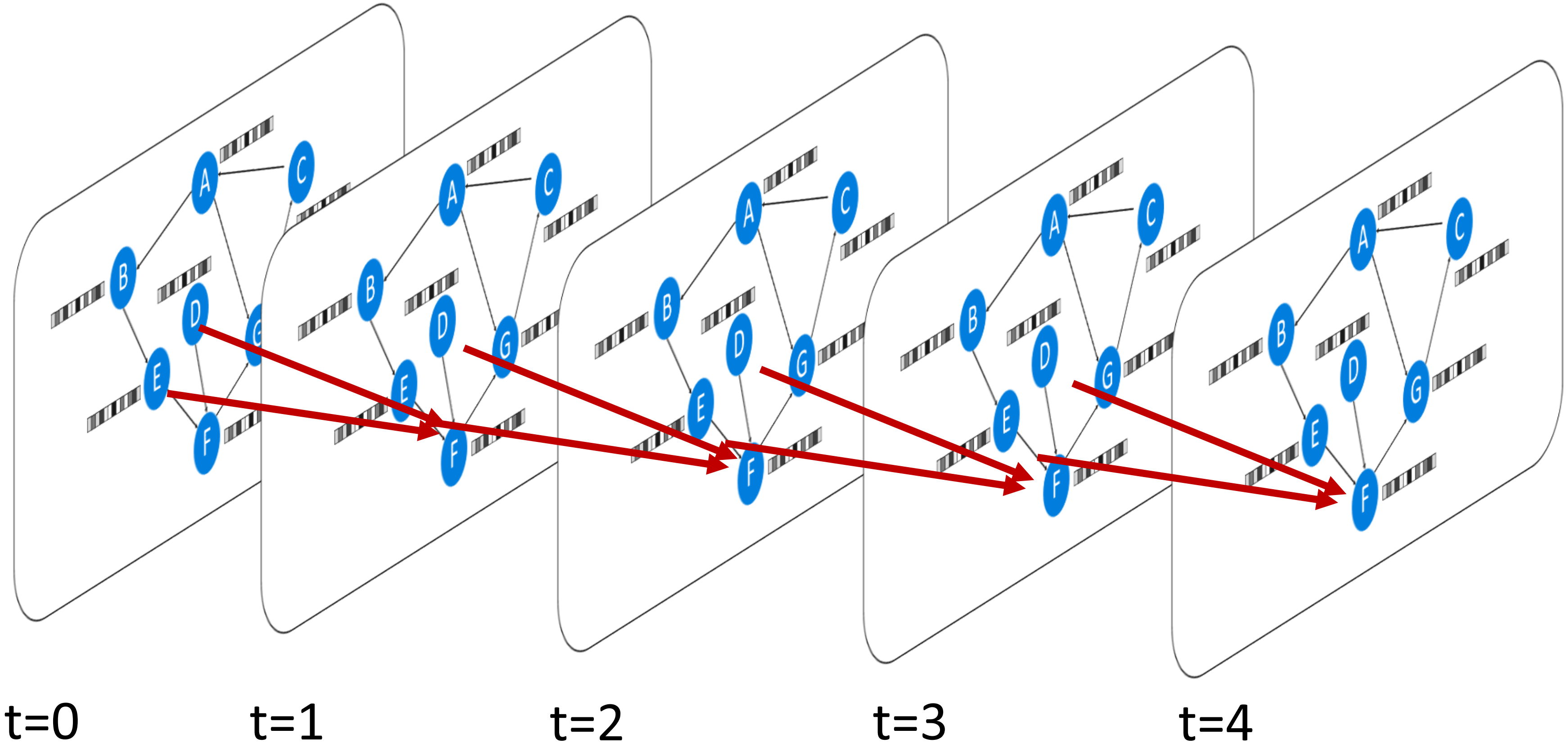
$$\boldsymbol{h}_t^n = q_t\left(\boldsymbol{h}_{t-1}^n, \bigcup_{\substack{k \\ n_j : n_j \to n}} f_t\left(\boldsymbol{h}_{t-1}^n, k, \boldsymbol{h}_{t-1}^{n_j}\right)\right)$$


A
B
C
D
E
F
G

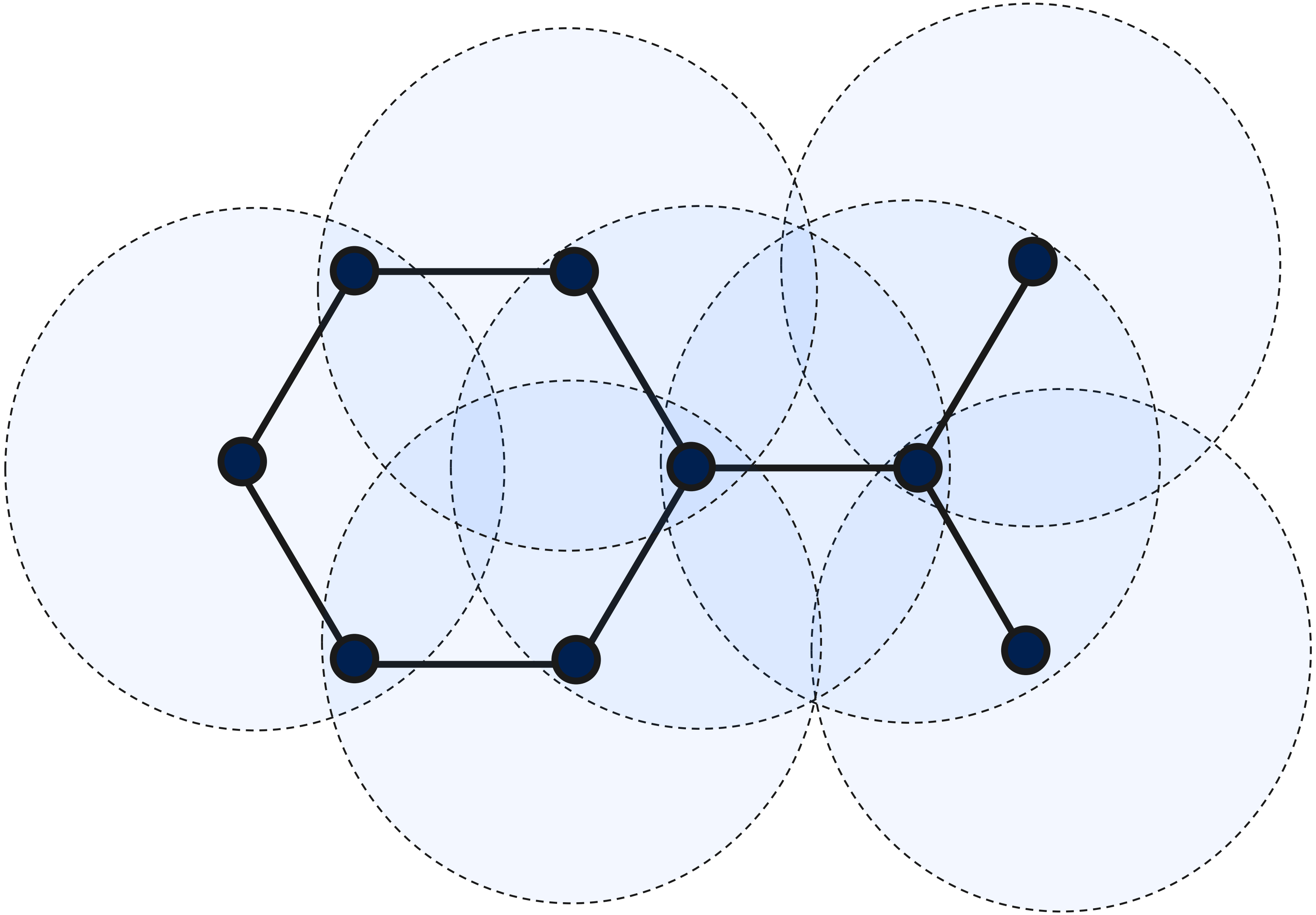


=f(A → B)
A
B
=f(C → A)
=f(A → G)
C
=f(B → E)
E
G
=f(G → C)
=f(E → G)
D
E
Current Neighbor States
=f(D → F)
=f(E → F)
Prepare "Message"
Summarize Received Information
F
h_{t-1}^n
Current Node State
$h_t^n = q_t(h_{t-1}^n, x)$
F
h_t^n
Next Node State

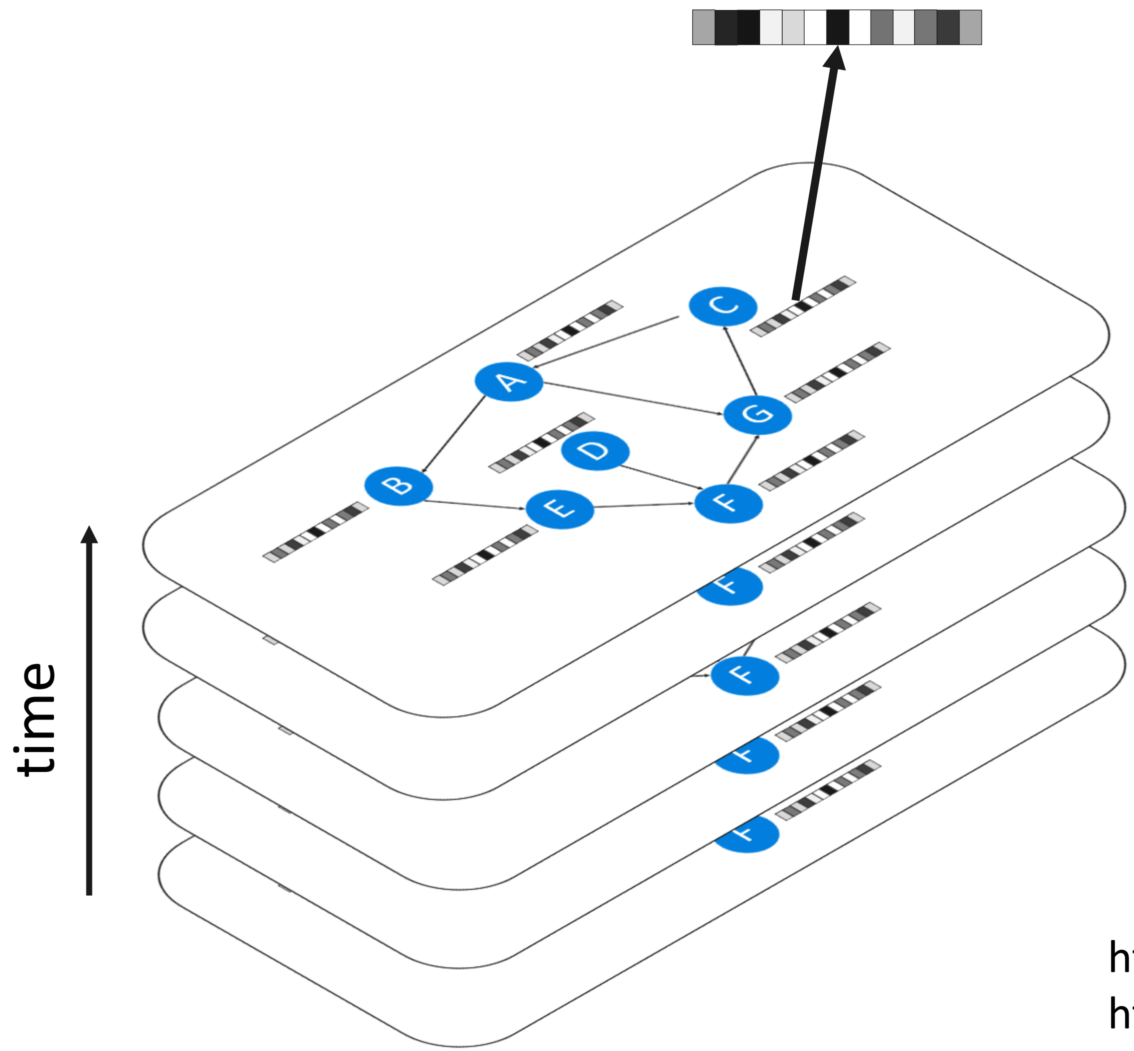
Graph Neural Networks: Message Passing



GNNs: Synchronous Message Passing



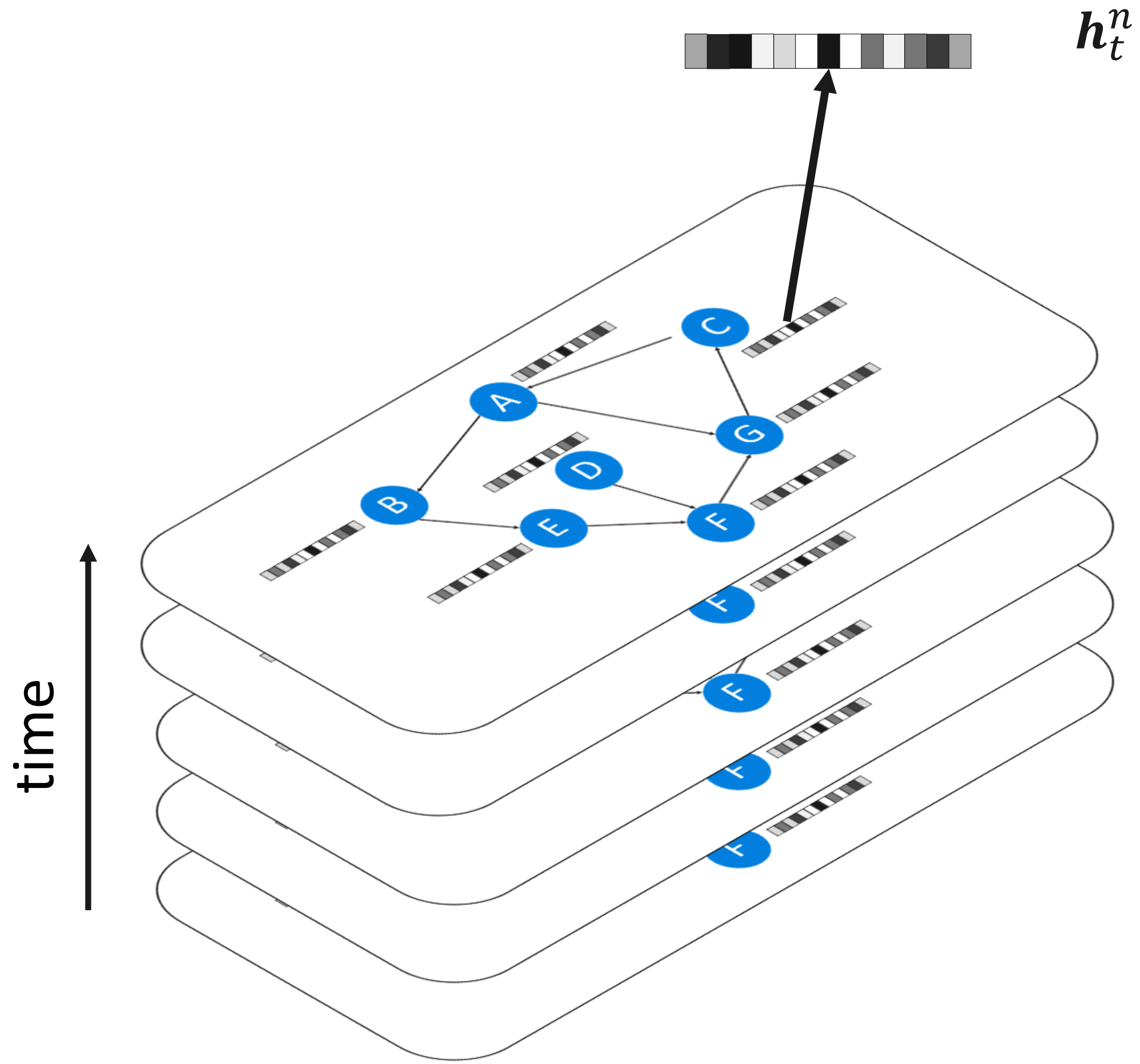
Graph Neural Networks: Output



- node selection
- node classification
- graph classification

https://github.com/microsoft/ptgnn/
https://github.com/microsoft/tf2-gnn/

Example: Node [Binary] Classification

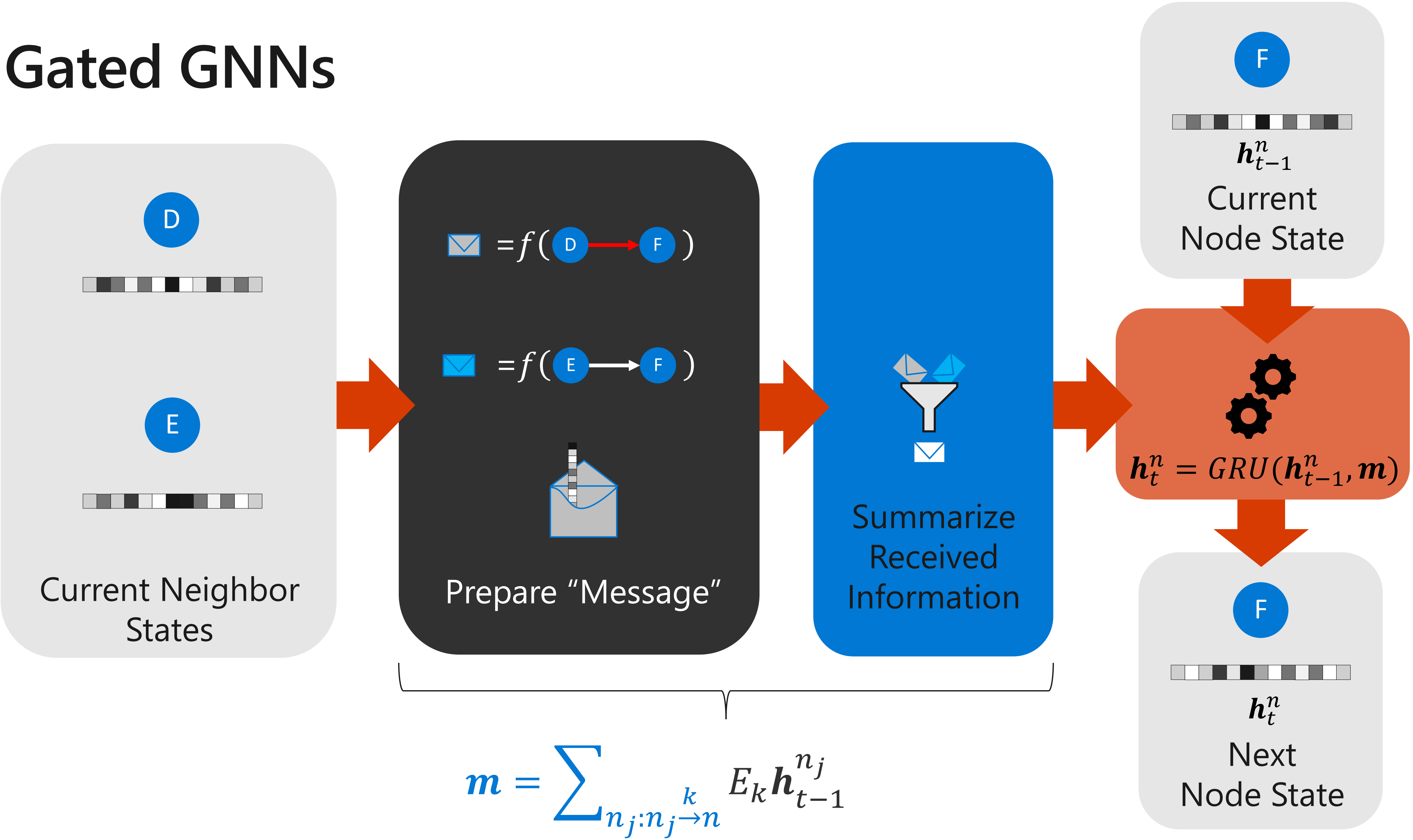


$$x_n = \sigma\left(\boldsymbol{w}^T \boldsymbol{h}_t^n + b\right)$$

Binary cross entropy

$$\mathcal{L}(x_n, y_n) = y_n \cdot \log x_n + (1 - y_n) \log(1 - x_n)$$

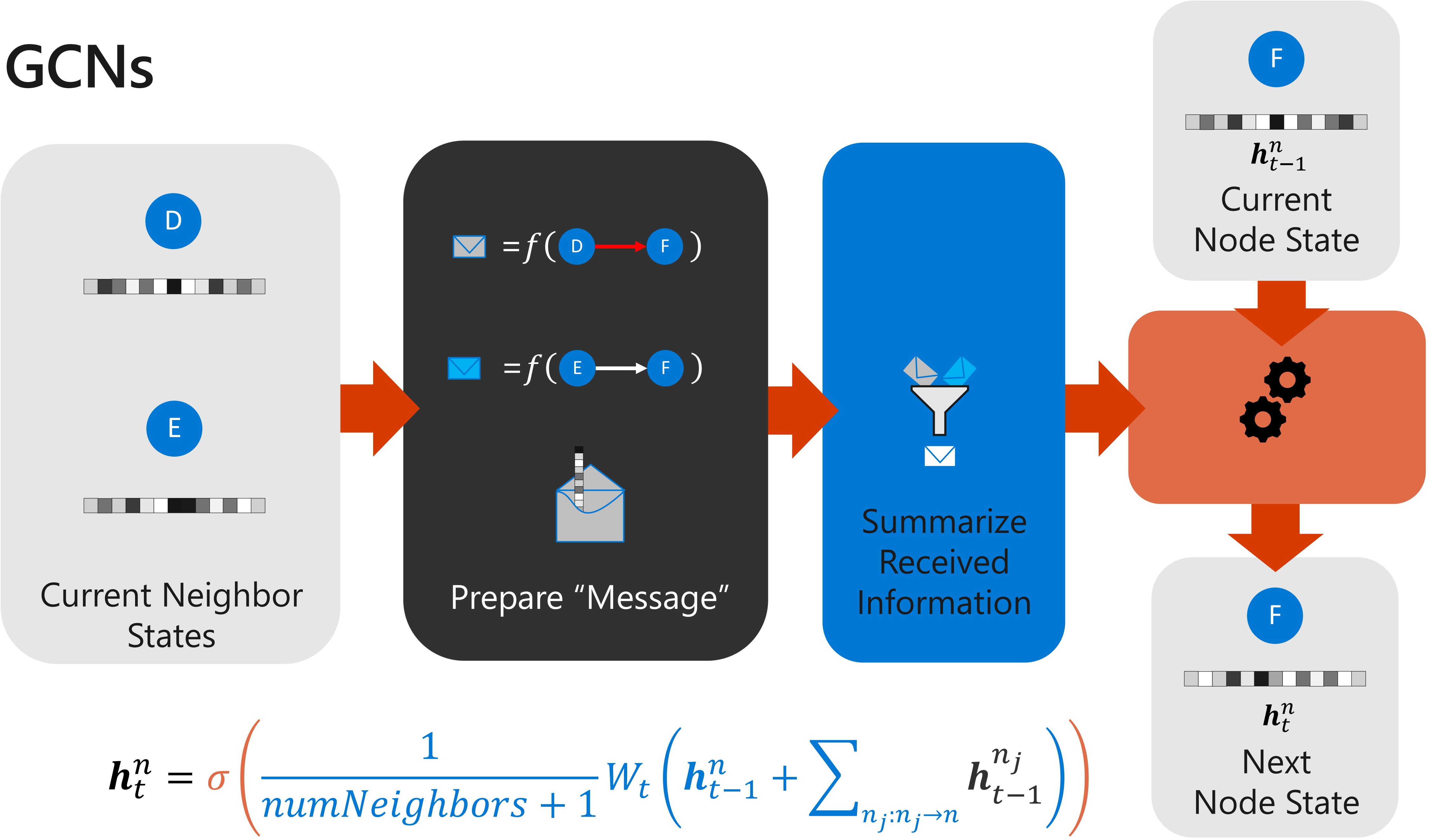
Gated GNNs



Li et al (2015). Gated graph sequence neural networks.

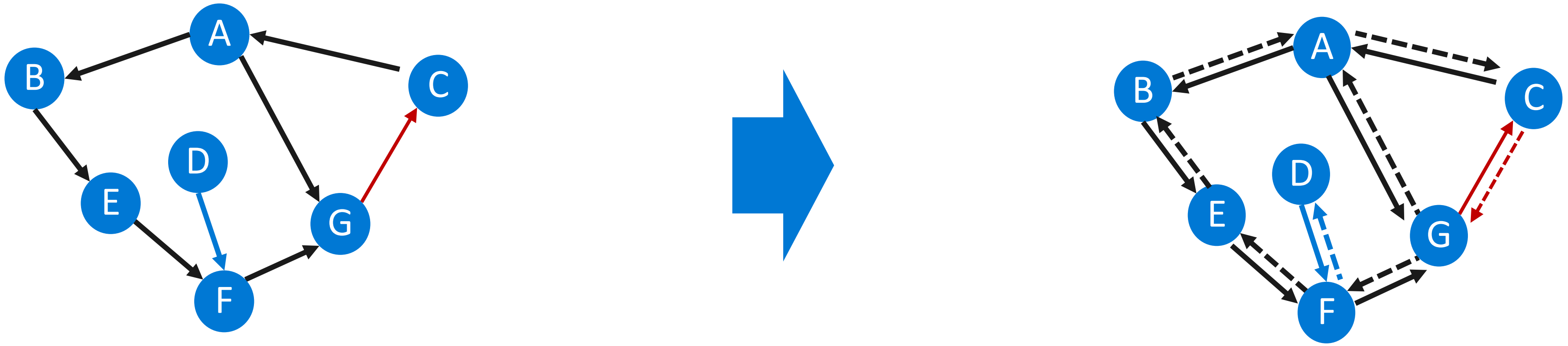
Kipf et al (2016). Semi-supervised classification with graph convolutional networks.

GCNs

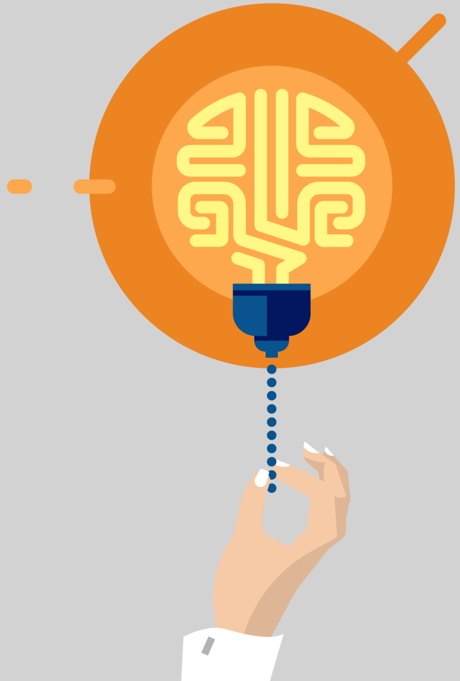


$$\boldsymbol{h}_t^n = \sigma\left(\frac{1}{numNeighbors + 1} W_t \left(\boldsymbol{h}_{t-1}^n + \sum\nolimits_{n_j : n_j \rightarrow n} \boldsymbol{h}_{t-1}^{n_j}\right)\right)$$

Trick 1: Backwards Edges

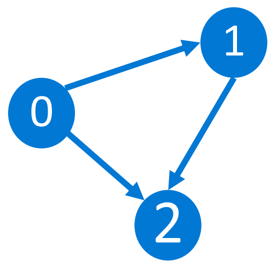


Expressing GGNNs as Matrix Operations



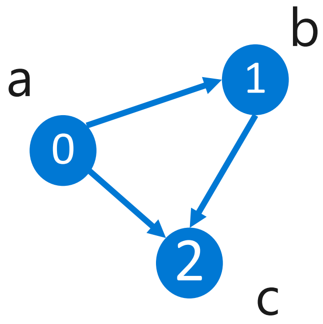
Graph Notation (2) — Adjacency Matrices

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$



Graph Notation (2) — Adjacency Matrices

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \ \mathrm{N} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

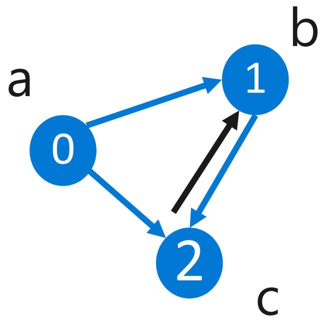


$$A \cdot N = \begin{bmatrix} 0 \\ a \\ a + b \end{bmatrix}$$

Graph Notation (2) — Adjacency Matrices

$$A_0 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix},$$

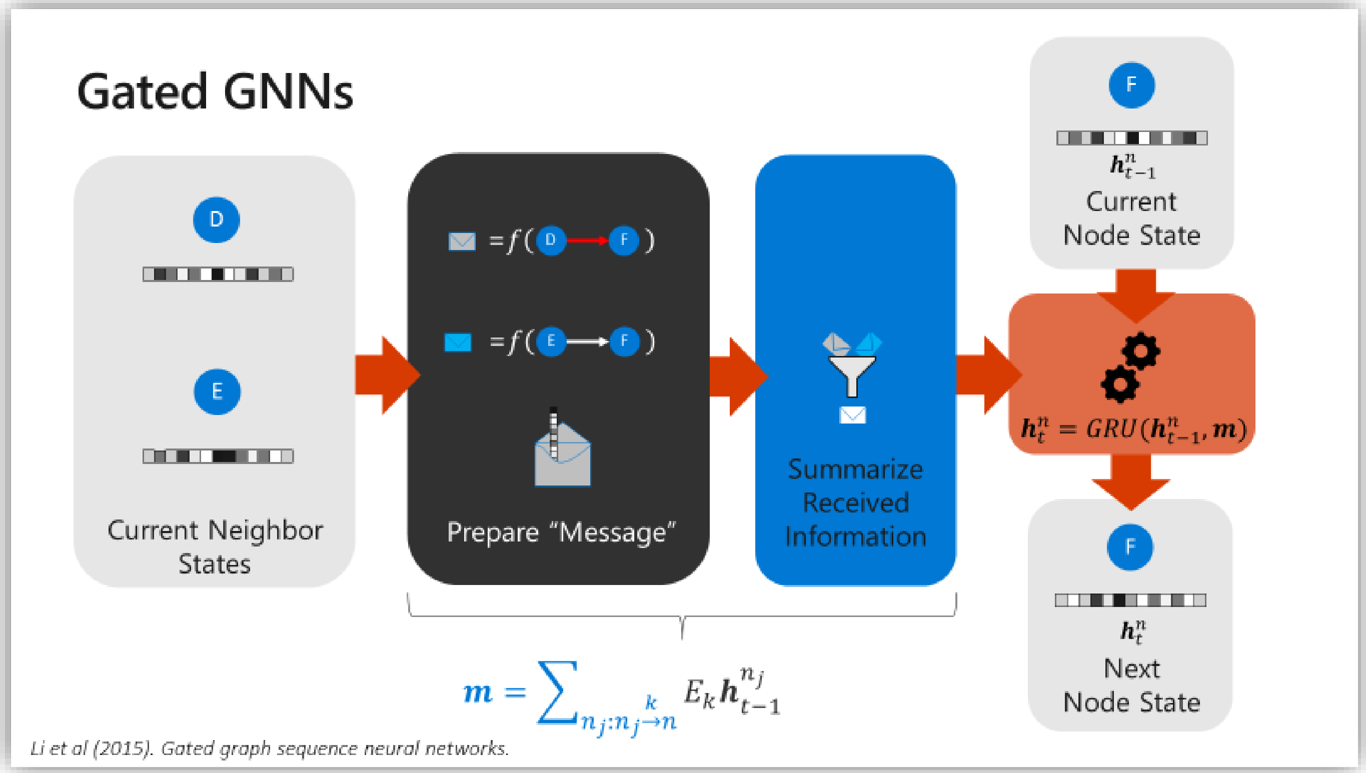
$$A_1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



GGNN as Matrix Operation

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix} \text{ (num\_nodes x D)}$$



Messages to-be sent

$$M_t^k = E_k H_t \text{ (num\_nodes x M)}$$

Received Messages

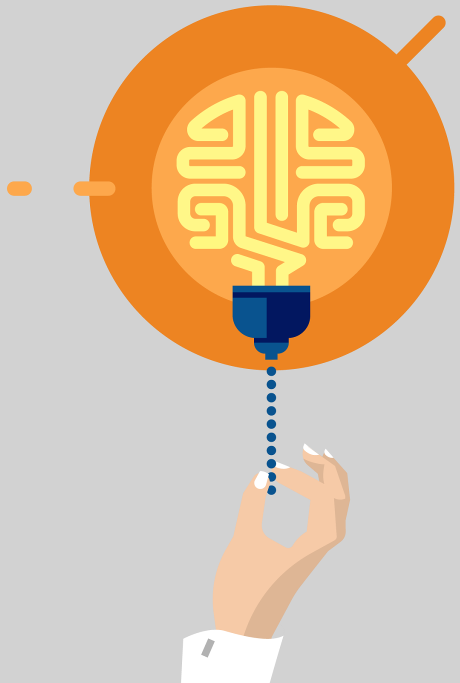
$$R_t = \sum_k A_k M_t^k \text{ (num\_nodes x M)}$$

Update $H_{t+1} = GRU(H_t, R_t)$

If we used a vanilla RNN instead

$$H_{t+1} = \sigma(\mathrm{U}H_t + WR_t)$$

Expressing GNN Matrix Operations as Code



einsum

```
C=np.einsum(‘td,qd->tq`, A, B)    #
```

$C_{t,q} = \sum_d A_{t,d} B_{q,d}$

```
D=np.einsum(‘abc,be,abq->cqe’, A, B,C)
```

\# $D_{c,q,e} = \sum_b \sum_a A_{a,b,c} B_{b,e} C_{a,b,q}$

GGNN as Pseudocode

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

  return node_states
```

N x N

GGNN as Pseudocode

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_state
     
     received_messages = zeros(num_nodes, M) # [N,
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', mes

     node_states = GRU(node_states, received_messag

  return node_states
```

Node States

$$H_t = \begin{bmatrix} \boldsymbol{h}_t^{n_0} \\ \vdots \\ \boldsymbol{h}_t^{n_K} \end{bmatrix}$$ (num_nodes x D)

Messages to-be sent

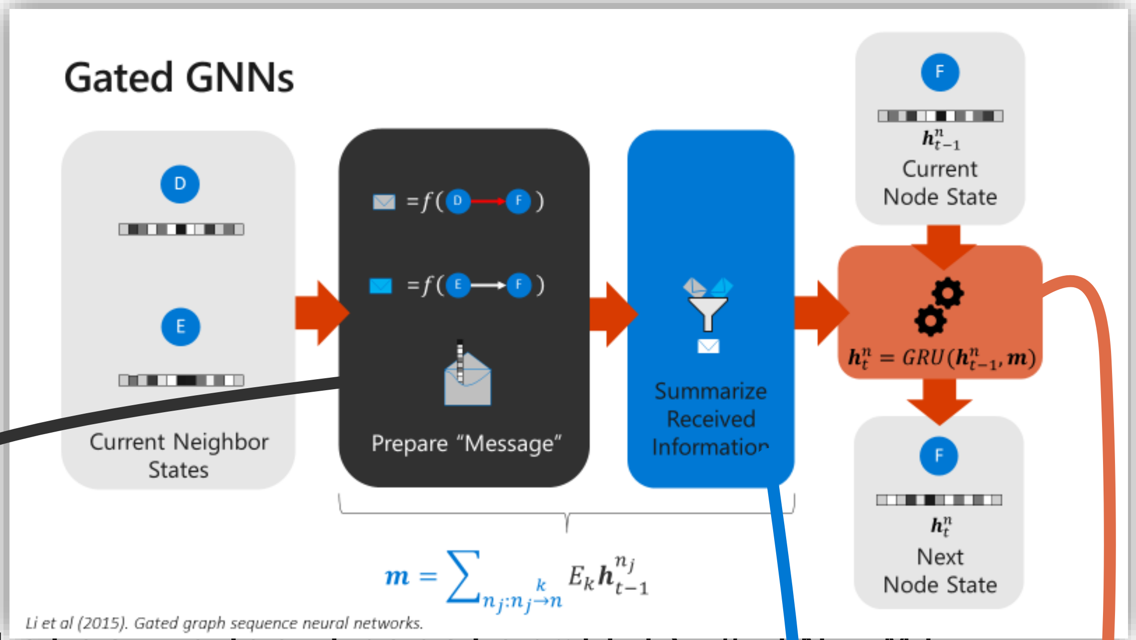
$M_t^k = E_k H_t$ (num_nodes x M)

Received Messages

$R_t = \sum_k A M_t^k$ (num_nodes x M)

Update $H_{t+1} = GRU(H_t, R_t)$

GGNN as Pseudocode




```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

  return node_states
```

N x N

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
     messages = {}
     for k in range(num_message_types):
       messages[k] = einsum('nd,dm->nm', node_states, edge_transform[k]) # [N, M]

     received_messages = zeros(num_nodes, M) # [N, M]
     for k in range(num_message_types):
       received_messages += einsum('nm,nl->lm', messages[k], adj[k])

     node_states = GRU(node_states, received_messages)

   return node_states
```

Find the parameters!

?

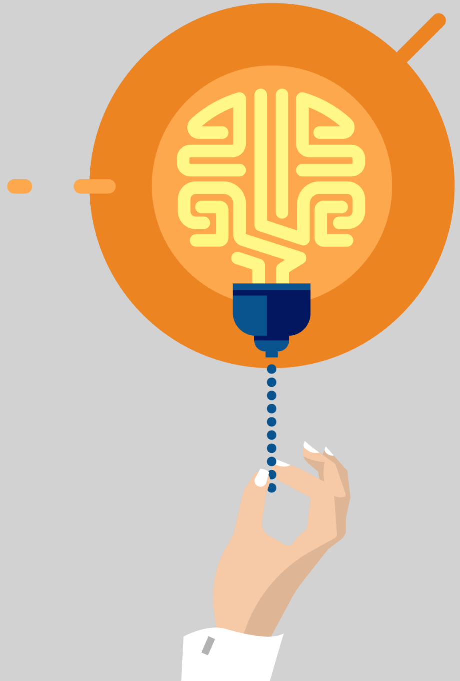
GGNN as Pseudocode: Sparsity

```
def GGNN(initial_node_states, adj, num_steps):
  node_states = initial_node_states  # [N, D]

  for i in range(num_steps):
    messages = {}
    for k in range(num_message_types):
      messages[k] = einsum(‘nd,dm->nm’, node_states, edge_transform[k]) # [N, M]

    received_messages = zeros(num_nodes, M) # [N, M]
    for k in range(num_message_types):
      received_messages += einsum(‘nm,nl->lm’, messages[k], adj[k])

    node_states = GRU(node_states, received_messages)

  return node_states
```

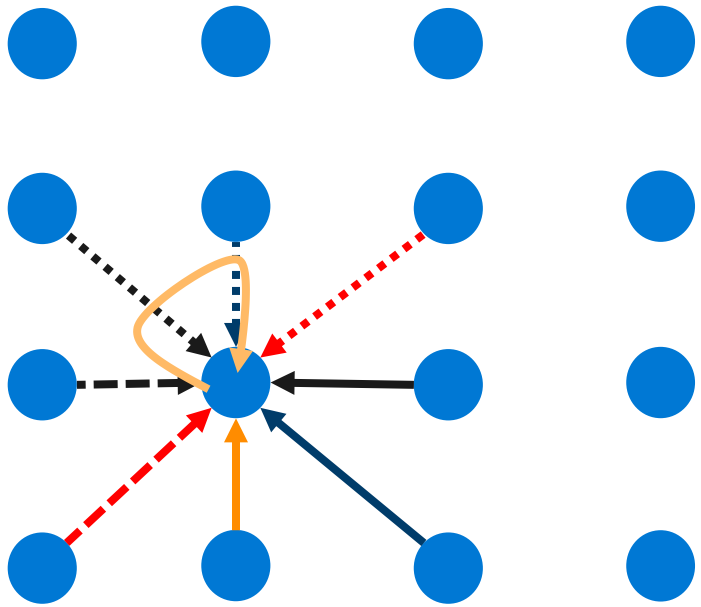
N x N

Memory

Other Models as Special Cases of GNNs



Special Case 1: Convolutions (CNN)



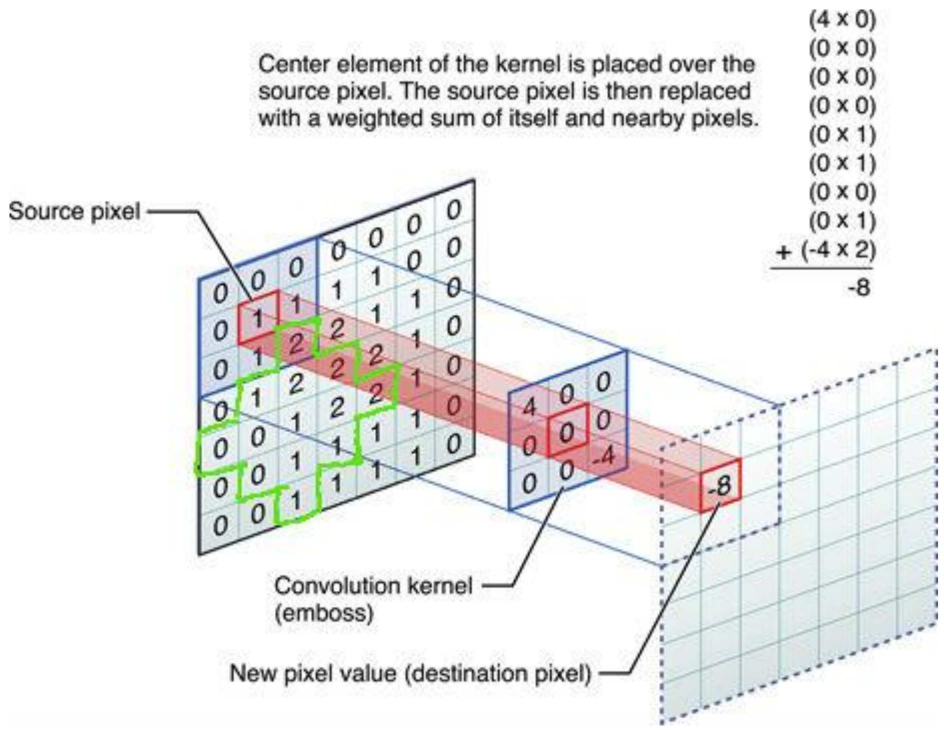


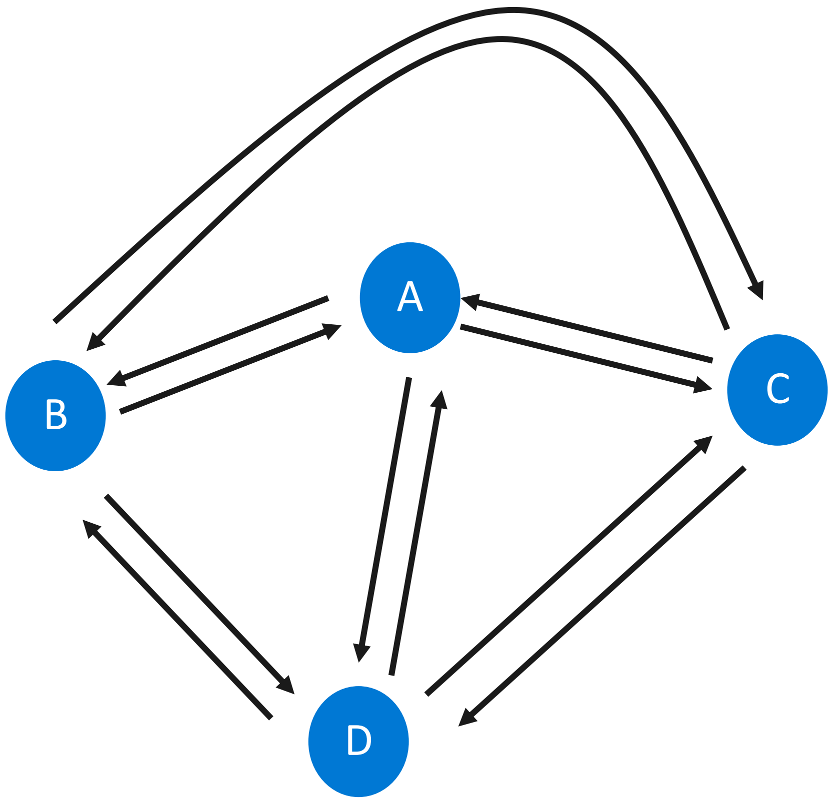
Image from: https://stats.stackexchange.com/questions/235032

Special Case 2: "Deep Sets"

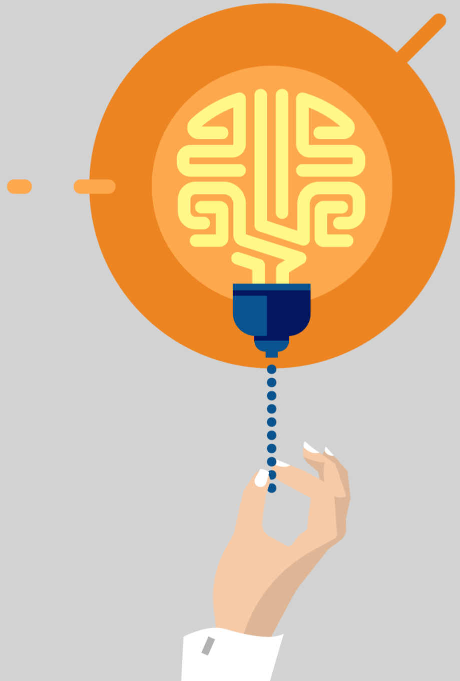
Set of Objects

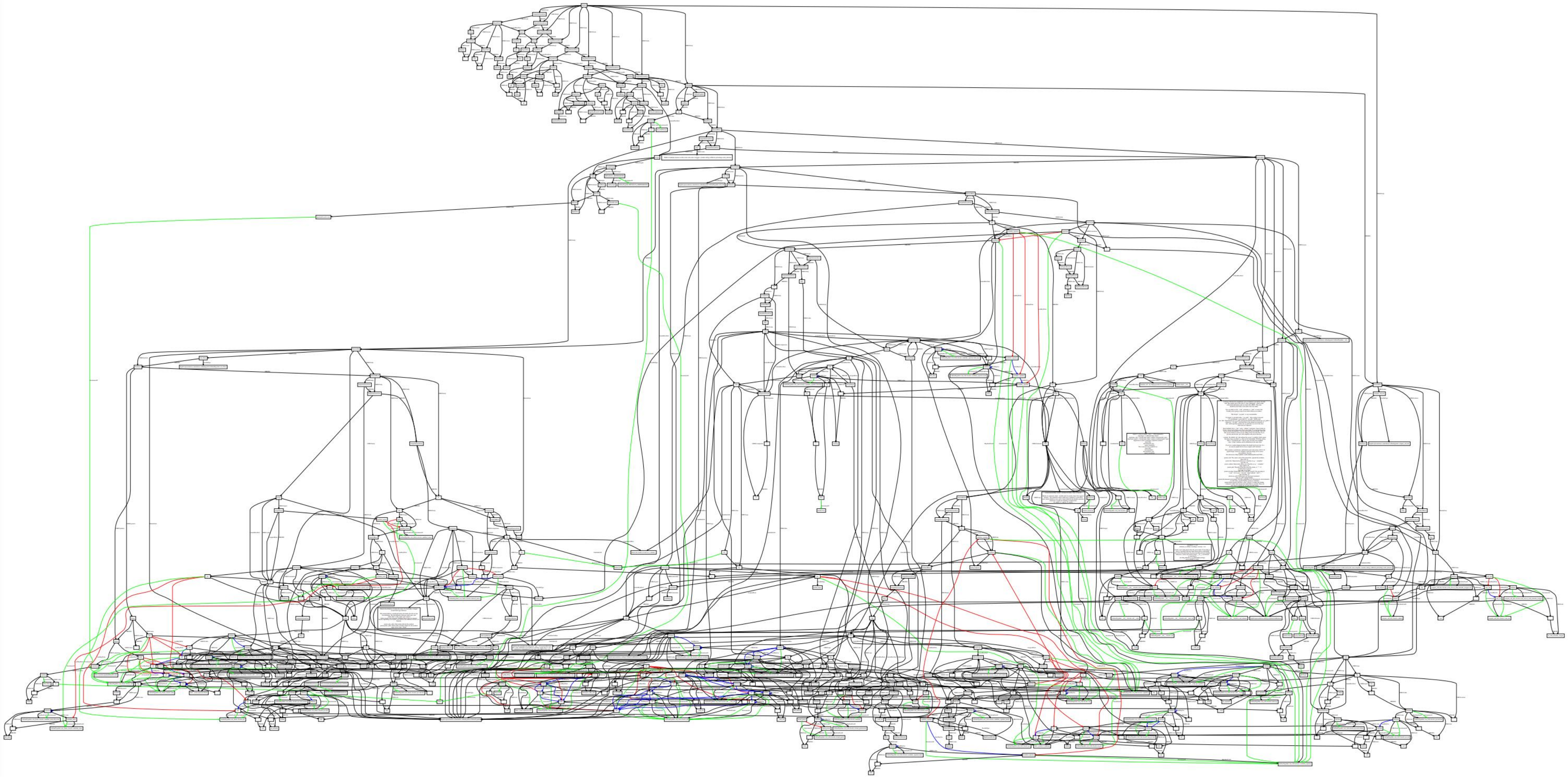
Representing a variable-sized set of objects

Special Case 2: Self-Attention (≈Transformers)



GNNs in Software Engineering

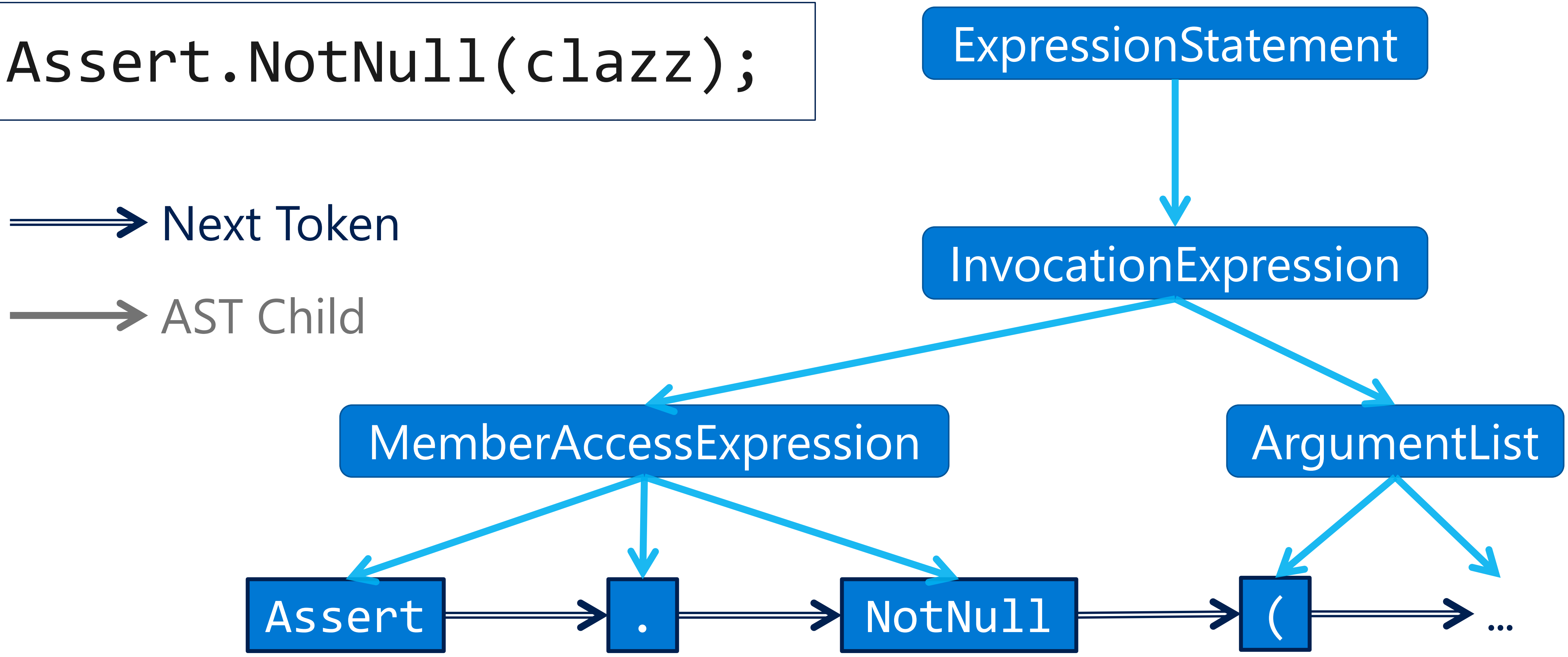




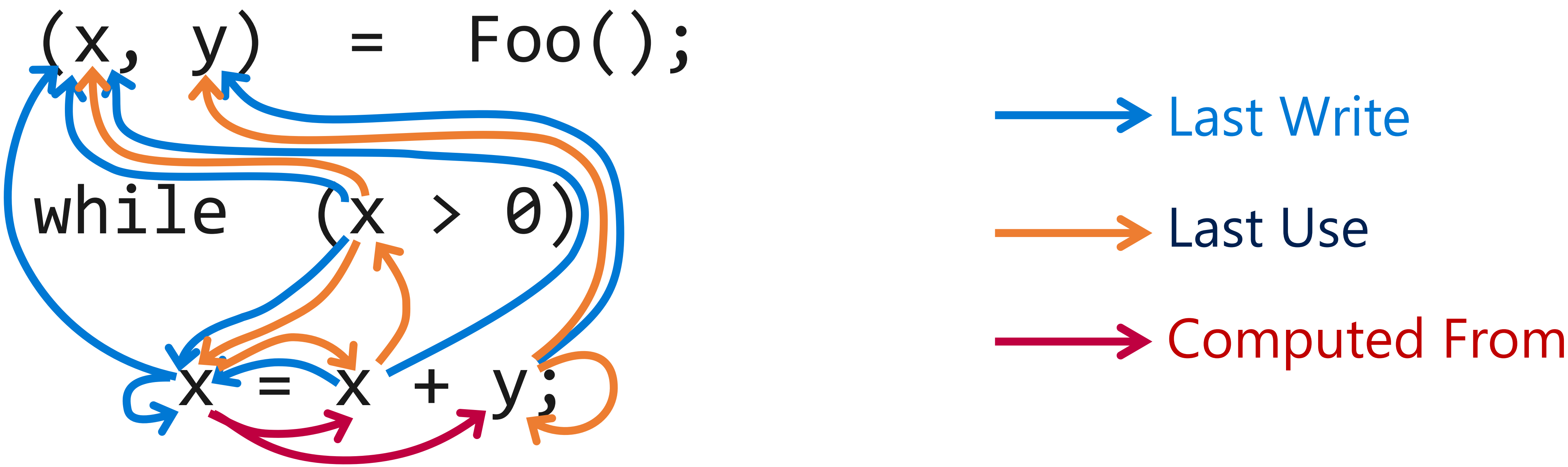
Programs are Graphs

Programs as Graphs: Syntax

```
Assert.NotNull(clazz);
```



Programs as Graphs: Data Flow



Representing Program Structure as a Graph

Additional Edge Types:

- ReturnsTo

```
int foo(int sum)
{
    …
    return x;
}
```

Representing Program Structure as a Graph

Additional Edge Types:

- ReturnsTo
- FormalArgName

```
b = foo(result);
```

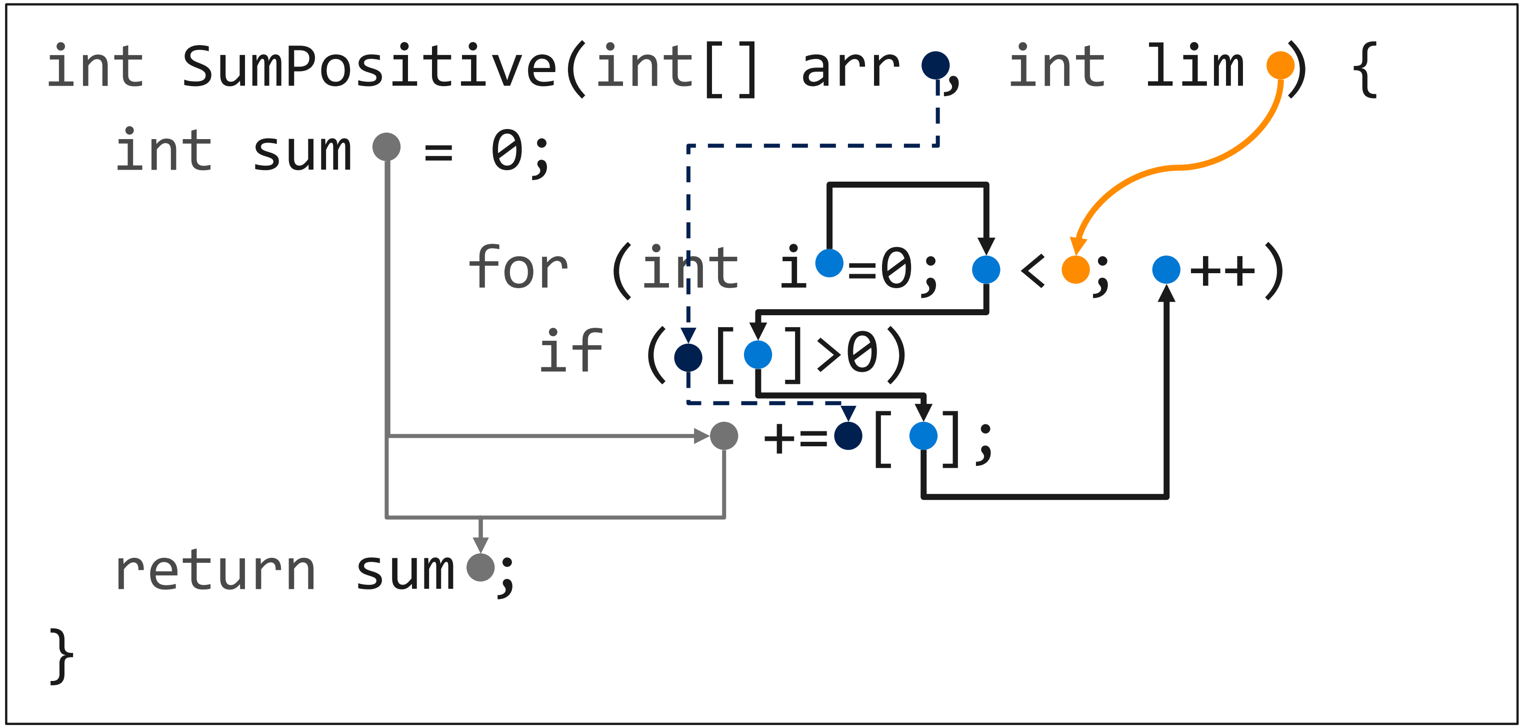
sum

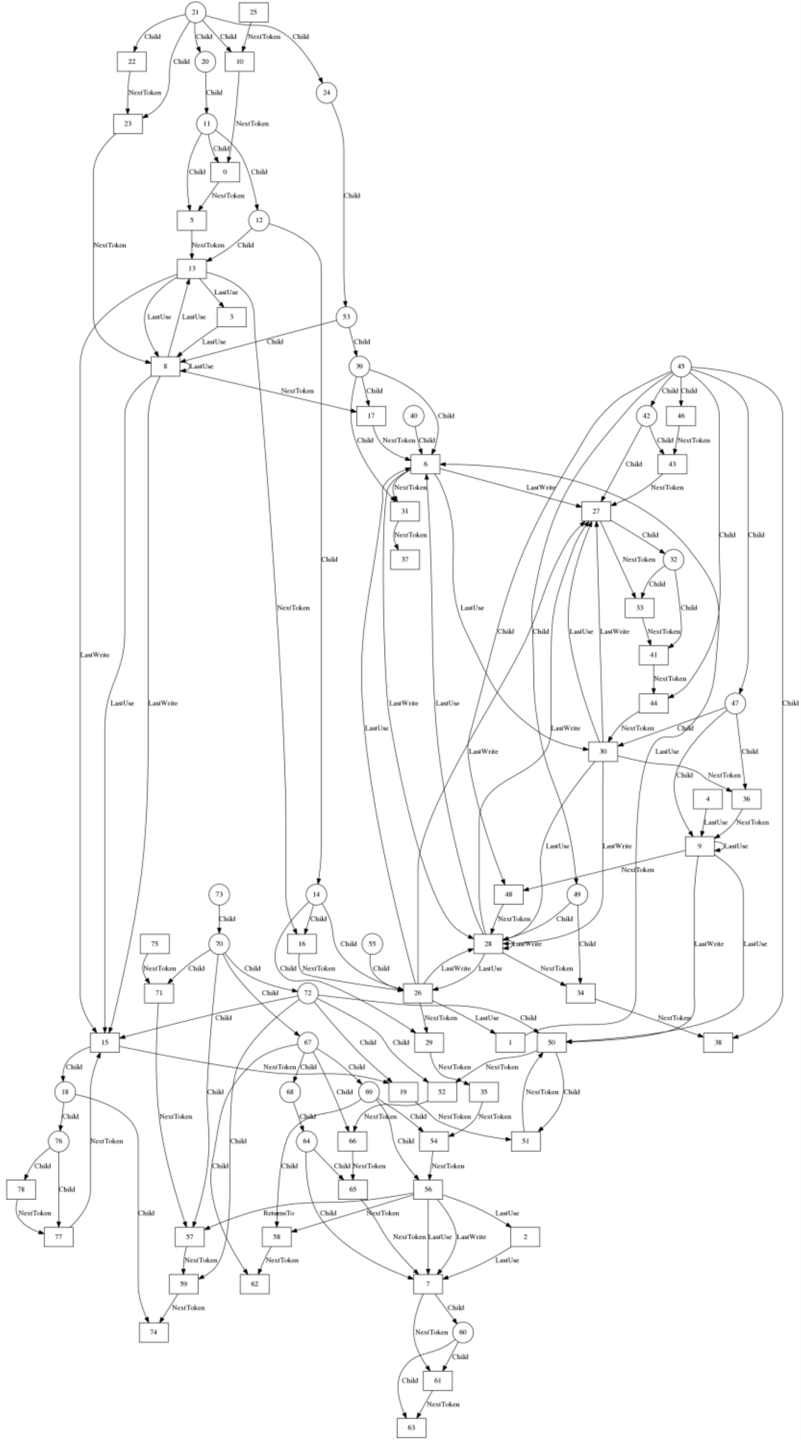
```
void foo(int sum) { … }
```

Representing Program Structure as a Graph

Entities in Programs → Nodes

Relationships among entities → Edges





Tasks Explored in Literature

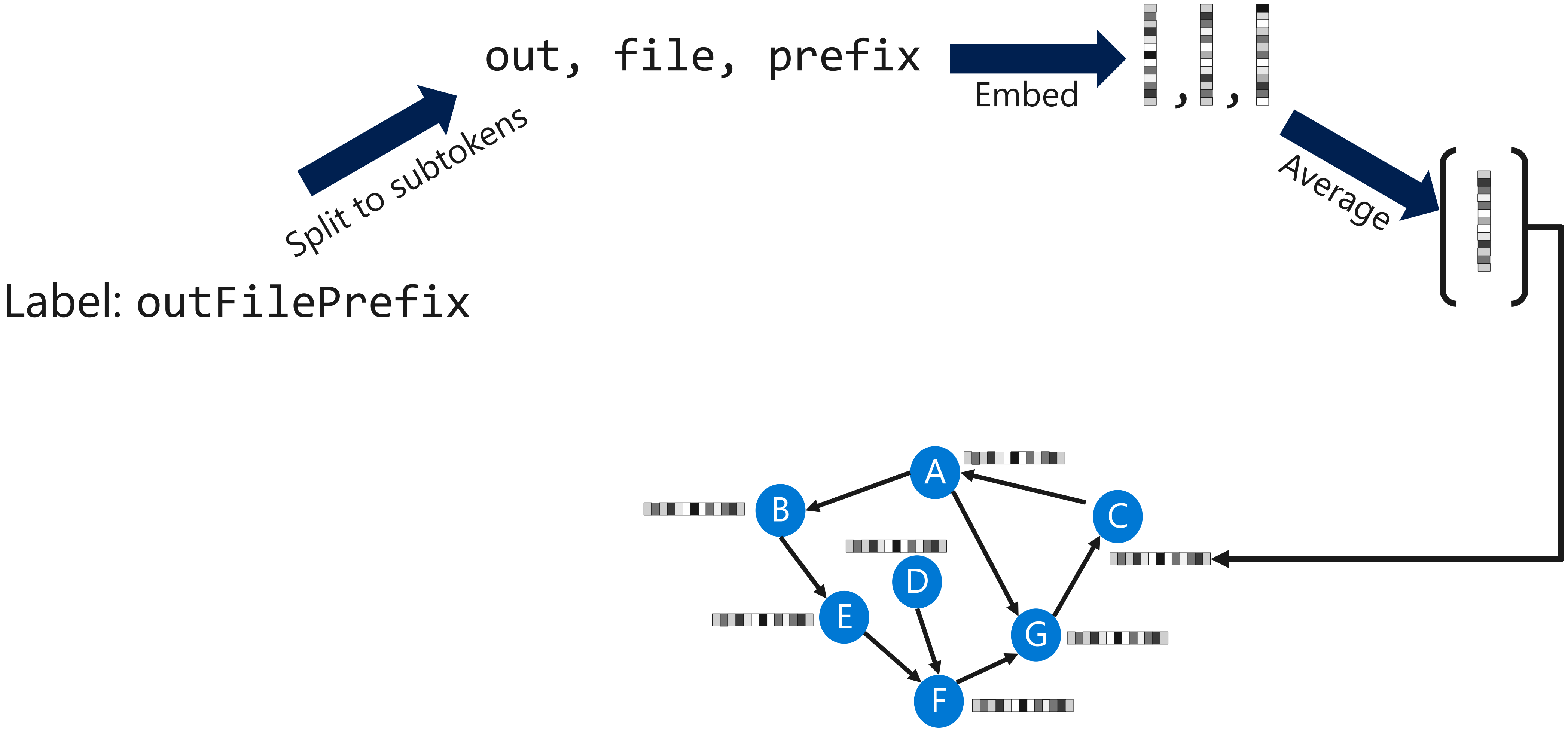
- VarMisuse Allamanis *et al.* 2018, Cvitkovic *et al.* 2019, Hellendoorn *et al.* 2020
- Method Naming Fernandes *et al.* 2019
- Code Generation or Repair Brockschmidt *et al.* 2019, Yasunaga *et al.* 2020
- Predicting Program Properties
  - Type Annotations in Dynamic Languages Wei *et al.* 2020, Allamanis *et al.* 2020, Schrouff *et al.* 2019
  - Performance Characteristics of Programs Tomczak *et al.* 2019

Variable Misuse Task

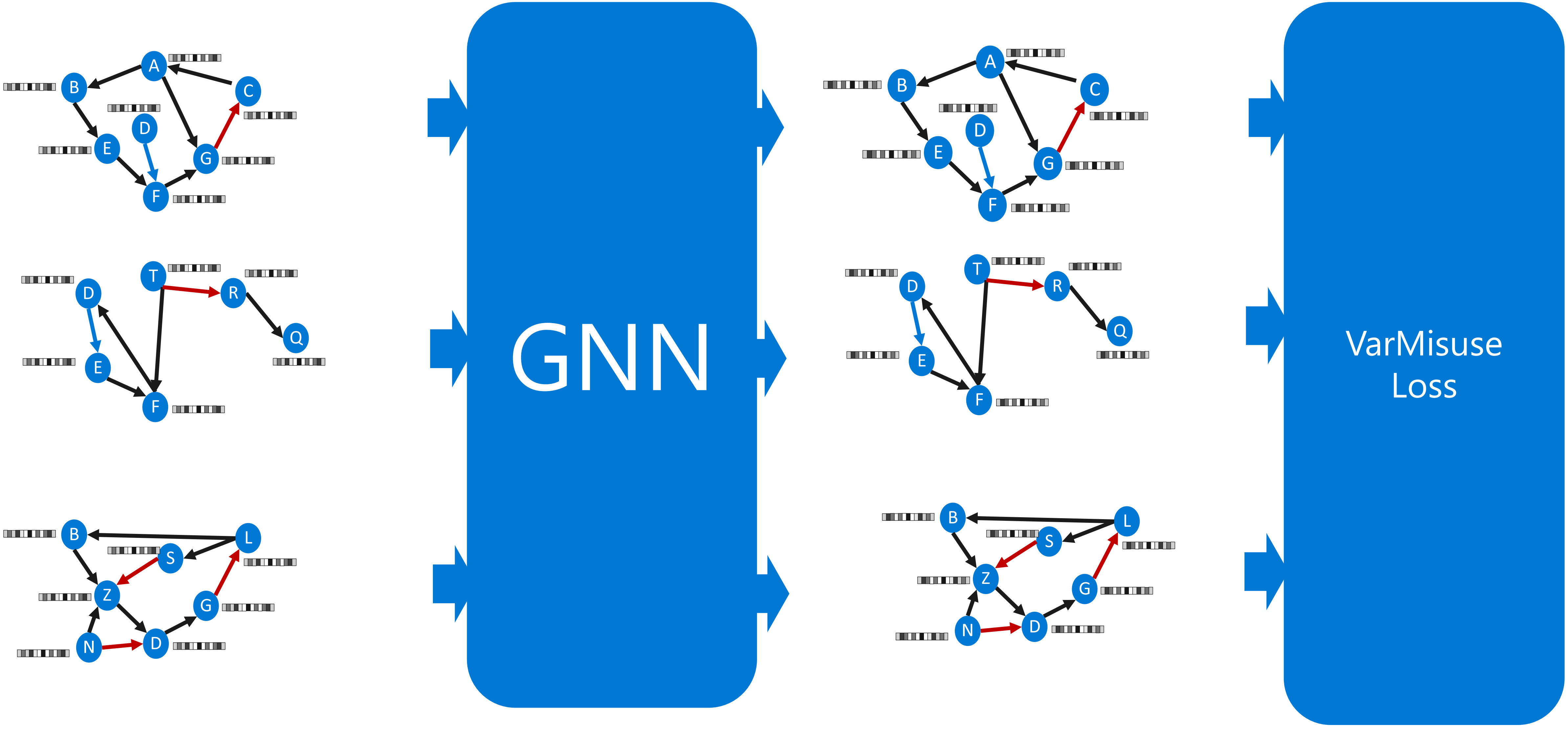
```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull( clazz );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

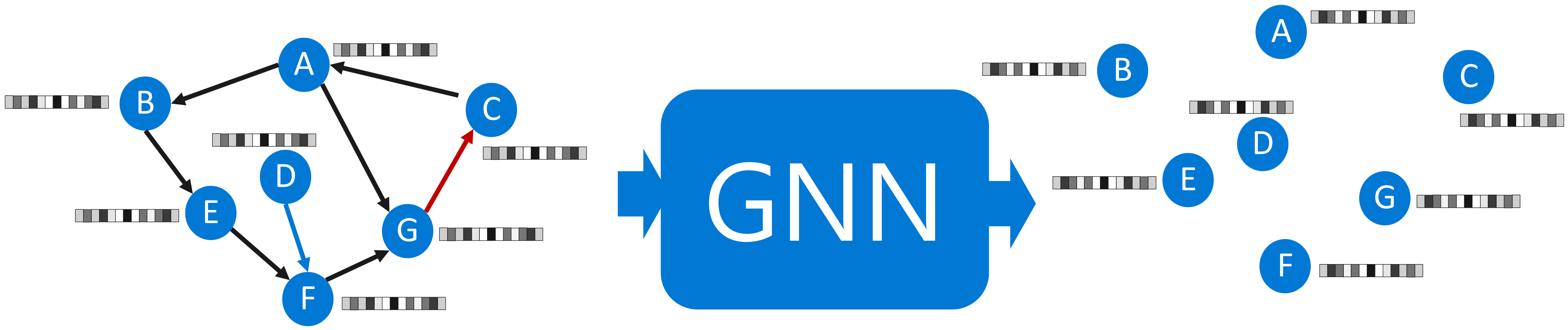
Allamanis *et al.* 2018, Cvitkovic *et al.* 2019, Hellendoorn *et al.* 2020

Initial Node Representations



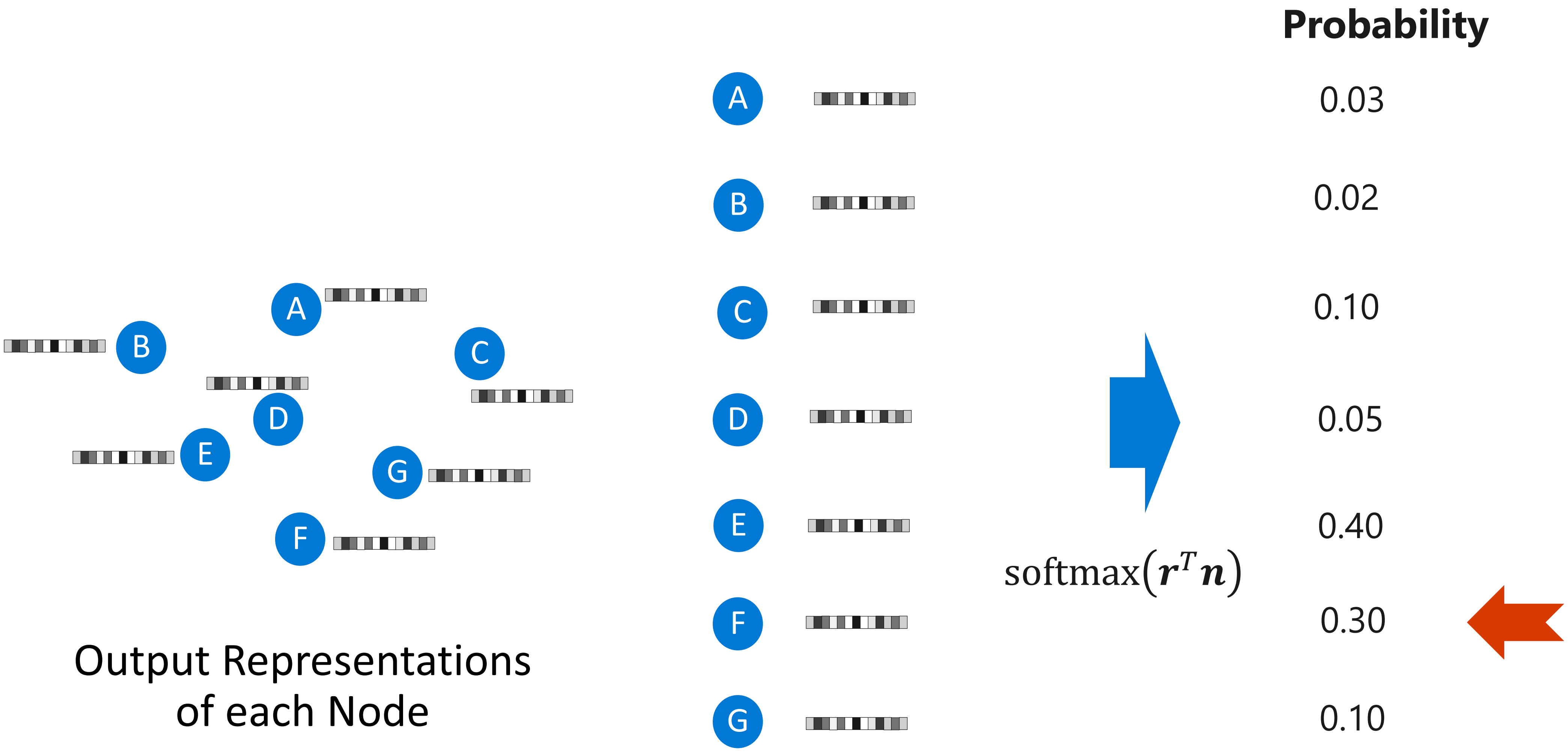
Variable Misuse



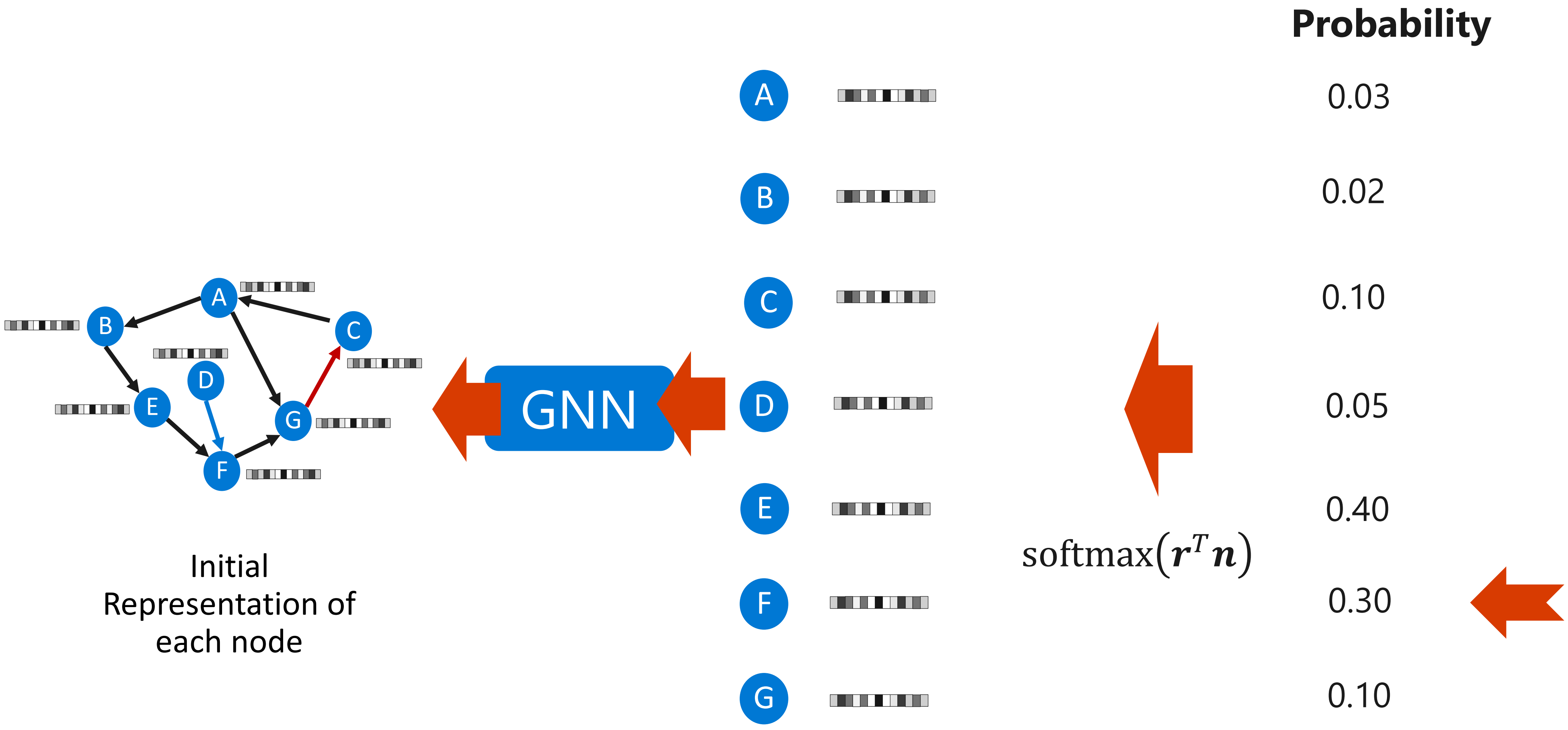


Initial Representation of each node

Output Representations of each Node

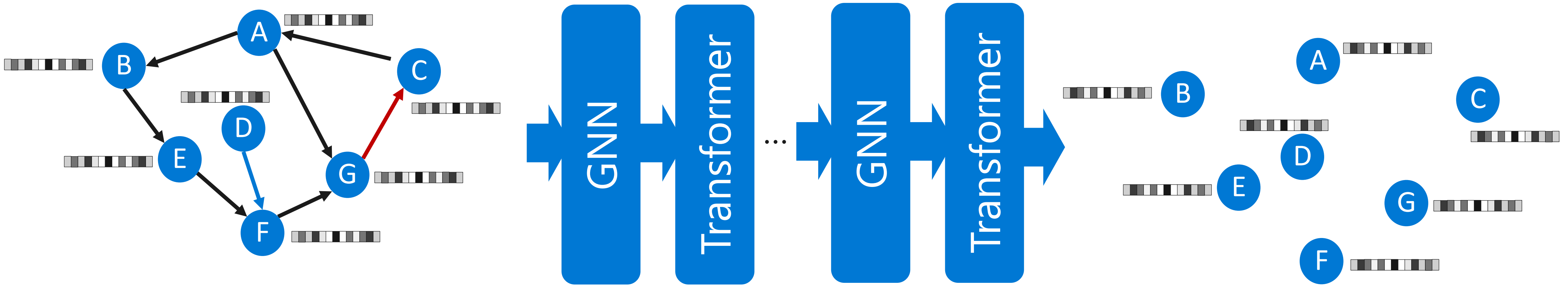


Localization objective: Pick the right node



Localization objective: Pick the right node

GNN Layers



Initial Representation
of each node

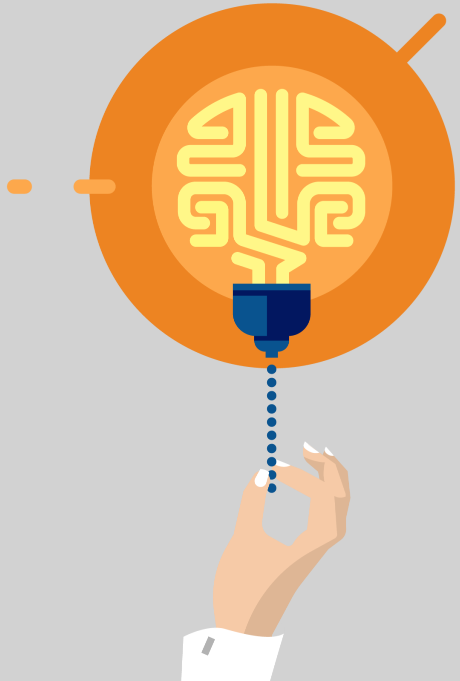
Output Representations
of each Node

Model Family	Class. Accuracy ≤ 250	Class. Accuracy ≤ 1000	Loc & Rep Accuracy ≤ 250	Loc & Rep Accuracy ≤ 1000	Parameters	Training Time
RNN[1]	71.8%	70.6%	44.4%	42.5%	4.3M	31.3h
Transformer	75.9%	73.2%	67.7%	63.0%	3.7M	41.5h
GGNN	81.4%	79.2%	64.0%	60.9%	5.5M	241h
RNN Sandwich	**82.5%**	**81.9%**	75.8%	**73.8%**	12.6M	109h
Transformer Sandwich	81.1%	78.1%	74.5%	71.4%	10M	161h
GREAT	80.1%	76.9%	**76.4%**	73.1%	7.9M	120h

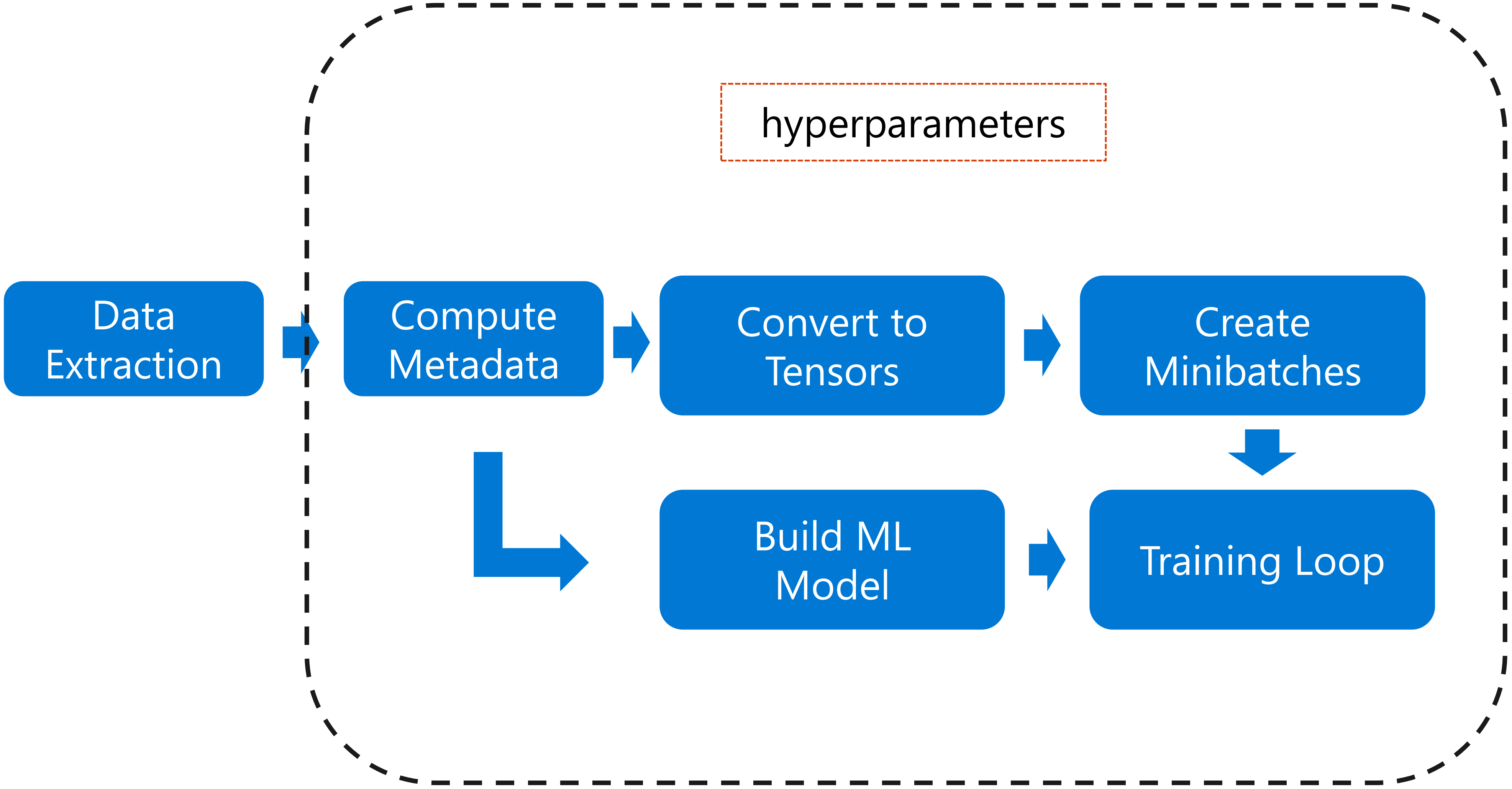
1: a stronger version of the model proposed in Vasic et al. (2019) (previous SOTA).

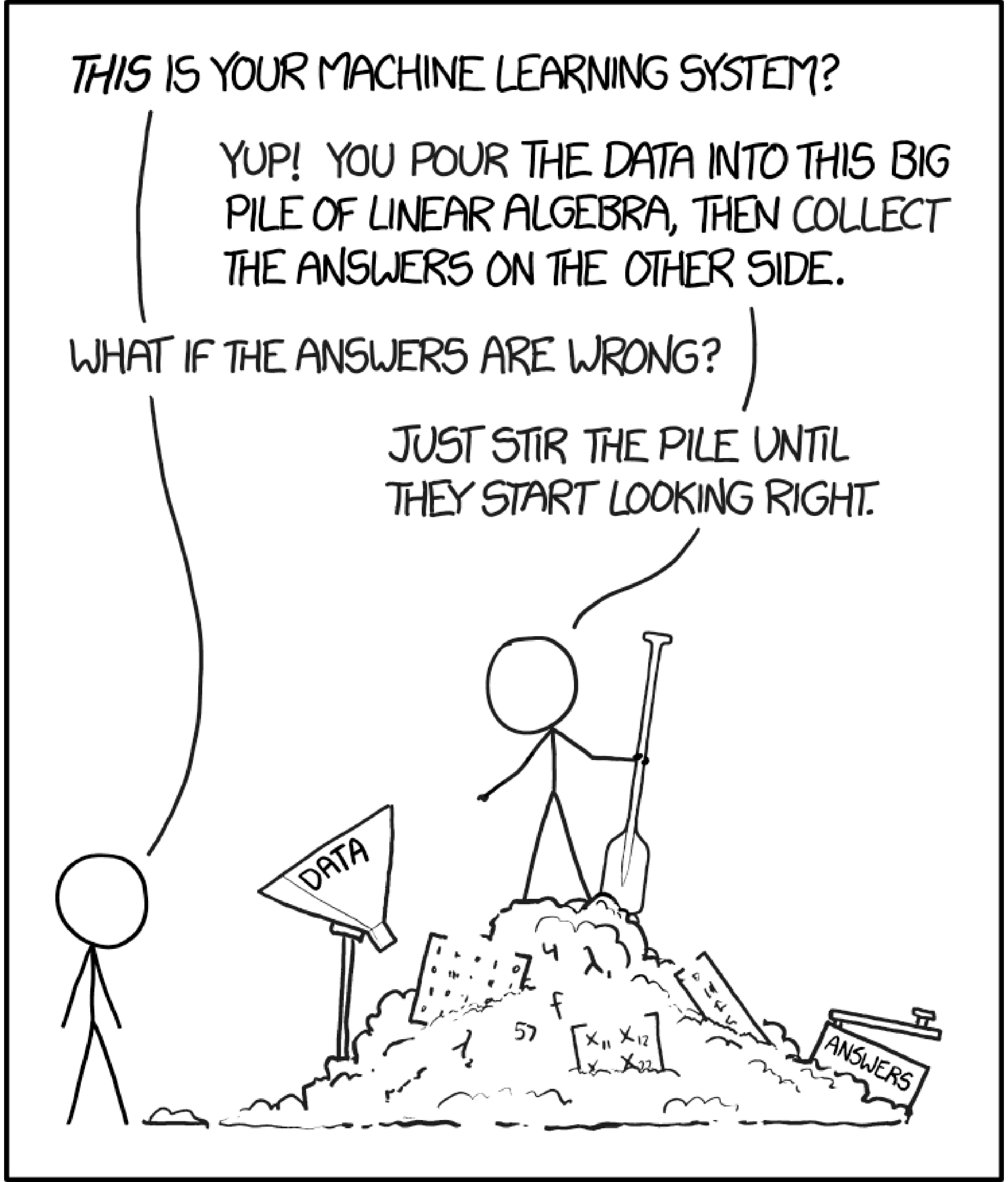
Hellendoorn *et al.* 2020

Machine Learning in Practice



Common Architecture of Deep Learning Code





https://xkcd.com/1838/

Practical(?) Tips on Debugging Machine Learning Models

Model Capacity (*what can the model learn?*)

- Overtrain on a small dataset
- Synthetic data

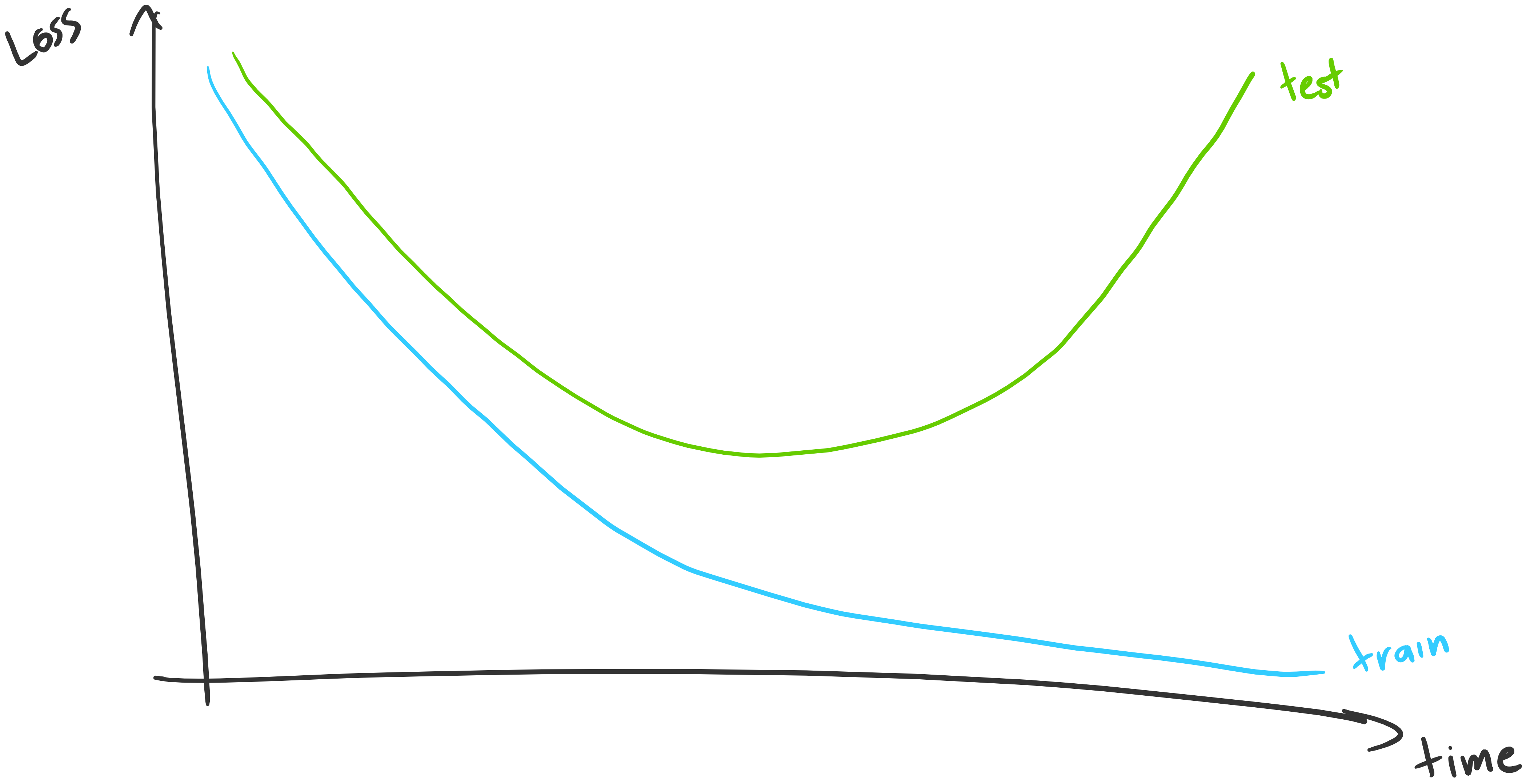
Optimization Issues (*can we make the model learn?*)

- Look at learning curves
- Monitor gradient update ratios
- Hand-pick parameters for synthetic data

Other model "bugs" (*is the model doing what I want it to do?*)

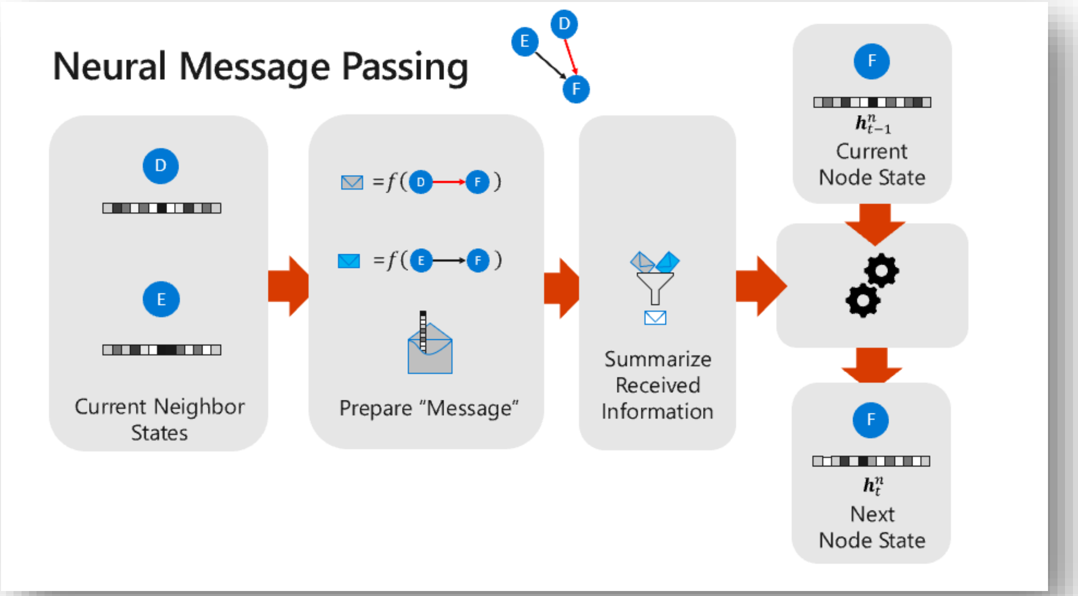
- Generate samples from your model (if you can)
- Visualize learned representations (*e.g.* embeddings, nearest neighbors)
- Error analysis (examples where the model is failing, most "confident" errors)
- Simplify the problem/model
- Increase capacity, sweep hyperparameters (*e.g.* increase size of **h** in LSTM)

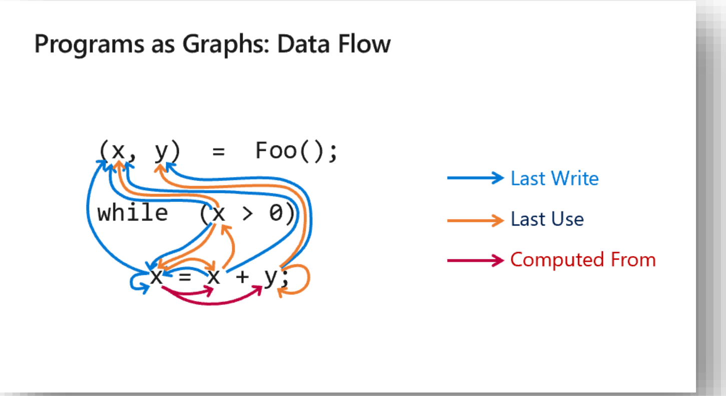
See also: https://youtu.be/oMB24_ao05A

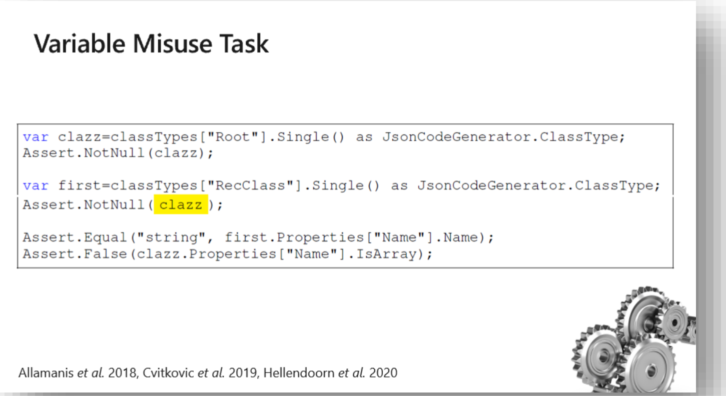


Learning Curve

Graph Neural Networks in SE Research







@miltos1

miltos.allamanis.com

Microsoft