A Graph placement methodology for fast chip design

Azalia Mirhoseini1,4 ✉, Anna Goldie1,3,4 ✉, Mustafa Yazgan2, Joe Wenjie Jiang1, Ebrahim Songhori1, Shen Wang1, Young-Joon Lee2, Eric Johnson1, Omkar Pathak2, Azade Nazi1, Jiwoo Pak2, Andy Tong2, Kavya Srinivasa2, William Hang3, Emre Tuncer2, Quoc V. Le1, James Laudon1, Richard Ho2, Roger Carpenter2 & Jeff Dean1

Contributions

- Deep reinforcement learning method that outperforms/matches human expert performance on chip floorplanning
- Generates placements in under 6 hours, whereas human-expert baselines take weeks or months at a high operation and opportunity cost
- Superhuman chip floorplans generated by this method were used in Google’s latest AI accelerator (TPU)!

한줄 요약 : 인간보다 더 잘 배치한다 (성능), 인간보다 빠르다(성능) + 새로운 칩에서도 배치가 가능하다(일반화)

반도체 생태계


Virtual Fab 개념도
설계기업
(Fabless)
반도체 설계기업
IP 설계 기업
로직
Design House
① MPW 프로그램 운영
② 소량 생산의뢰
③ 설계지원
④ 디자인서비스
도면
위탁 생산
(Foundry)
SAMSUNG
삼성전자
SK 하이닉스
동부하이텍
MagnaChip

로직과 도면

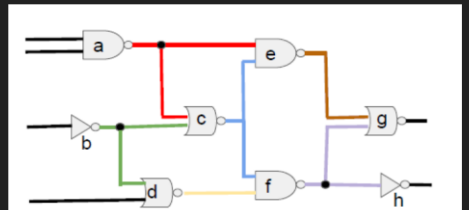
```
module c17 (N1,N2,N3,N6,N7,N22,N23);
input N1,N2,N3,N6,N7;
output N22,N23;
wire N10,N11,N16,N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

로직

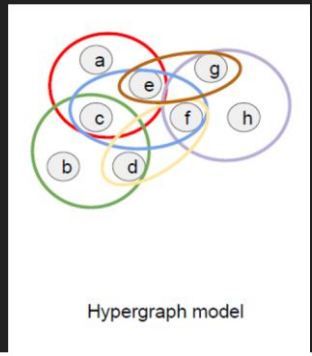
도면

디자인 하우스 (Using chip design model)

```
module c17 (N1,N2,N3,N6,N7,N22,N23);
input N1,N2,N3,N6,N7;
output N22,N23;
wire N10,N11,N16,N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

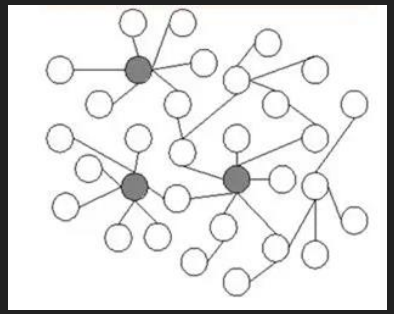


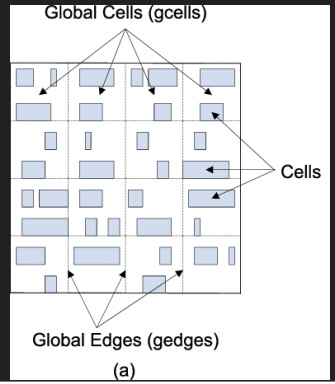
Gate-level circuit

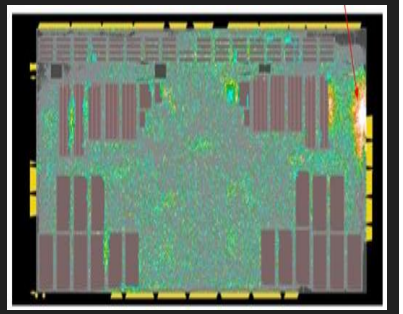


Hypergraph model

GNN+Multi armed bandit



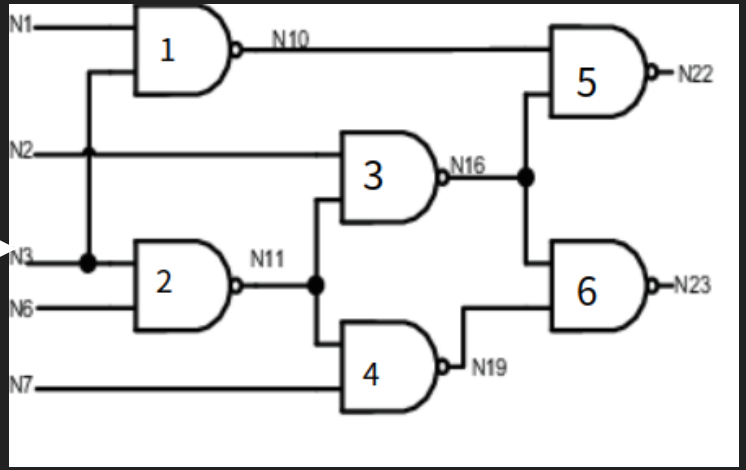




로직과 도면

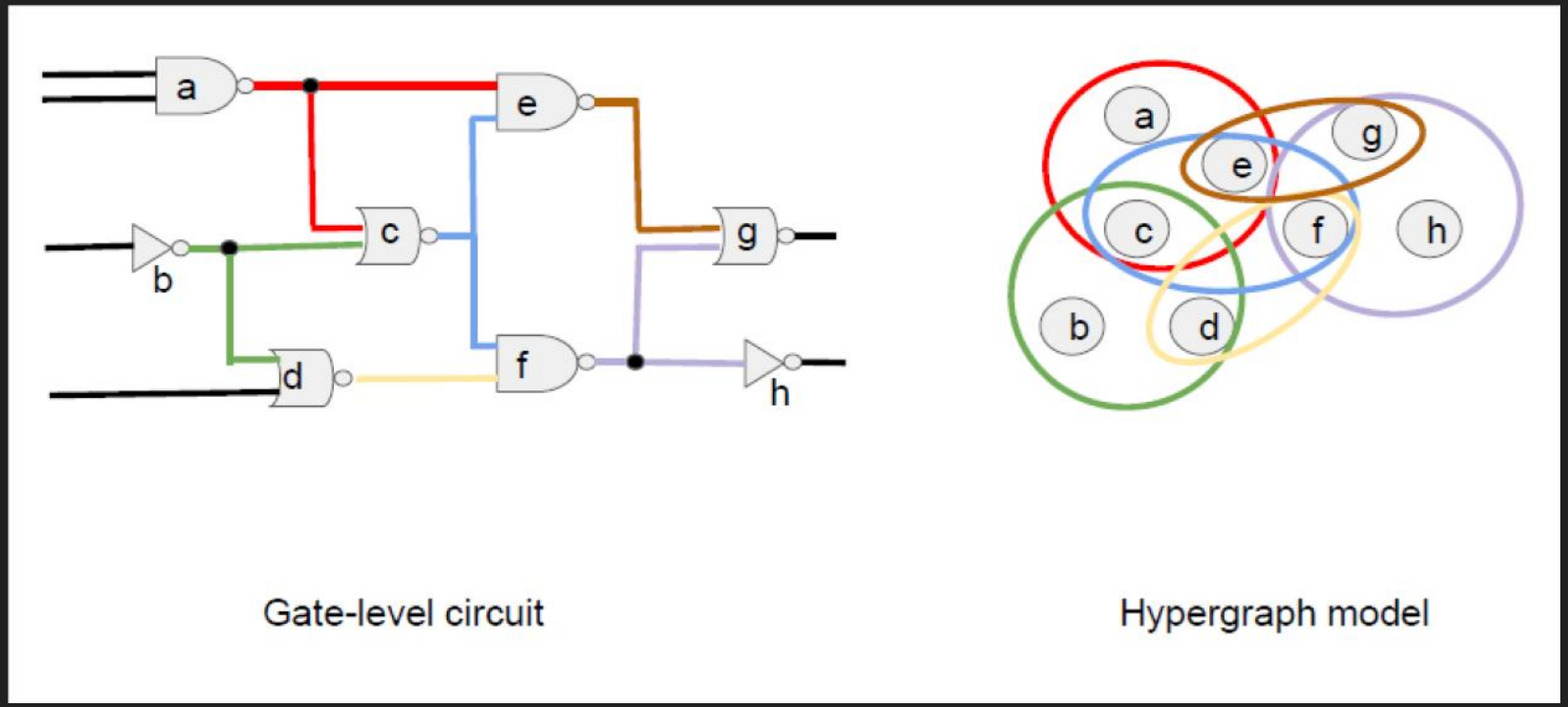
```
module c17 (N1,N2,N3,N6,N7,N22,N23);
input N1,N2,N3,N6,N7;
output N22,N23;
wire N10,N11,N16,N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

로직



회로도

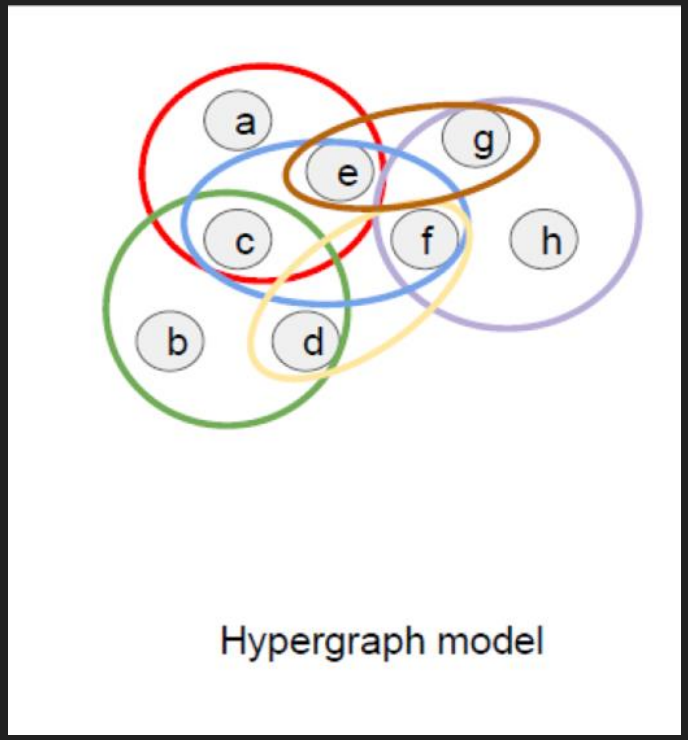
하이퍼 그래프 (generalization of graph)



netlist: ((a,c,e), (e,g), (b,c,d), (c,e,f), (d,f), (f,g,h))

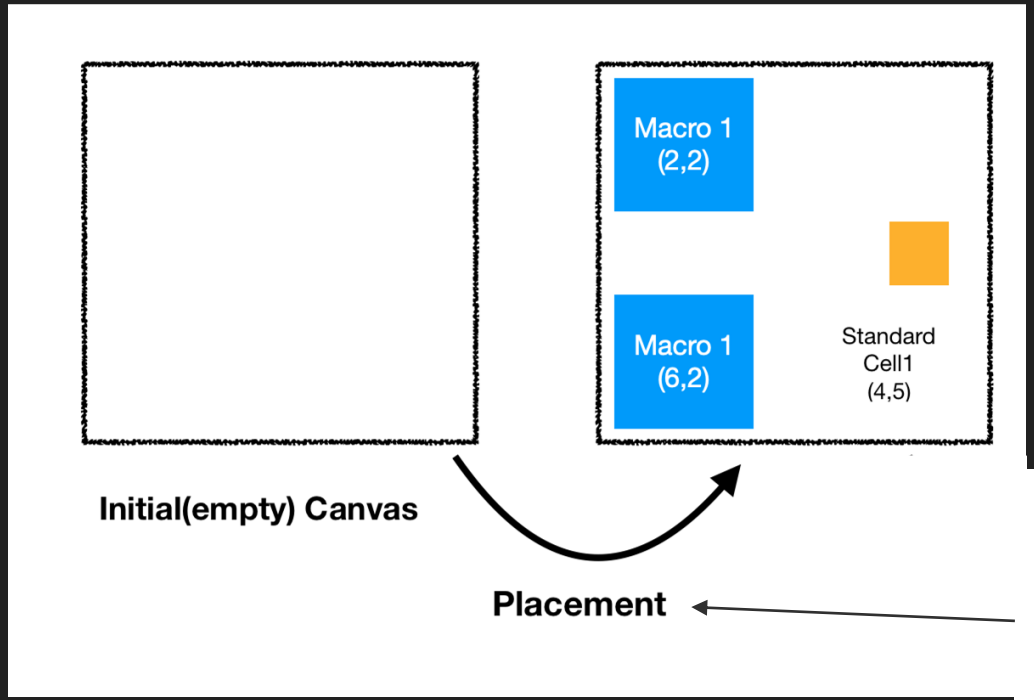
하이퍼 그래프 -> 그래프

Register-distance?

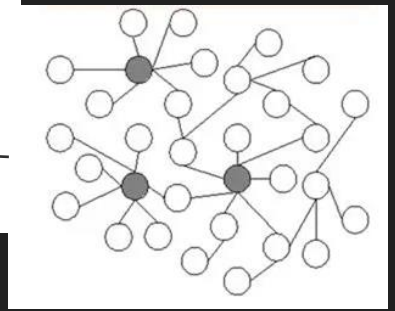


	a	b	c	d	e	f	g	h
a			1/4		1/4			
b			1/4	1/4				
c					1/2	1/4		
d						1/2		
e						1/4		
f							1/4	1/4
g								1/4
h								

최종 문제 - Graph Placement

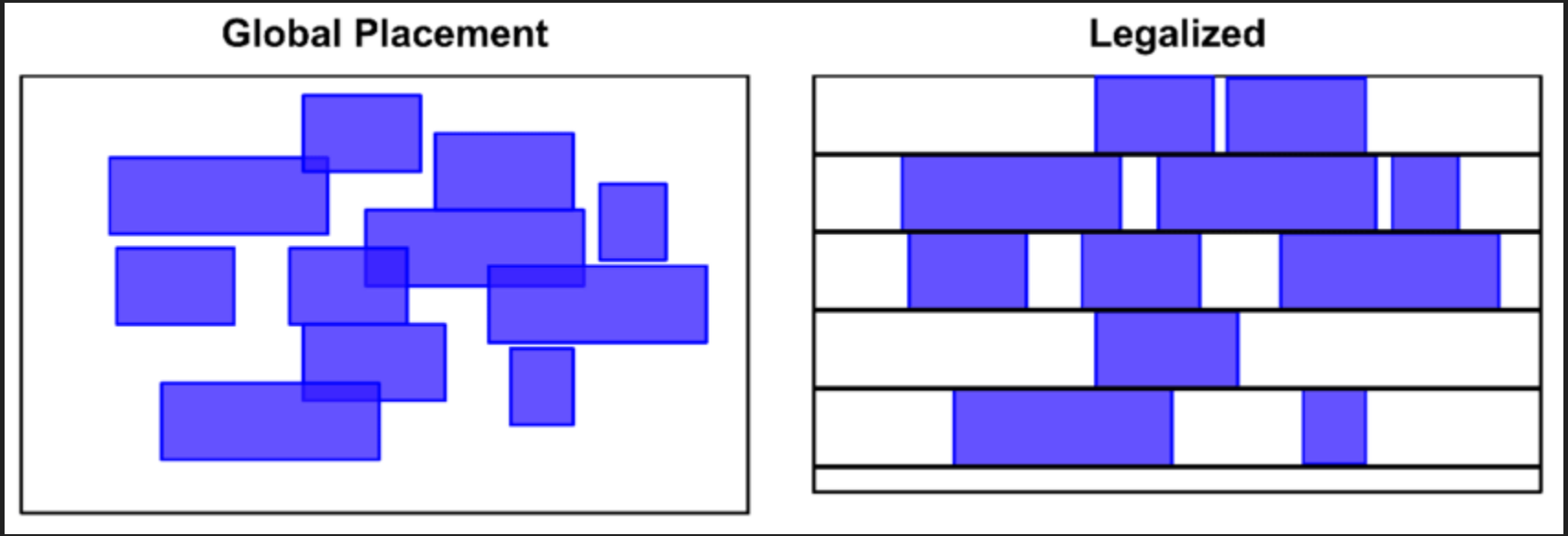


그래프 정보를 캔버스에 배치



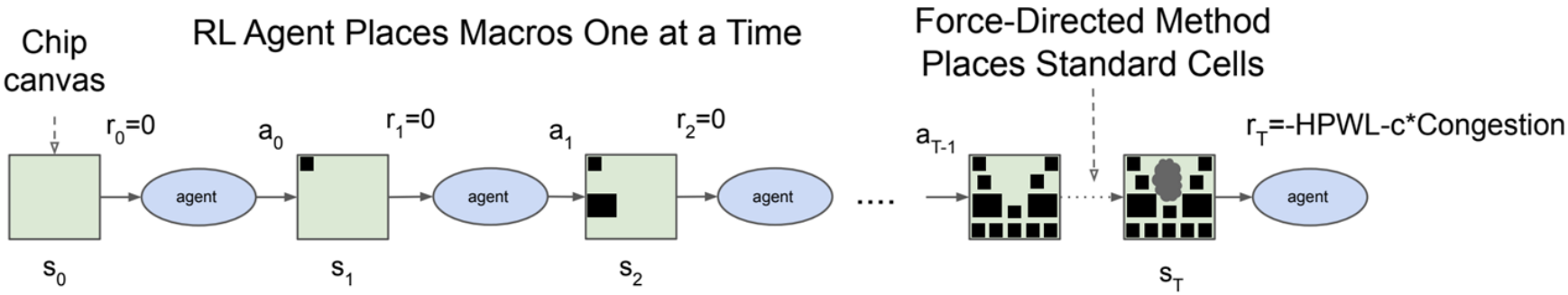
Legalization (Discrete -> continuous)

Row에 맞게 위치를 조정해준 뒤,겹치는 공간이 없게 조정(DP)



강화학습 formulation

작은 애들은 알아서 배치해주는 방법 (고전 graph placement)



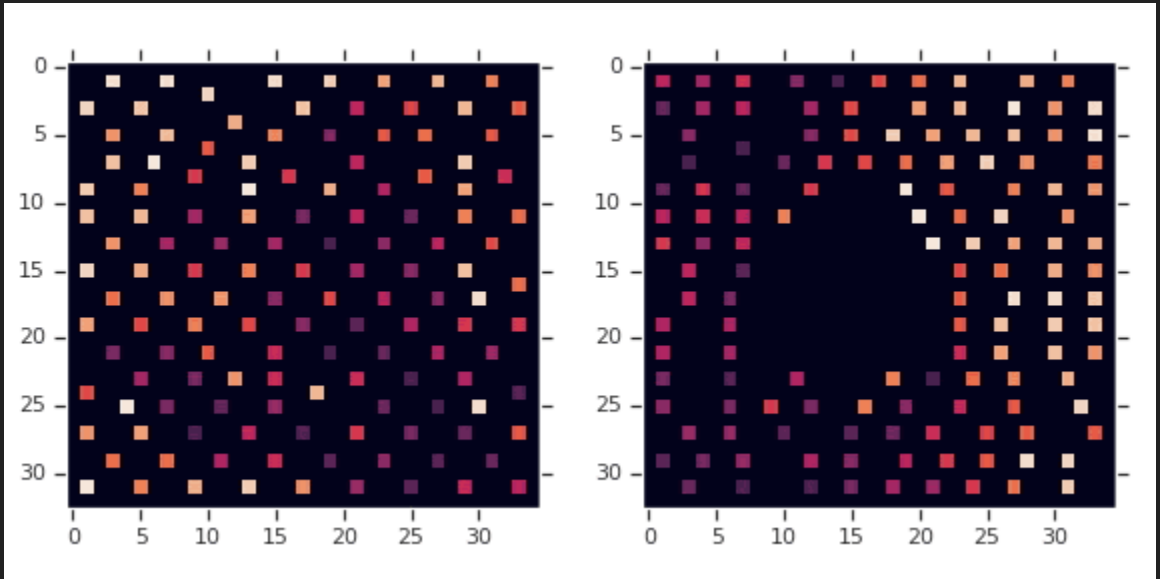
$$J(\theta, G) = \frac{1}{K} \sum_{g \sim G} E_{g, p \sim \pi_\theta}[R_{p,g}]$$

Set of training graphs G

K is size of training set

RL policy parameterized by theta

Placement (scratch vs pretrained)



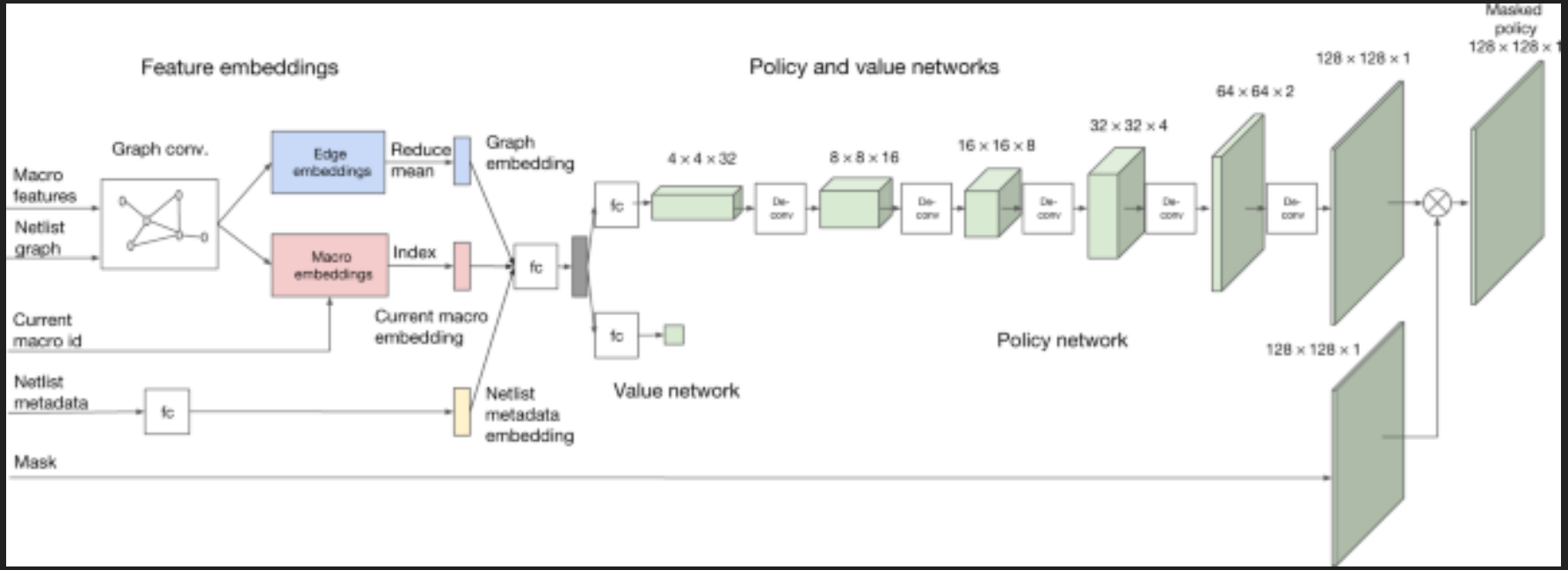
환경 설명

State : 소자들 간의 연결성에 대한 정보, chip canvas의 상태, step에서 배치될 macro에 대한 정보, macro를 배치할 수 있는 영역에 관한 정보

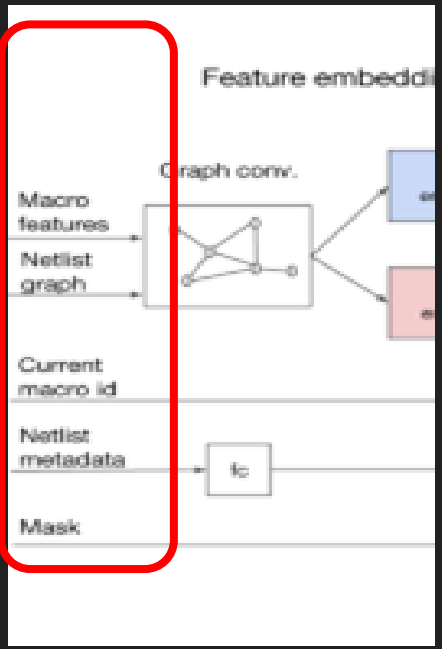
Action : chip canvas를 여러 개의 grid 영역으로 나눠서 action space를 구성

Reward : Wire length, Congestion, Density (power, area, timing)

전체 모델


Feature embeddings
Policy and value networks
Graph conv.
Macro features
Netlist graph
Edge embeddings
Reduce mean
Graph embedding
Macro embeddings
Index
Current macro id
Current macro embedding
Netlist metadata
fc
Netlist metadata embedding
Mask
fc
4 × 4 × 32
8 × 8 × 16
16 × 16 × 8
32 × 32 × 4
64 × 64 × 2
128 × 128 × 1
De-conv
Value network
Policy network
Masked policy
128 × 128 × 1

Input에 대한 정보



Macro Feature는 배치 대상되는 Macro의 특성 정보들을 말합니다. 여기서의 특성 정보에는 Macro의 타입, 크기, 위치 정보등이 포함되어 있습니다.

Netlist Graph는 소자들의 연결관계를 표현하는 Adjacency Matrix입니다.

Current Macro id는 현재 배치하고자 하는 Macro가 무엇인지 알려주는 정보입니다.

Netlist Metadata는 말 그대로 Netlist의 메타 정보들로서, Netlist를 구성하는 Macro의 개수, Wire의 개수, Standard Cell Cluster의 개수, 배치 Grid의 크기 등이 들어갑니다.

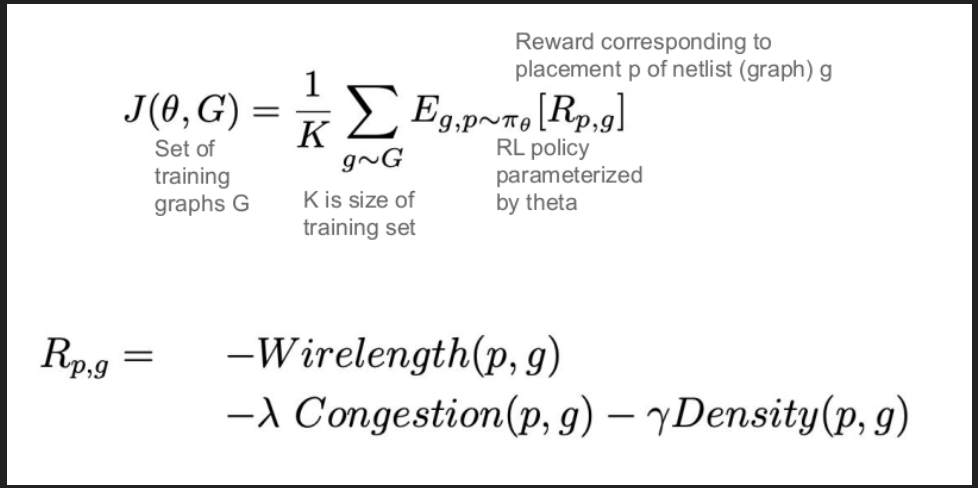
Mask는 배치가 가능한 영역을 알려주는 정보로, Action을 Masking 하여 배치 배치 불가능한 영역에 대해서는 배치가 이뤄지지 않도록 합니다.

Objective

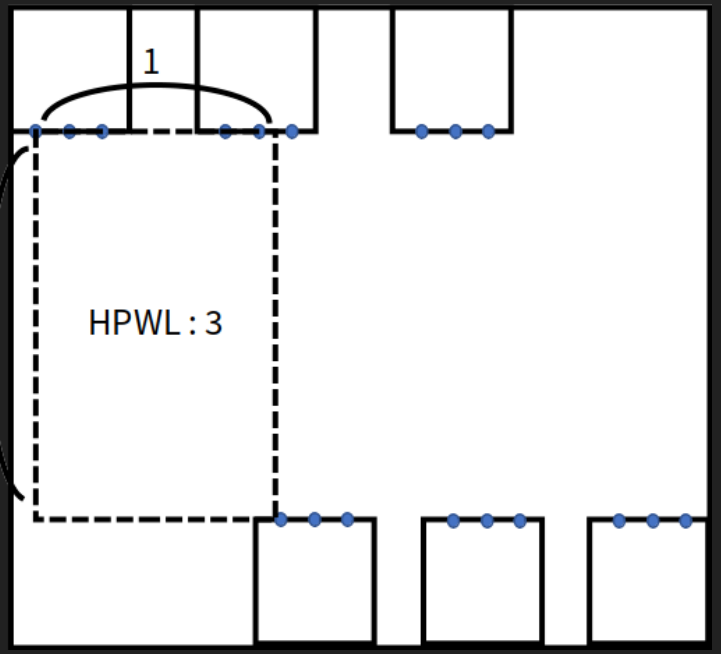
Power, Timing, Area가 좋을수록 좋지만 강화학습 step 중에 계산하기 어려움.

(Routing이 필요함)

따라서 다음과 같은 Objective function을 만듬.

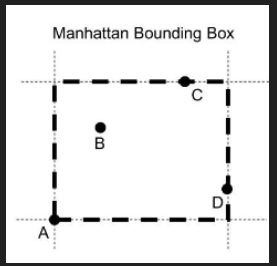


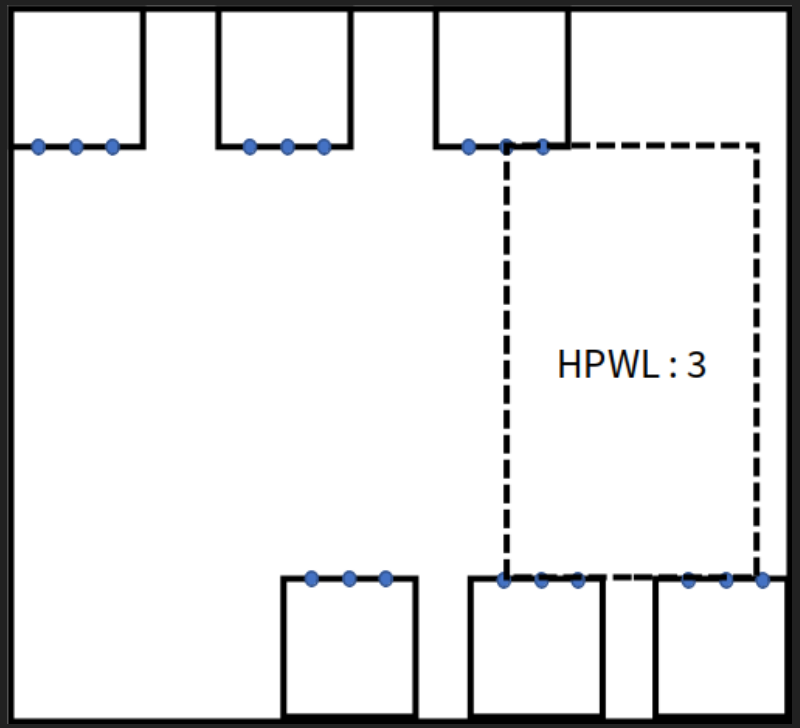
Wire length : Half perimeter wire length(HPWL)



((1,2), (1,5), (2,3,4), (3,5,6), (4,6))

HPWL = max(x_net) - min(x_net) + max(y_net) - min(y_net)





((1,2), (1,5), (2,3,4), (3,5,6), (4,6))

Summation of HPWL : 3+6+3+3+1.5 = 16.5

Congestion

Flylight Analysis로 구함

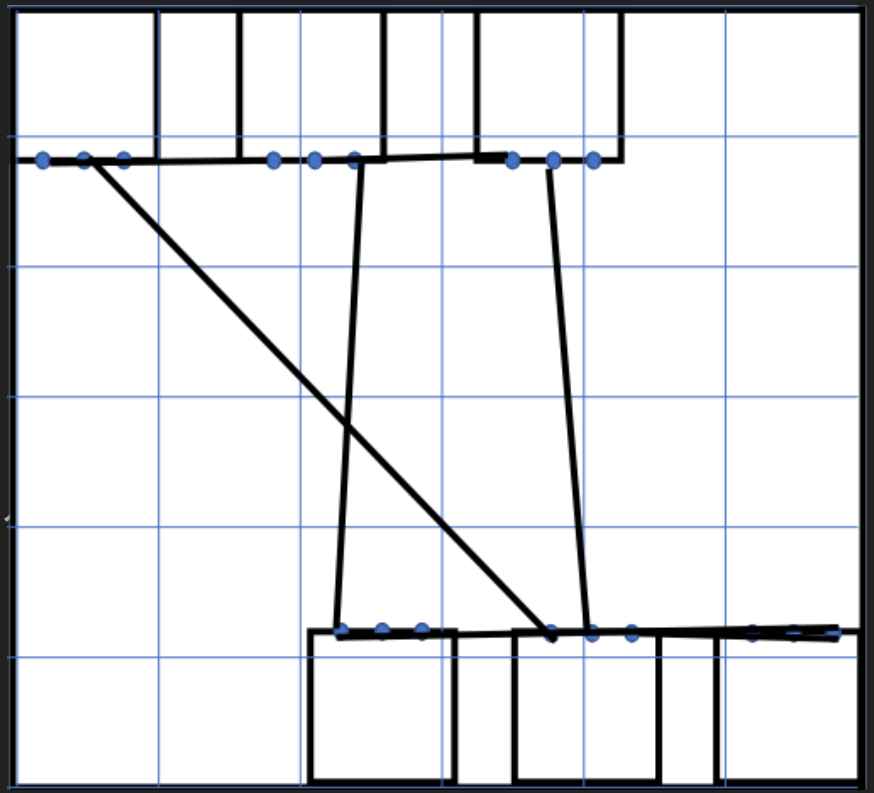
Logical direction에 따라 직선을 그은 다음,

netlist: ((a,c,e), (e,g), (b,c,d), (c,e,f), (d,f), (f,g,h))

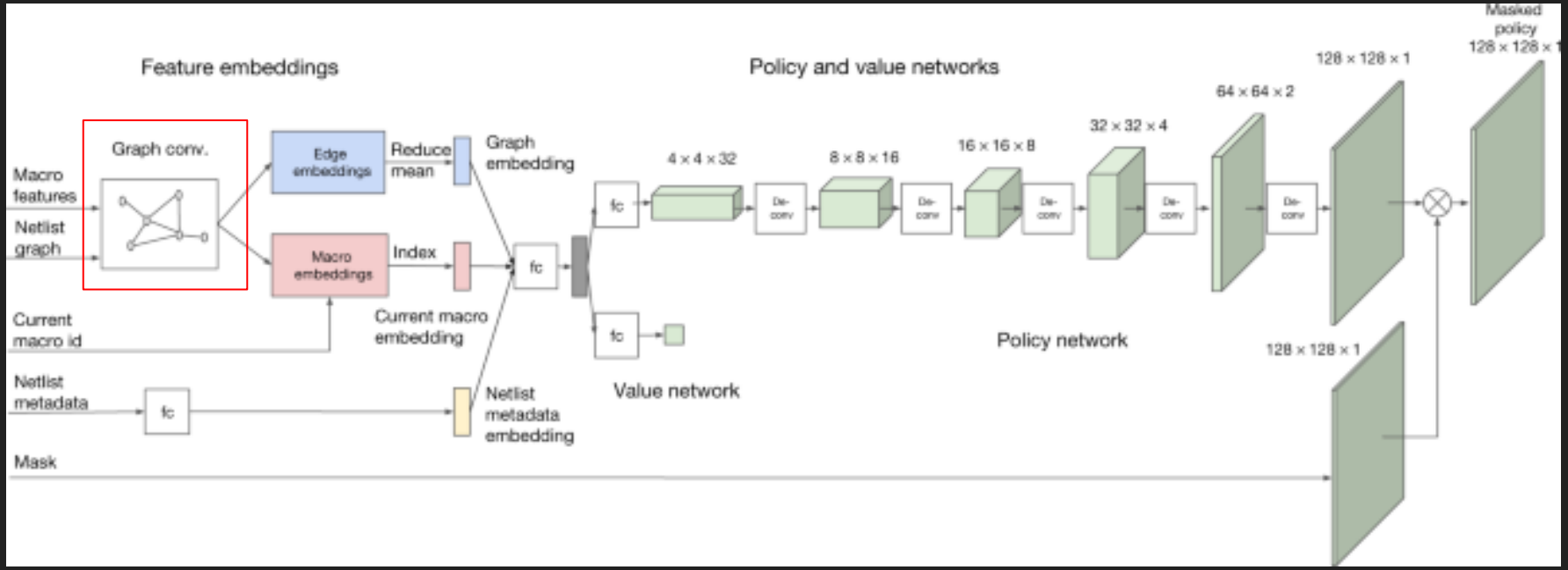
상하좌우를 통과한 수만큼 계산

Congestion = Routing demand - Capacity

구글에서는 네 방향 Congestion을 합쳐서 계



전체 모델


Feature embeddings
Policy and value networks
Graph conv.
Macro features
Netlist graph
Edge embeddings
Reduce mean
Graph embedding
Macro embeddings
Index
Current macro id
Current macro embedding
fc
Netlist metadata
Netlist metadata embedding
Mask
4 × 4 × 32
8 × 8 × 16
16 × 16 × 8
32 × 32 × 4
64 × 64 × 2
128 × 128 × 1
De-conv
Value network
Policy network
Masked policy
128 × 128 × 1

Graph model

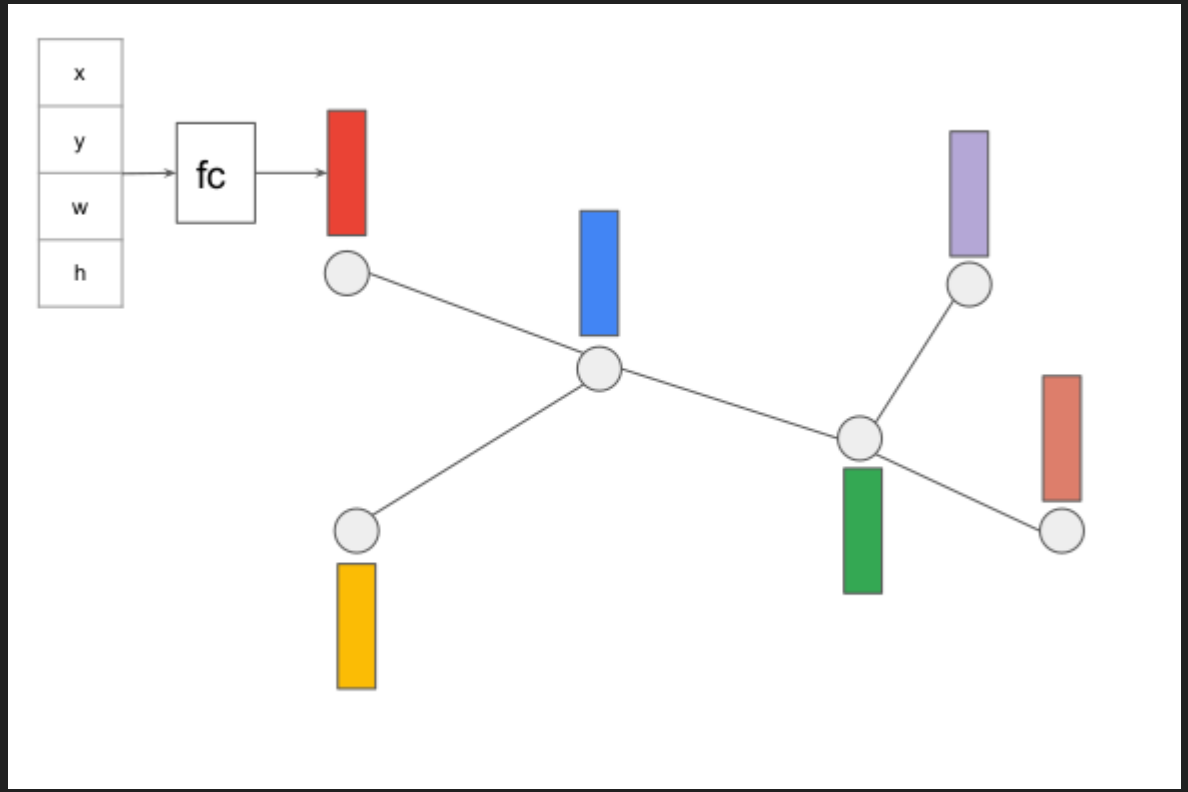
- Edge GNN(paper) vs GCN

while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end

- Edge GNN의 경우에는 edge도 update된다는 특징이 있음.

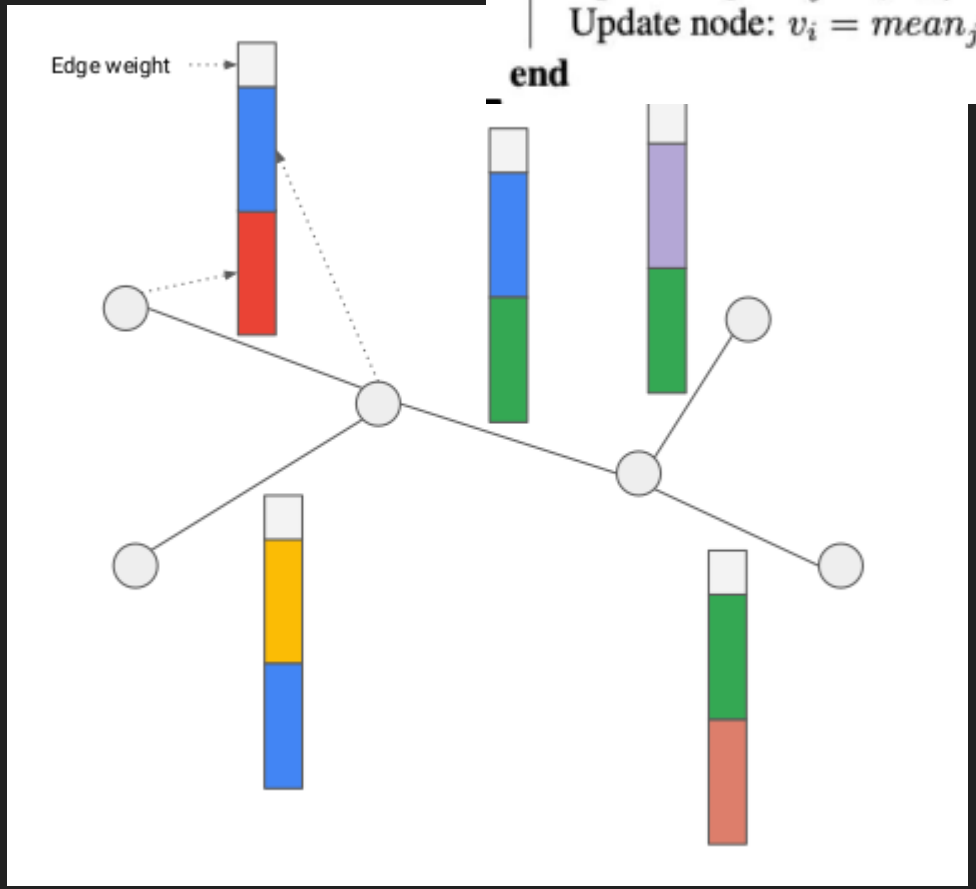
step 1

while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end



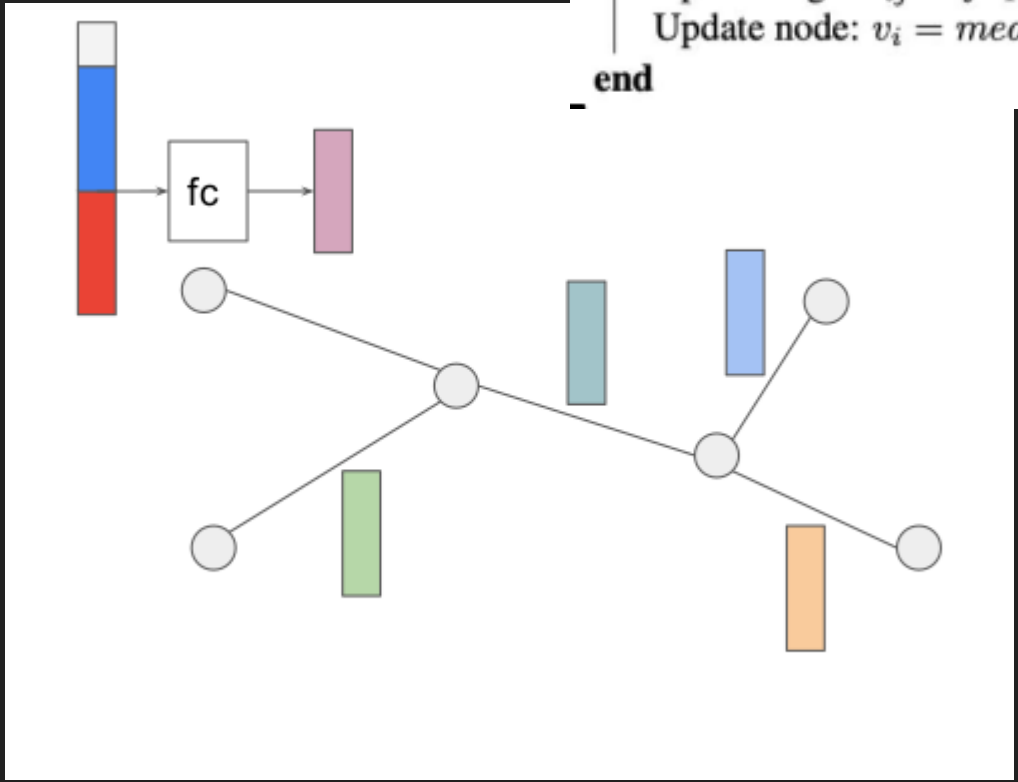
step 2

while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end



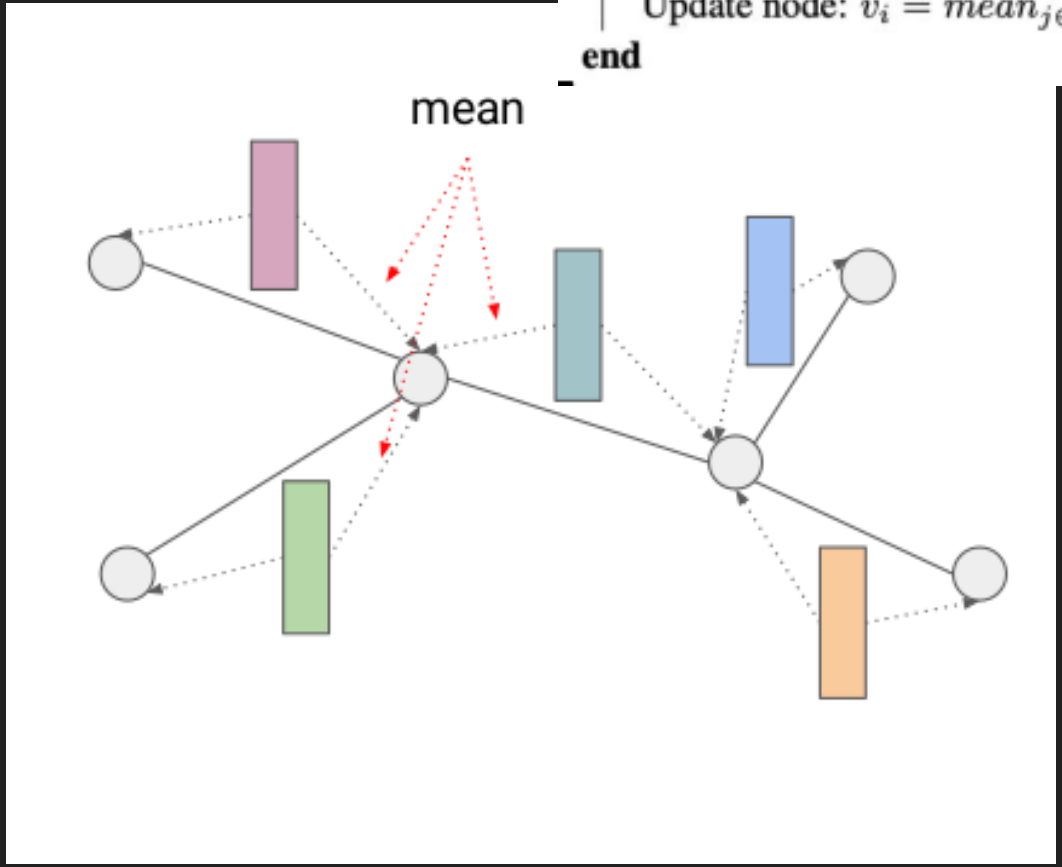
step 3

while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end



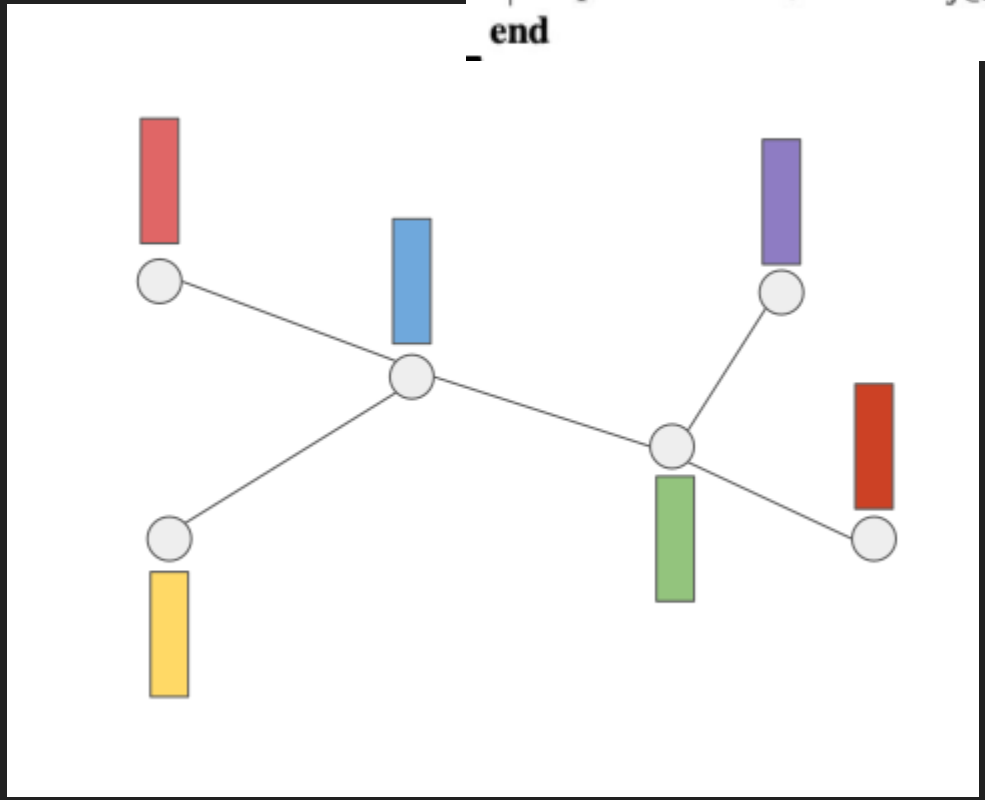
step 4

while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end

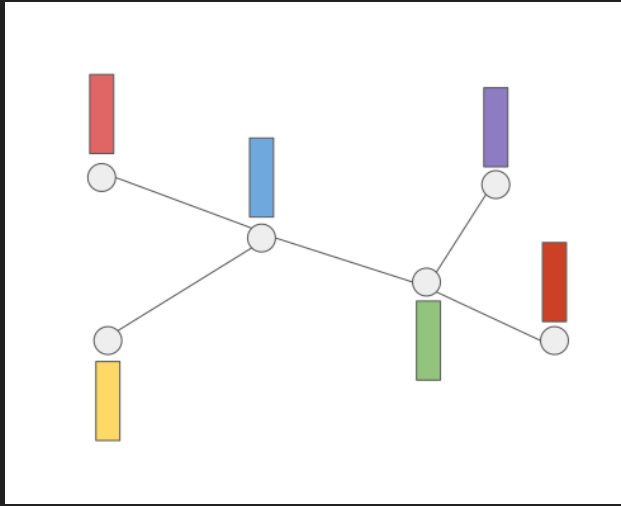


Repeat

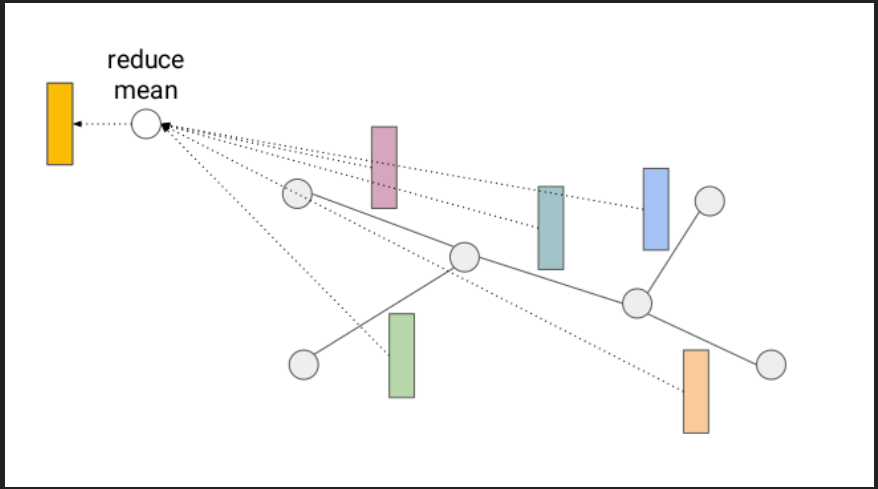
while *Not converged* **do**
  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w_{ij}^e])$
  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end



Macro embedding vs graph embedding

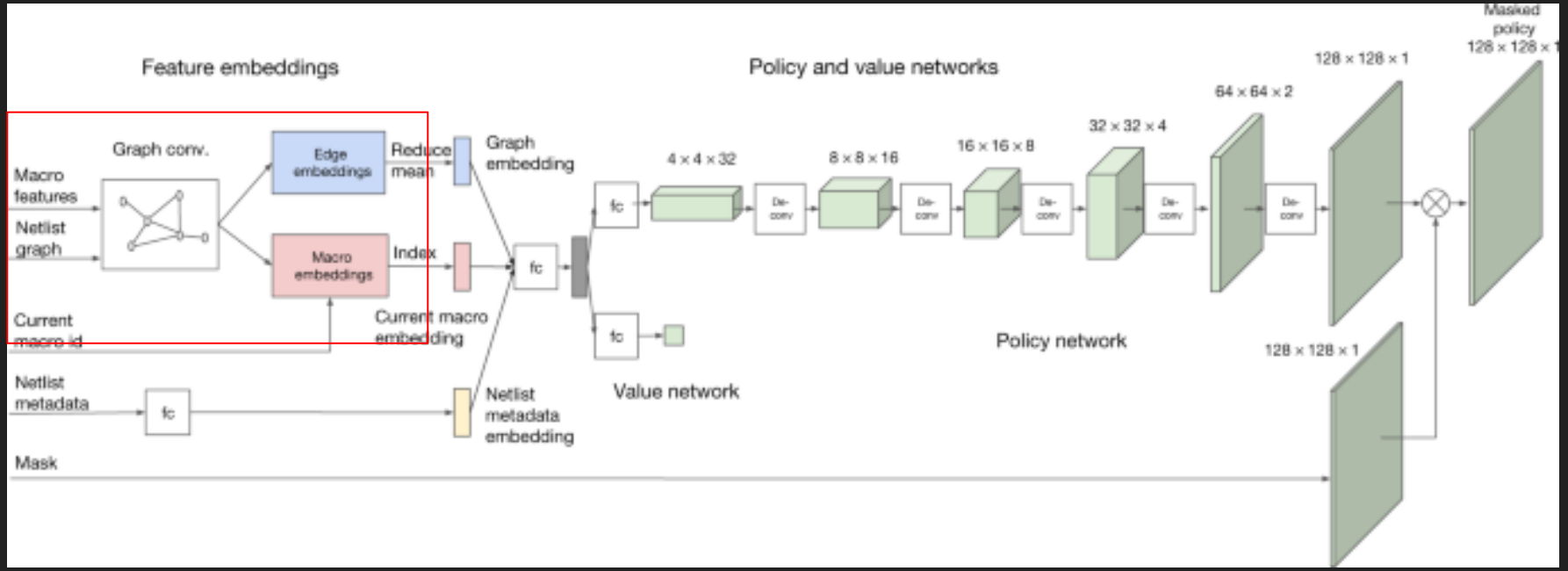


macro(node) embedding

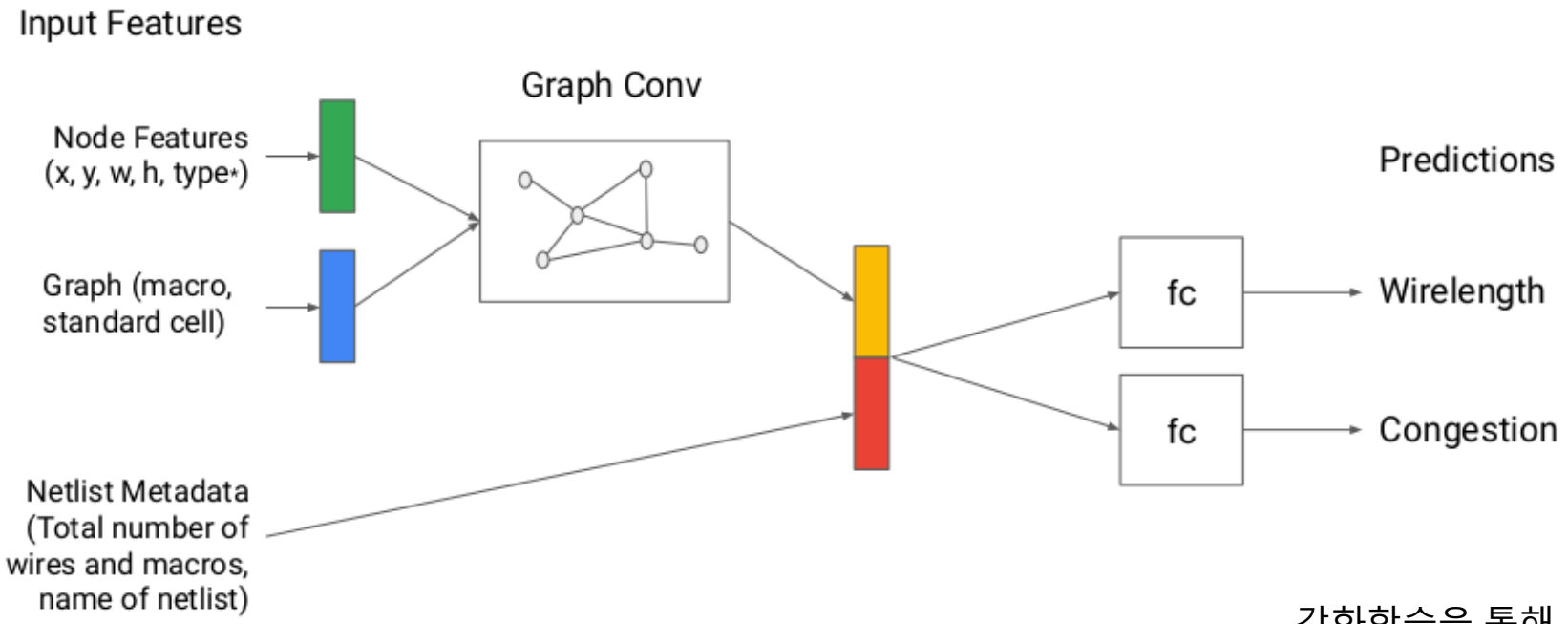


graph embedding

전체 모델

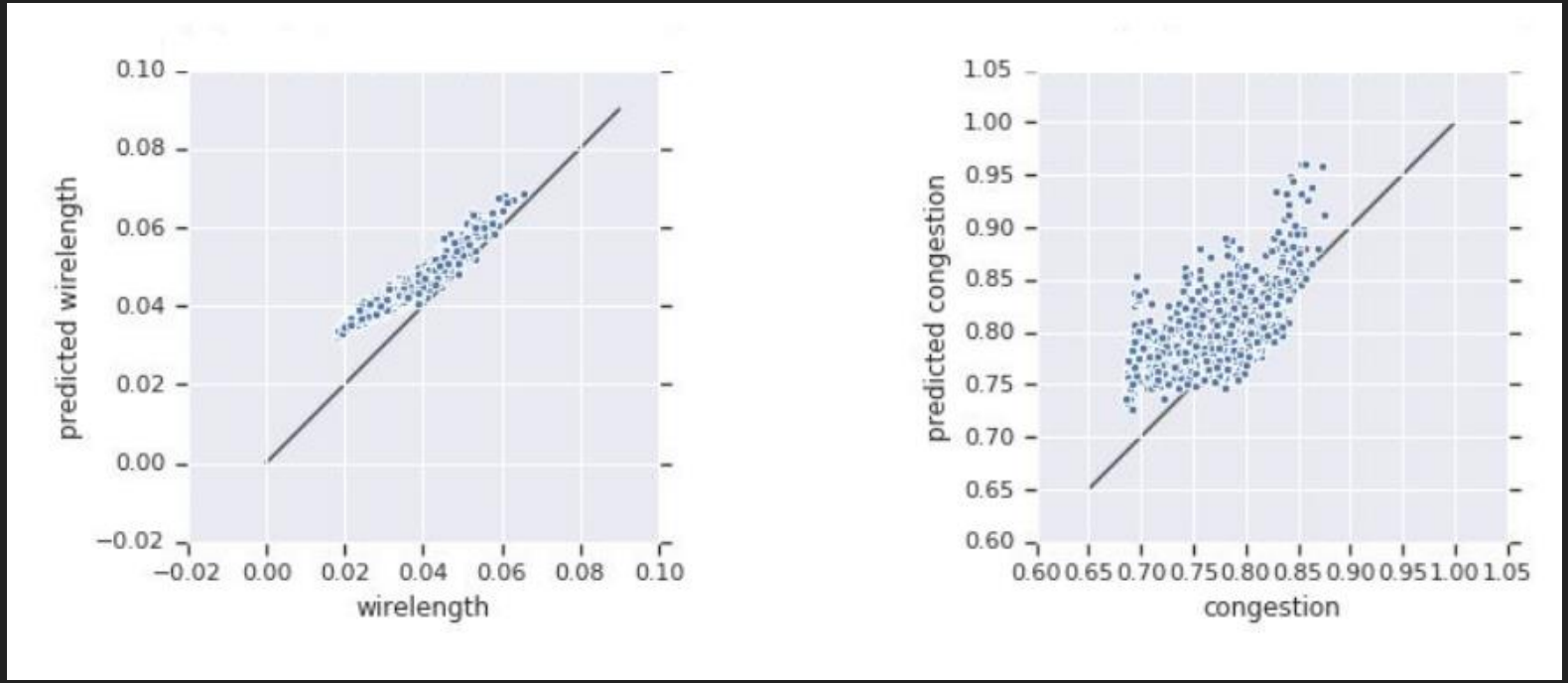

Feature embeddings
Policy and value networks
Graph conv.
Macro features
Netlist graph
Edge embeddings
Reduce mean
Graph embedding
Macro embeddings
Index
Current macro id
Current macro embedding
Netlist metadata
fc
Netlist metadata embedding
Mask
4 × 4 × 32
8 × 8 × 16
16 × 16 × 8
32 × 32 × 4
64 × 64 × 2
128 × 128 × 1
De-conv
Value network
Policy network
Masked policy

Embedding model (Pretrain)

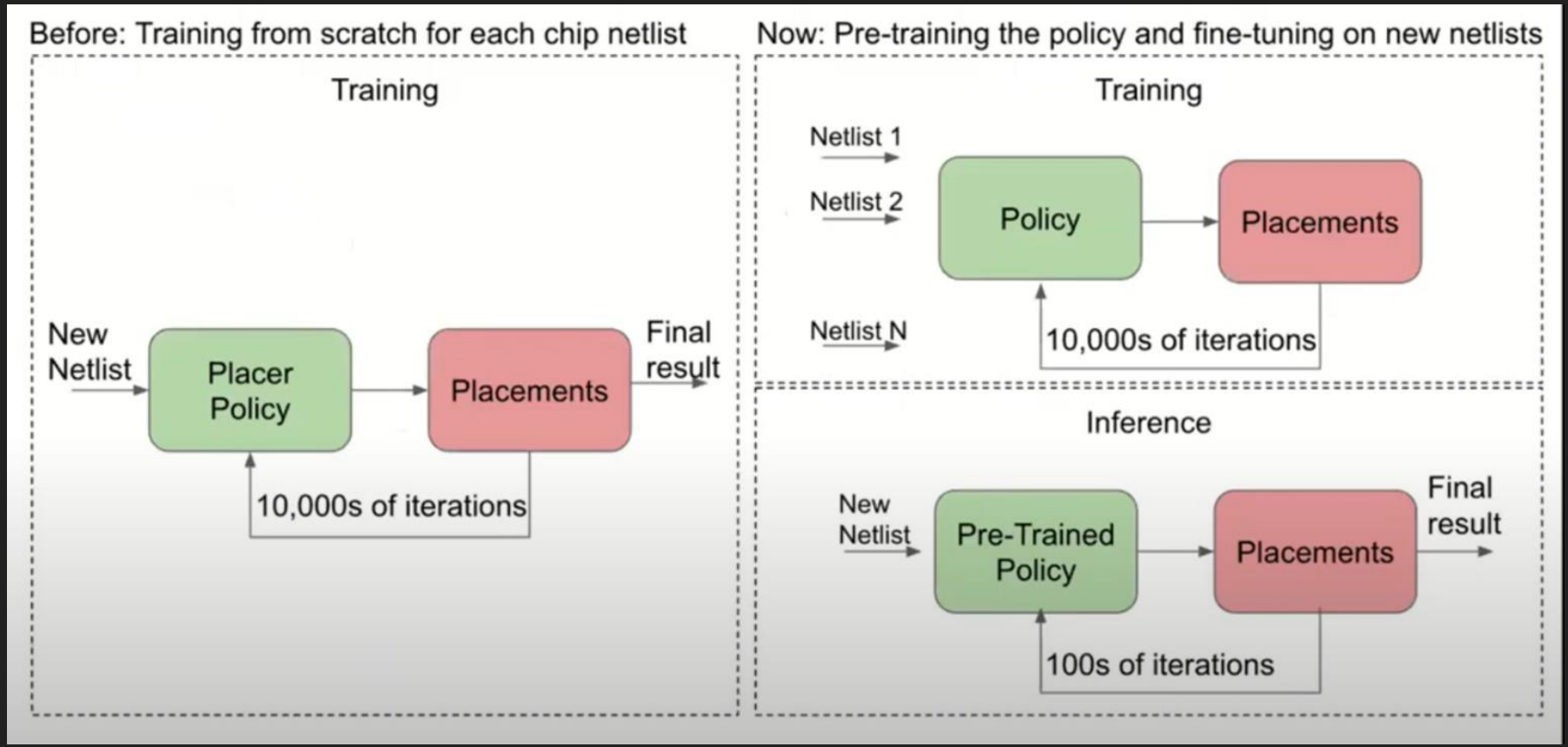


강화학습을 통해 데
이터 수집

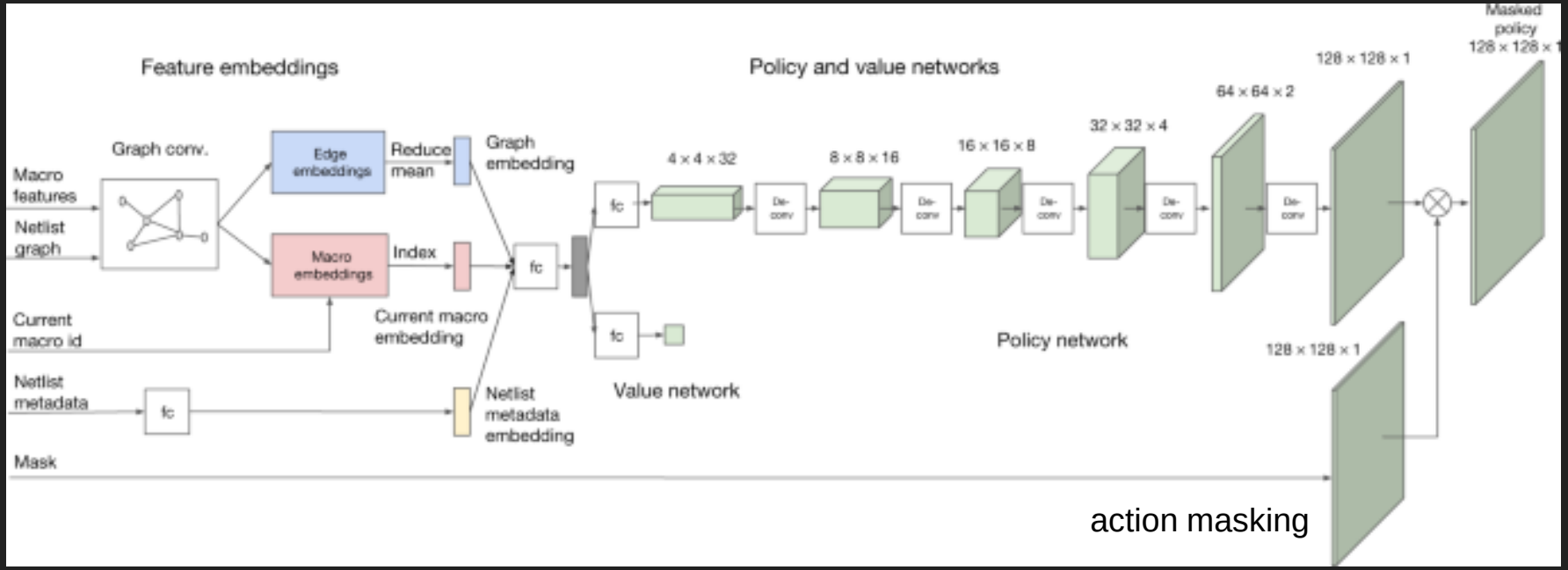
Pretrained model 성능 체크



moving towards generalized placement

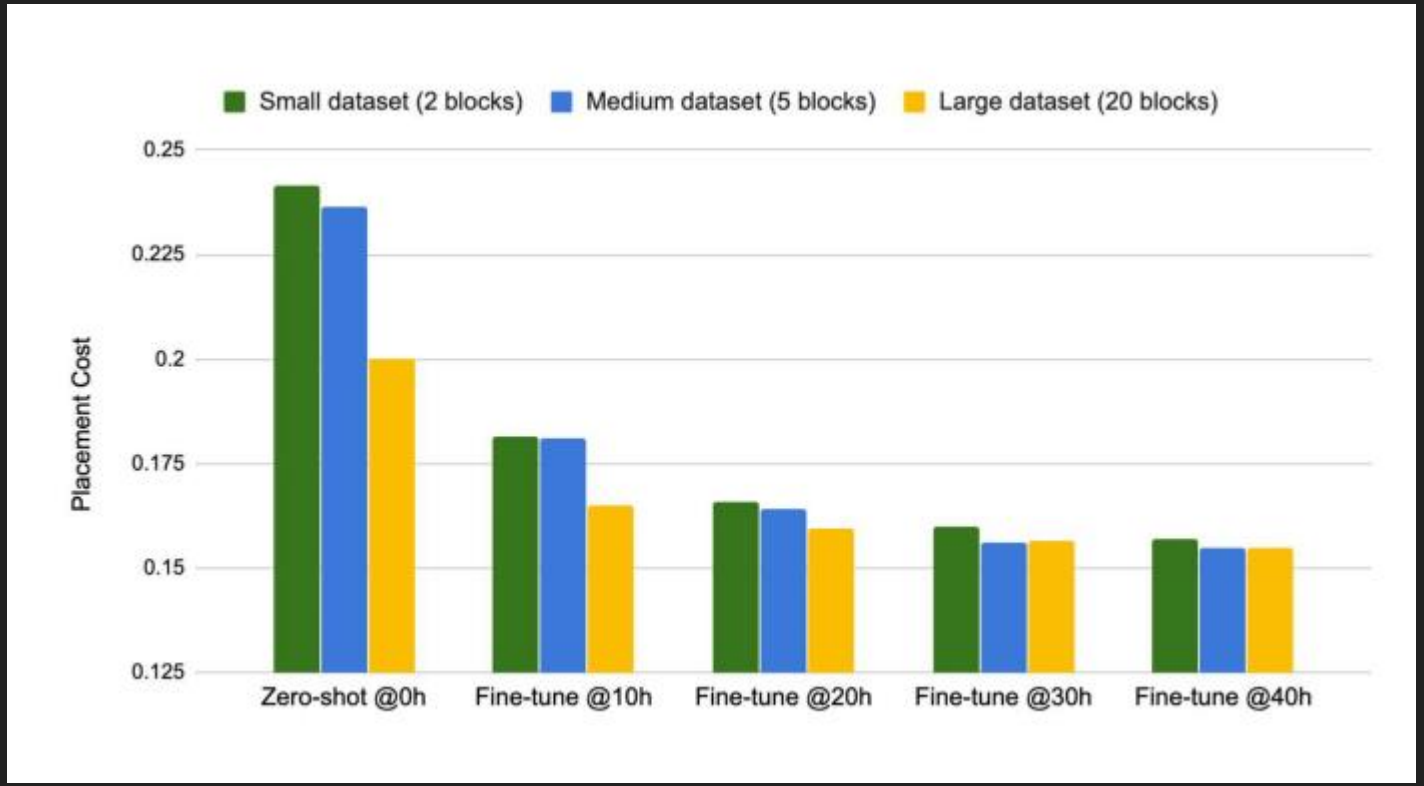


전체 모델


Feature embeddings
Policy and value networks
Graph conv.
Macro features
Netlist graph
Edge embeddings
Reduce mean
Graph embedding
Macro embeddings
Index
Current macro id
Current macro embedding
Netlist metadata
fc
Netlist metadata embedding
Mask
fc
4 × 4 × 32
8 × 8 × 16
16 × 16 × 8
32 × 32 × 4
64 × 64 × 2
128 × 128 × 1
De-conv
Value network
Policy network
Masked policy
128 × 128 × 1
action masking

Pretrain 수렴 속도

Pretrained 데이터셋 크기의 영향



벤치마킹

Name	Method	Timing		Area	Power	Wirelength	Congestion	
		WNS (ps)	TNS (ns)	Total (μm^2)	Total (W)	(m)	H (%)	V (%)
Block 1	RePlAce	374	233.7	1693139	3.70	52.14	1.82	0.06
	Manual	136	47.6	1680790	3.74	51.12	0.13	0.03
	Ours	84	23.3	1681767	3.59	51.29	0.34	0.03
Block 2	RePlAce	97	6.6	785655	3.52	61.07	1.58	0.06
	Manual	75	98.1	830470	3.56	62.92	0.23	0.04
	Ours	59	170	694757	3.13	59.11	0.45	0.03
Block 3	RePlAce	193	3.9	867390	1.36	18.84	0.19	0.05
	Manual	18	0.2	869779	1.42	20.74	0.22	0.07
	Ours	11	2.2	868101	1.38	20.80	0.04	0.04
Block 4	RePlAce	58	11.2	944211	2.21	27.37	0.03	0.03
	Manual	58	17.9	947766	2.17	29.16	0.00	0.01
	Ours	52	0.7	942867	2.21	28.50	0.03	0.02
Block 5	RePlAce	156	254.6	1477283	3.24	31.83	0.04	0.03
	Manual	107	97.2	1480881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1472302	3.28	36.59	0.01	0.03