Profiling and analyzing GraphSaint with MH-Aug

20230503
박현우

Algorithm 1 GraphSAINT using MH-Aug

```
Input: target distribution P, proposal distribution Q, original graph G
Output: GCN parameter θ
Initialize: t ← 0, G_s^(0) ← G_s
while not convergence do
    while True do
        Draw G' from Q(G'|G^(t))
        Draw u from Uniform(0,1)
        G^(t+1) ← G'
        if u < A then
            break
        end if
    end while
    for each batch do
        G_s(V_s, E_s) ← Sampled subgraph of G
        G_s^(t)(V_s, E_s) ← Sampled subgraph of G^(t)
        G_s^(t+1)(V_s, E_s) ← Sampled subgraph of G^(t+1)
        Update θ with L(G_s, G_s^(t), G_s^(t+1), θ)
        t ← t + 1
    end for
end while
```

← Augmentation Part

10/88

Training is very slow and as a result, the model doesn’t converge well.

= Are there any bottlenecks?

Run profiling in training of one sample graph in the dataset.

```python
def generate_aug_graph(g, model,
                       sigma_delta_e=0.03, sigma_delta_v=0.03, mu_e=0.6, mu_v=0.2,
                       lam1_e=1, lam1_v=1, lam2_e=0.0, lam2_v=0.0,
                       a_e=100, b_e=1, a_v=100, b_v=1):
    # Original Graph Feature and Metadata Extraction, Preprocessing
    num_nodes = g.num_nodes()
    num_edges = g.num_edges()

    coo_mat = g.edges(form='uv')
    coo_mat = th.tensor([list(coo_mat[0]), list(coo_mat[1])], device='cuda:0')
    n_list = th.ones(num_nodes)

    # Create Aggregate Model
    agg_model = AGGNet(num_hop=2)
    agg_model.cuda()

    while True:
        # Calculate Delta Value
        delta_G_e = 1 - coo_mat.shape[1] / num_edges
        delta_G_e_aug = our_truncnorm(0, 1, delta_G_e, sigma_delta_e, mode='rvs')

        delta_G_v = 1 - n_list.sum().item() / num_nodes
        delta_G_v_aug = our_truncnorm(0, 1, delta_G_v, sigma_delta_v, mode='rvs')

        # Graph Augmentation According To Delta Value
        aug_g, aug_n_list = augment(g, delta_G_e_aug, delta_G_v_aug)
        aug_g = dgl.add_self_loop(aug_g)

        # message_passing_g = copy.deepcopy(g)
        message_passing_g = g.clone()
        message_passing_g.ndata['feat'] = th.ones(num_nodes, 1, device='cuda:0')
```

```python
# Split into sub functions for profiling.
def _generate_aug_graph(g, model,
                        sigma_delta_e=0.03, sigma_delta_v=0.03, mu_e=0.6, mu_v=0.2,
                        lam1_e=1, lam1_v=1, lam2_e=0.0, lam2_v=0.0,
                        a_e=100, b_e=1, a_v=100, b_v=1):

    # Original Graph Feature and Metadata Extraction, Preprocessing
    def initialize():
        num_nodes = g.num_nodes()
        num_edges = g.num_edges()

        coo_mat = g.edges(form='uv')
        coo_mat = torch.tensor([list(coo_mat[0]), list(coo_mat[1])]).to(DEVICE)
        n_list = torch.ones(num_nodes).to(DEVICE)

        # Create Aggregate Model
        agg_model = AGGNet(num_hop=2)
        agg_model.cuda()

        return num_nodes, num_edges, coo_mat, n_list, agg_model

    def calculate_delta_value():
        # Calculate Delta Value
        delta_G_e = 1 - coo_mat.shape[1] / num_edges
        delta_G_e_aug = our_truncnorm(0, 1, delta_G_e, sigma_delta_e, mode='rvs')

        delta_G_v = 1 - n_list.sum().item() / num_nodes
        delta_G_v_aug = our_truncnorm(0, 1, delta_G_v, sigma_delta_v, mode='rvs')

        return delta_G_e, delta_G_e_aug, delta_G_v, delta_G_v_aug
```

Algorithm 1 GraphSAINT using MH-Aug

Input: target distribution P, proposal distribution Q, original graph G
Output: GCN parameter θ
Initialize: $t \leftarrow 0, {G_s}^{(0)} \leftarrow G_s$

while *not convergence* **do**
  while *True* **do**
    Draw G' from $Q(G'|G^{(t)})$
    Draw u from $Uniform(0,1)$
    $G^{(t+1)} \leftarrow G'$
    if $u < A$ **then**
      break
    end if
  end while
  for *each batch* **do**
    $G_s(V_s, E_s) \leftarrow$ Sampled subgraph of G
    ${G_s}^{(t)}(V_s, E_s) \leftarrow$ Sampled subgraph of $G^{(t)}$
    ${G_s}^{(t+1)}(V_s, E_s) \leftarrow$ Sampled subgraph of $G^{(t+1)}$
    Update θ with $L(G_s, {G_s}^{(t)}, {G_s}^{(t+1)}, \theta)$
    $t \leftarrow t + 1$
  end for
end while

← Augmentation Part

10/88

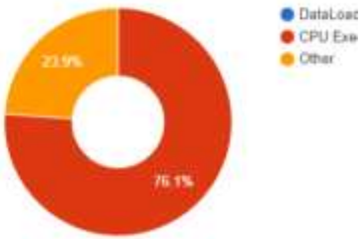
```python
with record_function("initialize"):
    num_nodes, num_edges, coo_mat, n_list, agg_model = initialize()

while True:

    with record_function("calculate_delta_value"):
        delta_G_e, delta_G_e_aug, delta_G_v, delta_G_v_aug = calculate_delta_value()

    with record_function("graph_augmentation"):
        aug_g = graph_augmentation()

    with record_function("message_passing"):
        message_passing_g, message_passing_aug_g = message_passing()

    with record_function("calculate_ego_graph_message_passing_value"):
        org_ego = calculate_ego_graph_message_passing_value()

    with record_function("calculate_augmented_delta"):
        delta_g_e, delta_g_aug_e, delta_g_v, delta_g_aug_v = calculate_augmented_delta()

    with record_function("calculate_distribution"):
        h_loss_op = calculate_distribution()

    with record_function("compute_model"):
        output = compute_model()

    with record_function("compute_ent"):
        ent = compute_ent()

    with record_function("compute_p_aug"):
        p, p_aug = compute_p_aug()

    with record_function("compute_q_aug"):
        q, q_aug = compute_q_aug()
```

Configuration

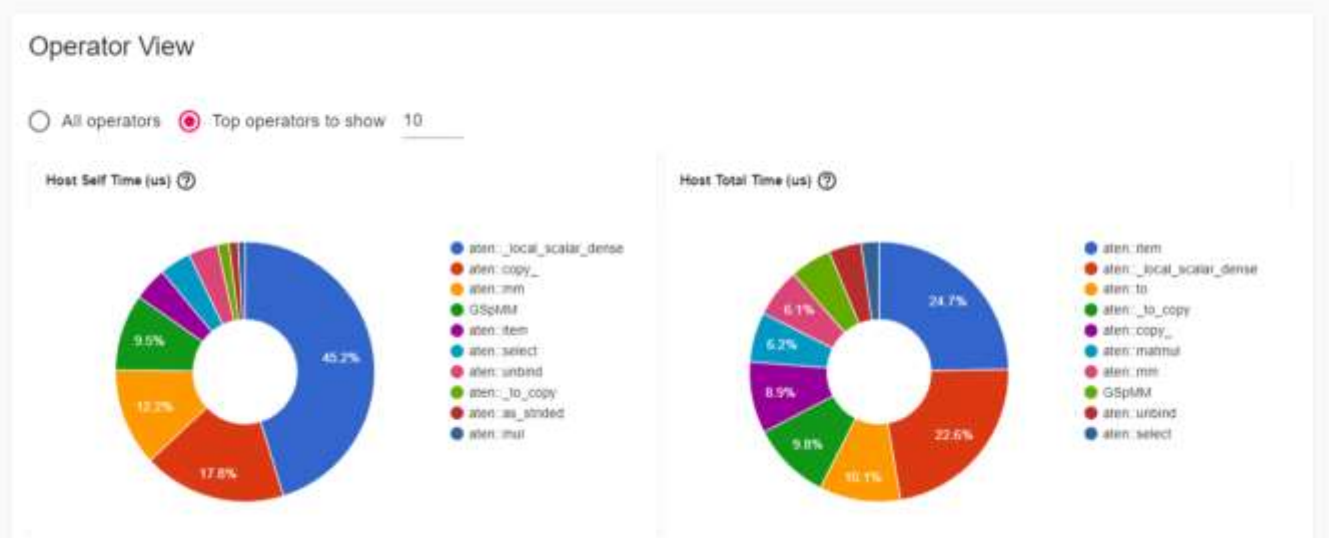
Number of Worker(s)	**4**
Device Type	**CPU**

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	6,957,929	100
DataLoader	0	0
CPU Exec	5,294,977	76.1
Other	1,662,952	23.9



Step Time Breakdown ⓘ

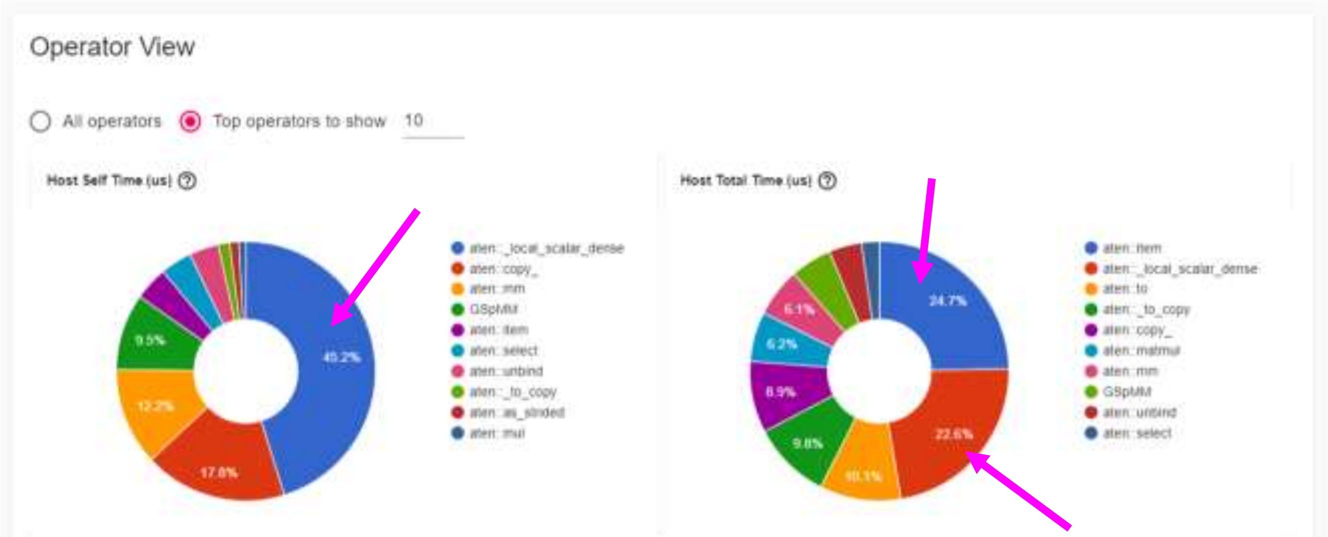


(left) Host Self Time

: Accumulated time of operator itself only.

(right) Host Total Time

: Accumulated time of both operator and its all children operators.



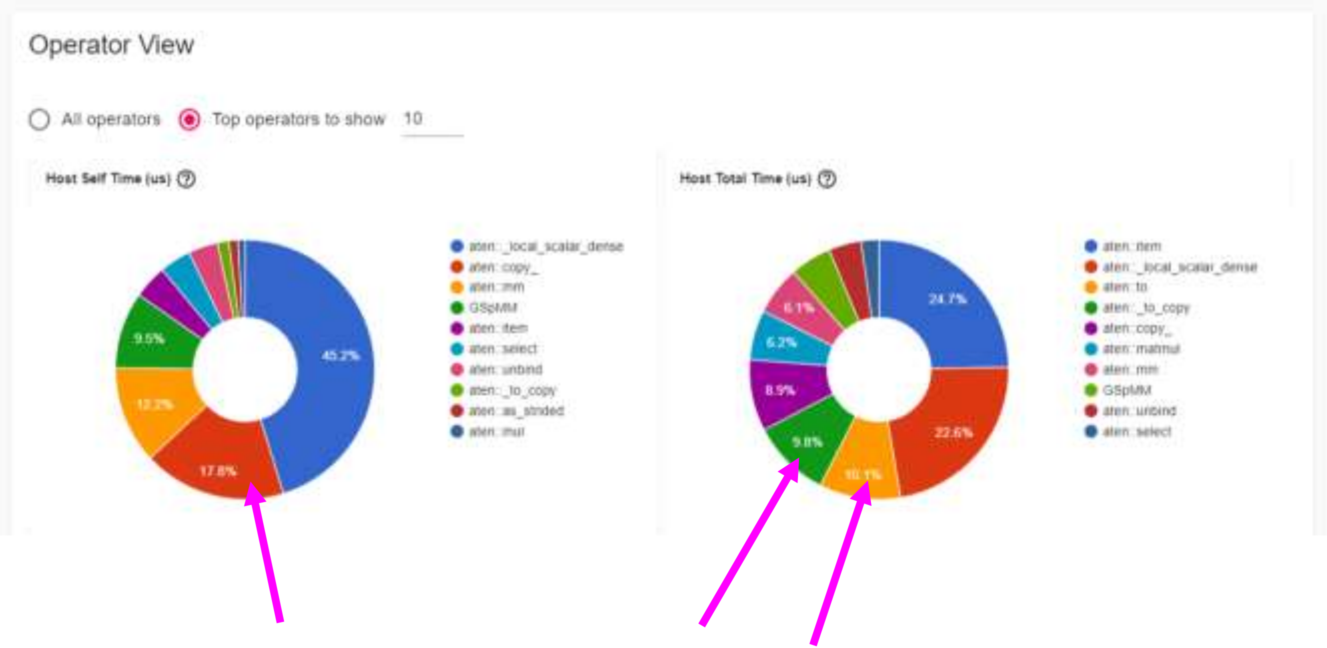
(left) / (right 2)

aten::_local_scalar_dense

Dense operation

(right) aten::item

Ordered_dict for parameter_dict, containing and handling parameters.



(left) aten::copy_

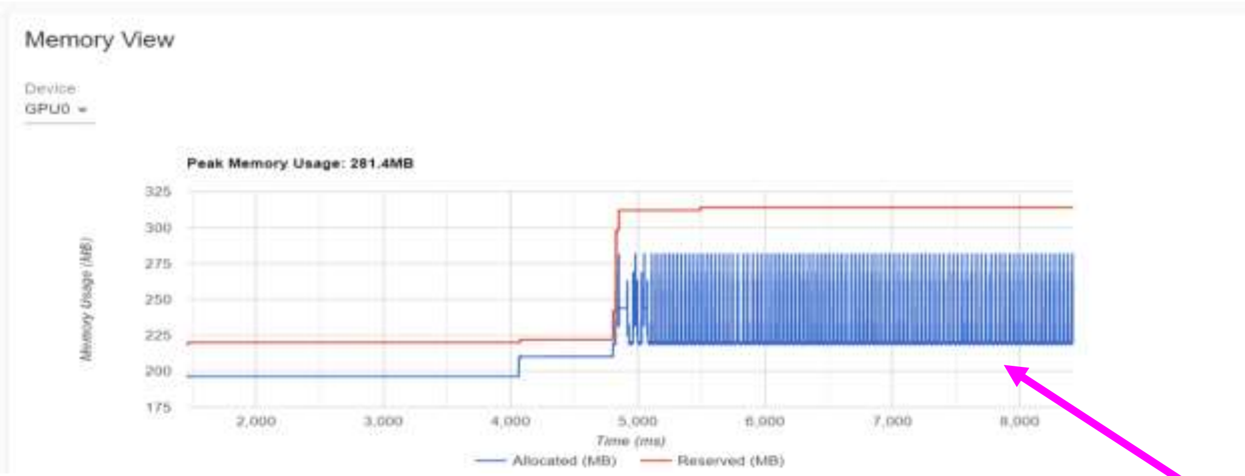
Copy from device memory into host memory

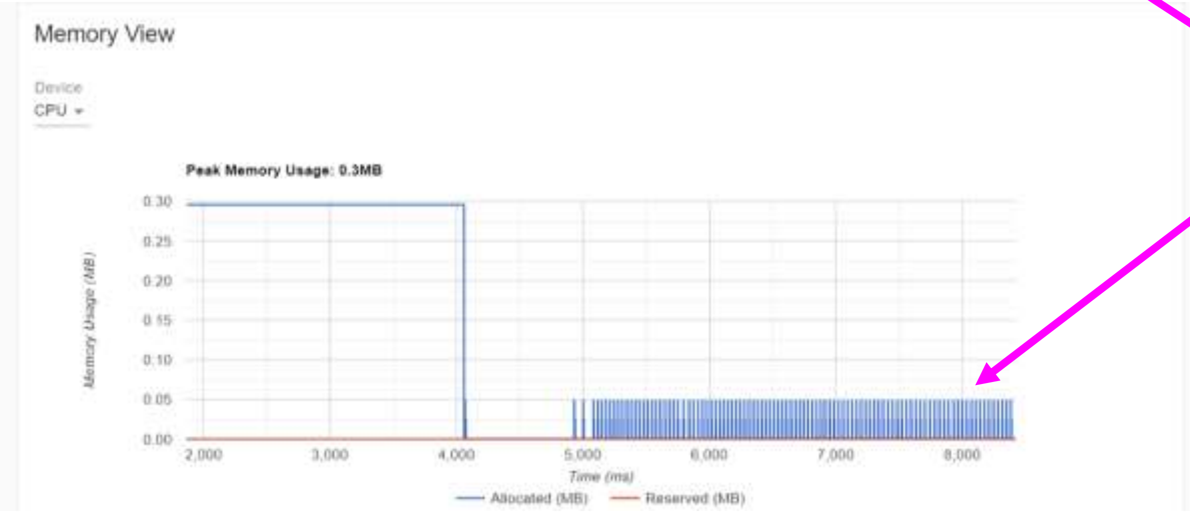
(right) aten::to

Move memory to other memory (H2D or D2H)

(right) aten::_to_copy

Copy from host memory into device memory



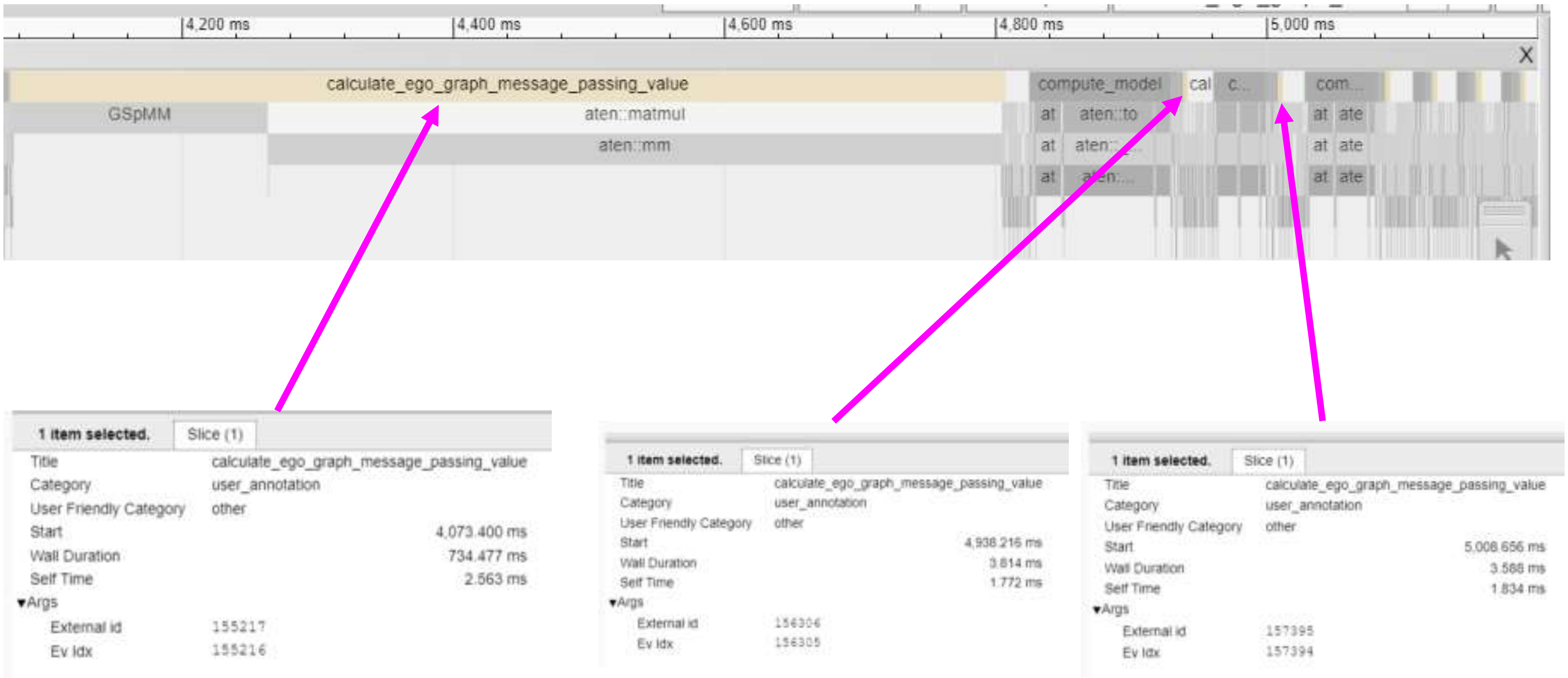


A lot of H2D and D2H copies

calculate_ego_graph_message_
passing_value

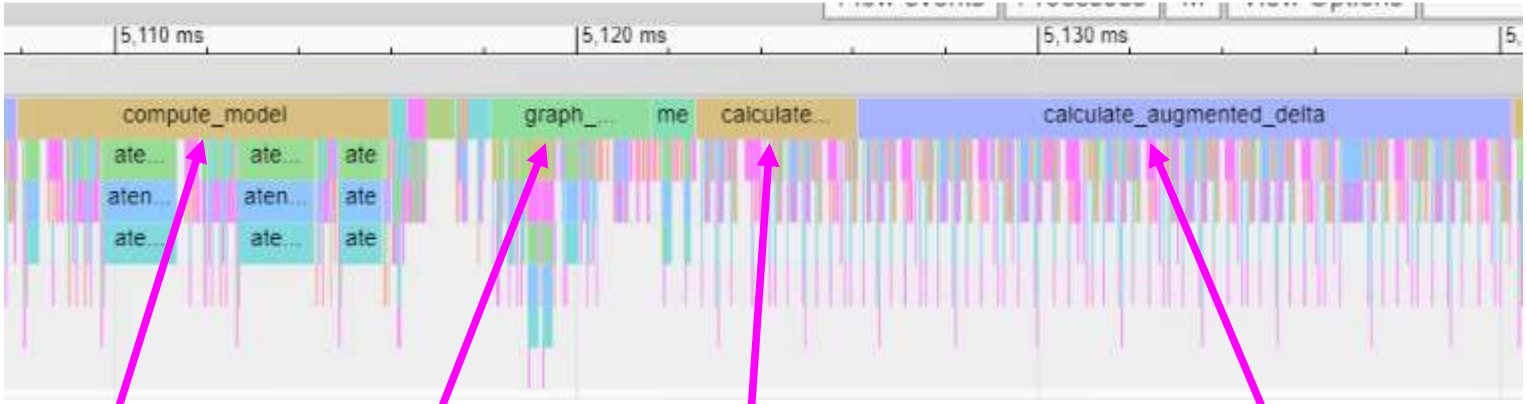
Initialization

Augmentation loop



May be warming up? (= loaded on device and cached)

One cycle of augmentation loop



1. compute_model

2. graph_augmentation

4. calculate_augmented_delta

3. calculate_ego_graph_message_passing_value

Top 4 Time taken : 4 > 1 > 3 > 2

```python
def calculate_ego_graph_message_passing_value():
    # Calculate ego-graph's message passing value
    with torch.no_grad():
        org_ego = aggregate(message_passing_g, agg_model)

    return org_ego
```

```python
def compute_model():
    with torch.no_grad():
        output = model(g)

    return output
```

```python
def calculate_augmented_delta():
    # Calculate Augmented Delta Value
    with torch.no_grad():
        delta_g_e = 1 - (aggregate(message_passing_g, agg_model) / org_ego).squeeze(1)
        delta_g_aug_e = 1 - (aggregate(message_passing_aug_g, agg_model) / org_ego).squeeze(1)
        delta_g_v = 1 - (aggregate(message_passing_g, agg_model) / org_ego).squeeze(1)
        delta_g_aug_v = 1 - (aggregate(message_passing_aug_g, agg_model) / org_ego).squeeze(1)

    return delta_g_e, delta_g_aug_e, delta_g_v, delta_g_aug_v
```

```python
def graph_augmentation():
    # Graph Augmentation According To Delta Value
    aug_g, aug_n_list = augment(g, delta_G_e_aug, delta_G_v_aug)
    aug_g = dgl.add_self_loop(aug_g)

    return aug_g
```

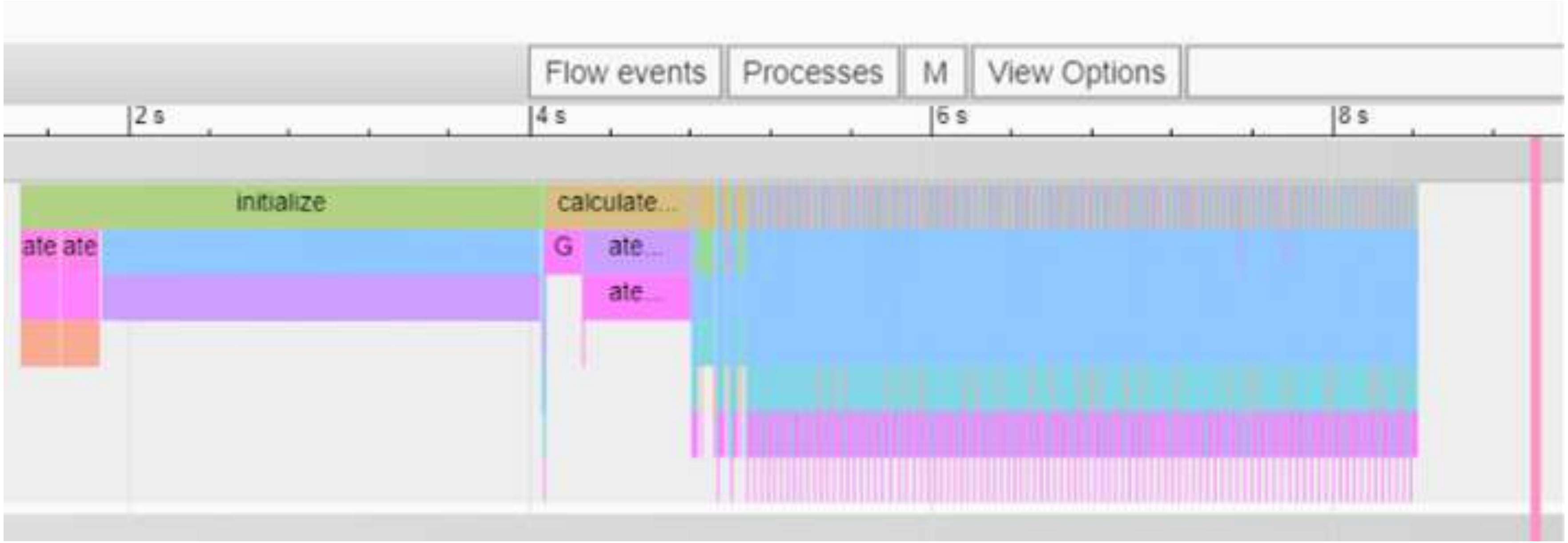
They use aggregate() function
= aggregate() is the bottleneck?

```python
class AGGNet(nn.Module):
    def __init__(self, num_hop, in_feats=1, hidden_feats=1, out_feats=1, dropout=0, use_bn=False):
        super(AGGNet, self).__init__()
        self.use_bn = use_bn
        self.dropout = dropout
        self.num_hop = num_hop

        self.convs = nn.ModuleList()

        if self.use_bn:
            self.bns = nn.ModuleList()
            self.bns.append(nn.BatchNorm1d(hidden_feats))

        self.convs.append(GraphConv(in_feats, out_feats, allow_zero_in_degree=True))

        for _ in range(self.num_hop - 1):
            if self.use_bn:
                self.bns.append(nn.BatchNorm1d(hidden_feats))
            self.convs.append(GraphConv(out_feats, out_feats, allow_zero_in_degree=True))

    def forward(self, graph):
        for i in range(self.num_hop - 1):
            feat = graph.ndata['feat']

            # graph = dgl.add_self_loop(graph)
            feat = self.convs[i](graph, feat)
            if self.use_bn:
                feat = self.bns[i](feat)
            feat = F.dropout(feat, p=self.dropout, training=self.training)
        feat = self.convs[-1](graph, feat)

        return feat
```

```python
# from aug.py

def aggregate(graph, agg_model):
    s_vec = agg_model(graph)
    return s_vec
```

Future work #1

Optimize aggregate() function (= AGGNet)

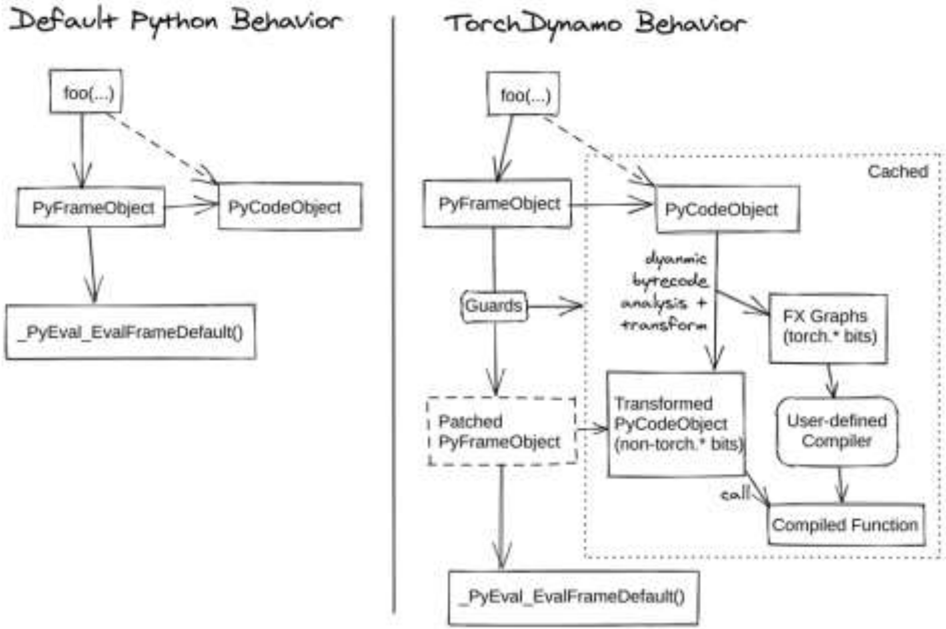


Augmentation loop

= Inside the loop is repeated a lot of times

TorchDynamo (Torch Script)

TorchDynamo is a Python-level JIT compiler for accelerating PyTorch code.



TorchDynamo (Torch Script)

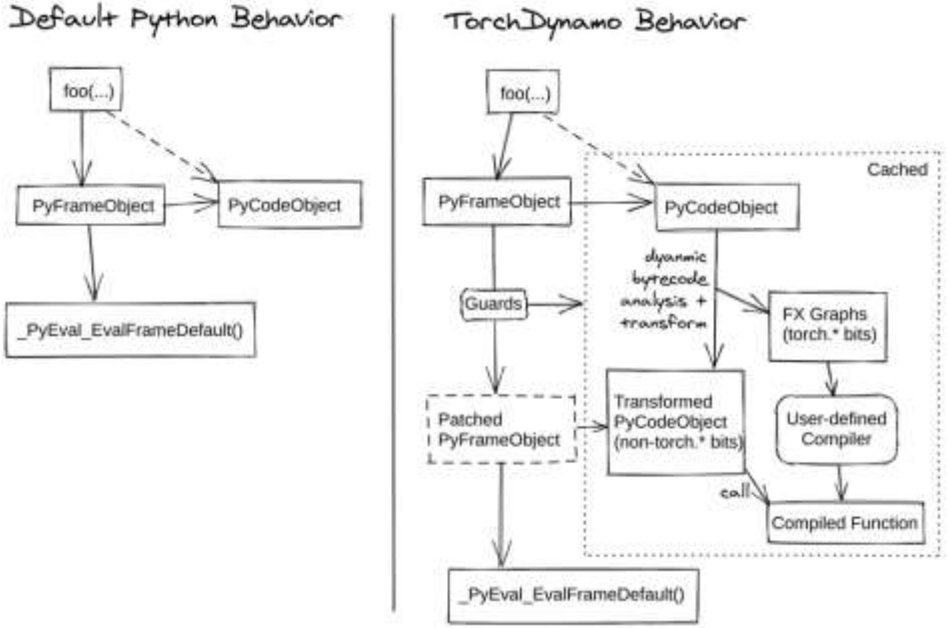
TorchDynamo dynamically modifies or “compiles” Python bytecode right before it is executed.

Then, it rewrites or “optimizes” Python bytecode in order to extract sequences of PyTorch operations.

Extracted PyTorch ops sequences are optimized, thus much faster than plain python codes or normal Pytorch code.

TorchDynamo

Furthermore, once compiled, these extracted PyTorch ops sequences are Cached.



```python
with record_function("initialize"):
    num_nodes, num_edges, coo_mat, n_list, agg_model = initialize()

while True:

    with record_function("calculate_delta_value"):
        delta_G_e, delta_G_e_aug, delta_G_v, delta_G_v_aug = calculate_delta_value()

    with record_function("graph_augmentation"):
        aug_g = graph_augmentation()

    with record_function("message_passing"):
        message_passing_g, message_passing_aug_g = message_passing()

    with record_function("calculate_ego_graph_message_passing_value"):
        org_ego = calculate_ego_graph_message_passing_value()

    with record_function("calculate_augmented_delta"):
        delta_g_e, delta_g_aug_e, delta_g_v, delta_g_aug_v = calculate_augmented_delta()

    with record_function("calculate_distribution"):
        h_loss_op = calculate_distribution()

    with record_function("compute_model"):
        output = compute_model()

    with record_function("compute_ent"):
        ent = compute_ent()

    with record_function("compute_p_aug"):
        p, p_aug = compute_p_aug()

    with record_function("compute_q_aug"):
        q, q_aug = compute_q_aug()
```

Future work #2

Compile loop contexts.

NOTE : TorchDynamo is beta; unstable and only works for Plain python code and PyTorch - unable to apply to external libraries;
e.g. DGL or Sci-py