GraphDL Libraries

Below are some of the libraries available for GraphDL:

- PyTorch Geometric (PyG)
- Tensorflow GNN (alpha release)
- Deep Graph Library (DGL)
- Graph4NLP: Deep learning on Graphs for NLP

GraphDL Libraries

PyTorch Geometric (PyG)

Tensorflow GNN (alpha release)

Deep Graph Library (DGL)

Graph4NLP: Deep learning on Graphs for NLP

PyG Architecture

Key Features:

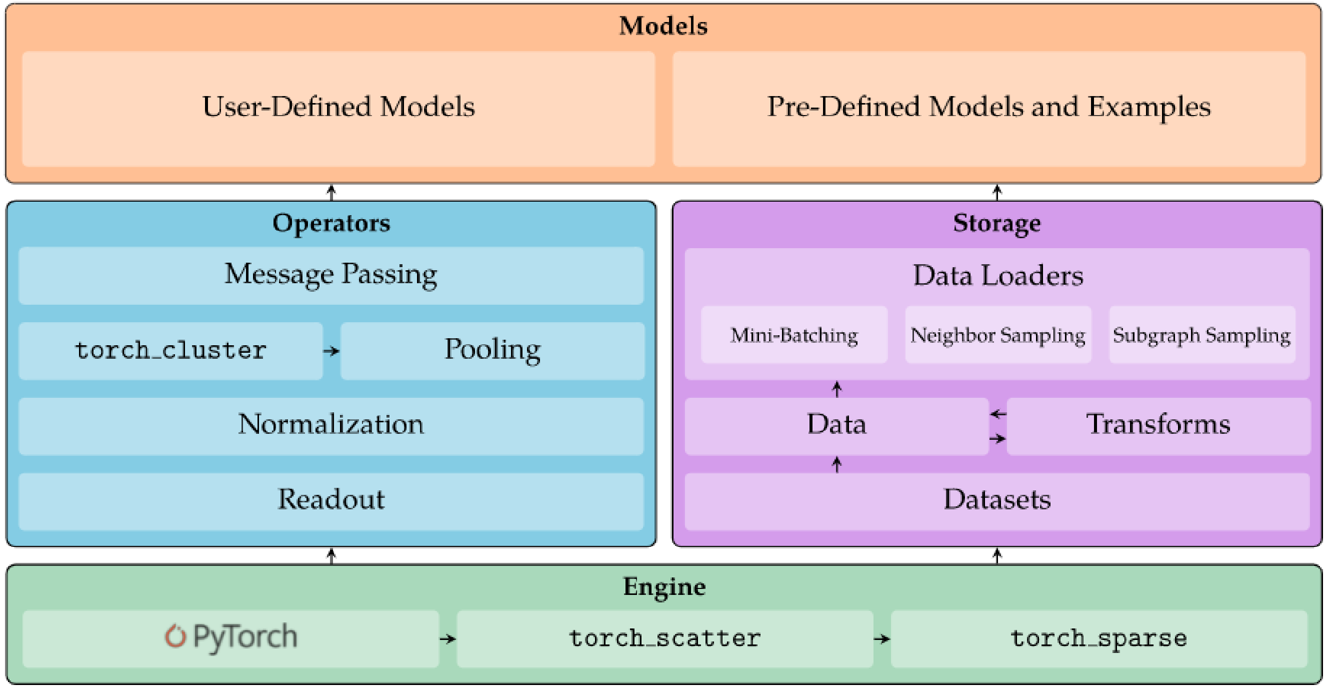
1. Easy-to-use: If you are already familiar with PyTorch, then utilizing PyG is straightforward!

2. State of the art GNN models: Models can be used as is or extended for research.

3. GraphGym integration: GraphGym lets users easily reproduce GNN experiments, can launch and analyze thousands of different GNN configurations.

4. Large scale real-world models: Scalable GNNs for graphs with millions of nodes; dynamic GNNs for node predictions over time; heterogeneous GNNs with multiple node types and edge types.

5. Heterogenous graphs support



https://pytorch-geometric.readthedocs.io/en/latest/index.html

PyG example code for 2-layer GCN

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 16)
        self.conv2 = GCNConv(16, dataset.num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

Initializing 2 GCN conv layers

https://pytorch-geometric.readthedocs.io/en/latest/index.html

GraphDL Libraries

PyTorch Geometric (PyG)

Tensorflow GNN (alpha release)

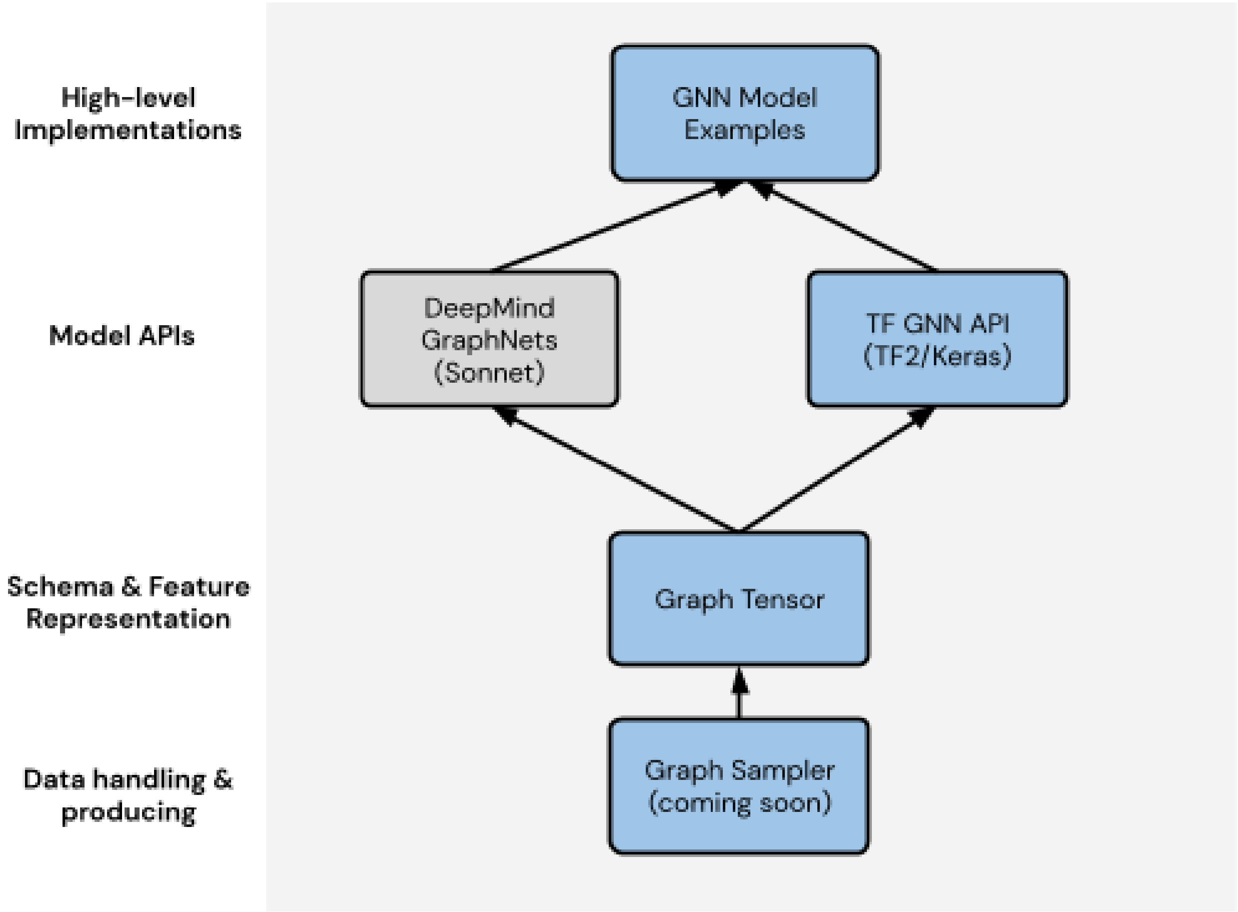
Deep Graph Library (DGL)

Graph4NLP: Deep learning on Graphs for NLP

TF-GNN Components (Initial release)

Key Features:

1. A high-level Keras-style API: To create GNN models that can easily be composed with other types of models.
2. Heterogenous graphs support
3. 'GraphTensor' composite tensor type: Holds graph data, can be batched, and has graph manipulation routines available.



https://github.com/tensorflow/gnn

GraphDL Libraries

PyTorch Geometric (PyG)

Tensorflow GNN (alpha release)

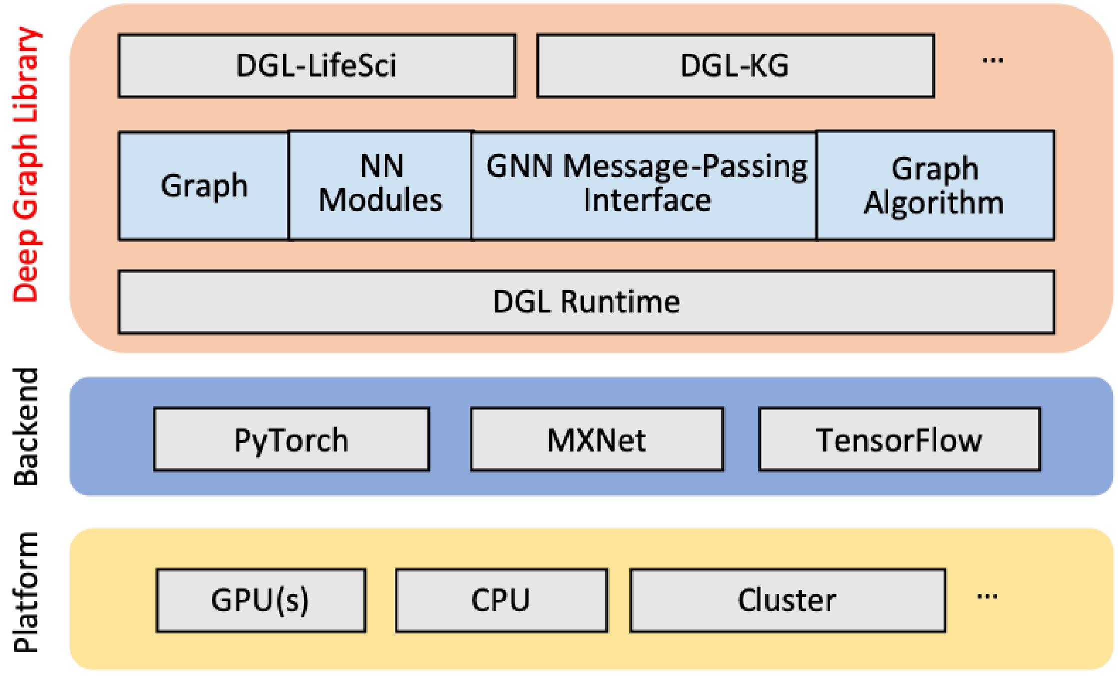
Deep Graph Library (DGL)

Graph4NLP: Deep learning on Graphs for NLP

DGL Architecture

Key Features:

1. GPU-ready graph library: Provides a powerful graph object that can reside on either CPU or GPU.
2. Models, modules and benchmarks for GNN researchers: Rich set of example implementations of popular GNN models of a wide range of topics.
3. Easy to learn and use: Plenty of learning materials for all kinds of users from ML researcher to domain experts.
4. Scalable and efficient: Convenient to train models using DGL on large-scale graphs across multiple GPUs or multiple machines.
5. Works with MXNet/Gluon, PyTorch and Tensorflow



https://github.com/dmlc/dgl

DGL Example: Graph Classification

```
from dgl.dataloading import GraphDataLoader
from torch.utils.data.sampler import SubsetRandomSampler

num_examples = len(dataset)
num_train = int(num_examples * 0.8)

train_sampler = SubsetRandomSampler(torch.arange(num_train))
test_sampler = SubsetRandomSampler(torch.arange(num_train, num_examples))

train_dataloader = GraphDataLoader(
    dataset, sampler=train_sampler, batch_size=5, drop_last=False)
test_dataloader = GraphDataLoader(
    dataset, sampler=test_sampler, batch_size=5, drop_last=False)
```

Split dataset into train and test

Dataloader loads 5 batches at once

DGL Example: Graph Classification

```
from dgl.nn import GraphConv

class GCN(nn.Module):
    def __init__(self, in_feats, h_feats, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, h_feats)
        self.conv2 = GraphConv(h_feats, num_classes)

    def forward(self, g, in_feat):
        h = self.conv1(g, in_feat)
        h = F.relu(h)
        h = self.conv2(g, h)
        g.ndata['h'] = h
        return dgl.mean_nodes(g, 'h')
```

Define 2 GCN layers

Take mean of all node-representations to get representation at graph-level

https://github.com/dmlc/dgl

GraphDL Libraries

PyTorch Geometric (PyG)

Tensorflow GNN (alpha release)

Deep Graph Library (DGL)

Graph4NLP: Deep learning on Graphs for NLP

Graph4NLP Architecture

Graph4NLP

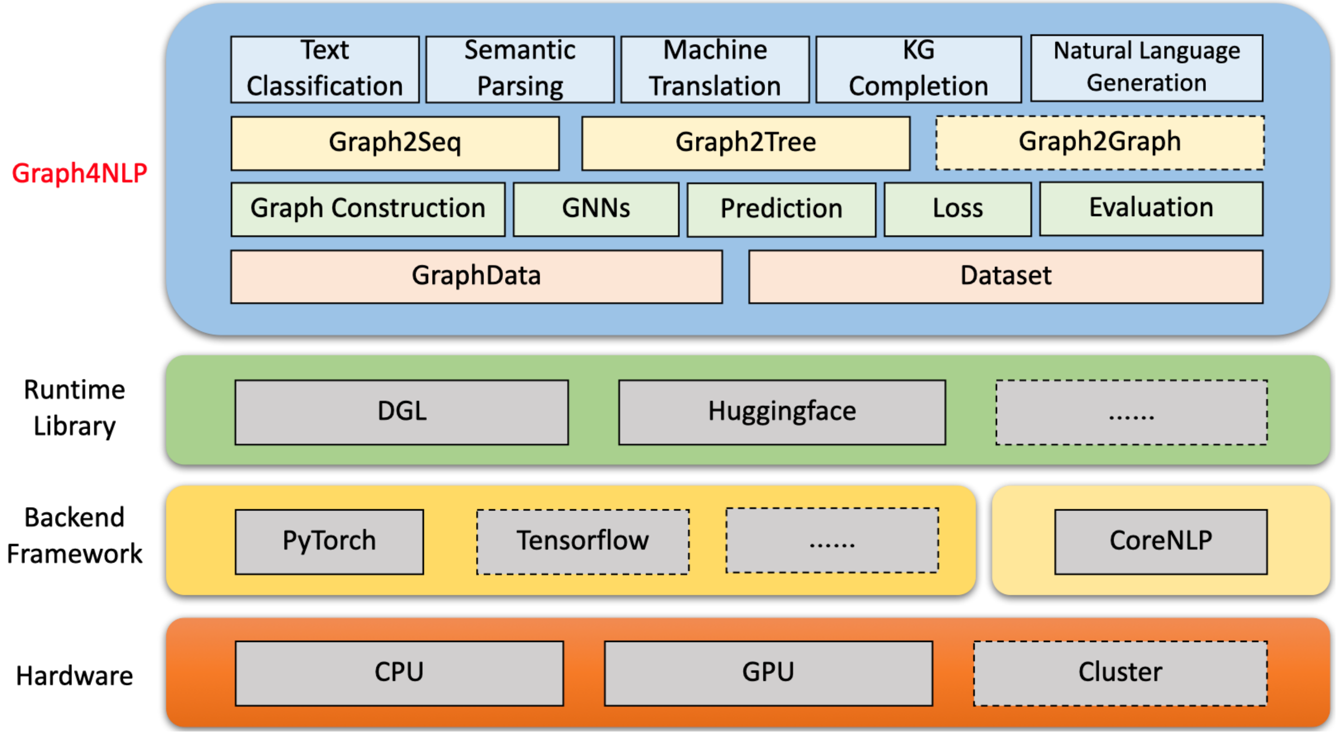
Key Features:

1. Easy-to-use and Flexible: Provides both full implementations of state-of-the-art models and also flexible interfaces to build customized models with whole-pipeline support.

2. Rich set of Learning Resources: Provide a variety of learning materials including code demos, code documentations, research tutorials and videos, and paper survey.

3. High running efficiency and extensibility: Build upon highly-optimized runtime libraries including DGL and provide highly modulized blocks.

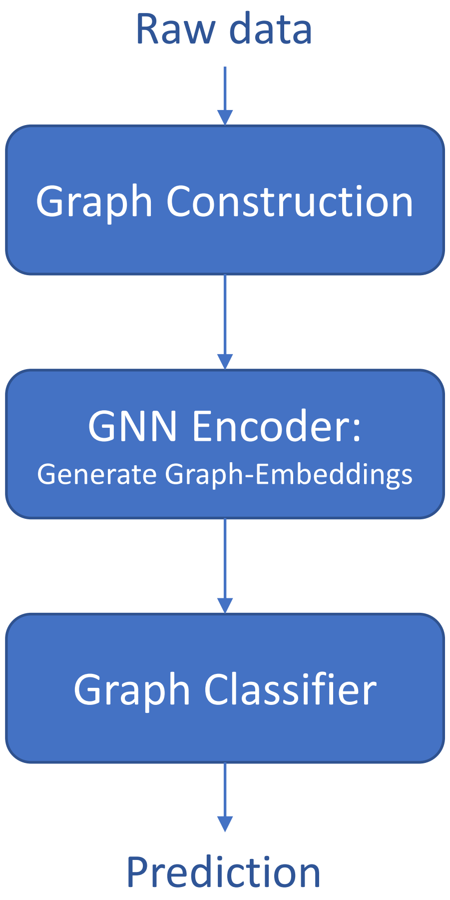
4. Comprehensive code examples: Provide a comprehensive collection of NLP applications and the corresponding code examples for quick-start.



https://dlg4nlp.github.io/index.html

Graph4NLP Example: Text Classification

Graph4NLP




```python
def forward(self, graph_list, tgt=None, require_loss=True):
    # build graph topology
    batch_gd = self.graph_topology(graph_list)

    # run GNN encoder
    self.gnn(batch_gd)

    # run graph classifier
    self.clf(batch_gd)
    logits = batch_gd.graph_attributes['logits']

    if require_loss:
        loss = self.loss(logits, tgt)
        return logits, loss
    else:
        return logits
```

```python
self.graph_topology = DependencyBasedGraphConstruction(
                        embedding_style=embedding_style,
                        vocab=vocab.in_word_vocab,
                        hidden_size=config['num_hidden'],
                        word_dropout=config['word_dropout'],
                        rnn_dropout=config['rnn_dropout'],
                        fix_word_emb=not config['no_fix_word_emb'],
                        fix_bert_emb=not config.get('no_fix_bert_emb', False))
```

```python
self.gnn = GAT(config['gnn_num_layers'],
               config['num_hidden'],
               config['num_hidden'],
               config['num_hidden'],
               heads,
               direction_option=config['gnn_direction_option'],
               feat_drop=config['gnn_dropout'],
               attn_drop=config['gat_attn_dropout'],
               negative_slope=config['gat_negative_slope'],
               residual=config['gat_residual'],
               activation=F.elu)
```

```python
self.clf = FeedForwardNN(2 * config['num_hidden'] \
                if config['gnn_direction_option'] == 'bi_sep' \
                else config['num_hidden'],
                config['num_classes'],
                [config['num_hidden']],
                graph_pool_type=config['graph_pooling'],
                dim=config['num_hidden'],
                use_linear_proj=config['max_pool_linear_proj'])

self.loss = GeneralLoss('CrossEntropy')
```

https://dlg4nlp.github.io/index.html

Scenario: How do we handle huge-datasets where each sample is a graph?

Mini-batching support

Mini-batching is crucial for letting the training of a deep learning model scale to huge amounts of data

Different graphs in the same dataset can have different no. of node and edges! How can we achieve parallelization then?

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_n \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_n \end{bmatrix}, \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_n \end{bmatrix}.$$

In PyG, **adjacency matrices are stacked in a diagonal fashion**, and node and target features are simply concatenated in the node dimension.

1. GNN operators that rely on a message passing scheme do not need to be modified since messages still cannot be exchanged between two nodes that belong to different graphs.

2. There is no computational or memory overhead: The resultant graph is stored in a sparse fashion.

https://pytorch-geometric.readthedocs.io/en/latest/index.html

Hands-on Session

Some starting points:

- Pubmed document-classification using GCN and GAT
- GAT on Cora dataset
- GCN on Cora dataset