1. Implementation Details

The project was implemented using Python 3.11 with key deep learning libraries:

1. Torch 2.2.1
2. PytorchGeometric (PyG) 2.5.3
3. Pytorch Lightning 2.2.1

The project pipelines and processing flow was implemented using Kedro framework (0.19.3) and MLFlow (2.11.1) to ensure replicability of experiments.

The project code is freely available on Github under the following address:
https://github.com/maddataanalyst/HexGIN

Some portions of the project, related to heterogeneous graph processing and extraction from structured data were moved to a separate implementation, that will be a part of a separate research paper in the future. This new project (as of July 2024 under development), called **HexTractor** is freely available on Github under the following address: https://github.com/maddataanalyst/hextractor.

2. HexGIN and GIN Equivalence

Provided that all neighboring nodes u across all relations share the same dimensionality d, the relational mapping neural network is formulated as an identity function $\mathbf{\Phi}_{\boldsymbol{r}}^{(k)}: \mathbb{R}^{\mathrm{d}} \rightarrow \mathbb{R}^{\mathrm{d}}$, AGG function is a simple summation, and the node-update neural network $\mathbf{\Phi}_{\boldsymbol{v}}^{(k)}$ has its first layer as non-learnable 1d convolutional layer with identity kernel of size d: $k = [1, 1, \dots, 1], k \in \mathbb{N}^{\mathrm{d}}$, stride equal to 1 and dilation parameter (O'Shea & Nash, 2015) set to d, then we can express equations (2), (3) and (4) in the form presented below.

$$m_{u,r\in\,\mathcal{N}_r(v)}^{(k)} = \sum_{r\in R(v)} \sum_{u\in\mathcal{N}_r(v)} h_{u,r}^{(k-1)} = \sum\nolimits_{u\in\,\mathcal{N}(v)} h_u^{(k-1)},$$

$$h_v^{(k)} = \mathbf{\Phi}_{\boldsymbol{v}}^{(\boldsymbol{k})}\left(\left(1+\epsilon^{(k)}\right)h_v^{(k-1)} \,||\, m_v^{(k)}\right) =$$

$$= \mathrm{MLP}^{(\mathrm{k})}\left(\boldsymbol{conv1d}_{\boldsymbol{k=\mathbb{I}}}^{\boldsymbol{dil=du}}\left(\left(1+\epsilon^{(k)}\right)h_v^{(k-1)} \,||\, m_v^{(k)}\right)\right)$$

$$\equiv \mathrm{MLP}^{(\mathrm{k})}\left(\left(1+\epsilon^{(k)}\right)h_v^{(k-1)} + \sum\nolimits_{u\in\,\mathcal{N}(v)} h_u^{(k-1)}\right) \blacksquare$$

Considered the properties and the behavior of the convolution operator with selected dilatation and stride parameters, the formulation of HexGIN and original GIN (Xu et al., 2018) can be equivalent. That formulation is directly equivalent to the original GIN definition provided by equation (1) in the paper.

3. Model Architectures

3.1. Technical Aspects

For GraphSAGE model, the official implementation, aligned with the original paper, from the library was used.

HexGIN was implemented using PyTorch Geometric basic components and elements, therefore it is new, original model.

3.2. Embedding and Preprocessing Layers

Each model in the study utilized the same embedding and preprocessing layers architecture. The original entity, filing and country IDs were passed to embedding layers that mapped them to initial numerical representation vectors (embeddings).

1. Entity embedding dimensionality: 64.
2. Filing embedding dimensionality: 64.
3. Country embedding dimensionality: 12.

This process can be depicted as in the Figure 1.

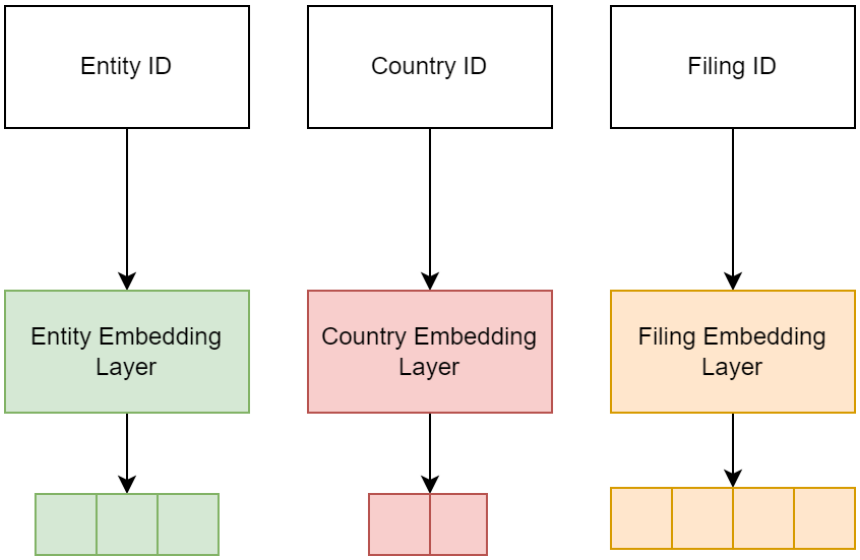

Figure 1. Entity, Country and Filing embedding

Source: own work.

Each set of colored squares at the bottom represents numerical vector representation (embedding).

3.3. GraphSAGE Architecture

GraphSAGE model consists of:

1. Embedding layers, as described above.
2. Two graph convolution layers (SAGE Convolution aligned with the paper – the official implementation), separate for each relation type.
3. Linear link prediction layer for beneficiary detection.

Detailed model characteristics are presented below (output from the library):

```
+-----------------------------------------+--------------+----------------+----------+
| Layer                                   | Input Shape  | Output Shape   | #Param   |
|-----------------------------------------+--------------+----------------+----------|
| GraphModule                             |              |                | -1       |
| ├─(conv1)ModuleDict                     | --           | --             | -1       |
| │    └─(entity__sends__filing)SAGEConv  | [2, 4507]    | [4507, 64]     | -1       |
| │    └─(filing__benefits__entity)SAGEConv | [2, 1894]  | [2823, 64]     | -1       |
| │    └─(filing__concerns__entity)SAGEConv | [2, 24491] | [2823, 64]     | -1       |
| ├─(hidden_act)ModuleDict                | --           | --             | --       |
| │    └─(entity)ReLU                     | [2823, 64]   | [2823, 64]     | --       |
| │    └─(filing)ReLU                     | [4507, 64]   | [4507, 64]     | --       |
| ├─(batch_norm1)ModuleDict               | --           | --             | 256      |
| │    └─(entity)BatchNorm                | [2823, 64]   | [2823, 64]     | 128      |
| │    │    └─(module)BatchNorm1d         | [2823, 64]   | [2823, 64]     | 128      |
| │    └─(filing)BatchNorm                | [4507, 64]   | [4507, 64]     | 128      |
| │    │    └─(module)BatchNorm1d         | [4507, 64]   | [4507, 64]     | 128      |
| ├─(conv2)ModuleDict                     | --           | --             | -1       |
| │    └─(entity__sends__filing)SAGEConv  | [2, 4507]    | [4507, 32]     | -1       |
| │    └─(filing__benefits__entity)SAGEConv | [2, 1894]  | [2823, 32]     | -1       |
| │    └─(filing__concerns__entity)SAGEConv | [2, 24491] | [2823, 32]     | -1       |
| ├─(batch_norm2)ModuleDict               | --           | --             | 128      |
| │    └─(entity)BatchNorm                | [2823, 32]   | [2823, 32]     | 64       |
| │    │    └─(module)BatchNorm1d         | [2823, 32]   | [2823, 32]     | 64       |
| │    └─(filing)BatchNorm                | [4507, 32]   | [4507, 32]     | 64       |
| │    │    └─(module)BatchNorm1d         | [4507, 32]   | [4507, 32]     | 64       |
| ├─(out_act)ReLU                         | --           | --             | --       |
+-----------------------------------------+--------------+----------------+----------+
```

3.4. MLP Architecture

MLP is a basic neural network, that operates on matrix/tabular data. It shares the same preprocessing, embedding layers as remaining models in the study. Detailed model characteristics are presented below (output from the library):

```
+-------------------------------------+--------------+----------------+----------+
| Layer                               | Input Shape  | Output Shape   | #Param   |
|-------------------------------------+--------------+----------------+----------|
| MlpModel                            | [7330, 7330] | [811, 1]       | 544,137  |
| ├─(embed_model)FinCENPreprocModel   |              |                | 465,648  |
| │    └─(embedding_layers)ModuleDict | --           | --             | 465,648  |
| │    │    └─(entity)Embedding       | [2823]       | [2823, 64]     | 175,552  |
| │    │    └─(filing)Embedding       | [4507]       | [4507, 64]     | 288,512  |
| │    │    └─(country)Embedding      | [2823]       | [2823, 12]     | 1,584    |
| ├─(h_layers)Sequential              | [811, 140]   | [811, 1]       | 78,489   |
| │    └─(0)BatchNorm1d               | [811, 140]   | [811, 140]     | 280      |
| │    └─(1)Linear                    | [811, 140]   | [811, 256]     | 36,096   |
| │    └─(2)LeakyReLU                 | [811, 256]   | [811, 256]     | --       |
| │    └─(3)BatchNorm1d               | [811, 256]   | [811, 256]     | 512      |
| │    └─(4)Linear                    | [811, 256]   | [811, 128]     | 32,896   |
| │    └─(5)LeakyReLU                 | [811, 128]   | [811, 128]     | --       |
| │    └─(6)BatchNorm1d               | [811, 128]   | [811, 128]     | 256      |
| │    └─(7)Linear                    | [811, 128]   | [811, 64]      | 8,256    |
| │    └─(8)LeakyReLU                 | [811, 64]    | [811, 64]      | --       |
| │    └─(9)BatchNorm1d               | [811, 64]    | [811, 64]      | 128      |
| │    └─(10)Linear                   | [811, 64]    | [811, 1]       | 65       |
+-------------------------------------+--------------+----------------+----------+
```

3.5. HexGIN Architecture

HexGIN model contains two HexGIN convolutions, which are modified version of Graph Isomorphism Networks convolutions, as described in the paper.

It shares the same preprocessing, embedding layers as remaining models in the study, and the same beneficiary link prediction layer at the end.

Detailed model characteristics are presented below (output from the library):

```
+-----------------------------------+--------------+----------------+----------+
| Layer                             | Input Shape  | Output Shape   | #Param   |
|-----------------------------------+--------------+----------------+----------|
| HeteroGNNLinkPredModel            | [7330, 7330] | [811, 1]       | 631,347  |
| ├─(preproc_model)FinCENPreprocModel |            |                | 465,648  |
| │    └─(embedding_layers)ModuleDict | --         | --             | 465,648  |
| │    │    └─(entity)Embedding     | [2823]       | [2823, 64]     | 175,552  |
| │    │    └─(filing)Embedding     | [4507]       | [4507, 64]     | 288,512  |
| │    │    └─(country)Embedding    | [2823]       | [2823, 12]     | 1,584    |
| ├─(gnn_conv)HexGINModel           |              |                | 163,586  |
| │    └─(hexgin_layers)ModuleList  | --           | --             | 163,586  |
| │    │    └─(0)HexGINLayer       |              |                | 117,055  |
| │    │    └─(1)HexGINLayer       |              |                | 46,531   |
| ├─(link_pred_subnet)Sequential    | [811, 64]    | [811, 1]       | 2,113    |
| │    └─(0)Linear                  | [811, 64]    | [811, 32]      | 2,080    |
| │    └─(1)LeakyReLU               | [4507, 128]  | [4507, 128]    | --       |
| │    └─(2)Linear                  | [811, 32]    | [811, 1]       | 33       |
+-----------------------------------+--------------+----------------+----------+
```

A single HexGIN convolution can be depicted as the following model:

```
HexGINConv(nn=Sequential(
  (0): BatchNorm1d(216, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (1): Linear(in_features=216, out_features=128, bias=True)
  (2): LeakyReLU(negative_slope=0.005)
  (3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): LeakyReLU(negative_slope=0.005)
))
```

References

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *ArXiv: 1511.08458*.
Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How Powerful are Graph Neural Networks? *ArXiv: 1810.00826*.