(1) Data Sampling:

Step 1: Down-sample the majority class

- Omit many of the majority class examples from the training data to make the dataset more balanced

Step 2: Upweight the down-sampled class

- Down-sampling introduces "predictive bias" by showing the model an artificial world where the classes are more balanced than the real world
- To solve this, we "upweight" the loss in the majority class by the factor by which we have down-sampled the majority class

Idea Reference: https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets

(2) Transactional Graph Construction (using Pytorch-geometric):

- Splitting of the dataset into train-test-validation temporally so that future information is not leaked during the training
- Performed SMOTE-like oversampling by duplicating fraud transactions with slight noise in training dataset keeping validation and test dataset intact to increase the number of fraud transactions in the training dataset

(3) Graph Transformers Model (i.e. GAT model):

Link to the paper on GAT : https://arxiv.org/pdf/1710.10903

The graph attentional operator from the "Graph Attention Networks" paper.

$$\mathbf{x}_i' = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{i,j} \mathbf{\Theta}_t \mathbf{x}_j,$$

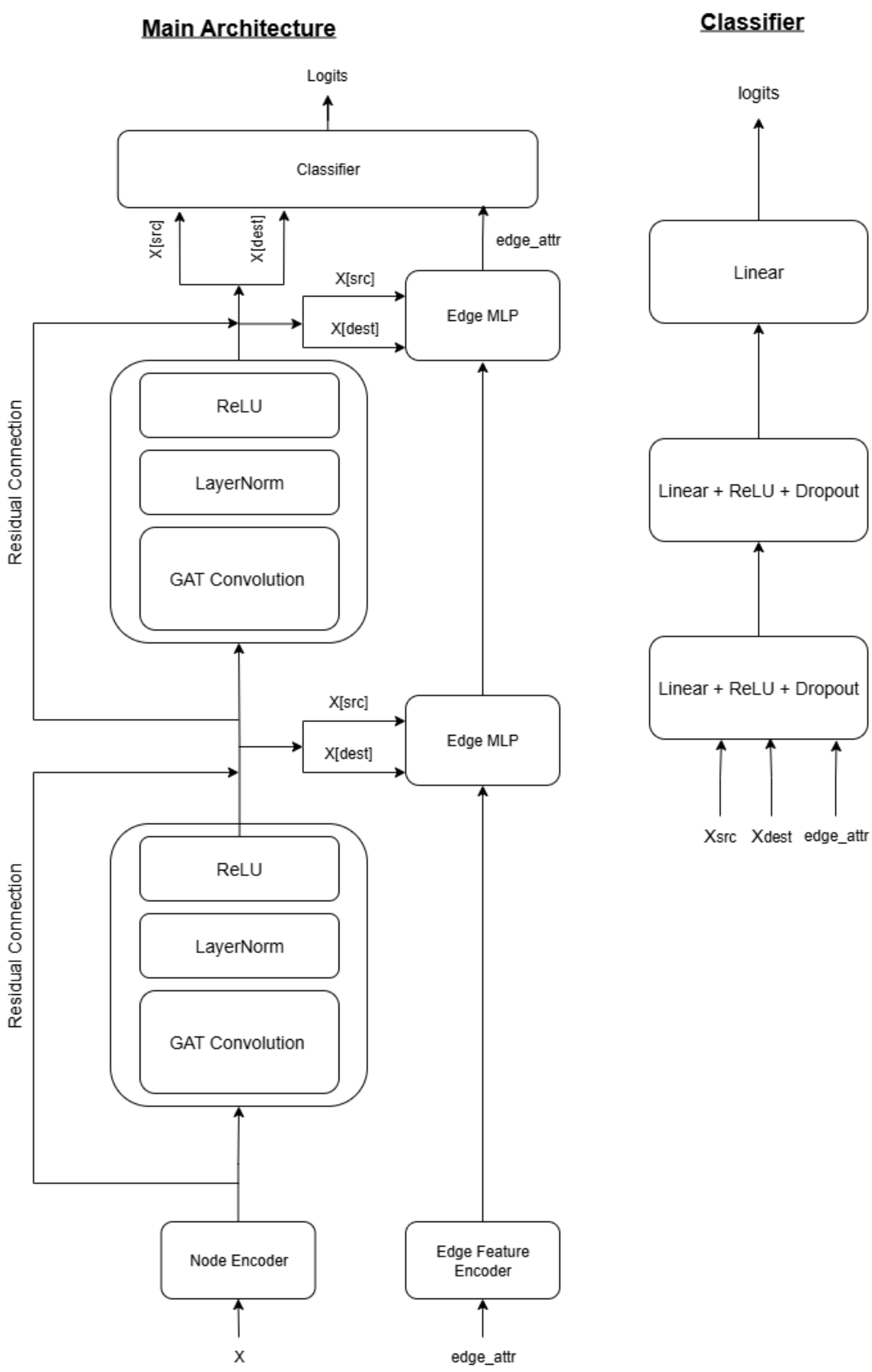
where the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_j\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_k\right)\right)}.$$

If the graph has multi-dimensional edge features $\mathbf{e}_{i,j}$, the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_j + \mathbf{a}_e^\top \mathbf{\Theta}_e \mathbf{e}_{i,j}\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}_s^\top \mathbf{\Theta}_s \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta}_t \mathbf{x}_k + \mathbf{a}_e^\top \mathbf{\Theta}_e \mathbf{e}_{i,k}\right)\right)}.$$

If the graph is not bipartite, $\mathbf{\Theta}_s = \mathbf{\Theta}_t$.


Main Architecture
Logits
Classifier
X[src]
X[dest]
edge_attr
X[src]
X[dest]
Edge MLP
ReLU
LayerNorm
GAT Convolution
Residual Connection
X[src]
X[dest]
Edge MLP
ReLU
LayerNorm
GAT Convolution
Residual Connection
Node Encoder
Edge Feature Encoder
X
edge_attr
Classifier
logits
Linear
Linear + ReLU + Dropout
Linear + ReLU + Dropout
Xsrc
Xdest
edge_attr

Training:

(1) BCE Loss $= -[pos_{weight} * y * log(p) + (1 - y) * log(1 - p)]$

Where:
y = true label(0 or 1)
p = predicted probability
pos_{weight} = $class\_count[0]$ / $class\_count[1]$

(2) Focal Loss = $-[\ \alpha\ *\ (1-p)\text{^}\ \gamma\ *\ log(p)]$

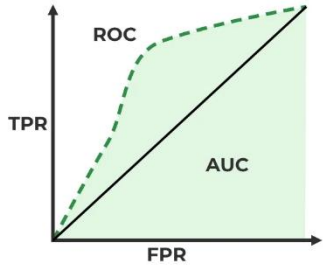
Where:
α (alpha): Balances positive/negative examples $(= 1\ -\ (fraud\_count)\ /\ total)$
γ (gamma): Focuses learning on hard examples $(= 2)$
p: Model's confidence for the correct class

(3) BCE with logits = $\begin{cases} \log\big(1\ +\ \exp(-\text{logit})\big) = \log\ (\text{sigmoid}(+\text{logit})), & target = 1 \\ \log\big(1\ +\ \exp(+\text{logit})\big) = \log\ (\text{sigmoid}(-\text{logit})), & target = 0 \end{cases}$

(4) Total Loss = 0.7 * Focal Loss + 0.3 * BCE Loss

Evaluation:

(1) ROC-AUC curve (Receiver Operating Characteristic Area under the curve):
- Curve of TPR(true positive rate) vs FPR(false positive rate)
- $TPR\ =\ TP\ /\ (TP\ +\ FN),\ FPR\ =\ FP\ /\ (FP\ +\ TN)$
- Ideally, TPR should be 1 and FPR should be 0



(2) $Precision\ =\ TP\ /\ (TP\ +\ FP)$
(3) $Recall\ =\ TP\ /\ (TP\ +\ FN)$
(4) $F1-Score\ =\ (2\ *\ Precision\ *\ Recall)\ /\ (Precision\ +\ Recall)$