Temporal Link Prediction with GConvGRU on Dynamic Graphs

Abdurrahman Kürşat Özkan
Department of Computer Science, Akdeniz University, Antalya, Türkiye
Email: kursatozkan.job@gmail.com

Alperen Cevahiroğlu
Department of Computer Science, Akdeniz University, Antalya, Türkiye
Email: 20200808016@ogr.akdeniz.edu.tr

***Abstract*—We present a modular pipeline for temporal link prediction on large-scale dynamic graphs, built with PyTorch Geometric Temporal. Key components include ID remapping and streaming sampling to handle 27M+ edges, a custom `TemporalLinkPredictionDataset`, a recurrent GNN encoder (GConvGRU with $K = 2$ hops and hidden size 64), and a two-layer MLP link scorer. We validate on held-out future edges (last 10% of timestamps) and evaluate using ROC AUC, average precision, and PR AUC. Our experiments on a sampled subset (2.7M edges) demonstrate robust performance and clear guidelines for scaling to full datasets.**

***Index Terms*—Temporal link prediction, dynamic graphs, GConvGRU, negative sampling, streaming preprocessing**

I. Introduction

Temporal link prediction forecasts future interactions in evolving networks, important for applications such as social event forecasting, e-commerce demand, and recommender systems. Static methods ignore sequence effects; our approach models spatial-temporal dependencies via a graph-convolutional GRU, enabling richer representations of node dynamics. We describe a full workflow: preprocessing, dataset construction, model design, training, validation query generation, inference, and evaluation.

II. Data Preprocessing

Handling 27 million raw edges on an Apple M1 requires streaming and sampling:

- **ID map generation:** `build_id_maps` reads the first column of `node_features.csv` and `edge_type_features.csv`, enumerates unique raw IDs to contiguous zero-based integers, and writes `node_id_map.json` and `etype_id_map.json`.
- **Static feature remapping:** `remap_and_save_static` applies these maps to the static feature tables, producing `node_features_mapped.csv` and `edge_type_features_mapped.csv`.
- **Streaming edge sampling:** `remap_edges_stream` reads `edges_train_A.csv` in 1,000,000-row chunks, randomly samples a fraction (`sample_frac`=0.000005) of each chunk (yielding approximately 5 edges per chunk), remaps source, destination, and edge-type IDs, and appends the results to `edges_train_A_mapped.csv`. A fixed random seed (`seed`=42) ensures reproducibility.

III. Dataset Construction

In `src/dataset.py`, we implement `TemporalLinkPredictionDataset`:

- Load remapped edges (columns `src,dst,etype,ts`) and sort unique timestamps $\{t_0, \ldots, t_T\}$.
- For each index i:
  1) Historical edges $\{e \mid ts_e \leq t_i\}$ form graph snapshot G_{t_i} with `edge_index`.
  2) Positive pairs: edges with $t_i < ts_e \leq t_{i+1}$.
  3) Negative pairs: uniformly sample K random node pairs (K equals the number of positives).
  4) Return `x` (node features), `edge_index`, `pairs`, and `labels`.

IV. Model Architecture

Our `GConvGRULinkPredictor` (in `src/model.py`) comprises:

- **GConvGRU encoder:** $F \to H$ node feature transform with two graph-convolutional hops ($K = 2$) and hidden dimension $H = 64$.
- **Link MLP:** concatenate $h_u, h_v \in \mathbb{R}^H$ to $\mathbb{R}^{2H}$, then Linear$(2H, H)$–ReLU–Linear$(H, 1)$–Sigmoid for edge probability.

V. Training

Scripts in `src/train.py` perform training with:

- `DataLoader(batch_size=None)` streams one snapshot per iteration.
- Device: `mps` if available, otherwise `cpu`.
- Optimizer: Adam (lr $= 10^{-3}$); Loss: Binary Cross-Entropy.
- Epochs and hidden size configurable via CLI flags (`--epochs`, `--hidden`).
- Progress tracked via `tqdm`.
- Final model saved to `model/model.pth`.

VI. Validation Query Generation

We hold out the latest 10% of edges by timestamp using `make_val.py`:

1) Compute cutoff $t_{\text{cut}} = \text{quantile}(ts, 0.9)$.
2) Positives: edges with $ts > t_{\text{cut}}$ in windows $(t-1,\, t]$, label $= 1$.
3) Negatives: an equal number of random (src, dst) pairs with random edge types in the same windows, label $= 0$.
4) Shuffle and export to `data/val_queries.csv`.

VII. Inference and Evaluation

Inference (`src/inference.py`) loads the checkpoint, computes full-graph embeddings, and scores queries into `output/*.csv`. Evaluation (`src/evaluate.py`) computes:

- **ROC AUC**: $\text{roc\_auc\_score} = 0.7908$
- **Average Precision**: $\text{average\_precision\_score} = 0.8479$
- **PR AUC**: area under the precision-recall curve $= 0.8429$

VIII. Discussion

Our pipeline achieves efficient preprocessing and strong performance on held-out data:

- The GConvGRU encoder effectively captures temporal dynamics, yielding a ROC AUC of 0.7908.
- Streaming sampling and negative sampling strategies proved critical for learning with large-scale data.
- Precision–Recall metrics demonstrate high confidence in top-ranked predictions (AP 0.8479, PR AUC 0.8429).

Future work includes scaling to the full 27M-edge graph via streaming TGN and exploring mixed-precision on MPS.

IX. Conclusion

We deliver a scalable, modular system for temporal link prediction, validated on a large dynamic graph sample. The method generalizes to other domains requiring time-aware interaction forecasting.