A Appendix

A.1 Seastar

Seastar [9] is a system for programming GNN models using a vertex-centric user-defined function. The benefit of this approach is two-fold. A deep-learning practitioner can implement the GNN logic quickly, and a learner can ascertain the model's purpose from the vertex-centric implementation. Seastar also optimizes models like GCN [5] and GAT [6] by bridging the gaps in other GNN frameworks like DGL and PyG. Seastar takes a vertex-centric program as input. It returns an executor object capable of invoking a series of CUDA kernels during forward and backward propagation of the GNN model. This process is made possible with the help of the following components:-

1. **Tracer**: infers all operations on the central node and its neighboring nodes from the vertex-centric function and produces the graph-aware intermediate representation (GIR).
2. **GIR**: The Graph-aware Intermediate Representation (GIR) is a directed acyclic graph (DAG) representing the computational graph. Each node in the DAG represents an operation, for which the input and outputs are both tensors.
3. **GIR Optimizations and Code Generation**: GIR optimizations such as dead-code elimination, common subexpression elimination, and operator fusion are introduced before the code generation phase. The code generation phase emits CUDA kernels for both forward and backward propagation.
4. **Runtime Executor** The CUDA kernels for both forward and backward passes are wrapped in an executor object. At runtime, the executor invokes the backend (PyTorch, TensorFlow, etc.) to run the generated CUDA kernels for forward and backward propagation.

Seastar is successful in achieving lesser memory consumption and faster execution in comparison to PyG and DGL. However, Seastar can only represent simple GNN models such as GCN and GAT, with no support for TGNNs.

A.2 Other Frameworks

A majority of real-world graphs exhibit dynamic characteristics, as seen in social networks, traffic-flow networks, etc. Dynamic graphs hold immense potential and are packed with rich information, which necessitates the development of fast and efficient frameworks curated for writing and training TGNN models. PyG-T [10] is the first open-source deep-learning tool for training TGNNs on evolving graphs. However, PyG-T is built on top of PyG which suffers from performance and memory issues itself. Additionally, when dealing with dynamic graphs, PyG-T stores each graph snapshot separately which in turn incurs high memory overhead. Hence, there is a need for a fast and memory-efficient framework for building TGNNs. Table 1 summarizes properties of a list of deep learning libraries on graphs. *The `STGraph` framework is the only framework, listed in Table 1, that offers a backend-agnostic approach to processing TGNNs.*

Table 1: Deep Learning Libraries on Graphs

Library	Backend	Static Graph	Temporal Graph
PyTorch Geometric (PyG)	PyTorch	✓	×
DGL	Agnostic	✓	×
GraphNets	TensorFlow	✓	×
Spektral	TensorFlow	✓	×
Seastar	Agnostic	✓	×
PyTorch Geometric Temporal (PyG-T)	PyTorch	✓	✓
`STGraph`	Agnostic	✓	✓

A.3 Experimental Evaluation

The hardware platform used for the experimental evaluation of the proposed work is equipped with an Intel(R) Xeon(R) Bronze 3204 CPU (1.90GHz) and an NVIDIA GeForce RTX 2080 Ti GPU with device memory of 11GB. `STGraph` is written in Python 3.10 using Pytorch-2.0.0 [14] as the backend and CUDA Python 12.1.0 as the CUDA host API. The generated kernels are compiled using CUDA

11.7. The graph abstractions use CUDA C++ routines exposed via PyBind11-2.10.4 [15]. Thrust [16] and CUB libraries provide additional support in developing these routines.

Table 2: Summary of Benchmarking Datasets

S.No	Dataset	# Nodes	# Edges	Graph Type
1	Wikipedia Vital Mathematics (WVM)	1068	27K	Static
2	Windmill Output (WO)	319	102K	Static
3	Hungary Chickenpox (HC)	20	102	Static
4	Montevideo Bus(MB)	675	690	Static
5	Pedal Me (PM)	15	225	Static
6	sx-superuser	194K	1443K	Dynamic
7	sx-stackoverflow *	194K	2000K	Dynamic
8	sx-mathoverflow	25K	507K	Dynamic
9	sx-askubuntu	159k	964K	Dynamic
10	email-Eu-core	986	332K	Dynamic

* The dataset has been pruned to the first 2 million edges, which is the maximum size supported on our experimental hardware

Datasets. We perform the experimental evaluation on a total of ten graph datasets (See Table 2), with five being static-temporal datasets (1-5) [10], and the other five being dynamic graph datasets (6-10) [12].

Baseline. The `STGraph` framework is compared against PyG-T v0.54.0. The model considered for this comparison is the default configuration of TGCN [17] since it serves as a basic TGNN model with both temporal and GNN components. Each test was run for hundred epochs. The first three epochs were ignored to account for GPU warm-up time. The loss for models compiled with PyG-T and `STGraph` are similar over all tests.

Tasks. The static-temporal graph datasets considered in this work contain node labels for each timestamp, these datasets are hence trained on the *node classification task* using *Mean Squared Error* as the loss criterion. The *link prediction task* is used for benchmarking on DTDGs because these datasets mostly contain information about the presence/absence of edges at different timestamps. The *Binary Cross Entropy Loss with Logits* criterion calculates the loss in this case.

Data Preprocessing. For static-temporal graphs no preprocessing was necessary. Dynamic graph datasets considered in this benchmarking contain a list of edges with their corresponding timestamps. The datasets are preprocessed to discrete-time snapshots. The first half of the dataset is the first snapshot. Then the window is moved to obtain a second snapshot such that the percent change between any two consecutive snapshots is always less than 10%. In the case of `STGraph-GPMA`, the data is further processed to only contain changes between consecutive snapshots, i.e., the addition and deletion of edges.

A.4 Acknowledgement

This project is supported by the National Supercomputing Mission (NSM), Department of Science and Technology (DST), India. Additionally, we thank Dr. Muralikrishnan K, the point of contact for this project at the National Institute of Technology Calicut, for his guidance and support.