GNNPointerNet: A Data-driven Approach to Generate 2D Delaunay Triangular Mesh

Wanzhou Lei [1] *Advised by Prof. Per-Olof Persson*

[1]University of California, Berkeley

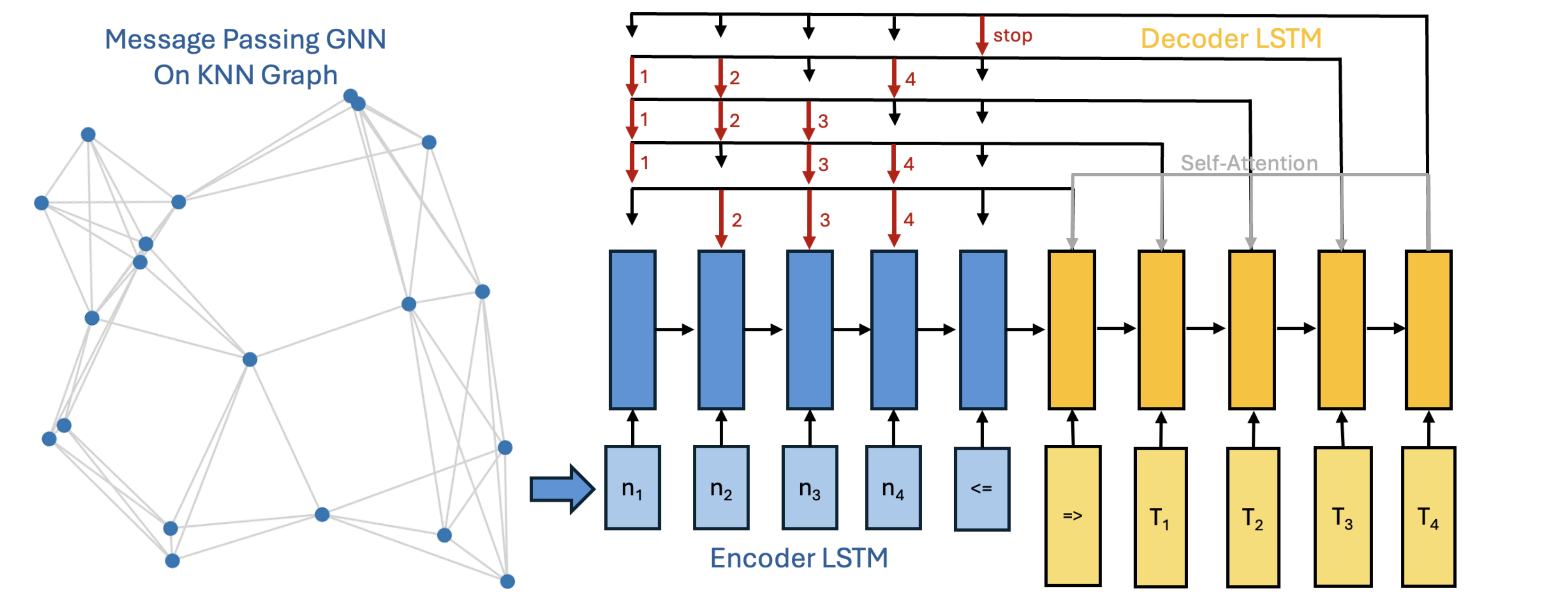
Introduction

Tetrahedral meshing plays a critical role in computational simulations, with Delaunay triangulation serving as a foundational deterministic method. However, classical Delaunay algorithms often generate meshes with problematic elements, such as slivers or poorly-shaped tetrahedra, adversely affecting numerical stability and simulation accuracy. Modern refinement techniques, including Weighted Squared Volume Minimization, Advancing-Front methods, and parallel refinement algorithms, have improved mesh quality but inherently remain constrained by their dependence on initial Delaunay triangulations. These refinement methods optimize existing meshes rather than constructing optimal structures from the outset.

Motivated by these limitations, this research proposes a data-driven approach leveraging machine learning to generate tetrahedral meshes directly. We introduce GNNPointerNet, a novel model combining a Message Passing Neural Network (MPNN) and a Pointer Network implemented through an LSTM architecture. As an initial step, our framework targets 2D triangulation to validate model performance against standard 2D Delaunay methods, ensuring feasibility before extending to complex 3D meshing.

Model Structure

The GNNPointerNet consists consists of two parts: a message passing GNN and a Pointer Network that is made of a LSTM. The model takes a sequence of N 2D points coordinates $\{(x_i, y_i)\}_{i=1}^N$ as input and outputs a sequence of triangles $\{\tilde{T}_i\}_{i=1}^m$. Below is the figure of model structure:



Given inputs $\{(x_i, y_i)\}_{i=1}^N$, the model first builds an undirected kNN graph. The initial node embeddings are inputs and the initial edge embeddings are 4D vectors that contains the positional, orientation and length information of each edge. After several layers of message passing, we obtain final node embeddings $\{n_i\}_{i=1}^N$. The LSTM only contains one layer. The encoder block takes current node embedding n_t, previous long term memory state h_{t-1}^e, previous short term memory state c_{t-1}^e as inputs:

$$h_t^e, c_t^e = EncoderBlock(n_t, h_{t-1}^e, c_{t-1}^e)$$

$h_t^e \in \mathbb{R}^H$, where hyperparameter H is the hidden dimension of LSTM. After processing N input nodes, at $N+1^{th}$ step, the model takes an embedding vector "<=" of stop token and produces a final hidden state h_{N+1}^e. "<=" is a model parameter. We used teacher forcing to train the network. At t^{th} step of the decoder LSTM:

$$h_t^d, c_t^d = DecoderBlock(T_{t-1}, h_{t-1}^d, c_{t-1}^d)$$

Model Structure (Continued)

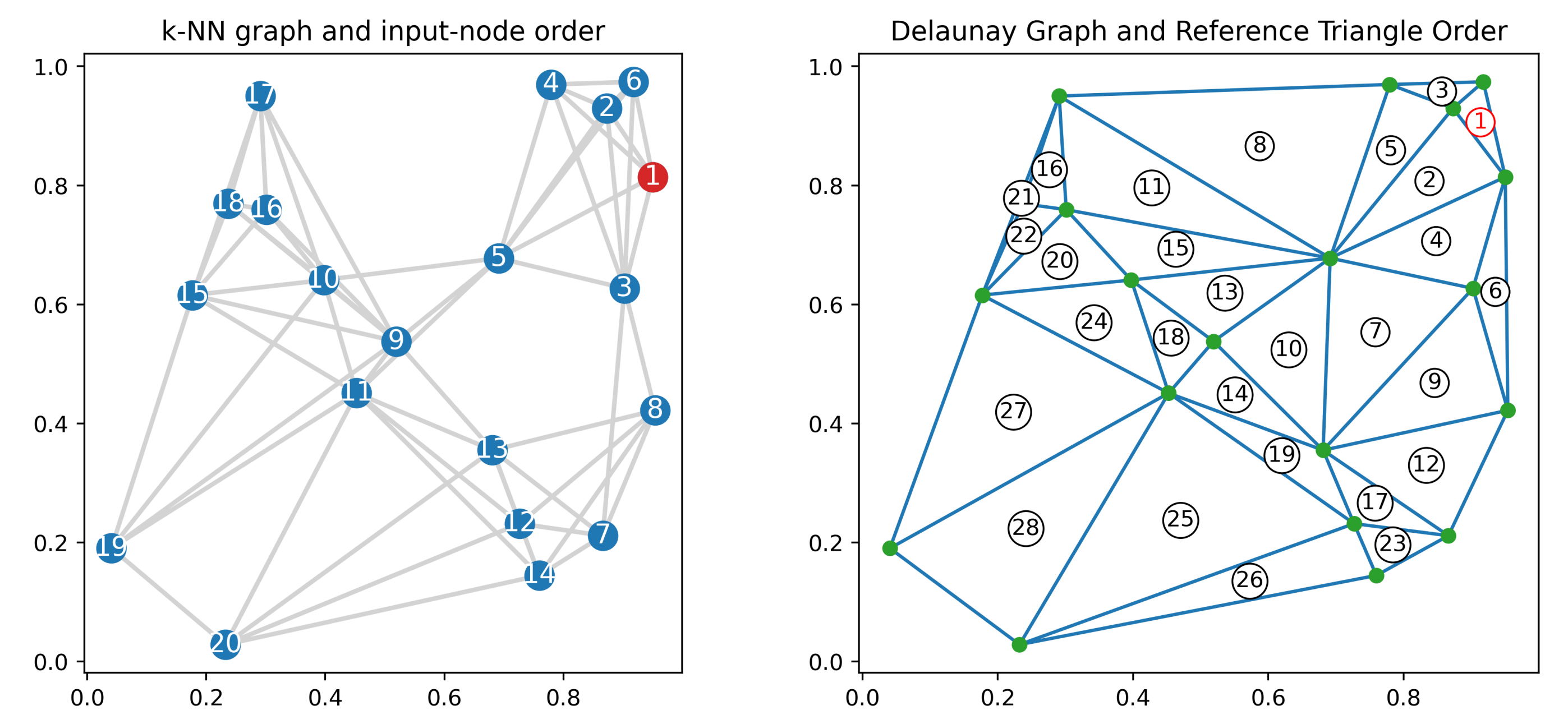
The t^{th} input $T_{t-1} \in \mathbb{R}^{3H}$ is the concatenation 3 of encoder hidden states (h_i^e, h_j^e, h_k^e) that corresponds to the three nodes in the $t-1^{th}$ triangle in the reference triangle sequence. The first input to the decoder is an embedding "=>" of start token, which is also a model parameter. The model also has a MLP N_Θ and at the t^{th} step of the decoder, it produces a query vector $q_t = N_\Theta(h_t^d)$. Using the query vector q_t, the model points back to each encoder hidden state to calculate the predicted probability of choosing the i^{th} node in the next triangle:

$$P_t(n_i) = softmax(q_t^T h_i^e)$$

We obtain a discrete probability distribution $\vec{P}_t \in \mathbb{R}^{N+1}$, where the last entry is the probability of stopping. The model picks the top 3 indices $\tilde{T}_t$ in the distribution as the prediction of the 3 indices of nodes of next triangle. If the last entry is among the top 3, the model predicts stop. The prediction mode, the model uses previous prediction as next input autoregressively.

Dataset, Loss Function and Evaluation Metric

Each data point is generated in the folloiwng way: first of all, 20 random points are generated uniformly in domain $[0,1]^2$. Then a kNN graph is built ($k=5$). Then, a 2D Delaunay triangulation is generated using the Delaunay algorithm. Note that, Delaunay triangulation is unique. Starting from every boundary node, the kNN graph is traversed in BFS order to create a sequence of inputs: $\{(x_i, y_i)\}_{i=1}^{20}$. Then, starting from the boundary triangle to the left of the starting node, all triangles in the Delaunay graph are traversed in BFS order to create a sequence of triangles: $\{\tau_i\}_{i=1}^m$, where t_is are the ordered triples node indices of each triangle.



For those 20 random points in the figure, 11 pairs of sequences $\{\{(x_i, y_i)\}_{i=1}^{20}, \{t_i\}_{i=1}^m\}$ are incorporated in the dataset. The train set is generated from 10,000 sets of 20 random points, which consists of 77,309 pairs of sequences. The test set is generated from 5,000 sets of 20 random points.

Suppose at the j^{th} databatch of size B. At the b^{th} data point in the batch, the reference triangle sequence has length N_b, $P_b^t(n_i)$ is the model's prediction of the probability of choosing the i^th node in the t^{th} step, we use the following negative log probability as our loss:

$$L(\Theta, DataBatch_j) = -\frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_b}\sum_{t=1}^{N_b}\sum_{i\in\tau_t^b} log(P_b^t(n_i))$$

where τ_t^b is the triple that contains the index of nodes in the reference triangle in the b^{th} data point at t^{th} step.

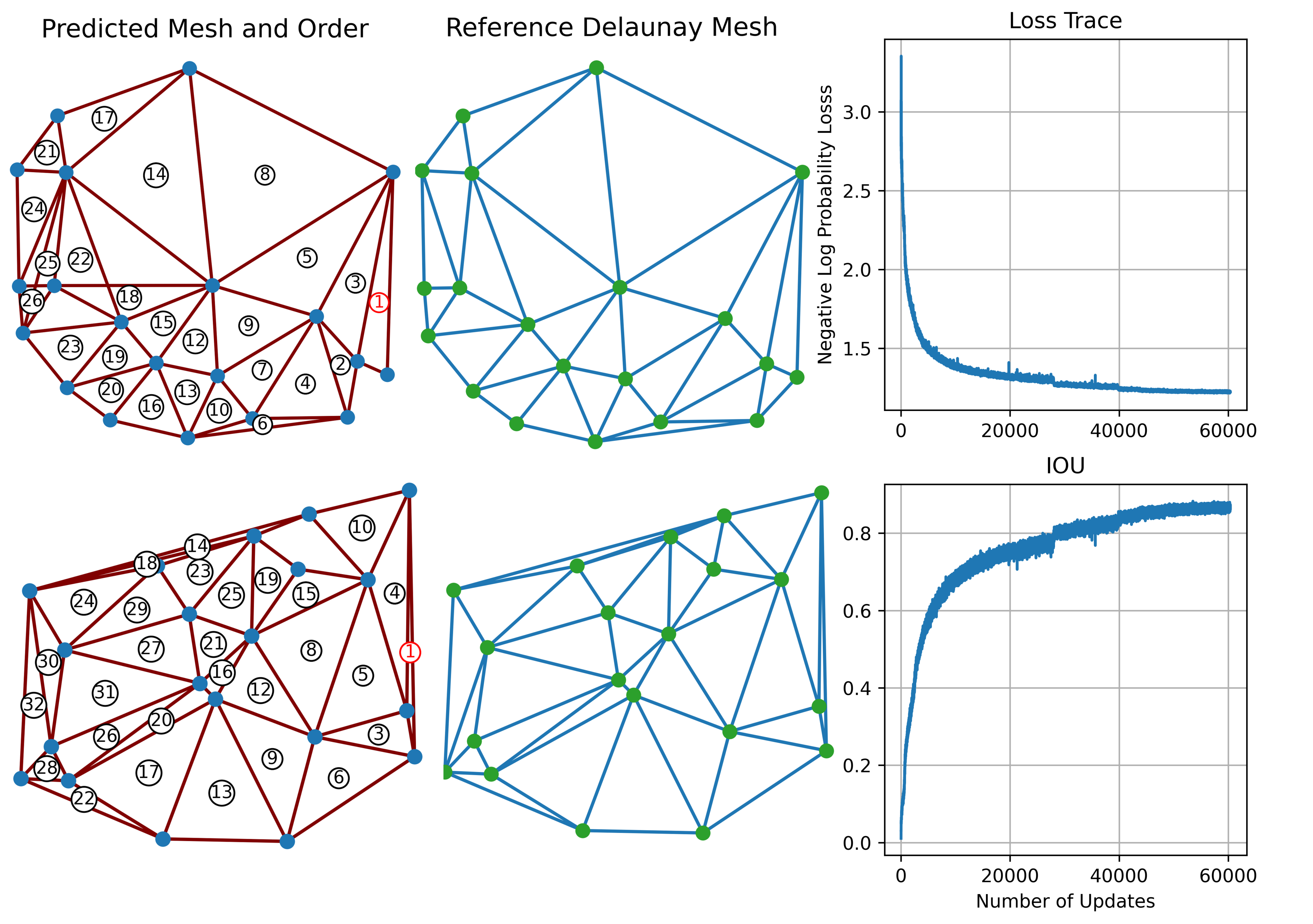
Dataset, Loss Function and Evaluation Metric (Continued)

We used **IoU** (Intersection over Union) between the predicted triangle set and reference triangle set as the metric. Note that this metric is invariant over the order of triangles in the predicted sequence.

Results

The embedding dimension of nodes in the GNN is set to 32. The GNN contains 5 layers of message passing. The hidden dimension of the LSTM is 256. The model also contains self-attention in the decoder LSTM. With a batch size of 256, we used an Adam optimizer of learning rate 1e-3 with a learning rate scheduler to train the model over 200 epochs.

The final train loss with teacher forcing is **1.22** and the final train IoU with teacher forcing is **0.86**.



We used **IoU** to evaluate the model over both train and test sets **without** teacher forcing. The the train **IoU** is **0.75** and the test **IoU** is **0.74**. The small different between these two figures suggests that the model did learn how to predict next triangle and generalizes well outside of the train set. The figure above shows two samples of the model's prediction on test set w/o teacher forcing.

Future Work

- Further improve the model by incorporating inductive biases into the model to prevent invalid triangle elements.
- Scale this model to generate 2D Delaunay mesh over larger number of points.
- Move into 3D Delaunay tetrahedral meshing. Pretrain the model using truth Delaunay reference and then fine tune the model using pre-defined evaluation metrics.