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

Wanzhou Lei

May 12, 2025

Abstract

In this project, we explored ways to build a data-driven approach to generate 2D triangular meshes that matches 2D Delaunay triangular meshes. The first tried model is a graph neural network that predicts the adjacency matrix of the mesh graph directly, which does not work well. The second tried model is a combination of a GNN and a pointer network that predicts triangular meshes sequentially, which works decently.

1 Introduction and Motivation

3D Tetrahedral meshing plays a critical role in computational simulations, with Delaunay triangulation serving as a foundational deterministic method. However, classical Delaunay algorithms in 3D often generate meshes with problematic elements, such as slivers or poorly-shaped tetrahedra, adversely affecting numerical stability and simulation accuracy [JT09]. Modern refinement techniques, including Weighted Squared Volume Minimization [ea24], Advancing-Front methods [Mav93], and parallel refinement algorithms [CC06], 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 explores data-driven approaches leveraging machine learning to generate tetrahedral meshes directly. As an initial step, this project targets 2D triangulation to validate model performance against standard 2D Delaunay methods, ensuring feasibility before extending to complex 3d meshing.

We first tried a graph neural network that predicts an adjacency matrix of the mesh graph directly. This model does not work well. We then introduce GNNPointerNet, a novel model combining a Message Passing Neural Network (MPNN) and a Pointer Network [OV17] implemented through an LSTM architecture.

2 Models We Have Explored

2.1 Naive Convolutional GNN Approach

The first step of my work is to build a naive convolutional GNN to predict the adjacency matrix of Delaunay graph given a sequence of 2D point coordinates.

2.1.1 Dataset

We created a train dataset of 10,000 truth Delaunay Graph of 20 nodes using the deterministic Delaunay algorithm. Each data point in the dataset contains the following elements: X: a sequence of 2d points in the graph as inputs (the order does not matter here). A: The adjacency matrix of the truth Delaunay Graph. The task is to build a model to predict an adjacency matrix based on the input coordinates and match with these truth adjacency matrices.

A test dataset of 5,000 truth Delaunay Graph of 20 nodes is also created using the same way.

2.1.2 Model Structure

The model is a lightweight graph convolutional neural network. Given a sequence of 2D coordinates, the model first builds a kNN graph over the sequence of input, where k is a hyperparameter chosen between 5 and 8. In each layer, the current representation of each node is mapped to a new representation of dimension 32. After all the layers, the model outputs a hidden state $H \in R^{B\times N\times 32}$, where B is the batch size, N is the number of points in each graph, and 32 is the embedding dimension.

Then, for each pair of nodes n_i and n_j, I concatenated the embeddings for these two nodes and pass it through a feed-forward neural network FNN_θ, where the last dimension is 1 and goes through a sigmoid function, such that $FNN_\theta(n_i, n_j)$ is the predicted probability of connection between the two vertices.

We connect these two parts and build the model. The model takes a sequence of N input coordinates and then output a symmetric N by N probability matrix P, where P_{ij} is the probability of connection between node i and node j.

2.1.3 Loss and Training

We defined several losses to train the model, some of them purely data-driven, some of them impose inductive biases as soft-constraints:

- adj_BCE_loss: is the weighted binary cross entropy between the model's prediction and the truth reference. More weights are added to penalize false negative prediction because of the class inbalance in the dataset (far more edges are not connected).
- unconnected_node_loss: Using a threshold of 0.5, the ratio of unconnected nodes in the predicted graph is used as a loss to impose soft-constraint such that all nodes are connected.
- MSE_edge_num: Using a threshold of 0.5, the MSE between the model's predicted edge number in each graph versus the real edge number in each graph.
- cross_edge_penalty: the number of corssing edges divided by the total number of edges in each graph, averaged by batch

We used an Adam optimizer, batch size is set to 32. The model is trained over 20 epochs. The total loss is stuck at some local minimum and still remains very high.

2.1.4 Results

To evaluate the model, we used the **IoU** between predicted edges and truth edges as metric. The **IoU** is calculated as the fraction between the number of common edges and the total edges of the predicted and true edges, averaged over all the test graphs. A perfect model that predicts all the true edges and nothing else will have an **IoU** of 1.0. This model achieved a test **IoU** of **0.09**.

The performance of the model is far from satisfactory. One sample of the model's prediction vs. reference is shown in Figure 3. As is shown in the figure, the model does not predict triangular elements and many edges are invalid, crossing each other.

Our hypothesis is that conventional convolutional graph neural networks (ConvGNNs) lack the expressiveness required for this task, resulting in poor performance. A key limitation observed is that the baseline model attempts to predict all edges simultaneously. This is inconsistent with how humans approach triangulation tasks—typically making predictions in a sequential manner, where each subsequent edge depends on the previously established ones. This insight motivates the exploration of more sophisticated models with sequence-to-sequence architectures, enabling autoregressive edge prediction conditioned on prior outputs.

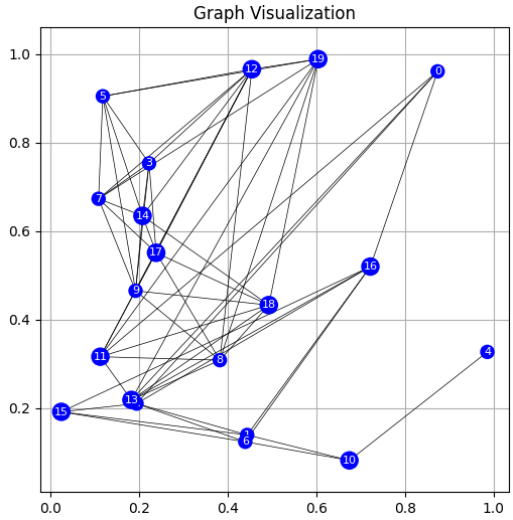

Figure 1: (a) Prediction

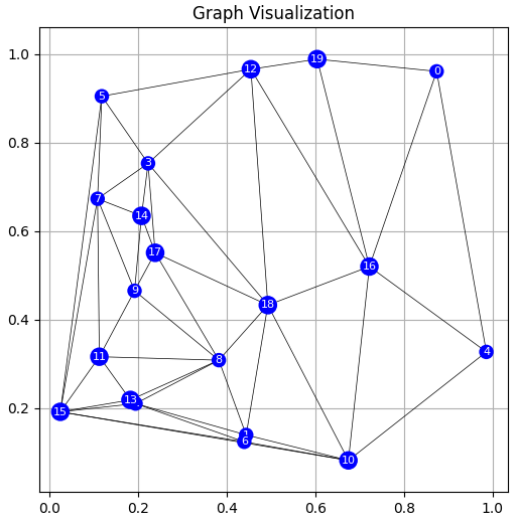

Figure 2: (b) Reference

Figure 3: ConvGNN's Prediction vs. Reference.

2.2 GNNPtrNet: A Combination of GNN and Pointer Network

The Pointer Network [OV17] is a sequence-to-sequence architecture capable of handling variable-sized output dictionary by leveraging an attention mechanism between encoder and decoder states. Delaunay triangulation can be naturally formulated as a combinatorial sequence-to-sequence problem, where the input is a sequence of points in space and the output is a sequence of triangles. Importantly, the size of output triangle dictionary is variable and depends on the spatial configuration of the input points.

Building on these insights, we propose GNNPtrNet, a modified Pointer Network that integrates a message-passing Graph Neural Network (GNN) with a Pointer Network based upon a LSTM. The model takes a variable-length sequence of 2D coordinates as input and generates a variable-length sequence of triangular elements, enabling autoregressive generation of mesh structures conditioned on learned geometric and topological features.

2.2.1 Dataset

Each data point is generated in the following way: first of all, 20 random points are generated uniformly in the domain $[0, 1]^2$. Then a kNN graph is built ($k = 5$). Then, a 2D Delaunay triangulation is generated using the deterministic 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 we build an undirected dual graph of triangles from the Delaunay graph in which triangles that share edges are connected. Starting from the boundary triangle to the left of the starting node, all triangles in the dual graph are traversed in BFS order to create a sequence of triangles: $\{\tau_i\}_{i=1}^{m}$, where τ_is are the ordered triples node indices of each triangle. For example, as is shown in Figure 4, for those 20 random points in the figure, 11 pairs of sequences $\{\{(x_i, y_i)\}_{i=1}^{20}, \{\tau_i\}_{i=1}^{m}\}$ are incorporated in the dataset, starting from different points on the boundary.

The training dataset is constructed from 10,000 sets of 20 randomly sampled 2D points, resulting in 77,309 input-output sequence pairs. The test dataset comprises 5,000 sets of 20 random points. Notably, the ordering of both input and output sequences plays a critical role in model performance and represents a key design decision. We evaluated several ordering strategies, including random order, depth-first search (DFS), and breadth-first search (BFS) orderings. Empirically, using BFS order for both input and output sequences yielded the best results. However, determining an optimal ordering strategy remains an open question and presents an interesting direction for future work.

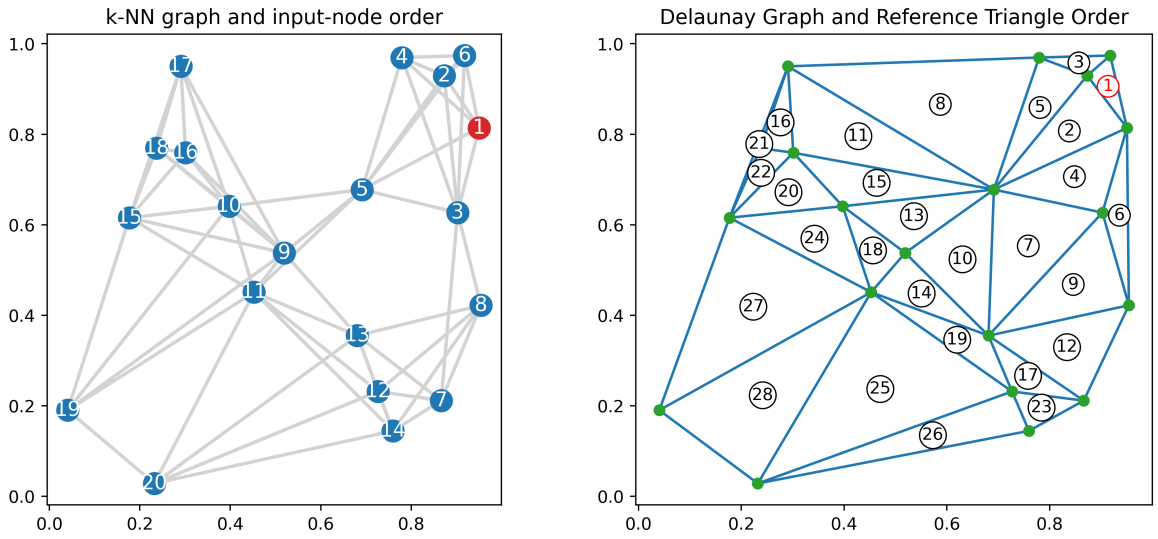

Figure 4: One Sample of Data Point

2.2.2 Model Structure

The GNNPtrNet 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 $\{\tilde{T}_i\}_{i=1}^{m}$. The structure of the model is shown in Figure 2.2.2.

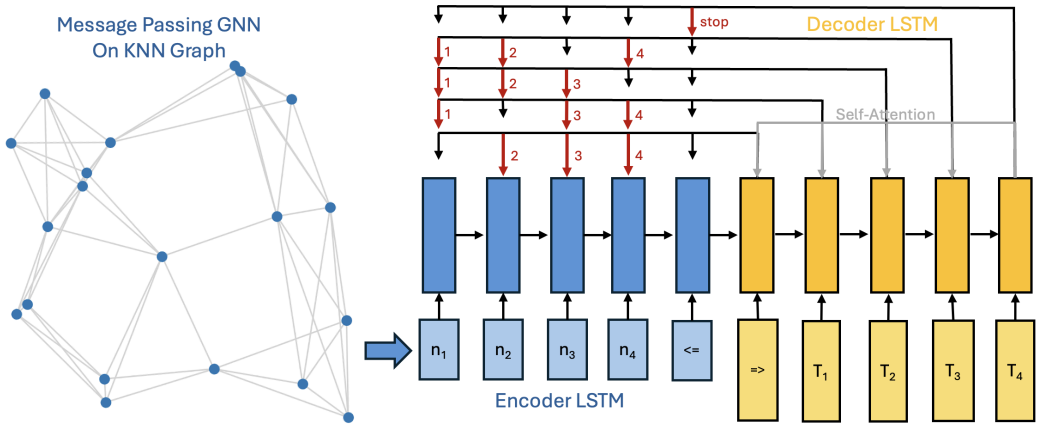

Figure 5: GNNPtrNet Structure

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:

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

$h_t^e \in R^H$, 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 $\Leftarrow$ of stop token and produces a final hidden state h_{N+1}^e. $\Leftarrow$ is a model parameter. We used teacher forcing to train the network. At t^{th} step of the decoder LSTM:

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

The t^{th} input $T_{t-1} \in R^{3H}$ 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 $\Rightarrow$ 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 logit of choosing the i^{th} node in the next triangle. Then all logits are passed through a softmax function to create a discrete probability distribution $\vec{P_t} \in R^{N+1}$, where the last entry is the probability of stop:

$$l_t(n_i) = q_t^T h_i^e$$
$$\vec{P_t} = softmax(\vec{l_t})$$

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. In prediction mode, the model uses previous prediction as next input auto-regressively.

Apart from the modified attention mechanism between the encoder and decoder, the primary distinction between GNNPtrNet and the vanilla Pointer Network lies in the incorporation of a message-passing Graph Neural Network (GNN) prior to the pointer module. Directly feeding raw 2D coordinates into the pointer network can result in limited contextual awareness, particularly for points appearing early in the input sequence, as their encoder hidden states are based on minimal preceding information. By introducing a message-passing GNN, each point's representation is enriched with global geometric context before being processed by the sequence model, thereby enhancing the encoder's ability to produce informative hidden states and improving overall prediction performance.

2.2.3 Loss and Training

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.

To evaluate the model, we used the following two metrics:

- **IoU**: Intersection over Union between predicted and reference triangular elements, averaged over all data.
- **Accuracy**: The percentage of triangles the model predicted correctly. (number of predicted correct triangular elements over total number of predicted triangular elements, averaged over all data).

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**. The loss trace is shown in Figure 2.2.4.

2.2.4 Results

We used **IoU** and **Accuracy** to evaluate the model over both train and test sets **without** teacher forcing. The train **IoU** is **0.75** and the test **IoU** is **0.74**. The train **Accuracy** is **0.87** and the test **Accuracy** is **0.86**. 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 the test set without teacher forcing.

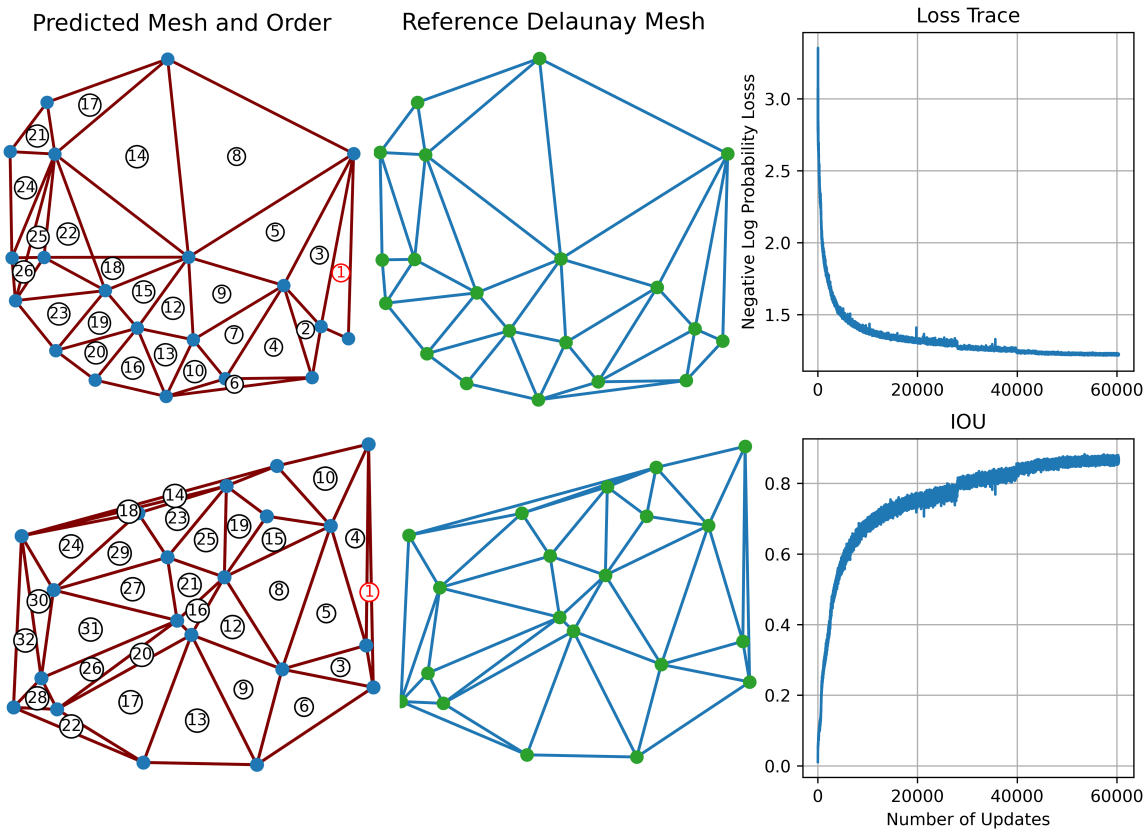

Figure 6: Sample Predictions and Loss Trace

In the original Pointer Network paper [OV17], the model was trained on **1 million** pairs of input and output sequences without enforcing any specific ordering in either sequence. One of the key innovations in our work is the explicit enforcement of breadth-first search (BFS) ordering in both the input and output sequences, which we find significantly aids the model in learning the underlying combinatorial structure of the task. The original Pointer Network [OV17], when trained on 1 million sequence pairs of 50-point inputs, achieved a test accuracy of **0.726**.

In contrast, our model—having a comparable number of parameters— was trained on a substantially smaller dataset consisting of only 70k input-output pairs (over 10× fewer samples), and achieved a test accuracy of 0.86 on 20-point inputs. While our experiments were limited to the 20-point setting due to time constraints, the substantial performance improvement highlights the potential of our approach. We attribute this gain in accuracy to two main factors: (1) the integration of a message-passing GNN that encodes rich contextual information prior to sequence modeling, and (2) the use of a carefully designed BFS-based sequence ordering that enhances the model's ability to generalize.

3 Conclusion

In this project, we explored two approaches to address the challenging problem of Delaunay triangulation. The baseline convolutional GNN model failed to produce satisfactory results, whereas the proposed GNNPtrNet model demonstrated strong performance and clear advantages over the vanilla Pointer Network. These results validate the feasibility of using data-driven machine learning models—specifically GNN-based sequence predictors—to generate triangulations and, by extension, offer promise for future extensions to 3D tetrahedral mesh generation.

A key direction for future work is to extend the current model to handle significantly larger 2D Delaunay triangulation problems, such as meshes involving 500 points. While exact deterministic algorithms with $O(nlogn)$ complexity exist for this task [GS85], [For86], our goal is to demonstrate the scalability and potential of learned models in such settings. Notably, Delaunay triangulation in 2D is inherently local: given a predicted triangle, the next triangle to be generated typically lies in its immediate neighborhood. This locality can be exploited by modifying the attention mechanism in the decoder to attend only to a constant number of nearby points at each step. Such a localized attention mechanism would allow the model to scale linearly with input size, reducing its computational complexity to $O(n)$.

Another important direction for future research is extending this framework to 3D tetrahedral meshing with appropriate modifications. While existing deterministic 3D Delaunay tetrahedralization algorithms can serve as references for pretraining, our objective is to surpass these traditional methods in terms of mesh quality and adaptability. To achieve this, it is essential to develop carefully designed evaluation metrics that capture the geometric and topological quality of predicted meshes. These metrics can then be incorporated directly into the training objective, enabling the model to optimize for task-specific criteria beyond simple replication of deterministic outputs.

References

[CC06] Andrey N. Chernikov and Nikos P. Chrisochoides. Parallel guaranteed quality planar delaunay mesh generation by concurrent point insertion. *SIAM Journal on Scientific Computing*, 28(5):1907–1926, 2006.

[ea24] Kaixin Yu et al. Weighted squared volume minimization (wsvm) for generating uniform tetrahedral meshes. *arXiv preprint arXiv:2409.05525*, 2024.

[For86] Steven Fortune. A sweepline algorithm for voronoi diagrams. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 313–322. ACM, 1986.

[GS85] Leonidas J. Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics (TOG)*, 4(2):74–123, 1985.

[JT09] Camille Wormser Mathieu Desbrun Jérémie Tournois, Pierre Alliez. Perturbing slivers in 3d delaunay meshes. *ACM Transactions on Graphics (TOG)*, 28(3):1–7, 2009.

[Mav93] D. J. Mavriplis. An advancing front delaunay triangulation algorithm designed for robustness. *NASA Technical Memorandum 104606*, 1993.

[OV17] Navdeep Jaitly Oriol Vinyals, Meire Fortunato. Pointer networks. *NeurIPS*, 2017.