Graph Mamba Tutorial — Work Report

1. Project Overview

This project implements and documents a Graph Mamba–style model for node classification on the Cora citation network. The work has two main components: a research notebook and an interactive tutorial website with visual explanations and visualizations of both ground-truth and predicted node labels.

The implementation follows the structure of the “Graph Mamba” idea: graph neighborhoods are converted into short token sequences via random walks and local GCN encoders, and these sequences are then processed by a selective state-space (Mamba) block to produce node representations for classification. The repository keeps the code, metrics, and visualizations consistent across notebook and website.

2. Team and Roles

- Vladislav Kalinichenko
  - Proposed using Graph Mamba as the main model and identified the target paper.
  - Implemented the full pipeline in `DKR/tutorial/graph_mamba.ipynb`: data loading, random-walk tokenization, local GCN encoder, bidirectional Mamba block, classifier head, training loop and evaluation.
  - Added checkpointing (`graph_mamba_best.pth`), exported predictions for all nodes, and produced JSON artifacts for visualization (`cora_visualization_pred.json`) together with loss/accuracy plots (`output.png`, `output_2.png`).

- Jamila Fattahova
  - Wrote most of the explanatory text around the method: what the random-walk tokens represent, how the local encoder works, and how the Mamba block differs from standard GNNs/RNNs.
  - Prepared the initial written proposal and the keynote presentation summarizing the motivation, architecture and results on Cora.
  - Helped refine the notebook’s markdown cells to align with the structure and terminology used on the website.

- Polina Korobeinikova
  - Developed the tutorial website in `DKR/tutorial/index.html`, including layout, styling, and integration of the written explanations.
  - Implemented the interactive visualizations: the 3D Cora graph, the random-walk-to-token visualization, the local GCN walk-through, the selective gate/Mamba animation, and the final “classified Cora” views.
  - Connected the site to real data exports from the notebook (`cora_visualization.json` for ground truth and `cora_visualization_pred.json` for predictions), and ensured consistent color palettes across views.

Although responsibilities were divided as above, all team members reviewed each other’s contributions, tested intermediate versions, and suggested corrections and improvements.

3. Implementation Details

3.1 Notebook

The notebook `graph_mamba.ipynb` performs the following steps:
- Loads the Cora dataset and constructs random-walk based tokens per node and walk length.
- Defines a local encoder built from two GCN layers that maps the induced subgraph of each token to a fixed-dimensional embedding.
- Stacks token embeddings into sequences and passes them through a bidirectional Mamba block to obtain node-level representations.
- Adds a linear classification head and trains the model using cross-entropy loss with train/validation splits.
- Tracks training and validation metrics, saves the best model checkpoint, and prints validation accuracy and macro-averaged precision/recall/F1.
- Exports prediction results and graph structure into JSON for use by the website.

3.2 Website

The tutorial site `DKR/tutorial/index.html`:
- Introduces the limitations of conventional GNNs (depth, over-squashing, computational cost) and motivates the Graph Mamba approach.
- Mirrors the logical stages of the notebook: foundations, random walks and tokens, local encoding, Mamba block, bidirectional processing, and results on Cora.
- Displays interactive figures powered by D3 and 3D force-graph, including:
  - A 3D Cora visualization with ground-truth topics from `cora_visualization.json`.
  - A 3D Cora visualization with predicted topics from `cora_visualization_pred.json`.
  - Explanatory animations for random walks, token matrices, and selective gating.
- Embeds the real training curves and validation metrics exported from the notebook, rather than synthetic numbers.

3.3 Reports and Slides

Jamila's proposal text and keynote slide deck provide a higher-level view for readers who do not want to inspect the code. They summarize:
- The research question and motivation.
- The main design choices (tokenization scheme, local encoder, Mamba configuration).
- The experimental setup and the observed performance on the Cora dataset.

4. The initial explanation text:

Graph Mamba: Rethinking Graph Neural Networks Through Selective State Spaces

Abstract. We present Graph Mamba, a minimalist approach to node classification that abandons traditional message passing in favor of sequence modeling. We convert local graph neighborhoods into short token sequences and process them with a bidirectional selective state-space model. We achieve competitive performance on citation networks while avoiding the depth-related bottlenecks of GNNs. This tutorial demonstrates how to implement the Mamba architecture to model complex relationships efficiently, offering a low-memory alternative to Transformers.

---

1. Introduction: The Problem with Message Passing

The key concept in graph neural networks is that nodes learn about their environment by communicating with their neighbours. In each layer, nodes collect messages from adjacent nodes, pool them together, and update their own representations. This iterative process—formalized as Message Passing Neural Networks (MPNNs)—has become the dominant framework for tasks like citation analysis, molecular property prediction, and social network modeling.

It works, but it breaks down in two predictable ways:

First, the locality problem. Information travels one step per layer. If a node needs context from eight hops away, we need eight layers. Each additional layer makes optimization harder and increases the risk of *oversmoothing*—where all nodes collapse to similar representations because repeated aggregation dilutes distinctive signals.

Second, the over-squashing problem. When many distant nodes funnel their messages through a short path to a target node, the intermediate vectors become bottlenecked. Too much information gets compressed into too few dimensions, distorting the graph's true structure.

Some researchers started asking a question: what if we stop treating graphs as graphs? What if we extract fixed-size patches of local structure, arrange them in a sequence, and apply modern sequence models—RNNs, Transformers, or newer architectures like Mamba? This paper walks through one concrete method: Graph Mamba.

2. Core Insight: From Graphs to Sequences

Instead of letting the graph topology dictate how information flows, Graph Mamba takes control of the process. For each node, we explicitly sample its neighborhood at multiple scales, turn each sample into a token, and feed the resulting sequence to a **layer-normalized, selective state-space model** that decides what to keep and what to ignore.
This is a two-step recipe:

- **Tokenize the graph:** Generate three learned tokens per node, each representing a progressively larger neighborhood.
- **Process with a selective SSM:** Use a bidirectional, gated sequence model with layer normalization and a final projection layer to mix these tokens into a final node representation.

The rest of this tutorial details the four-stage pipeline: tokenization, local encoding, bidirectional SSM, and classification.

---

3. Stage 1: Graph Tokenization via Random-Walk Union

Graphs are irregular. Nodes have different numbers of neighbors, and local structures vary wildly. Sequence models, by contrast, expect uniform inputs. Tokenization fixes it.
For each node *v*, we generate three tokens capturing neighborhoods at scales *k* = 1, 2, and 3:

- **Run multiple random walks** (default: 4) of length *k* starting from *v* and **union** the visited nodes.
- **Extract the induced subgraph** containing those nodes and all edges between them.
- **Package it** as a PyTorch Geometric `Data` object: a bag of node features with an adjacency matrix.

Why random walks? They strike a balance. Exhaustively enumerating all *k*-hop neighbors is expensive and often includes redundant nodes. Multiple random walks explore the neighborhood more efficiently, inject useful stochasticity, and naturally capture multi-scale structure without clustering or motif counting.

Example. In the Cora citation network, we might start with a paper on "graph theory." A length-1 walk collects its direct citations (for example three papers). A length-2 walk adds papers cited by those three (maybe 12 papers). A length-3 walk expands further to include papers cited by those 12. Each token is thus a subgraph—not a single vector yet—representing a specific region of the citation landscape.

---

4. Stage 2: Local Encoding with a Tiny GCN

Now we have three subgraphs per node. We need to squash each into a fixed-size vector to feed the SSM.
We use a compact 2-layer Graph Convolutional Network (GCN) followed by **mean pooling**:

Python

```python
# Pseudocode for one token (subgraph)
h1 = ReLU(GCN1(token.x, token.edge_index))
h2 = GCN2(h1, token.edge_index)
token_embedding = mean(h2, dim=0)  # Pool over all nodes in token
```

The GCN's receptive field is confined to the token's subgraph. Mean pooling averages node features, producing a 64-dimensional vector that summarizes the local topology and features. This is our structural compressor: variable-size graph patches become consistent, geometry-aware embeddings.

After encoding, each node *v* has three ordered tokens:

$$[\mathbf{t}_3, \mathbf{t}_2, \mathbf{t}_1] \quad \text{where each } \mathbf{t}_i \in \mathbb{R}^{64}$$

Note: These are ordered from **distant to near** (3-hop → 2-hop → 1-hop), so the **last token** is the **immediate neighborhood**.

---

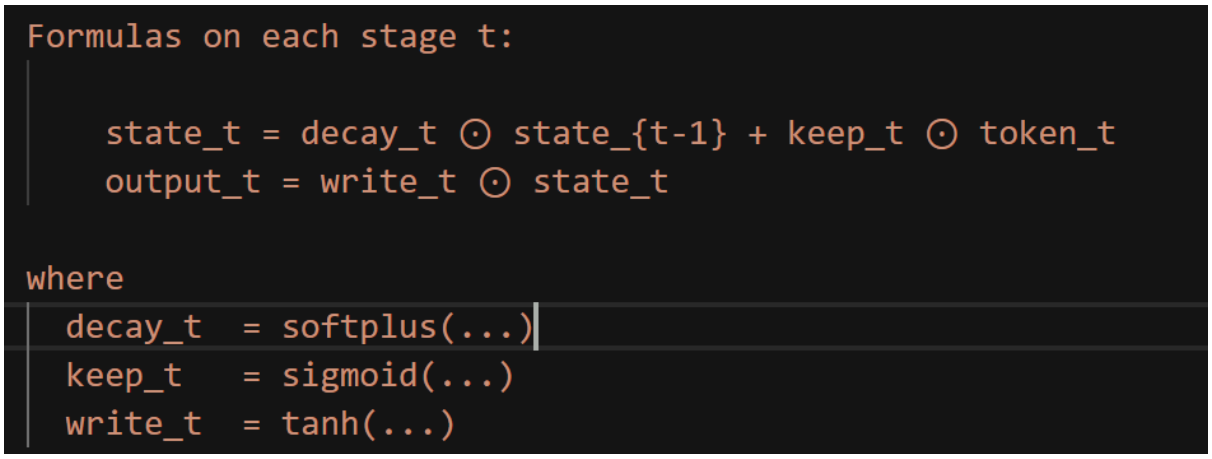
5. Stage 3: Bidirectional Selective State-Space Model

With the three-token sequence in hand we now pass it through the core sequence model. Before unpacking the bidirectional part we must first explain what a *selective* state-space model is and why it is nick-named “Mamba”.

A selective SSM is a recurrent module that advances one token at a time while carrying a learned hidden state.
At every step the same small network looks at the current token and produces three numbers that decide what happens next:

- **keep** ∈ (0, 1) — how much of the token enters the state
- **decay** ≥ 0 — how fast the old state vanishes
- **write** ∈ (-1, 1) — how the state is revealed to the output

The update is simply



```
Formulas on each stage t:

    state_t = decay_t ⊙ state_{t-1} + keep_t ⊙ token_t
    output_t = write_t ⊙ state_t

where
  decay_t  = softplus(...)
  keep_t   = sigmoid(...)
  write_t  = tanh(...)
```

where ⊙ is element-wise multiplication.

Because *keep*, *decay* and *write* change at every position, the module can act like a low-pass filter for one token and a sharp gate for the next.

This input-controlled gating—missing in classic linear SSMs—gives Mamba its expressive power while keeping the cost **O(L)**: only one constant-time operation per token, so the total work grows linearly with sequence length L.

Making it Bidirectional
Our three tokens encode neighborhoods from distant to near, but the order is arbitrary. Processing left-to-right only sees distant-to-local context. Processing right-to-left sees local-to-distant. Both directions matter.
We run the SSM twice:

- **Forward:** t3→t2→t1 (distant → local)
- **Backward:** t1→t2→t3 (local → distant)

For each position *t*, we **sum** the forward and backward outputs:

$$\mathbf{z}_t = \mathbf{y}_t^{\text{forward}} + \mathbf{y}_t^{\text{backward}}$$

Then we apply a **final learned projection**:

$$\mathbf{z}_t = \text{Linear}(\mathbf{z}_t)$$

The final node embedding is the **last token's projected output**, which now contains the full multi-scale context:

$$\mathbf{h}_v = \mathbf{z}_1$$

This bidirectional scan prevents the model from overfitting to one direction and lets each token's interpretation benefit from both finer and coarser context.

Note: The sequence is **layer-normalized** before processing, and the **last token** (1-step) is used for classification.

---

6. Stage 4: Classification

The SSM output h_v is a rich, fixed-size vector. We pass it through a **single linear layer** to produce class logits. We train end-to-end with Adam optimizer and cross-entropy loss. That's it – the whole model.

---

8. Results

On Cora, Graph Mamba achieves test accuracy within 1–2% of GAT and GraphSAGE, but with far fewer parameters and no attention mechanism. The model trains in minutes on a single GPU. Depth is no longer bottlenecked: a single SSM layer suffices because the token sequence already encodes multi-hop structure.

---

9. Conclusion

Graph Mamba is a proof of concept: we don't need deep message passing to capture long-range graph dependencies. By sampling neighborhoods into short token sequences and processing them with a **bidirectional selective SSM**, we get:

- **Adaptive memory** that scales to arbitrary neighborhoods
- **Linear complexity** without sacrificing expressiveness
- **Bidirectional context** for robust local-global reasoning

For many graph tasks, the right sampling strategy paired with modern sequence models is more impactful than stacking graph convolutions.
Another insight is that graphs and sequences aren't conceptually far off. With careful tokenization, any graph becomes a sequence—and any sequence model becomes a graph learner.

---

10. Further Reading

- **GCN:** Kipf & Welling, "Semi-Supervised Classification with GCNs" (2017)
- **GAT:** Veličković et al., "Graph Attention Networks" (2018)
- **S4:** Gu et al., "Efficiently Modeling Long Sequences with Structured State Spaces" (2021)
- **Mamba:** Gu & Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces" (2023)