Time Complexity Experiment

1 Motivation and Setup

1.1 Why count FLOPs instead of Time?

While measuring wall-clock time (in seconds) is intuitive, it is heavily influenced by hardware specifics (GPU vs. CPU), background processes, and Python interpreter overhead. To ensure our replication is **hardware-agnostic**, we count Floating Point Operations (FLOPs). This provides a theoretical guarantee of the algorithm's scaling behavior that holds true regardless of the machine used.

1.2 Experimental Parameters

To exactly replicate Figure 3 from the original paper, we fixed the following hyperparameters:

- **Graph Topology:** 1-Dimensional Grid Graph (Linear chain).
- **Dimensions:** Hidden dimension $d = 8$, Feature dimension $m = 8$.
- **Random Walks:** $n = 4$ walkers per node, with termination probability $p_{halt} = 0.5$.
- **Stochasticity:** Results for GRF are averaged over 10 random seeds to account for variance in sparsity.

To acknowledge: The Hidden Dimension (d) and Feature Dimension (m) are terms referring to the size of the vectors used inside the Transformer and the Linear Attention mechanism.

1. Hidden Dimension (d)
    - **What it is:** This is the size of the vector representing a single token (a graph node) inside the attention head.
    - **Context:** In a Transformer, your input might be a vector of size 512. Inside the "Multi-Head Attention" block, this vector is split into smaller "heads." If you have 8 heads, d=512/8=64.
    - **In the experiment:** d=8. This means every query (q_i), key (k_i), and value (v_i) is a vector of 8 numbers.
    - **Role:** It determines the "capacity" or "richness" of the information each token holds during the attention step.
2. Feature Dimension (m)
    - **What it is:** This is the size of the projected vector after applying the feature map $\phi(\cdot)$ in Linear Attention.
    - **Context:** Linear Attention works by approximating the softmax function. It takes a query vector q_i (size d) and maps it to a new feature vector $\phi(q_i)$ of size m.

- **The Transformation:** $\mathbb{R}^d \to \mathbb{R}^m$.
- **In your experiment:** $m = 8$.
- **Role:** It determines the "rank" of the low-rank approximation. If m is small, the approximation is faster but less accurate. If m is large, it's more accurate but slower. In standard Softmax attention, this concept doesn't exist (or rather, m=N, which is why it's slow).

2 Experiment details

The `run_experiment()` loop is the engine that drives the graph generation. It systematically increases the size of the graph (N) and, for each size, calculates the theoretical computational cost (FLOPs) for the three different methods.
Here is the breakdown of the logic inside that loop and the three counting functions.

2.1 The loop structure

The loop iterates through powers of 2 (64, 128, 256, etc.).

```
for N in N_VALUES:
```

At every step, N represents the number of nodes (tokens) in the graph. The goal is to see how the cost grows as N gets massive.

2.1.1 `count_flops_softmax(N, d)`

Complexity: O(N^2) (Quadratic).

This represents the standard Transformer attention: $Att = softmax(QK^T)V$.

Step A: (QK^T): You multiply a matrix of size $(N \times d)$ by $(d \times N)$.

- The result is an $(N \times N)$ matrix.
- Every single cell in that $(N \times N)$ matrix requires a dot product of size d.
- Cost: $N^2 \times 2d$. (Multiplied by 2 because typically 1 multiply + 1 add per element).

Step B: $(A \times V)$: You multiply the resulting $(N \times N)$ attention matrix by the Value matrix $(N \times d)$.

- The result is $(N \times d)$.
- Every cell in the output requires a dot product of size N.
- Cost: $N \times d \times N = N^2 d$.

Total: We sum them up. The dominant term is N^2, making this Quadratic. As N doubles, the cost quadruples.

2.1.2 `count_flops_linear(N, m, d)`

Complexity: O(N) (Linear).

This represents Linear Attention: $Att = \phi(Q)(\phi(K)^T V)$. The order of multiplication changes.

Step A: ($\phi(K)^T V$): You multiply the Key feature map ($m \times N$) by the Value matrix ($N \times d$).

- The result is a tiny matrix of size ($m \times d$). Crucially, the dimension N disappears from the matrix size here.
- Cost: $m \times d \times N$.

Step B: ($\phi(Q) \times \ldots$): You multiply the Query feature map ($N \times m$) by that tiny result ($m \times d$).

- Cost: $N \times d \times m$.

Total: The cost is proportional to N, not N^2. As N doubles, the cost doubles.

2.1.3 `count_flops_grf(...)`

Complexity: O(N) (Linear with a larger constant).

This represents the paper's method: $Att = \hat{\Phi}_Q(\hat{\Phi}_K^T V)$. The formula looks like Linear Attention, but $\hat{\Phi}$ is a Sparse Matrix (mostly zeros).

The Simulation (Why we need it): We cannot just use a formula like $N \times m$. We need to know exactly how many non-zero entries are in the matrix. The function *simulate_unique_visits* runs a random walk simulation:

1. Start at Node 0. Walk 4 times.
2. How many unique nodes did we touch? (e.g., node 0, 1, 2). That's 3 unique nodes.
3. Repeat for all N nodes.
4. Sum them up. This is the *avg_unique_visits*.

The Calculation:

- **Non-Zeros (NNZ):** If a node visits 3 unique neighbors, and the feature dimension m=8, that node contributes $3 \times 8 = 24$ non-zero numbers to the sparse matrix.
- **Sparse Multiplication:** When multiplying sparse matrices, you only do math on the non-zeros.
- **Cost:** $\approx 4 \times$ Total NNZ $\times d$.

Why it is Linear: Even though the graph grows to 1,000,000 nodes, a walker starting at Node 1 will typically only wander 3-4 steps away before stopping (due to *p_halt*). The "local neighborhood size" is constant (C). Therefore, Total $NNZ \approx N \times C \times m$. Since C and m are constants, the complexity is O(N).

3 Interpretation of Results

3.1 Quantitative Scaling Analysis

We measured the computational cost (in MFLOPs, scaled by 10^6) for graph sizes ranging from $N = 1$ to $N = 4096$. The results reveal three distinct regimes of operation.

The Crossover Point ($N \approx 8$)

Contrary to the intuition that "Linear Attention is always faster," our data shows that for extremely small graphs ($N < 8$), standard Softmax attention is more efficient.

- At $N = 1$: Softmax (3.20×10^{-5}) is an order of magnitude faster than Linear (2.56×10^{-4}).
- At $N = 4$: Softmax is still faster (5.12×10^{-4} vs 1.02×10^{-3}).
- At $N = 8$: The methods reach a **Crossover Point** where costs are identical (2.05×10^{-3} MFLOPs).

Theoretical Explanation: Softmax complexity is $4N^2d$ while Linear is $4Nmd$. With our parameters ($d = 8, m = 8$), the costs equalize when $N^2 = Nm$, i.e., $N = m = 8$. Below this threshold, the overhead of projecting features to dimension m dominates the cost.

The Divergence ($N > 8$)

As N grows beyond the feature dimension m, the quadratic scaling of Softmax causes a catastrophic increase in cost compared to the linear methods.

- At $N = 4096$: Softmax requires **537 MFLOPs**.
- At $N = 4096$: Linear requires only **1.05 MFLOPs**.

This represents a speedup factor of $\approx$ **511**$\times$, empirically confirming the necessity of linear approximations for large-scale graph transformers.

3.2 Analyzing the GRF Overhead

The "GRF (Ours)" method scales linearly (slope ≈ 1 in the log-log plot), mirroring the behavior of unmasked Linear attention. However, it incurs a consistent overhead.

- At $N = 4096$, GRF cost is 3.30 MFLOPs compared to Linear's 1.05 MFLOPs.
- This yields an overhead ratio of ≈ 3.14.

Interpretation: This constant factor corresponds to the sparsity of the Graph Random Features. In our 1D grid experiment with $p_{halt} = 0.5$, a random walker visits on average ≈ 3.1 unique nodes (the node itself plus immediate neighbors). Therefore, while the complexity class remains $\mathcal{O}(N)$, the absolute cost is roughly $3\times$ that of unmasked attention due to the local neighborhood aggregation.

3.3 Summary Table

The following table summarizes the critical transitions in computational cost.

N (Nodes)	**Softmax** (10^6)	**Linear** (10^6)	**GRF** (10^6)
1	3.20×10^{-5}	2.56×10^{-4}	2.56×10^{-4}
8	$\mathbf{2.05 \times 10^{-3}}$	$\mathbf{2.05 \times 10^{-3}}$	6.48×10^{-3}
64	1.31×10^{-1}	1.64×10^{-2}	5.38×10^{-2}
512	8.39×10^{0}	1.31×10^{-1}	4.11×10^{-1}
4096	5.37×10^{2}	1.05×10^{0}	3.30×10^{0}

Table 1: Selected data points showing the crossover at $N = 8$ and the divergence at large N.