Time Complexity Experiment

1 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.

1.1 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.

1.1.1 `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.

1.1.2 `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.

1.1.3 `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).