CS 8803 Lab 4 Report

NANDAN BHARAT PARIKH

December 2025

1 Task 1

1.1 Temperature Variation

Temperature is a hyperparameter used in sampling and generation within a language model. It controls the randomness of the output by adjusting the probability distribution of the next token the model selects.

- **Low Temp (closer to 0)**: This makes the model more deterministic and conservative. The model is heavily biased toward the highest-probability tokens, resulting in output that is highly consistent, predictable, and often repetitive across multiple generations. I also observed this phenomenon with my output where all the answers
- **High Temp (closer to 1)**: This makes the model more random and creative. It flattens the probability distribution, giving lower-probability tokens a greater chance of being selected. This results in output that is more diverse, unexpected, and potentially less coherent.

1.2 Input Length Variation

Input Length	Mean Latency (in ms)	Standard Deviation (in ms)
10	39.62935028	0.7585042846
50	38.60608063	0.0733482765
100	39.26296921	0.6141216734
200	39.45884781	0.09938426995
400	45.47365723	0.618296105
800	71.67130432	0.1240165387

Table 1: Latency Summary

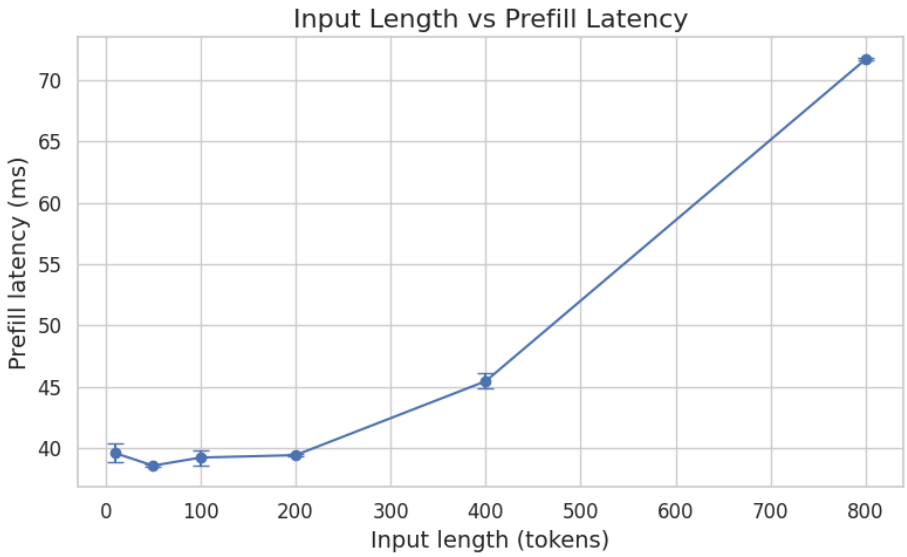

Figure 1: Input length vs latency

Observation: As we can see from the table and the plot that the latency does not scale linearly with the input range.

- **Low Input Regime (10–200 tokens):** Latency is effectively constant. The difference between processing 10 tokens (39.63ms) and 200 tokens (39.46ms) is negligible. In fact, due to variance (noise), the 50-token run was slightly faster than the 10-token run.
- **High Input Regime (200–800 tokens):** Latency increases sharply. Moving from 400 to 800 tokens results in a 57% increase in latency (45.47ms $\rightarrow$ 71.67ms).

Potential Explanation:

- Overhead Dominance: In the low input regime (¡200 tokens), the GPU is underutilized. The time taken is dominated by fixed overheads rather than the actual computation of attention. These overheads include kernel launch latency, memory access and the decode step
- **Compute/Memory Saturation:** As the input length exceeds $\approx$ 200 tokens, the sheer volume of calculations (specifically the $O(N^2)$ complexity of the attention matrix calculation and linear projection layers) begins to saturate the available GPU resources. The latency becomes dependent on the sequence length.

1.3 GPU Memory Analysis

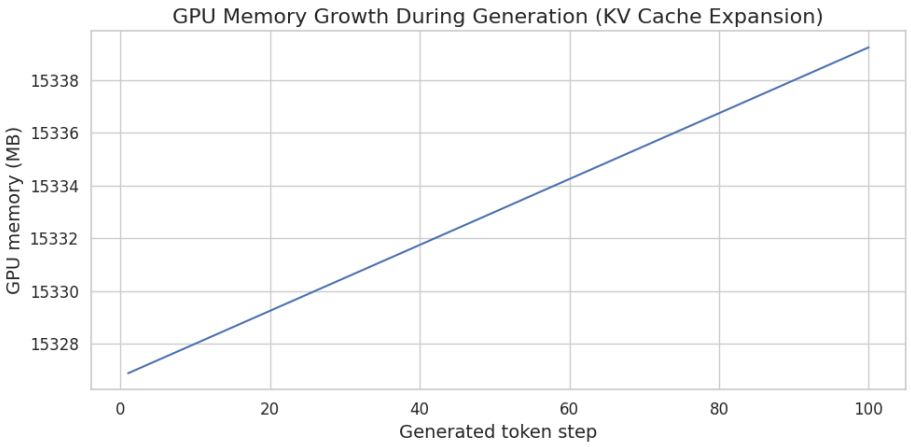

Figure 2: Input length vs latency

Table 2: Statistical Analysis of GPU Memory Growth (Linear Regression)

Metric	**Value**
Slope (Marginal Cost)	0.1250 MB/token
Intercept (Base Footprint)	15,326.75 MB
Coefficient of Determination (R^2)	1.000000
Standard Error	1.0×10^{-6}
P-value	0.00

- **Observation:** The GPU memory usage exhibits a perfectly linear growth trend ($R^2 = 1.0$) with respect to the output length.
  - **Base Footprint (Intercept):** This is almost 15,327 MB. This represents the static cost of loading the model weights (FP16) and initializing the CUDA context.
  - **Marginal Cost (Slope):** 0.125 MB per token. For every new token generated, memory consumption increases by exactly 0.125 MB.
- **Explanation:** This linear growth is caused by the KV Cache (Key-Value Cache).
  - To avoid re-computing attention for previous tokens at every step, the model caches the Key and Value vectors for the entire sequence in VRAM.
  - As the sequence length (N) increases by 1, a new set of KV vectors is appended to the cache.

2 Task 2

2.1 Attention Maps

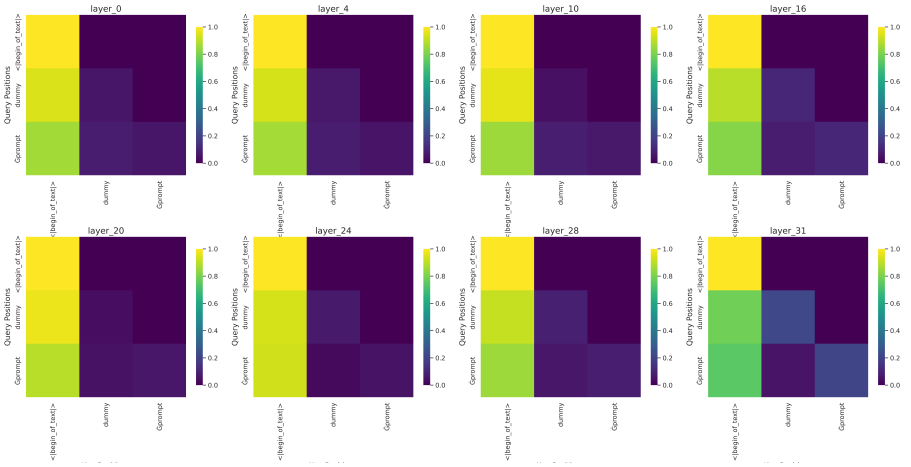

Figure 3: Short Dummy Attention

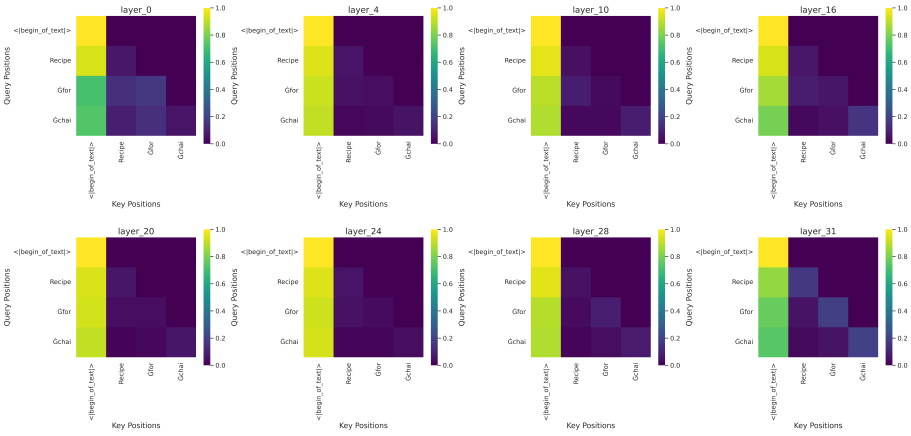

Figure 4: Short Meaningful Attention

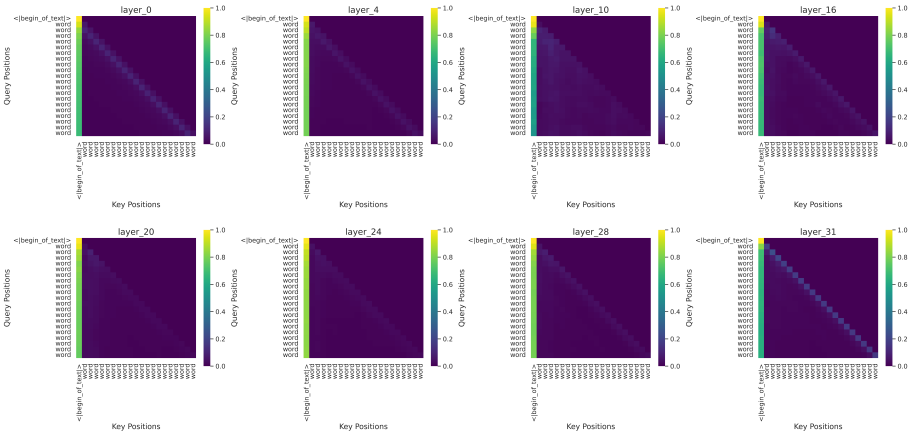

Figure 5: Medium Dummy Attention

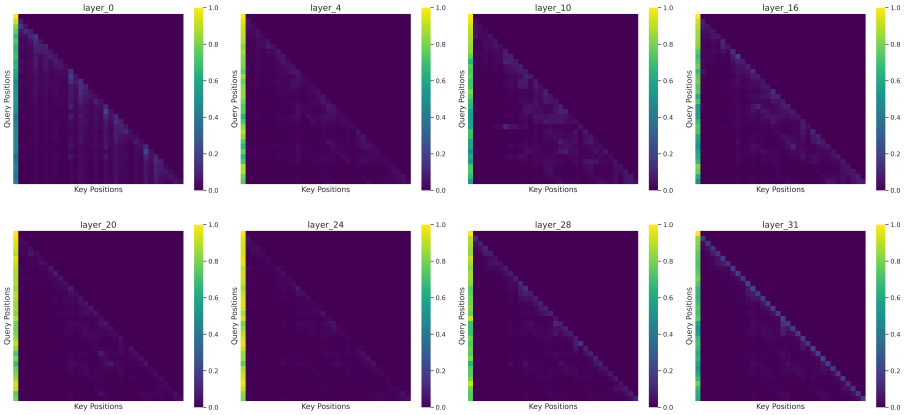

Figure 6: Medium Meaningful Dummy Attention

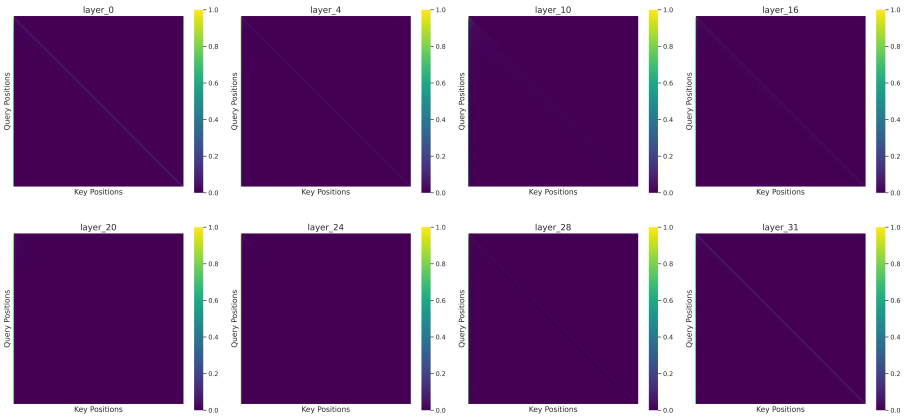

Figure 7: Long Dummy Attention

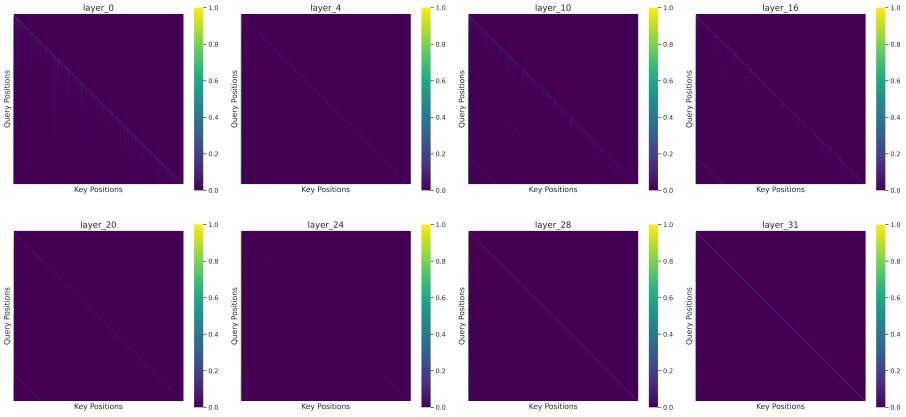

Figure 8: Long Meaningful Attention

2.2 In Depth Attention Sink Analysis

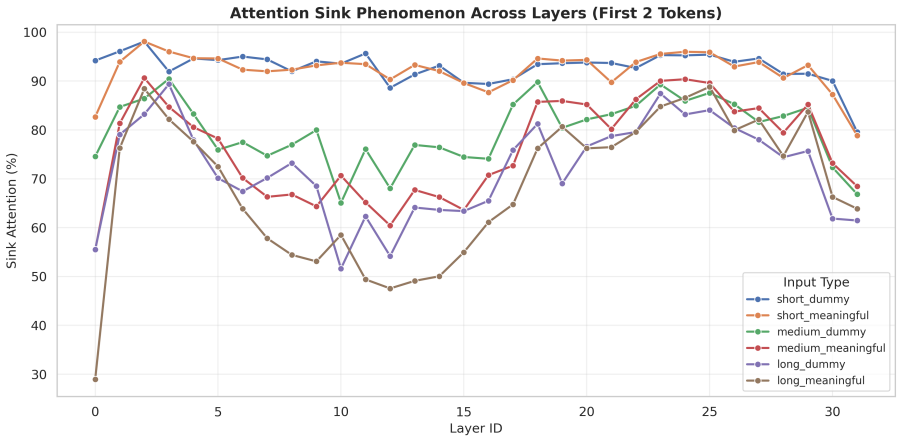

Figure 9: Attention Sink Analysis

2.2.1 Existence of Attention Sinks

Observation: The attention sink phenomenon is clearly observed across the majority of the network layers. As illustrated in the results, the model consistently allocates a dominant proportion of attention probability mass to the first two tokens ("sink tokens").

- **Magnitude:** For layers 3 through 28, the attention score allocated to the sink tokens consistently ranges between **80% and 95%**, regardless of the input type or length.
- **Significance:** This behavior aligns with the findings of *StreamingLLM*, confirming that the model utilizes the initial tokens as a "sink" to stabilize the Softmax distribution, even when those tokens lose semantic relevance in longer sequences.

2.2.2 Distribution Analysis: Layer ID and Input Length

The existence of the sink is not uniform; it evolves distinctly across the depth of the network and is influenced by input length in the initial layers.

A. Impact of Layer ID The phenomenon exhibits three distinct phases across the network depth:

1. **Phase 1: Emergence (Layers 0–2):** The sink is not structural at initialization. In Layer 0, attention is highly dependent on input type. Notably, for the `long_meaningful` input, sink attention is minimal (≈

29%), indicating that the first layer prioritizes local semantic information over the sink. However, by Layer 2, the sink mechanism dominates across all inputs.

2. **Phase 2: The Anchor Plateau (Layers 3–28):** This region represents the structural core of the model. Regardless of whether the input is dummy text or meaningful context, the sink attention converges to a high stable range (90%–95%). This suggests the sink serves as a necessary global anchor for the attention heads during complex feature extraction.

3. **Phase 3: The Output Shift (Layers 29–31):** In the final layers, there is a marked decline in sink attention (dropping to 60%–70% at Layer 31). This shift likely occurs because the final layers must redirect attention from the sink back to the most recent local tokens to accurately predict the next token's logits.

B. Impact of Input Text Length Input length primarily affects the *onset* of the sink behavior:

- **Short Inputs:** For short sequences (e.g., `short_meaningful`), Layer 0 shows high sink attention ($> 80\%$) immediately. With limited context available, the model defaults to attending to the start tokens.

- **Long Inputs:** For longer sequences, there is a delay. The `long_meaningful` input starts with low sink attention, as the model initially distributes probability mass to the abundant local tokens. However, the mechanism forces a correction by Layer 2, bringing the long-context inputs in line with the sink behavior.

3 Task 3

3.1 Visualization and Analysis of Weight and Activation Distributions

3.1.1 Llama-2-7B Visualizations

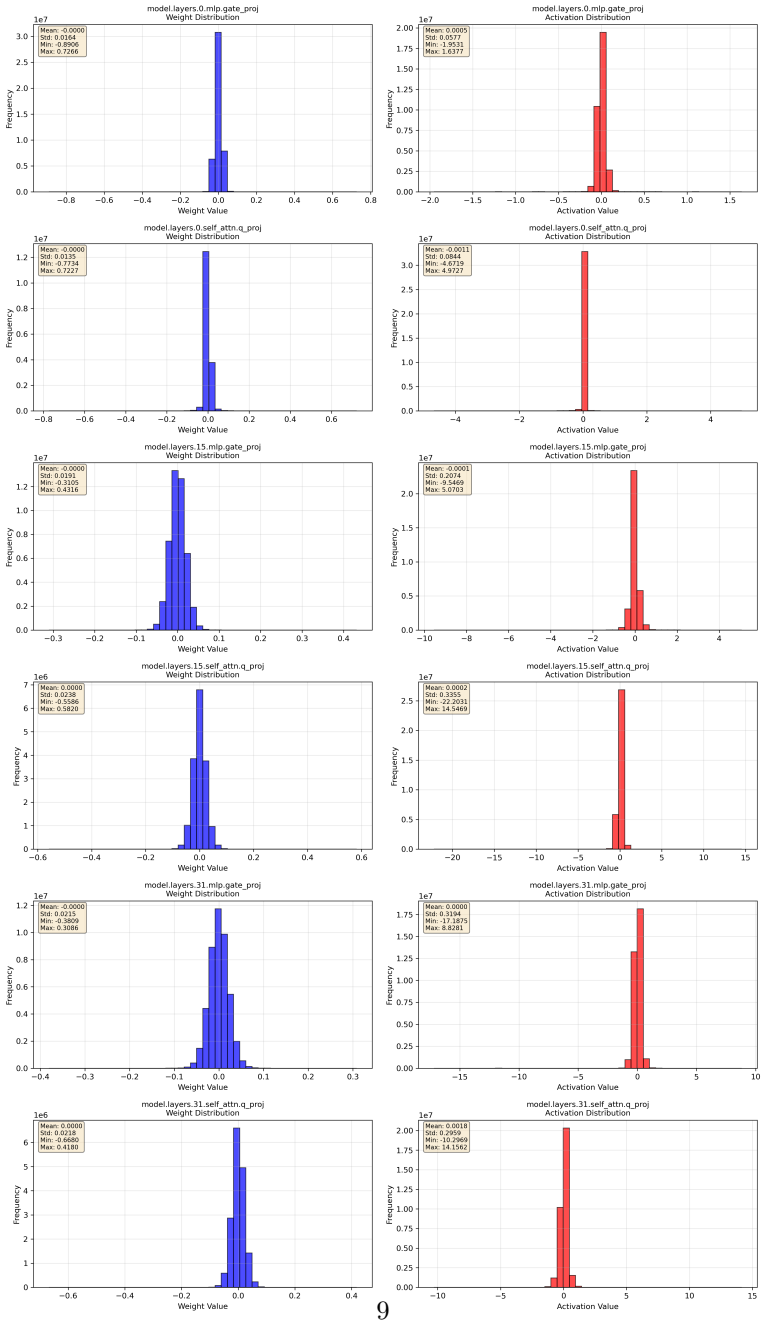

Figure 10: Comparison of weight and activation distributions for Llama-2-7B across Layers 0, 15, and 31. Note the significant difference in scale and outlier presence between weights (left) and activations (right).

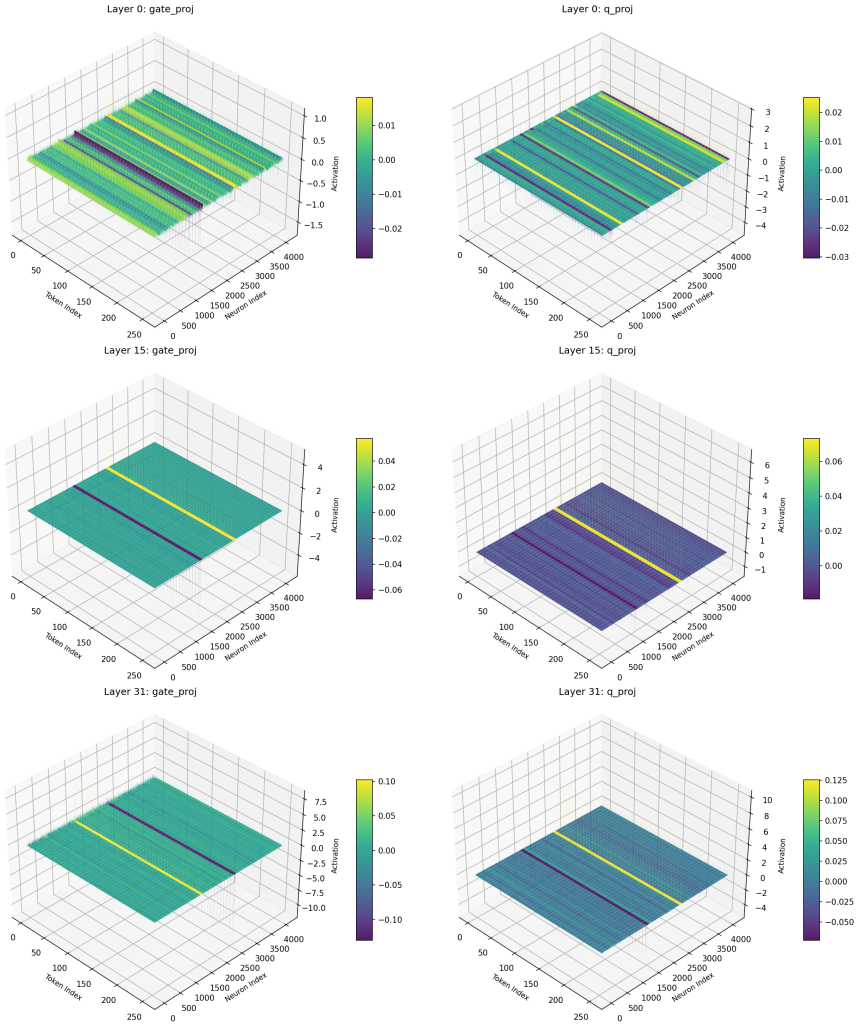

Figure 11: 3D visualization of activations and weights

3.1.2 Llama-3-8B Visualizations

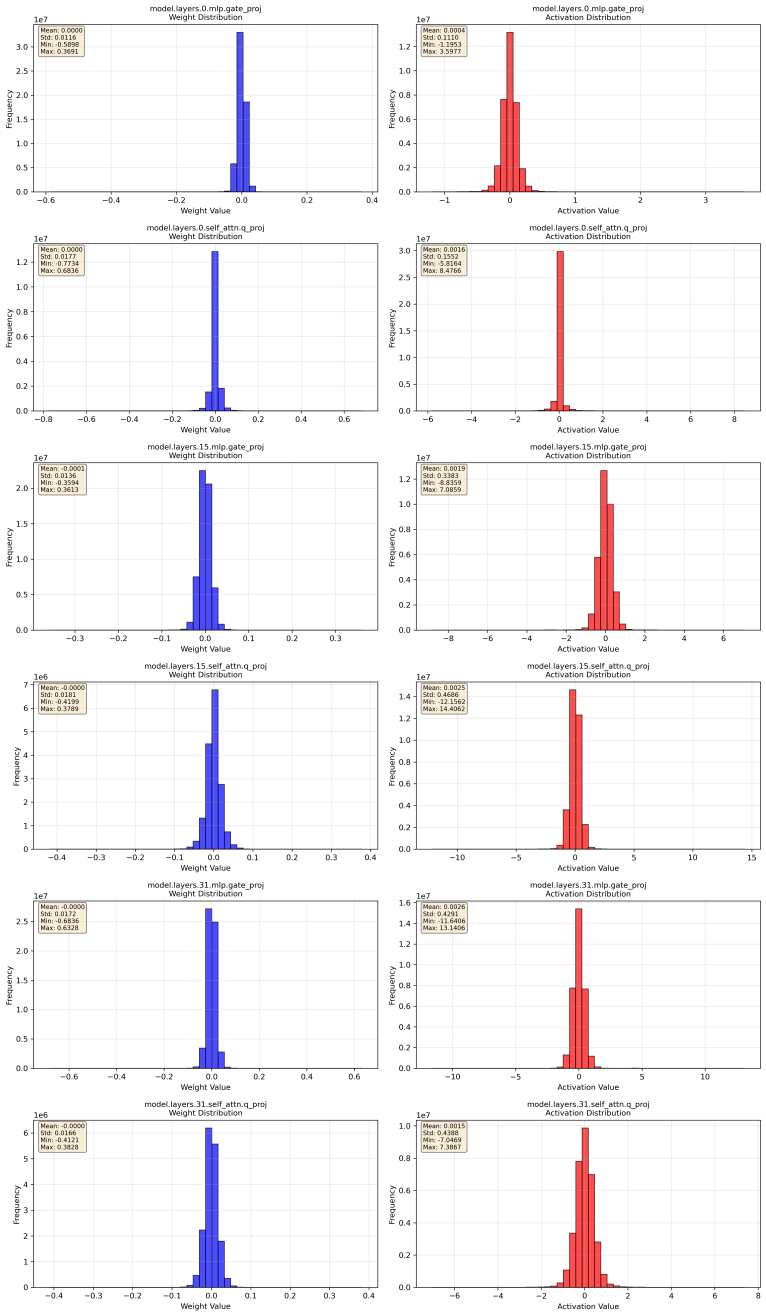

Figure 12: Comparison of weight and activation distributions for Llama-3-8B across Layers 0, 15, and 31. Note the significant difference in scale and outlier presence between weights (left) and activations (right).

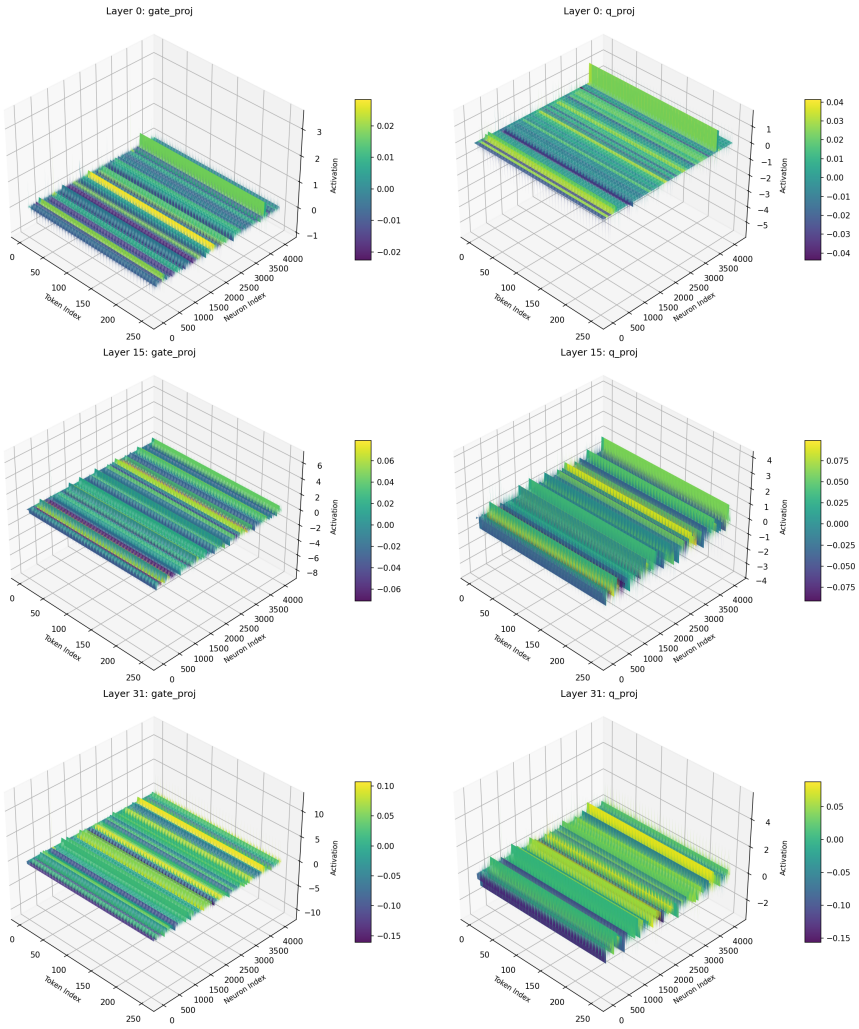

Figure 13: 3D visualization of activations and weights

3.1.3 Analysis of Observed Distributions

Based on the visualization of the Llama-2-7B model (Figure 10), we observe distinct statistical behaviors between weights and activations:

1. Weights vs. Activations (The Outlier Disparity) There is a fundamental difference in the value distribution of weights compared to activations:

- **Weights (Easy to Quantize):** The weight distributions (left column) are stationary, symmetric, and Gaussian-like. They are tightly bounded within a small range (e.g., typically between $[-0.5, 0.5]$). For instance, in

Layer 15 (`mlp.gate_proj`), the weights are confined to $[-0.31, 0.43]$. This lack of outliers makes weights suitable for standard uniform quantization.

- **Activations (Hard to Quantize):** The activation distributions (right column) are highly non-Gaussian and distinctively heavy-tailed. While most values cluster near zero, they exhibit **extreme systematic outliers**. In Layer 15, while the weights remain small, the corresponding activations extend significantly to $[-22.20, 14.54]$. These outliers dictate the quantization range, causing a loss of precision for the majority of smaller values.

2. Variation Across Layers The challenge of activation outliers is not uniform; it evolves as the signal propagates through the network depth:

- **Early Layers (Layer 0):** The activations are relatively bounded. The `self_attn.q_proj` activations in Layer 0 range from $[-4.67, 4.97]$.
- **Middle Layers (Layer 15):** We observe a drastic expansion in the dynamic range due to outlier emergence. The minimum value drops to -22.20, representing a nearly **5x increase in magnitude** compared to the first layer. This confirms that activation outliers are a feature that accumulates and persists in the deeper layers of the Transformer, representing the primary bottleneck for INT8 quantization.

3. Comparison: Llama-2 vs. Llama-3 Despite architectural improvements in Llama-3 (such as Grouped Query Attention and a larger vocabulary), the "outlier phenomenon" persists and is notably similar in magnitude to Llama-2.

Table 3: Comparison of Dynamic Ranges at Layer 15

Model	**Weight Range**	**Activation Range**	**Act/Weight Ratio**
Llama-2-7B	$[-0.55, 0.58]$	$[-22.20, 14.54]$	$\approx 40\times$
Llama-3-8B	$[-0.36, 0.36]$	$[-12.15, 14.40]$	$\approx 40\times$

Conclusion: The persistence of these systematic outliers across model generations confirms that they are an intrinsic feature of the Transformer architecture (likely driven by LayerNorm and residual dynamics). Therefore, activation smoothing techniques like SmoothQuant remain essential for the efficient deployment of Llama-3-8B, just as they were for Llama-2.

3.2 Result Table

Table 4: Quantization Performance Comparison (Perplexity on WikiText-2)

Method	Llama-3-8B	Llama-2-7B
Baseline (FP16)	**6.12**	**5.51**
Baseline (W8A8)	6.26	5.62
SmoothQuant (W8A8)	6.24	5.62
Baseline (W4A4)	6792.47	8.50×10^7
SmoothQuant (W4A4)	5198.40	1.12×10^6

Note: Lower is better. W4A4 settings resulted in model collapse.

3.3 Analysis of Calibration and Evaluation Results

We evaluated the quantization performance using Perplexity (PPL) on the WikiText-2 dataset. The comparison includes the original FP16 model, a naive baseline quantization, and the SmoothQuant method across both 8-bit (W8A8) and 4-bit (W4A4) precision settings.

3.3.1 1. Efficacy of 8-Bit Quantization (W8A8)

In the 8-bit regime, both models demonstrated high resilience to quantization, with SmoothQuant showing a measurable advantage for the Llama-3 architecture.

- **Minimal Degradation:** For Llama-2-7B, the degradation from FP16 (5.51) to W8A8 (5.62) is minimal (+0.11 PPL). Interestingly, SmoothQuant performed identically to the naive baseline here (5.62), suggesting that at 8-bit precision, Llama-2's outliers are still manageable within the quantization grid without aggressive smoothing.
- **SmoothQuant Advantage on Llama-3:** For Llama-3-8B, the naive baseline degradation was slightly higher ($6.12 \rightarrow 6.26$). SmoothQuant successfully recovered some of this accuracy loss, achieving a perplexity of **6.24**. While the absolute difference is small (0.02), it indicates that Llama-3's more aggressive activation outliers (identified in the previous section) benefit more from the smoothing transformation than Llama-2.

3.3.2 2. Catastrophic Collapse at 4-Bit (W4A4)

The experimental results for 4-bit quantization reveal a critical threshold for these models.

- **Model Collapse:** Both models suffered catastrophic failure when quantized to W4A4. The perplexity scores exploded to unusable levels (> 5000 for Llama-3 and $> 10^6$ for Llama-2).

- **Analysis of Failure:** This collapse confirms that while SmoothQuant effectively mitigates activation outliers for W8A8, it is insufficient on its own for W4A4. The dynamic range of activations is simply too wide to be compressed into 16 bins (2^4) without significant information loss. To make W4A4 viable, SmoothQuant would likely need to be combined with weight-only techniques (like GPTQ or AWQ) or fine-grained group-wise quantization.

3.3.3 Conclusion

SmoothQuant proves effective for W8A8 quantization, preserving near-FP16 performance by mitigating the impact of systematic outliers. However, the extreme perplexity explosion at W4A4 indicates that the outlier smoothing technique alone cannot compensate for the precision loss in low-bit regimes (INT4) for these architectures.