Computational Flow and Complexity Analysis of Multi-Head Attention Mechanism

Understanding Operations, Dimensions, and Computational Costs

Technical Report

October 28, 2025

Abstract

This paper provides a systematic analysis of the Multi-Head Attention (MHA) mechanism, the core component of Transformer-based large language models (LLMs), from a computational graph perspective. We begin by mathematically defining and characterizing fundamental operations—matrix multiplication, element-wise operations, softmax, layer normalization, transpose, and reshape—that are essential for understanding MHA diagrams. We then trace the complete data flow of both forward and backward passes step-by-step, quantitatively deriving the time complexity (FLOPs) and space complexity (memory) at each stage. Special attention is given to analyzing how the $O(S^2)$ complexity of self-attention affects long sequence processing, and we discuss practical optimization strategies including FlashAttention, Multi-Query Attention, and gradient checkpointing.

Keywords Transformer; Multi-Head Attention; Computational Complexity; Memory Analysis; Forward Pass; Backward Pass; Gradient Computation; LLM Optimization.

Contents

1 Introduction

The Transformer architecture [1] has become the foundation of modern natural language processing and large language models (LLMs). State-of-the-art models such as the GPT series, BERT, and LLaMA all utilize the Multi-Head Attention (MHA) mechanism as their core component, and the performance and efficiency of these models heavily depend on the effective implementation of MHA.

1.1 Motivation

As the scale of LLMs increases dramatically (e.g., GPT-3 with 175B parameters, GPT-4 with an estimated 1.7T parameters), accurate understanding of computational and memory costs for each operation has become essential. Key considerations include:

- **Increasing sequence length** S: As context windows expand from 4K → 32K → 128K, self-attention with $O(S^2)$ complexity becomes the primary bottleneck
- **Growing model dimension** D: Larger embedding dimensions increase the computational cost of linear projections
- **Batch processing**: Optimizing batch size B to maximize GPU utilization
- **Gradient computation**: Memory requirements for the backward pass are 2-3x higher than the forward pass

1.2 Contributions

The contributions of this paper are as follows:

1. **Computational graph-based analysis**: Visualizing MHA data flow through graphs with nodes (operations) and edges (tensors)
2. **Detailed operation analysis**: Mathematical definitions and complexity derivations for each primitive operation (MatMul, Add, Softmax, LayerNorm, etc.)
3. **Forward & Backward pass analysis**: Step-by-step FLOPs and memory analysis for forward and backward propagation
4. **Scaling laws**: Characterizing complexity scaling with respect to B, S, D, and N_H
5. **Optimization strategies**: Discussion of practical optimization techniques including FlashAttention and KV caching

1.3 Organization

The paper is organized as follows:

- **Section 2**: Overall Transformer architecture and data flow
- **Section 3**: Computational graph notation and visual conventions
- **Section 4**: Mathematical definitions and characteristics of basic operations
- **Section 5**: Step-by-step complexity analysis of the forward pass
- **Section 6**: Gradient computation analysis of the backward pass
- **Section 7**: Overall complexity summary and optimization strategies

2 Overall Transformer Architecture

Before diving into the details of Multi-Head Attention, we first present the big picture of how a complete Transformer processes data. Figure 1 provides a bird's-eye view of the Transformer's computation and training signal path.

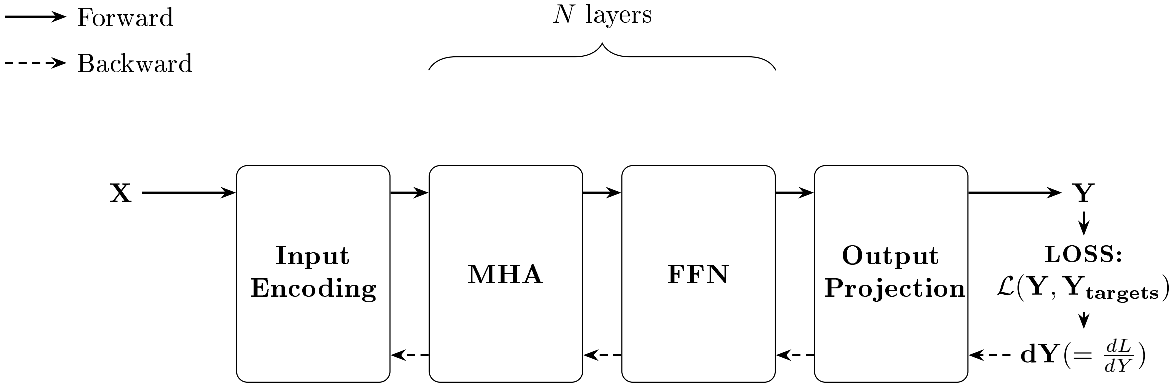

Figure 1: Overall Transformer forward (solid arrows) and backward (dashed arrows) flow. The pipeline moves from *Input Encoding* through *MHA* and *FFN* blocks (repeated N times) to *Output Projection*. During training, the loss gradient $\mathbf{dY}$ propagates backwards through all components.

2.1 Forward Path (Solid Arrows)

The forward computation proceeds left-to-right through the following stages:

1. Input Encoding Raw input tokens (typically integers representing vocabulary indices) are transformed into continuous vector representations:

- **Token Embedding**: Maps each token to a D-dimensional vector via lookup table $\mathbf{E} \in \mathbb{R}^{V \times D}$
- **Positional Encoding**: Adds position information (absolute, learned, or RoPE)
- Output: $\mathbf{X} \in \mathbb{R}^{[B,S,D]}$

2. Core Transformer Blocks (Repeated N Times) Each layer consists of two main components:

- **Multi-Head Attention (MHA)**: The focus of this paper. Allows tokens to attend to other tokens in the sequence
- **Feed-Forward Network (FFN)**: Applies position-wise transformations, typically with expansion: $D \to D_{\text{ff}} \to D$ where $D_{\text{ff}} = 4D$
- Both components include residual connections and layer normalization

3. Output Projection Transforms the final hidden states to vocabulary logits:

- Linear transformation: $\mathbf{W}_{\text{lm}} \in \mathbb{R}^{D \times V}$
- Often tied with input embedding matrix: $\mathbf{W}_{\text{lm}} = \mathbf{E}^{\top}$
- Softmax for probability distribution over vocabulary
- Output: $\mathbf{Y} \in \mathbb{R}^{[B,S,V]}$

4. Loss Computation For training, predictions are compared with target tokens:

$$\mathcal{L} = \text{CrossEntropy}(\mathbf{Y}, \mathbf{Y}_{\text{targets}}) \tag{1}$$

Typically averaged over batch and sequence dimensions:

$$\mathcal{L} = -\frac{1}{B \cdot S} \sum_{b=1}^{B} \sum_{s=1}^{S} \log P(y_{\text{target}}^{(b,s)} | \mathbf{Y}^{(b,s)}) \tag{2}$$

2.2 Backward Path (Dashed Arrows)

During training, gradients flow right-to-left through backpropagation:

1. Loss Gradient The gradient of the loss with respect to predictions initiates the backward pass:

$$\mathbf{dY} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \in \mathbb{R}^{[B,S,V]} \tag{3}$$

2. Backward Through Output Projection Computes gradients for:

- Weight matrix: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\text{lm}}}$
- Hidden states: $\frac{\partial \mathcal{L}}{\partial \mathbf{H}_{\text{final}}}$

3. Backward Through Transformer Blocks For each of the N layers (in reverse order):

- Backward through FFN: Updates FFN weights, produces gradient w.r.t. FFN input
- Backward through MHA: Updates attention weights ($\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_O$), produces gradient w.r.t. MHA input
- Through residual connections and layer normalization

4. Backward Through Input Encoding Updates embedding matrix $\mathbf{E}$ and (if applicable) learned positional encodings.

2.3 Key Observations

Sequential Processing Unlike recurrent models, Transformers process entire sequences in parallel during forward pass. The quadratic $O(S^2)$ attention complexity is the price for this parallelism.

Layer Stacking Deep networks ($N = 24$ for GPT-2, $N = 96$ for GPT-3, $N = 80$ for LLaMA-2 70B) allow learning complex representations through hierarchical feature extraction.

Residual Connections Skip connections around both MHA and FFN are crucial for:

- Gradient flow in deep networks
- Stable training
- Allowing identity mappings (network can learn to skip layers if needed)

Memory Requirements During training, must store:

- All activations from forward pass (for backward computation)
- All gradients during backward pass
- Optimizer states (momentum, variance for Adam)
- Peak memory is typically 3-4x the model parameter count

2.4 Scope of This Paper

The remainder of this paper focuses specifically on the **Multi-Head Attention (MHA)** component, which is:

- The defining innovation of the Transformer architecture
- The computational bottleneck for long sequences (due to $O(S^2)$ complexity)
- The memory bottleneck (attention scores scale as $O(S^2)$)
- The primary target for optimization research

Sections 3–7 provide a detailed, operation-level analysis of MHA's forward and backward passes, enabling precise understanding of computational costs and optimization opportunities.

3 Computational Graph Notation

3.1 Graph Structure: Nodes and Edges

Neural network computation can be represented as a Directed Acyclic Graph (DAG), where:

- **Nodes**: Represent operations
- **Edges**: Represent tensor data flow
- **Edge labels**: Indicate tensor names and shapes (e.g., $\mathbf{X}\ [B, S, D]$)

3.2 Node Types

The following node types are used in our diagrams:

Matrix Multiplication $\odot$

- **Symbol**: Circular node with $\bullet$ symbol
- **Meaning**: Matrix multiplication of two tensors
- **Inputs**: Two tensors (one marked with double arrow)
- **Output**: Product tensor

Element-wise Addition $\oplus$

- **Symbol**: Circular node with + symbol
- **Meaning**: Element-wise addition of two tensors (broadcasting allowed)
- **Usage**: Residual connections, bias addition

Auxiliary Operations $\boxed{\text{LN}}$, $\boxed{\text{SM}}$, $\boxed{\text{S}}$, $\boxed{\text{T}}$, $\boxed{\text{R}}$, $\boxed{\text{C}}$, $\boxed{\text{DO}}$

- **Symbol**: Rectangular node with operation abbreviation
- **Types**:
  - $\boxed{\text{LN}}$: Layer Normalization
  - $\boxed{\text{SM}}$: Scale + Mask (for attention scores)
  - $\boxed{\text{S}}$: Softmax
  - $\boxed{\text{T}}$: Transpose
  - $\boxed{\text{R}}$: Reshape (head split/merge)
  - $\boxed{\text{C}}$: Concatenate
  - $\boxed{\text{DO}}$: Dropout

Reduction Operations $\textcircled{\Sigma}$

- **Symbol**: Small circle with $\sum$ symbol
- **Meaning**: Summation over specific axes (e.g., for bias gradient computation)
- **Label**: Indicates reduction dimensions (e.g., $\sum_{B,S}$)

3.3 Edge Conventions

Single Arrow → Represents general data flow. In forward pass, carries activations; in backward pass, carries gradients.

Double Arrow ⇒ Explicitly marks the second operand (typically weight matrix) of matrix multiplication $\odot$. This clarifies that W is the right operand in $Z = XW$.

Example:

- Forward: $\mathbf{X} \rightarrow \odot \Leftarrow \mathbf{W} \rightarrow \mathbf{Z}$
- Meaning: $\mathbf{Z} = \mathbf{XW}$

3.4 Shape Notation

All tensors are annotated with their shapes:

- B: Batch size
- S: Sequence length
- D: Model dimension
- N_H: Number of attention heads
- D_h: Head dimension ($D_h = D/N_H$)
- V: Vocabulary size

Broadcasting notation: $\mathrm{BC}_{B,S}(\tilde{\mathbf{b}})$ indicates that a bias of shape $[D]$ is broadcast to shape $[B, S, D]$.

4 Understanding Basic Operations

4.1 Matrix Multiplication

4.1.1 Mathematical Definition

Matrix multiplication of $\mathbf{A} \in \mathbb{R}^{m \times k}$ and $\mathbf{B} \in \mathbb{R}^{k \times n}$:

$$\mathbf{C} = \mathbf{AB}, \quad C_{ij} = \sum_{p=1}^{k} A_{ip} B_{pj} \tag{4}$$

4.1.2 Batched Matrix Multiplication

With additional batch dimension: $\mathbf{A} \in \mathbb{R}^{[B,m,k]}$, $\mathbf{B} \in \mathbb{R}^{[B,k,n]}$

$$\mathbf{C}[b, :, :] = \mathbf{A}[b, :, :]\mathbf{B}[b, :, :] \quad \text{for each } b \in [1, B] \tag{5}$$

4.1.3 Computational Cost (FLOPs)

Single matrix multiplication $[m, k] \times [k, n] \rightarrow [m, n]$:

- Per output element: k multiplications + $(k - 1)$ additions $\approx 2k$ FLOPs
- Total output elements: $m \times n$
- **Total FLOPs**: $2mnk$

Batched matrix multiplication $[B, m, k] \times [B, k, n]$:

$$\text{FLOPs} = B \cdot 2mnk = 2Bmnk \tag{6}$$

4.1.4 Memory Requirements

- Inputs: $Bmk + Bkn$ elements
- Output: Bmn elements
- **Total**: $B(mk + kn + mn)$ (multiply by 4 for float32 bytes)

4.1.5 Backward Pass

Forward: $\mathbf{Z} = \mathbf{X}\mathbf{W}$ where $\mathbf{X} \in \mathbb{R}^{[B,S,D_{\text{in}}]}$, $\mathbf{W} \in \mathbb{R}^{[D_{\text{in}},D_{\text{out}}]}$

Gradients with respect to inputs:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Z}}\mathbf{W}^\top \quad [B, S, D_{\text{out}}] \times [D_{\text{out}}, D_{\text{in}}] \tag{7}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^\top \frac{\partial L}{\partial \mathbf{Z}} \quad [D_{\text{in}}, B \cdot S] \times [B \cdot S, D_{\text{out}}] \tag{8}$$

Each gradient computation also involves matrix multiplication with similar FLOPs.

4.2 Element-wise Addition

4.2.1 Mathematical Definition

For two tensors $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{[d_1,d_2,\ldots,d_n]}$ with the same shape:

$$\mathbf{C} = \mathbf{A} + \mathbf{B}, \quad C_{i_1,i_2,\ldots,i_n} = A_{i_1,i_2,\ldots,i_n} + B_{i_1,i_2,\ldots,i_n} \tag{9}$$

4.2.2 Broadcasting

NumPy/PyTorch broadcasting rules:

- Shape $[B, S, D]$ + shape $[D]$ $\to$ bias broadcast over (B, S)
- Shape $[B, S, D]$ + shape $[B, S, D]$ $\to$ element-wise addition

Example: Bias addition

$$\mathbf{Y} = \mathbf{X} + \mathrm{BC}_{B,S}(\tilde{\mathbf{b}}), \quad \mathbf{Y}[b, s, :] = \mathbf{X}[b, s, :] + \tilde{\mathbf{b}} \tag{10}$$

4.2.3 Computational Cost

Element-wise addition:

$$\text{FLOPs} = \text{total number of elements} = \prod_i d_i \tag{11}$$

Example: Addition of $[B, S, D]$ tensors requires BSD FLOPs

4.2.4 Memory

- Inputs: $2 \times$ size (smaller tensor reused with broadcasting)
- Output: size

4.2.5 Backward Pass

Forward: $\mathbf{C} = \mathbf{A} + \mathbf{B}$

Gradients:

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{\partial L}{\partial \mathbf{C}} \tag{12}$$

$$\frac{\partial L}{\partial \mathbf{B}} = \frac{\partial L}{\partial \mathbf{C}} \tag{13}$$

With broadcasting, sum gradient over broadcast dimensions:

$$\frac{\partial L}{\partial \tilde{\mathbf{b}}} = \sum_{b=1}^{B} \sum_{s=1}^{S} \frac{\partial L}{\partial \mathbf{C}}[b, s, :] \quad \text{(sum over } B, S\text{)} \tag{14}$$

This is represented by a $\textcircled{\scriptsize\Sigma}$node in the diagram.

4.3 Softmax

4.3.1 Mathematical Definition

For vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{15}$$

For multi-dimensional tensors, applied along a specific axis (typically the last axis).

Usage in MHA: For attention scores $\mathbf{A} \in \mathbb{R}^{[B, N_H, S, S]}$, apply softmax along the last axis (key dimension):

$$\mathbf{AS}[b, h, i, :] = \text{softmax}(\mathbf{A}[b, h, i, :]) \tag{16}$$

4.3.2 Numerical Stability

In practice, use the log-sum-exp trick to prevent overflow:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i - \max(\mathbf{x})}}{\sum_j e^{x_j - \max(\mathbf{x})}} \tag{17}$$

4.3.3 Computational Cost

Softmax over dimension of size n:

- Max computation: n comparisons
- Exp computation: n exponentials
- Sum computation: n additions
- Division: n divisions
- **Total**: $\approx 4n$ operations per vector

For tensor $[B, N_H, S, S]$ with softmax along last axis:

$$\text{FLOPs} = B \cdot N_H \cdot S \cdot (4S) = 4BN_HS^2 \tag{18}$$

4.3.4 Memory

- Input: BN_HS^2
- Output: BN_HS^2
- Intermediate values (max, sum): BN_HS

4.3.5 Backward Pass

Forward: $\mathbf{y} = \text{softmax}(\mathbf{x})$

Gradient:

$$\frac{\partial L}{\partial x_i} = y_i \left(\frac{\partial L}{\partial y_i} - \sum_j y_j \frac{\partial L}{\partial y_j} \right) \tag{19}$$

Vector form:

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{y} \odot \left(\frac{\partial L}{\partial \mathbf{y}} - \langle \mathbf{y}, \frac{\partial L}{\partial \mathbf{y}} \rangle \mathbf{1} \right) \tag{20}$$

where $\odot$ is element-wise multiplication and $\langle \cdot, \cdot \rangle$ is inner product.

4.4 Layer Normalization

4.4.1 Mathematical Definition

Normalization over feature dimension (D) for each sample and token:

Input: $\mathbf{x} \in \mathbb{R}^D$ (single token)

$$\mu = \frac{1}{D} \sum_{i=1}^{D} x_i \tag{21}$$

$$\sigma^2 = \frac{1}{D} \sum_{i=1}^{D} (x_i - \mu)^2 \tag{22}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{23}$$

$$y_i = \gamma_i \hat{x}_i + \beta_i \tag{24}$$

where $\gamma, \beta \in \mathbb{R}^D$ are learnable parameters and ϵ is a small constant for numerical stability (e.g., 10^{-5}).

4.4.2 Batch Processing

For tensor $\mathbf{X} \in \mathbb{R}^{[B,S,D]}$, apply LayerNorm independently at each (b, s) position:

$$\mathbf{Y}[b, s, :] = \text{LayerNorm}(\mathbf{X}[b, s, :]) \tag{25}$$

4.4.3 Computational Cost

Single vector $\mathbf{x} \in \mathbb{R}^D$:

- Mean computation: D additions
- Variance computation: D subtractions, D squares, D additions
- Normalization: D subtractions, D divisions, D square roots (amortized)
- Scale/shift: D multiplications, D additions
- **Total**: $\approx 6D$ operations

Tensor $[B, S, D]$:

$$\text{FLOPs} = B \cdot S \cdot 6D = 6BSD \tag{26}$$

4.4.4 Memory

- Input: BSD
- Output: BSD
- Parameters: $2D$ (γ, β)
- Cache for backward: $2BS$ (μ, σ^2) + BSD $(\hat{\mathbf{X}})$

4.4.5 Backward Pass

Values saved from forward: $\mu, \sigma^2, \hat{\mathbf{x}}$

Gradient w.r.t. parameters:

$$\frac{\partial L}{\partial \beta} = \sum_{b,s} \frac{\partial L}{\partial \mathbf{Y}}[b, s, :] \tag{27}$$

$$\frac{\partial L}{\partial \gamma} = \sum_{b,s} \frac{\partial L}{\partial \mathbf{Y}}[b, s, :] \odot \hat{\mathbf{X}}[b, s, :] \tag{28}$$

Gradient w.r.t. input (per token):

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \left[\frac{\partial L}{\partial \mathbf{y}} - \frac{1}{D} \sum \frac{\partial L}{\partial \mathbf{y}} - \hat{\mathbf{x}} \frac{1}{D} \sum \left(\frac{\partial L}{\partial \mathbf{y}} \odot \hat{\mathbf{x}} \right) \right] \tag{29}$$

4.5 Transpose

4.5.1 Mathematical Definition

Matrix transpose $\mathbf{A} \in \mathbb{R}^{[m,n]}$:

$$\mathbf{A}_{ij}^{\top} = \mathbf{A}_{ji} \tag{30}$$

For higher-dimensional tensors, exchange (permute) specific axes:

$$\mathbf{B} = \text{permute}(\mathbf{A}, \text{dims} = (0, 2, 1, 3)) \tag{31}$$

Usage in MHA: Transform $\mathbf{K} \in \mathbb{R}^{[B, N_H, S, D_h]}$ to $\mathbf{K}^{\top} \in \mathbb{R}^{[B, N_H, D_h, S]}$.

4.5.2 Computational Cost

Transpose only rearranges data:

$$\text{FLOPs} = 0 \quad \text{(no arithmetic operations)} \tag{32}$$

However, changes in memory access patterns can affect cache efficiency.

4.5.3 Memory

- If not in-place: requires memory for both input and output
- Contiguous memory requirement: may need reordering for subsequent operation efficiency

4.5.4 Backward Pass

Forward: $\mathbf{B} = \text{permute}(\mathbf{A}, \text{dims})$

Backward: Permute gradient in reverse order

$$\frac{\partial L}{\partial \mathbf{A}} = \text{permute}\left(\frac{\partial L}{\partial \mathbf{B}}, \text{inverse\_dims} \right) \tag{33}$$

4.6 Reshape

4.6.1 Mathematical Definition

Change tensor shape while preserving element order:

$$\mathbf{B} = \text{reshape}(\mathbf{A}, \text{new\_shape}) \tag{34}$$

Usage in MHA:

- Head split: $[B, S, D] \to [B, N_H, S, D_h]$ where $D = N_H \times D_h$
- Head merge: $[B, N_H, S, D_h] \to [B, S, D]$

4.6.2 Computational Cost

Reshape only changes metadata (shape information):

$$\text{FLOPs} = 0 \quad \text{(no arithmetic operations)} \tag{35}$$

4.6.3 Memory

- Typically a view operation: no additional memory required
- May trigger copy if contiguous memory is required

4.6.4 Backward Pass

Forward: $\mathbf{B} = \text{reshape}(\mathbf{A}, \text{shape}_B)$

Backward: Reshape gradient back to original shape

$$\frac{\partial L}{\partial \mathbf{A}} = \text{reshape}\left(\frac{\partial L}{\partial \mathbf{B}}, \text{shape}_A\right) \tag{36}$$

4.7 Concatenate

4.7.1 Mathematical Definition

Combine multiple tensors along a specific axis:

$$\mathbf{C} = \text{concat}([\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k], \text{dim} = d) \tag{37}$$

Usage in MHA: Combine outputs from each head

$$[B, S, N_H, D_h] \xrightarrow{\text{reshape}} [B, S, N_H \times D_h] = [B, S, D] \tag{38}$$

(Actually implemented as reshape, but conceptually concatenation)

4.7.2 Computational Cost

Concatenate only performs memory copying:

$$\text{FLOPs} = 0 \quad \text{(no arithmetic operations)} \tag{39}$$

4.7.3 Memory

- Inputs: $\sum_i \text{size}(\mathbf{A}_i)$
- Output: $\text{size}(\mathbf{C}) = \sum_i \text{size}(\mathbf{A}_i)$

4.7.4 Backward Pass

Forward: $\mathbf{C} = \text{concat}([\mathbf{A}_1, \ldots, \mathbf{A}_k], \text{dim} = d)$

Backward: Split gradient back to original tensors

$$\frac{\partial L}{\partial \mathbf{A}_i} = \text{split}\left(\frac{\partial L}{\partial \mathbf{C}}, \text{dim} = d\right)_i \tag{40}$$

4.8 Dropout

4.8.1 Mathematical Definition

During training, randomly set activations to zero:

$$\mathbf{Y} = \frac{\mathbf{M} \odot \mathbf{X}}{1 - p} \tag{41}$$

where $\mathbf{M} \sim \text{Bernoulli}(1 - p)$ is a binary mask and p is the dropout rate.

During inference: identity operation ($\mathbf{Y} = \mathbf{X}$)

4.8.2 Computational Cost

Training:

- Mask generation: size($\mathbf{X}$) random samples
- Element-wise multiplication: size($\mathbf{X}$)
- Scaling: size($\mathbf{X}$) divisions
- **Total**: $\approx 2 \times \text{size}(\mathbf{X})$ operations

Inference: 0 FLOPs (identity)

4.8.3 Memory

- Mask: size($\mathbf{X}$) bits (can be optimized)
- Cache for backward: mask must be stored

4.8.4 Backward Pass

Forward (training): $\mathbf{Y} = \mathbf{M} \odot \mathbf{X}/(1 - p)$

Backward:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\mathbf{M} \odot \frac{\partial L}{\partial \mathbf{Y}}}{1 - p} \tag{42}$$

Gradient is zero at positions where mask is zero.

5 Forward Pass Analysis

Having understood the basic operations, we now analyze the forward pass of Multi-Head Attention step-by-step. For each step, we specify input/output shapes, computational cost (FLOPs), and memory usage.

Figure 2 provides a complete visual representation of the MHA forward pass, showing all operations, data dependencies, and tensor shapes.

5.1 Reading the Forward Diagram

The diagram in Figure 2 should be read left-to-right, following the data flow:

Node Types

- **Circular nodes with •**: Matrix multiplication operations
- **Circular nodes with +**: Element-wise addition (with broadcasting)
- **Rectangular nodes**: Auxiliary operations with abbreviations

Arrow Types

- **Single arrows** (→): Data flow (activations)
- **Double arrows** (⇒): Second operand of matrix multiplication (weight matrices, often from below)

Key Observations from the Diagram

1. **Three parallel paths** for Q, K, V projections from normalized input
2. **Attention mechanism** in the center: Q · $K^\top$ → Scale+Mask → Softmax → × V
3. **Head operations**: Reshape splits into multiple heads before attention, then merges after
4. **Output path**: Concatenate → Linear projection → Bias add → Dropout

5.2 Notation

- $\mathbf{X} \in \mathbb{R}^{[B,S,D]}$: Input hidden states
- $\widetilde{\mathbf{W}}_Q, \widetilde{\mathbf{W}}_K, \widetilde{\mathbf{W}}_V \in \mathbb{R}^{[D,D]}$: Query, Key, Value projection weights
- $\widetilde{\mathbf{W}}_O \in \mathbb{R}^{[D,D]}$: Output projection weight
- $\tilde{\mathbf{b}}_O \in \mathbb{R}^D$: Output bias
- N_H: Number of attention heads
- $D_h = D/N_H$: Dimension per head

5.3 Step 0: Layer Normalization

Operation: $\boxed{\text{LN}}$

Input: $\mathbf{X}$ $[B, S, D]$

Output: $\mathbf{X}_{\text{norm}}$ $[B, S, D]$

Computation:

$$\mathbf{X}_{\text{norm}}[b, s, :] = \text{LayerNorm}(\mathbf{X}[b, s, :]) \tag{43}$$

FLOPs: $6BSD = O(BSD)$

Memory:

- Activations: $2BSD$;
- Cache: $2BS + BSD$;
- Parameters: $2D$

5.4 Step 1: Q, K, V Projections

Operation: $\odot\times 3$

Input: $\mathbf{X}_{\text{norm}}\ [B, S, D]$

Output: $\mathbf{Q}_{\text{flat}}, \mathbf{K}_{\text{flat}}, \mathbf{V}_{\text{flat}}$ each $[B, S, D]$

FLOPs: $3 \times 2BSD^2 = 6BSD^2 = O(BSD^2)$

Memory: Weights: $3D^2$; Outputs: $3BSD$

5.5 Step 2: Reshape to Multi-Head

Operation: $\boxed{\text{R}} \times 3$

Input/Output: $[B, S, D] \to [B, N_H, S, D_h]$

FLOPs: 0 (metadata operation)

5.6 Step 3: Attention Scores ($\mathbf{Q} \cdot \mathbf{K}^\top$)

Operation: $\boxed{\text{T}} + \odot$

Input: $\mathbf{Q}, \mathbf{K}$ each $[B, N_H, S, D_h]$

Output: $\mathbf{A}\ [B, N_H, S, S]$

FLOPs: $2BS^2D = O(BS^2D)$ **(Primary $O(S^2)$ bottleneck)**

Memory: BN_HS^2 **(Memory bottleneck for long sequences)**

5.7 Step 4: Scale and Mask

Operation: $\boxed{\text{SM}}$

FLOPs: $2BN_HS^2 = O(BN_HS^2)$

5.8 Step 5: Softmax

Operation: $\boxed{\text{S}}$

FLOPs: $4BN_HS^2 = O(BN_HS^2)$

5.9 Step 6: Attention × Value

Operation: $\odot$

FLOPs: $2BS^2D = O(BS^2D)$ **(Second $O(S^2)$ bottleneck)**

5.10 Steps 7-9: Concatenate, Output Projection, Dropout

FLOPs: $2BSD^2$ (output projection)

5.11 Forward Pass Total Complexity

Table 1: Forward Pass FLOPs Summary

Component	**FLOPs**
Linear Projections	$8BSD^2$
Attention Core	$4BS^2D$
Other Operations	$O(BN_HS^2)$
Total	$8BSD^2 + 4BS^2D + O(BN_HS^2)$

Complexity Analysis:

- **Short sequences ($S \ll D$)**: $O(BSD^2)$ dominates (linear projections)
- **Long sequences ($S \gg D$)**: $O(BS^2D)$ dominates (attention)
- **Crossover**: $S \approx 2D$

6 Backward Pass Analysis

The backward pass receives gradient $\frac{\partial L}{\partial \mathbf{A}_{\text{out}}}$ from the loss function and computes gradients for all parameters and inputs. Generally, backward pass computation is approximately 2x that of forward pass.

Figure 3 shows the complete backward pass through MHA, illustrating gradient flow and weight gradient computations.

6.1 Reading the Backward Diagram

The diagram in Figure 3 should be read right-to-left, following the gradient flow:

Gradient Flow Types

- **Activation gradients**: Flow through the graph structure (e.g., **dQ**, **dK**, **dV**)
- **Weight gradients**: Computed by matrix multiplications involving forward activations and upstream gradients
- **Bias gradients**: Computed via summation over batch and sequence dimensions

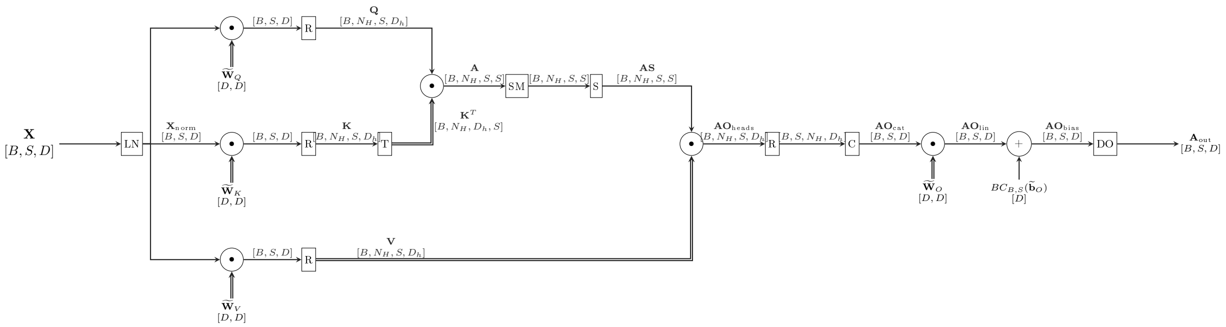

Figure 2: Complete Multi-Head Attention Forward Pass. Circular nodes with • represent matrix multiplications, circular nodes with + represent element-wise addition, and rectangular nodes represent auxiliary operations (LN=LayerNorm, R=Reshape, T=Transpose, SM=Scale+Mask, S=Softmax, C=Concatenate, DO=Dropout). Double arrows (⇒) mark the second operand of matrix multiplications (typically weight matrices). All tensor shapes are annotated on edges.

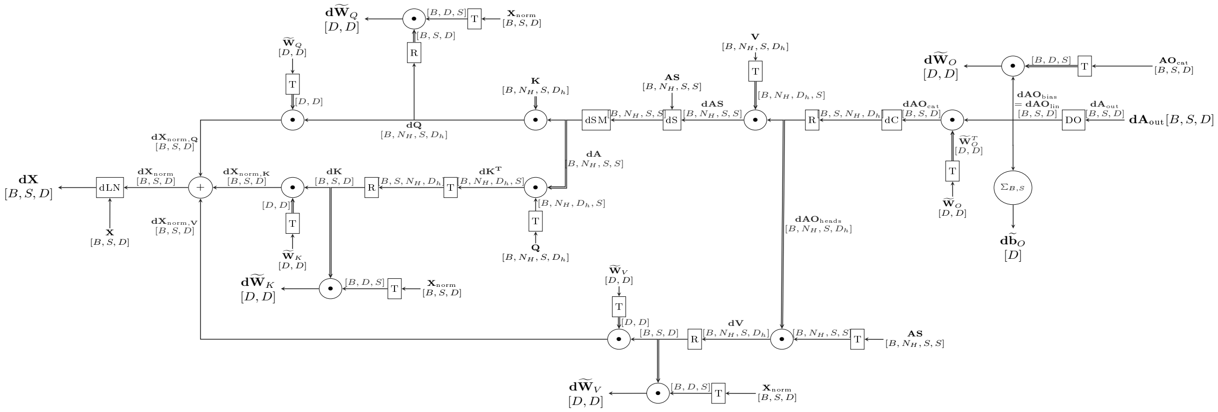

Figure 3: Complete Multi-Head Attention Backward Pass. Gradients flow right-to-left (opposite to forward). The diagram shows both activation gradients (flowing through the computational graph) and weight gradients (computed via additional matrix multiplications with transposed activations). Operations marked with 'd' prefix (dS, dSM, dC, dLN) represent the backward pass through their corresponding forward operations. The summation node $\sum_{B,S}$ computes bias gradients by reducing over batch and sequence dimensions.

Key Components

1. **Gradient entry point**: $\mathbf{dA}_{\text{out}}$ from the next layer
2. **Three gradient branches**: Separate paths for Q, K, V weight gradients
3. **Gradient merging**: The three branches ($\mathbf{dX}_{\text{norm},Q}$, $\mathbf{dX}_{\text{norm},K}$, $\mathbf{dX}_{\text{norm},V}$) sum to form $\mathbf{dX}_{\text{norm}}$
4. **Forward activation reuse**: Many forward activations (shown as separate nodes) are cached and reused for gradient computation

Critical Observations

1. **Backward operations** (dS, dSM, dLN): These are not simple reversals but have their own complex gradient computations (see Section 4)
2. **Double computation**: Each forward MatMul generates two backward MatMuls (one for input gradient, one for weight gradient)
3. **Transpose operations**: Extensive use of transpose for proper dimension alignment in gradient computations
4. **Memory intensive**: Must cache forward activations ($\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, $\mathbf{AS}$, $\mathbf{X}_{\text{norm}}$, etc.)

6.2 Key Observations

1. Each forward operation requires computing gradients for both inputs and weights
2. Total backward FLOPs: $16BSD^2 + 8BS^2D + O(BN_HS^2)$
3. **Ratio**: Backward is approximately 2x forward

6.3 Memory Considerations

- Must cache intermediate activations from forward pass
- Alternative: recompute activations during backward (gradient checkpointing)
- Trade-off: Memory vs computation time

7 Complexity Summary and Optimization

7.1 Overall Complexity

Table 2: Multi-Head Attention Total Complexity

Component	FLOPs	Memory
Forward		
Linear Projections	$8BSD^2$	$4D^2 + 4BSD$
Attention Core	$4BS^2D$	BN_HS^2
Backward		
Linear Projections	$16BSD^2$	$4D^2$
Attention Core	$8BS^2D$	BN_HS^2
Total (Fwd+Bwd)	$24BSD^2 + 12BS^2D$	$8D^2 + BN_HS^2$

7.2 Scaling Characteristics

7.2.1 Sequence Length S

Critical observation: Attention has $O(S^2)$ complexity in both FLOPs and memory.

Example: $B = 32, N_H = 32, S = 8192$, float32

$$\text{Memory}(\mathbf{A}) = 32 \times 32 \times 8192^2 \times 4 \text{ bytes} = 256 \text{ GB} \tag{44}$$

This exceeds single GPU memory (∼80GB)!

7.3 Optimization Strategies

7.3.1 FlashAttention

Problem: Standard attention requires $O(S^2)$ memory.

Solution: IO-aware attention using tiling and recomputation.

Benefits:

- Memory: $O(BS \cdot D)$ instead of $O(BN_H S^2)$ **($S^2 \to S$ reduction)**
- Speed: 2-4x faster wall-clock time
- Exact: Same results as standard attention

7.3.2 Multi-Query / Grouped-Query Attention

Problem: KV cache dominates inference memory.

Solution:

- **MQA**: All heads share single K, V (N_H heads, 1 KV pair)
- **GQA**: Groups of heads share K, V (N_H heads, G KV pairs)

Benefits:

- MQA: 32× KV cache reduction
- GQA: 4× reduction (with $G = 8$)
- Quality: GQA $\approx$ standard $>$ MQA

7.3.3 Gradient Checkpointing

Problem: Must store all activations for backward pass.

Solution: Store only layer boundaries, recompute during backward.

Trade-off:

- Memory: $O(L \cdot BSD) \to O(\sqrt{L} \cdot BSD)$
- Time: ∼33% increase

7.3.4 Mixed Precision Training

FP16/BF16 Benefits:

- Memory: 2x reduction
- Speed: 2-3x faster (with tensor cores)
- Stability: BF16 > FP16

7.4 Practical Recommendations

7.4.1 Training

1. Use gradient checkpointing for $L > 24$ layers
2. Enable FlashAttention (always)
3. Use BF16 mixed precision
4. Maximize batch size to 90-95% GPU memory

7.4.2 Inference

1. Use GQA or MQA for KV cache efficiency
2. Apply INT8 quantization (2-4x speedup)
3. Consider speculative decoding for latency
4. Use sliding window for very long contexts

8 Conclusion

This paper provided a systematic analysis of the Multi-Head Attention mechanism from a computational graph perspective. We explained the mathematical definitions and characteristics of each primitive operation, and quantitatively derived the computational and memory costs at each stage of both forward and backward passes.

Key Findings :

1. **Dual bottleneck**: Linear projections for short sequences, attention for long sequences
2. **Memory bottleneck**: $O(S^2)$ attention scores, cumulative KV cache
3. **Backward is 2x forward**: Due to computing both input and weight gradients

Optimization Direction :

- FlashAttention (essential): $O(S^2) \to O(S)$ memory
- GQA/MQA: 4-32x KV cache reduction
- Mixed precision + fused kernels: 2-3x speedup
- Gradient checkpointing: $\sqrt{L}$ memory reduction

This analysis provides LLM researchers and engineers with the understanding needed to design efficient Transformer implementations.

References

[1] Vaswani, A., et al. (2017). Attention is all you need. NeurIPS.

[2] Dao, T., et al. (2022). FlashAttention: Fast and Memory-Efficient Exact Attention. NeurIPS.

[3] Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. arXiv:1911.02150.

[4] Ainslie, J., et al. (2023). GQA: Training Generalized Multi-Query Transformer Models. arXiv:2305.13245.