Transformer Parallelism: A Visual, Dimension-Oriented Guide

November 4, 2025

Contents

1 Neural Network Basics

In this section, we briefly introduce the core ideas of neural networks for readers with no prior background. We cover the notion of tensors, linear layers, activation functions, and the backpropagation algorithm at a high level.

1.1 What is a Neural Network?

Here we describe the basic idea of a neural network as a composition of linear transformations and non-linear activation functions, operating on vector or matrix inputs.

1.2 Fully-Connected Layers and MLPs

We introduce the multi-layer perceptron (MLP):

- Input/output shapes $[B, D]$.
- Linear layer $W \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$, bias $b \in \mathbb{R}^{D_{\text{out}}}$.
- Activation functions such as ReLU or GELU.

This provides the foundation for understanding the Transformer MLP block.

1.3 Backpropagation at a Glance

We explain the key idea of backpropagation:

- Gradients flowing from the loss to earlier layers.
- Parameter updates using optimizers such as SGD or Adam.

The detailed backward diagrams in later sections are concrete instances of this general principle.

2 From Neural Networks to Transformers

2.1 Sequence Modeling Motivation

We consider sequence modeling tasks such as natural language processing, time series forecasting, and speech processing. In these settings the input is not a single vector but a *sequence* of tokens, each of which is mapped to an embedding. The model must capture dependencies both between nearby tokens and between tokens that are far apart in the sequence.

2.2 The Self-Attention Idea

Transformers address sequence modeling by using self-attention instead of explicit recurrence. At a high level:

- Each token is mapped to an input embedding.
- Self-attention allows every position to attend to every other position in the same sequence.
- Multi-head attention (MHA) uses several attention "heads" in parallel to capture different types of relationships.
- A position-wise feed-forward network (MLP block) processes each token independently after attention.
- Layer normalization, residual connections, and an output projection tie the blocks together and produce final logits or predictions.

In the following sections we make these ideas concrete using dimension-annotated diagrams of a Transformer layer and its parallel variants.

3 Gradients and Backpropagation Basics

This section introduces the basic notions of loss functions, gradients, and backpropagation from an abstract operator point of view. In the figures later in the document we mainly show how tensors flow along edges; the detailed Jacobian matrices are not drawn explicitly. Instead, we use a compact notation for local backward operators such as $\mathrm{d}f$, which map upstream gradients on the outputs of a node to gradients on its inputs.

3.1 Scalar Loss and Gradient Notation

Training a Transformer model is formulated as minimizing a scalar loss function $\mathcal{L}(\theta)$ with respect to the model parameters θ. For a batch of input–target pairs $(\mathbf{X}, \mathbf{Y}_{\text{targets}})$, the model produces predictions

$$\mathbf{Y} = f_\theta(\mathbf{X}),$$

and a scalar loss

$$\mathcal{L} = \mathcal{L}(\mathbf{Y}, \mathbf{Y}_{\text{targets}}).$$

We use the differential-style notation $\mathrm{d}\theta = \partial\mathcal{L}/\partial\theta$ for gradients. For example,

$$\mathrm{d}\mathbf{X} = \frac{\partial\mathcal{L}}{\partial\mathbf{X}}, \quad \mathrm{d}\mathbf{W} = \frac{\partial\mathcal{L}}{\partial\mathbf{W}}, \quad \mathrm{d}\mathbf{b} = \frac{\partial\mathcal{L}}{\partial\mathbf{b}}.$$

In the diagrams, these gradients appear as edges labeled $\mathrm{d}\mathbf{X}$, $\mathrm{d}\mathbf{W}$, etc. together with their tensor shapes such as $[B, S, D]$ or $[D, D]$.

3.2 Single-Input Nodes and Backward Operators

Consider a single node in a computation graph with forward computation

$$\mathbf{y} = f(\mathbf{x}),$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors or tensors. Let $\mathbf{J}_f(\mathbf{x})$ denote the Jacobian of f at $\mathbf{x}$. If the loss $\mathcal{L}$ depends on $\mathbf{y}$, then by the chain rule

$$\mathrm{d}\mathbf{x} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \mathbf{J}_f(\mathbf{x})^T \frac{\partial\mathcal{L}}{\partial\mathbf{y}} = \mathbf{J}_f(\mathbf{x})^T \mathrm{d}\mathbf{y}.$$

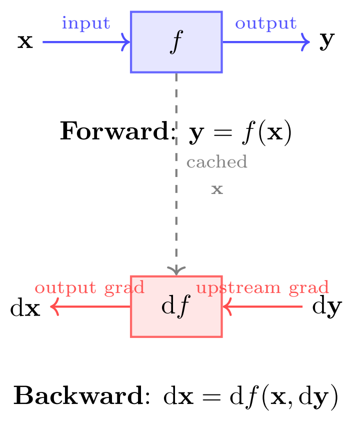
In the figures we do not materialize the Jacobian. Instead we introduce an **abstract backward operator** $\mathrm{d}f$ and write

$$\boxed{\mathrm{d}\mathbf{x} = \mathrm{d}f(\mathbf{x}, \mathrm{d}\mathbf{y})},$$

with the understanding that

$$\mathrm{d}f(\mathbf{x}, \mathrm{d}\mathbf{y}) \equiv \mathbf{J}_f(\mathbf{x})^T \mathrm{d}\mathbf{y}.$$

Graphical representation:



Conceptually, each backward node in the graph implements the local mapping

$$(\mathbf{x}, \mathrm{d}\mathbf{y}) \mapsto \mathrm{d}\mathbf{x},$$

where:

- **Input 1 (cached)**: The forward input $\mathbf{x}$ is cached during the forward pass
- **Input 2 (upstream)**: The upstream gradient $\mathrm{d}\mathbf{y}$ flows from the next layer
- **Output**: The resulting local gradient $\mathrm{d}\mathbf{x}$ flows to the previous layer

Softmax, dropout, layer normalization, and many other operators in a Transformer layer are special cases of this pattern. Their concrete backward rules are described in terms of such operators $\mathrm{d}f$.

3.3 Nodes with Multiple Inputs

Many nodes in a Transformer layer have several inputs. For example, a matrix multiplication node uses both an activation tensor and a weight matrix, and a layer-normalization node uses inputs as well as learned scale and shift parameters. Abstractly, we write

$$\mathbf{y} = f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k),$$

where each $\mathbf{x}_i$ may be a tensor of its own.

Given the upstream gradient $\mathrm{d}\mathbf{y} = \partial\mathcal{L}/\partial\mathbf{y}$, the chain rule yields gradients with respect to all inputs,

$$\mathrm{d}\mathbf{x}_i = \mathbf{J}_{f,\mathbf{x}_i}(\mathbf{x}_1, \ldots, \mathbf{x}_k)^T \mathrm{d}\mathbf{y}, \quad i = 1, \ldots, k,$$

where $\mathbf{J}_{f,\mathbf{x}_i}$ is the Jacobian of f with respect to the i-th input.

We encode this in an abstract backward operator

$$\mathrm{d}_{\mathbf{x}_i} f(\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathrm{d}\mathbf{y}) := \mathbf{J}_{f,\mathbf{x}_i}(\mathbf{x}_1, \ldots, \mathbf{x}_k)^T \mathrm{d}\mathbf{y},$$

so that, for each input,

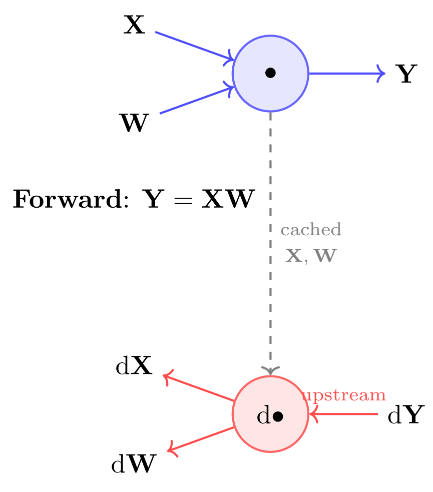
$$\mathrm{d}\mathbf{x}_i = \mathrm{d}_{\mathbf{x}_i} f(\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathrm{d}\mathbf{y}).$$

Collecting all input gradients together, we can also view the backward node as a single vector-valued operator

$$\boxed{(\mathrm{d}\mathbf{x}_1, \ldots, \mathrm{d}\mathbf{x}_k) = \mathrm{d}f(\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathrm{d}\mathbf{y})},$$

whose components are exactly the individual $\mathrm{d}_{\mathbf{x}_i} f(\cdot, \ldots, \cdot, \mathrm{d}\mathbf{y})$.

Example: Matrix Multiplication



Backward: $(\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{W}) = \mathrm{d}f(\mathbf{X}, \mathbf{W}, \mathrm{d}\mathbf{Y})$

Concrete formulas:

$$\mathrm{d}\mathbf{X} = \mathrm{d}\mathbf{Y}\mathbf{W}^T$$
$$\mathrm{d}\mathbf{W} = \mathbf{X}^T\mathrm{d}\mathbf{Y}$$

In the diagrams, this is represented as a single backward node (for example, a node labeled `dMatmul` or `dLN`) with multiple incoming edges carrying the necessary forward inputs and the upstream gradient, and multiple outgoing edges carrying $\mathrm{d}\mathbf{x}_1, \dots, \mathrm{d}\mathbf{x}_k$.

3.4 Computation Graph and Backpropagation

A full Transformer layer can be viewed as a composition of simpler operations:

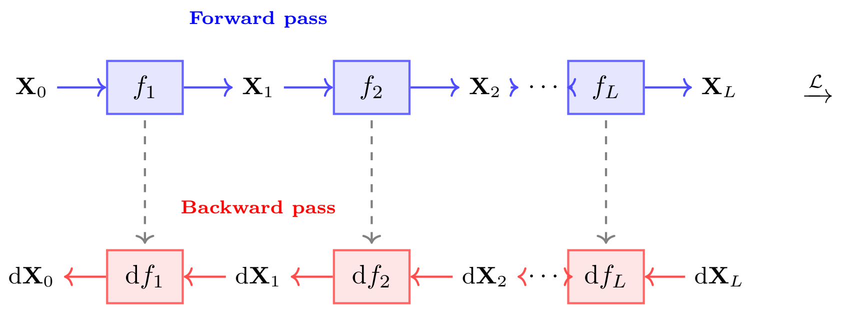
$$\mathbf{X}_0 \xrightarrow{f_1} \mathbf{X}_1 \xrightarrow{f_2} \mathbf{X}_2 \xrightarrow{\cdots} \mathbf{X}_L,$$

where $\mathbf{X}_0$ is the input to the layer and $\mathbf{X}_L$ is the final output before the loss. Each f_ℓ is a local operator such as a matrix multiplication, a nonlinearity, a dropout, or a normalization.

Backpropagation proceeds in reverse order. Starting from $\mathrm{d}\mathbf{X}_L = \partial\mathcal{L}/\partial\mathbf{X}_L$, we apply the corresponding backward operator for each node:

$$\mathrm{d}\mathbf{X}_\ell = \mathrm{d}f_\ell(\mathbf{X}_\ell, \mathrm{d}\mathbf{X}_{\ell+1}), \quad \ell = L-1, L-2, \dots, 0.$$

Graphical representation of a computation chain:



Key observations:

- **Forward**: Data flows left to right through function nodes f_ℓ
- **Backward**: Gradients flow right to left through backward nodes $\mathrm{d}f_\ell$
- **Cache**: Each backward node $\mathrm{d}f_\ell$ needs access to the cached forward state $\mathbf{X}_\ell$
- **Chain rule**: $\mathrm{d}\mathbf{X}_\ell = \mathrm{d}f_\ell(\mathbf{X}_\ell, \mathrm{d}\mathbf{X}_{\ell+1})$ for $\ell = L-1, \dots, 0$

If f_ℓ depends on additional inputs (e.g. parameters), then its backward operator also produces gradients with respect to those inputs, as discussed below.

In the diagrams, we emphasize this process by drawing:

- **forward edges** for $\mathbf{X}_\ell$ flowing into the forward nodes f_ℓ,
- **backward edges** for $\mathrm{d}\mathbf{X}_\ell$ flowing out of the corresponding backward nodes $\mathrm{d}f_\ell$.

The detailed formulas that define each local operator $\mathrm{d}f_\ell$ are hidden inside the node and explained in the operator dictionary of Section 4.

3.5 Parameter Gradients and Updates

Parameters such as weight matrices and bias vectors enter the graph as additional inputs to some node. For example, consider a matrix multiplication

$$\mathbf{Y} = \mathbf{X}\mathbf{W},$$

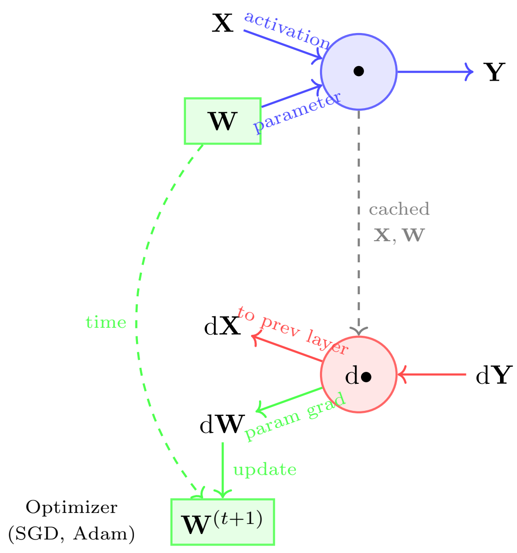
where $\mathbf{X}$ is an activation tensor and $\mathbf{W}$ is a weight matrix. We view this as a function of two inputs,

$$\mathbf{Y} = f(\mathbf{X}, \mathbf{W}).$$

The corresponding backward operator produces both activation and parameter gradients:

$$\boxed{(\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{W}) = \mathrm{d}f(\mathbf{X}, \mathbf{W}, \mathrm{d}\mathbf{Y})}.$$

Parameter gradient flow:



An optimizer then uses the parameter gradients to update the parameters. For example, stochastic gradient descent with learning rate η performs

$$\theta^{(t+1)} = \theta^{(t)} - \eta\, \mathrm{d}\theta^{(t)}.$$

Key distinction:

- **Activation gradients** ($\mathrm{d}\mathbf{X}$): Flow to the previous layer in the backward pass
- **Parameter gradients** ($\mathrm{d}\mathbf{W}$): Accumulated and used by the optimizer to update weights

In this document we do not draw optimizer steps in the diagrams; we only show how $\mathrm{d}\mathbf{W}$ and $\mathrm{d}\mathbf{b}$ are computed by the backward nodes.

3.6 Connection to the Diagrams

The detailed MHA, MLP, and output-projection figures in later sections are best read with this abstract picture in mind:

- Each **forward node** (e.g. `SM`, `S`, `DO`, `LN`, `matmul`) represents a mapping $\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_k)$.
- Each corresponding **backward node** (e.g. `dSM`, `dS`, `dDO`, `dLN`, `dMatmul`) represents the operator

$$(\mathrm{d}\mathbf{x}_1, \dots, \mathrm{d}\mathbf{x}_k) = \mathrm{d}f(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathrm{d}\mathbf{y}),$$

  implemented using the appropriate Jacobian-transpose formulas for that operator.

- Edges labeled with tensors such as $\mathbf{X}$, $\mathbf{H}$, $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, $\mathbf{AS}$, and their gradients $\mathrm{d}\mathbf{X}$, $\mathrm{d}\mathbf{Q}$, $\mathrm{d}\mathbf{W}$, etc., capture only the flow of data, together with compact shape annotations like $[B, S, D]$ or $[B, N_H, S, D_h]$.

In the next section we define the graphical notation and operator dictionary used in the figures, and we specialize the abstract backward operator $\mathrm{d}f$ to concrete nodes such as softmax (`S/dS`), scale/mask (`SM/dSM`), dropout (`DO/dDO`), and layer normalization (`LN/dLN`).

4 Graphical Notation and Figure Conventions

The diagrams in this document are designed to show how tensors flow through a Transformer layer and its parallel variants. This section summarizes the graphical notation, including tensor-shape labels, node types, edge styles, and the small operator dictionary for the most common nodes (matmul, softmax, scale/mask, dropout, layer normalization, communication, and broadcast).

Throughout the figures, the goal is to emphasize the flow of tensors along edges. The exact Jacobian matrices for each operation are not drawn; instead, the backward nodes are understood as the abstract operators $\mathrm{d}f$ introduced in Section 3.

4.1 Tensor Shapes and Index Notation

We use a consistent braced notation for tensor shapes. Instead of writing $\mathbb{R}^{B\times S\times D}$, we annotate edges in the diagrams with labels such as

$$[B, S, D], \quad [B, N_H, S, D_h], \quad [D, D_{ff}],$$

directly next to the arrows. This makes it easier to match each edge to a particular dimension ordering.

The main symbols are:

- B: batch size.
- S: sequence length (number of tokens per sequence).
- D: model (hidden) dimension.
- D_{ff}: intermediate MLP (feed-forward) dimension.
- N_H: number of attention heads.
- D_h: per-head dimension, typically $D_h = D/N_H$.

Typical tensor shapes in the diagrams include:

- $\mathbf{X} \in [B, S, D]$: input or hidden states.
- $\mathbf{H} \in [B, S, D]$: normalized or intermediate states.
- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in [B, N_H, S, D_h]$: projected queries, keys, and values.
- $\mathbf{AS} \in [B, N_H, S, S]$: attention scores after scaling/masking and softmax.
- $\mathbf{Y} \in [B, S, D]$: output of a Transformer block or layer.

Under tensor parallelism (TP) and data parallelism (DP) we mostly keep the same shape notation on edges. For example, a TP shard that actually stores a slice of width D/N_T may still be labeled $[B, S, D]$ in an end-to-end figure when we want to focus on the logical model dimension rather than the physical shard size. When necessary, shard dimensions such as $[B, S, D/N_T]$ or $[B, N_H/N_T, S, D_h]$ are written explicitly.

Gradients use the same shape conventions. For example:

$$\mathrm{d}\mathbf{X} \in [B, S, D], \quad \mathrm{d}\mathbf{W}_Q \in [D, D], \quad \mathrm{d}\mathbf{W}_{\mathrm{up}} \in [D, D_{ff}].$$

4.2 Nodes, Edges, and Arrow Styles

The diagrams are drawn as computation graphs. Nodes represent local operations; edges represent tensors flowing between them.

4.2.1 Forward vs. Backward Arrows

We distinguish between forward and backward flows:

- **Overall flow diagrams** (e.g. the top-level Transformer flow) use:
  - solid arrows for forward activations (e.g. $\mathbf{X}$ to $\mathbf{Y}$);
  - dashed arrows for backward gradients (e.g. $\mathrm{d}\mathbf{Y}$ to $\mathrm{d}\mathbf{X}$).
- **Detailed backward diagrams** (e.g. MHA backward, MLP backward) use thicker arrows with different styles (single vs. double) to distinguish:
  - gradient flow along the main backward path,
  - cached forward values reused as secondary inputs to backward nodes.

In all cases, the arrow direction follows the direction of computation for forward edges and the direction of gradient propagation for backward edges.

4.2.2 Node Types

We use a small set of recurring node types:

Matrix multiplication. A matrix multiplication node is drawn as a circle containing a dot:

$$\bullet$$

In the forward pass this corresponds to an operation such as $\mathbf{Y} = \mathbf{X}\mathbf{W}$. In the backward diagrams the corresponding dNode (e.g. `dMatmul`) implements the abstract backward operator

$$(\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{W}) = \mathrm{d}f(\mathbf{X}, \mathbf{W}, \mathrm{d}\mathbf{Y}),$$

as described in Section 3.

Addition and residuals. Elementwise addition is drawn as a circle containing a plus:

$$+$$

This is used for bias addition, combining residual connections, and aggregating multiple gradient contributions. Small circles labeled $\sum$ denote explicit summations over batch and/or sequence dimensions (e.g. $\sum_{B,S}$ for bias-gradient accumulation).

Generic rectangular operators. Many local operations (layer normalization, nonlinearity, dropout, scale/mask, reshape, transpose) are drawn as rectangles with short labels such as `LN`, `GL`, `DO`, `SM`, `R`, or `T`. Their backward counterparts are labeled with a leading "d", for example `dLN`, `dDO`, `dSM`. Each such dNode implements the corresponding backward operator $\mathrm{d}f(\cdot, \ldots, \cdot, \mathrm{d}\mathbf{y})$.

Communication nodes. Distributed communication collectives are drawn as small rectangular nodes with labels such as:

- `AR`: All-Reduce.
- `AG`: All-Gather.

The arrows entering/leaving these nodes indicate which tensors are participating in the collective, and the shape annotations show the logical tensor size before and after the communication.

4.3 Forward and Backward Nodes: Abstract View

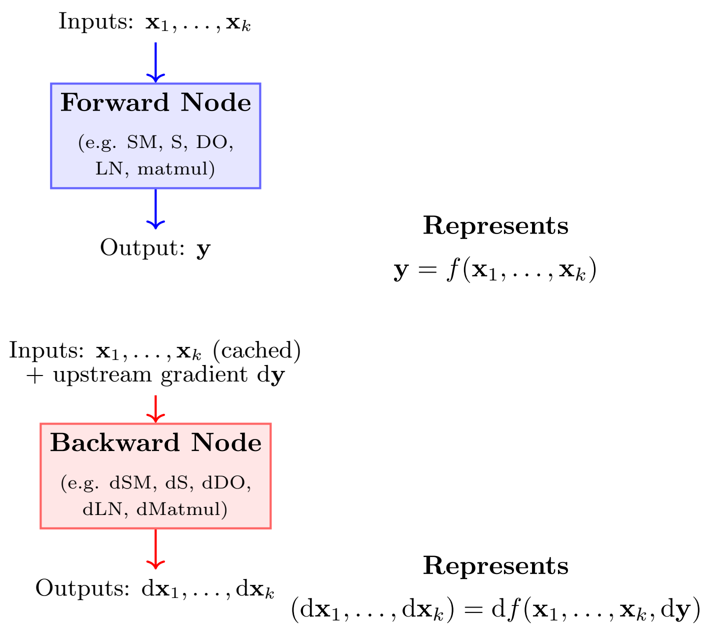
Most operators in a Transformer layer can be written abstractly as

$$\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_k),$$

where $\mathbf{x}_1, \dots, \mathbf{x}_k$ include both activations and parameters (such as weight matrices or bias vectors). In Section 3 we introduced the abstract backward operators

$$\mathrm{d}\mathbf{x}_i = \mathrm{d}_{\mathbf{x}_i} f(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathrm{d}\mathbf{y}), \quad i = 1, \dots, k.$$

How to read nodes in our diagrams:


Inputs: $\mathbf{x}_1, \dots, \mathbf{x}_k$
Forward Node
(e.g. SM, S, DO, LN, matmul)
Output: $\mathbf{y}$
Represents
$\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_k)$
Inputs: $\mathbf{x}_1, \dots, \mathbf{x}_k$ (cached) + upstream gradient $\mathrm{d}\mathbf{y}$
Backward Node
(e.g. dSM, dS, dDO, dLN, dMatmul)
Outputs: $\mathrm{d}\mathbf{x}_1, \dots, \mathrm{d}\mathbf{x}_k$
Represents
$(\mathrm{d}\mathbf{x}_1, \dots, \mathrm{d}\mathbf{x}_k) = \mathrm{d}f(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathrm{d}\mathbf{y})$

In the diagrams:

- A **forward node** (e.g. `S`, `SM`, `DO`, `LN`, `matmul`) represents the mapping $\mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_k)$.
- The corresponding **backward node** (labeled with a leading "d", e.g. `dS`, `dSM`, `dDO`, `dLN`) represents the mapping

$$(\mathrm{d}\mathbf{x}_1, \dots, \mathrm{d}\mathbf{x}_k) = \mathrm{d}f(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathrm{d}\mathbf{y}),$$

  i.e. it consumes the upstream gradient $\mathrm{d}\mathbf{y}$ and the necessary cached forward inputs, and produces gradients for all inputs.

The purpose of this section is not to re-derive all Jacobian formulas, but to provide a dictionary that tells the reader what each node means in the diagrams and how to read its inputs/outputs at the level of tensors and gradients.

4.4 Operator Dictionary: Forward and Backward

We now describe the most common node types used in the MHA, MLP, and output-projection diagrams. For each operator we briefly summarize the forward computation and the corresponding backward operator in the abstract notation $\mathrm{d}f(\cdot, \dots, \cdot, \mathrm{d}\mathbf{y})$.

4.4.1 Matrix Multiplication (Matmul)

Forward. A matmul node computes
$$\mathbf{Y} = \mathbf{X}\mathbf{W},$$
with shapes such as $\mathbf{X} \in [B, S, D]$, $\mathbf{W} \in [D, D]$, and $\mathbf{Y} \in [B, S, D]$. The same pattern appears in the MLP block with $\mathbf{W}_{\text{up}} \in [D, D_{ff}]$ or $\mathbf{W}_{\text{down}} \in [D_{ff}, D]$.

Backward. The backward node `dMatmul` implements
$$(\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{W}) = \mathrm{d}f(\mathbf{X}, \mathbf{W}, \mathrm{d}\mathbf{Y}),$$
with the usual formulas
$$\mathrm{d}\mathbf{X} = \mathrm{d}\mathbf{Y}\mathbf{W}^T, \quad \mathrm{d}\mathbf{W} = \mathbf{X}^T\mathrm{d}\mathbf{Y},$$
applied with appropriate reshaping for batched tensors. In the diagrams, $\mathbf{X}$ and $\mathbf{W}$ (or their transposes) are supplied to the `dMatmul` node via double arrows, and outgoing arrows carry $\mathrm{d}\mathbf{X}$ and $\mathrm{d}\mathbf{W}$.

4.4.2 Broadcast (BC)

In many diagrams we annotate bias addition by a term such as $\mathrm{BC}_{B,S}(\mathbf{b}_0)$, which denotes a logical broadcast of a 1-D bias vector $\mathbf{b}_0 \in [D]$ or $[D_{ff}]$ across the batch and sequence dimensions to match a tensor of shape $[B, S, D]$ or $[B, S, D_{ff}]$.

Forward. Conceptually,
$$\mathbf{Y} = \mathbf{X} + \mathrm{BC}_{B,S}(\mathbf{b}_0),$$
where $\mathrm{BC}_{B,S}(\mathbf{b}_0)$ is a tensor in $[B, S, D]$ obtained by repeating $\mathbf{b}_0$ over the B and S dimensions. In the diagrams we typically draw only the addition node and label the edge near the bias with $\mathrm{BC}_{B,S}(\mathbf{b}_0)$ to indicate that the bias is broadcast in this way.

Backward. The backward contribution to $\mathrm{d}\mathbf{b}_0$ is obtained by summing $\mathrm{d}\mathbf{Y}$ over the broadcast dimensions:
$$\mathrm{d}\mathbf{b}_0 = \sum_{b=1}^{B}\sum_{s=1}^{S} \mathrm{d}\mathbf{Y}_{b,s,:},$$
which is represented in the diagrams by a small summation node labeled $\sum_{B,S}$. The gradient $\mathrm{d}\mathbf{X}$ simply inherits $\mathrm{d}\mathbf{Y}$, since the addition is symmetric.

4.4.3 Scale/Mask Node (SM, dSM)

In the attention mechanism, raw scores $\mathbf{A}$ from $\mathbf{Q}\mathbf{K}^T$ are scaled and masked before softmax. This is represented by a node labeled `SM` (scale/mask).

Forward (SM). Given attention scores $\mathbf{A} \in [B, N_H, S, S]$, the `SM` node computes
$$\mathbf{Z} = \mathrm{SM}(\mathbf{A}) = \alpha\mathbf{A} + \mathbf{M},$$
where $\alpha = 1/\sqrt{D_h}$ is a scalar and $\mathbf{M}$ encodes the mask (e.g. large negative values at disallowed positions). The shape of $\mathbf{Z}$ matches $\mathbf{A}$. The forward pass caches the scaling factor and the mask pattern.

Backward (dSM). The backward node `dSM` is the abstract operator
$$\mathrm{d}\mathbf{A} = \mathrm{dSM}(\mathbf{A}, \mathrm{d}\mathbf{Z}),$$
with
$$\mathrm{dSM}(\mathbf{A}, \mathrm{d}\mathbf{Z}) = \alpha\,\mathrm{d}\mathbf{Z},$$
and no gradient is propagated into the fixed mask $\mathbf{M}$. In the diagram, $\mathbf{A}$ and $\mathrm{d}\mathbf{Z}$ enter the `dSM` node, and $\mathrm{d}\mathbf{A}$ exits as the gradient with respect to the raw attention scores.

4.4.4 Softmax Node (S, dS)

The node labeled `S` performs softmax over the key dimension of the attention scores.

Forward (S). For a fixed batch b, head h, and query position s, let $\mathbf{z} \in \mathbb{R}^S$ be the vector of scores over keys. Softmax produces

$$\mathbf{p} = \mathrm{softmax}(\mathbf{z}), \quad p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

This is applied to every (b, h, s), so the overall shape $[B, N_H, S, S]$ is preserved. The forward pass typically caches $\mathbf{p}$ (or $\mathbf{z}$).

Backward (dS). The backward node `dS` implements the local mapping

$$\mathrm{d}\mathbf{z} = \mathrm{dS}(\mathbf{z}, \mathrm{d}\mathbf{p}),$$

which, in Jacobian form, is

$$\mathrm{dS}(\mathbf{z}, \mathrm{d}\mathbf{p}) = \mathbf{J}_{\mathrm{softmax}}(\mathbf{z})^T \mathrm{d}\mathbf{p}.$$

In practice this is computed using the standard softmax backward formula. In the diagrams, the `dS` node has incoming edges carrying $\mathbf{p}$ (or $\mathbf{z}$) and $\mathrm{d}\mathbf{p}$, and an outgoing edge carrying $\mathrm{d}\mathbf{z}$.

4.4.5 Nonlinearities and Dropout (GL, dGL, DO, dDO)

Rectangular nodes labeled `GL`, `GELU`, or similar denote elementwise nonlinearities; nodes labeled `DO` denote dropout.

Forward (GL / DO). For a generic scalar nonlinearity g,

$$\mathbf{Y} = g(\mathbf{X})$$

is applied elementwise. For dropout we write

$$\mathbf{Y} = \mathrm{DO}(\mathbf{X}; \mathbf{m}) = \mathbf{m} \odot \mathbf{X},$$

where $\mathbf{m}$ is a binary mask of the same shape as $\mathbf{X}$ and $\odot$ denotes elementwise multiplication. The mask $\mathbf{m}$ is cached in the forward pass.

Backward (dGL / dDO). For a generic nonlinearity, the backward node `dGL` implements

$$\mathrm{d}\mathbf{X} = \mathrm{dGL}(\mathbf{X}, \mathrm{d}\mathbf{Y}) = \mathrm{d}\mathbf{Y} \odot g'(\mathbf{X}),$$

using the forward input $\mathbf{X}$ from the cache.

For dropout, the backward node `dDO` implements

$$\mathrm{d}\mathbf{X} = \mathrm{dDO}(\mathbf{X}, \mathrm{d}\mathbf{Y}) = \mathbf{m} \odot \mathrm{d}\mathbf{Y}.$$

In the diagrams, the node labeled `dDO` consumes the upstream gradient $\mathrm{d}\mathbf{Y}$ and the cached mask $\mathbf{m}$, and produces $\mathrm{d}\mathbf{X}$.

4.4.6 Layer Normalization (LN, dLN)

Layer normalization nodes are labeled `LN` in the forward pass and `dLN` in the backward pass.

Forward (LN). Given $\mathbf{X} \in [B, S, D]$, layer normalization computes

$$\mathbf{H} = \mathrm{LN}(\mathbf{X}; \gamma, \beta),$$

by normalizing each $[D]$-dimensional vector at fixed B, S, and applying learned scale and shift parameters $\gamma, \beta \in [D]$. The forward pass caches per-position mean/variance as well as γ and β.

Backward (dLN). The backward node `dLN` implements

$$(\mathrm{d}\mathbf{X}, \mathrm{d}\gamma, \mathrm{d}\beta) = \mathrm{dLN}(\mathbf{X}, \gamma, \beta, \mathrm{d}\mathbf{H}, \mathrm{stats}),$$

where stats denotes the cached means and variances. The explicit formulas follow from the standard layer-normalization backward derivation; in the diagrams we treat `dLN` as a single node that consumes $\mathbf{X}$, γ, the cached statistics, and $\mathrm{d}\mathbf{H}$, and produces three outgoing gradient edges $\mathrm{d}\mathbf{X}$, $\mathrm{d}\gamma$, and $\mathrm{d}\beta$.

4.4.7 Communication Nodes (AR, AG)

In tensor-parallel (TP), data-parallel (DP), and hybrid DP+TP settings, collective communication operations synchronize tensors across devices.

All-Reduce (AR). An All-Reduce node `AR` takes as input a tensor shard from each participant and outputs the elementwise reduced tensor (typically a sum), optionally divided by the number of participants for averaging. For example, in DP, weight gradients $\mathrm{d}\mathbf{W}$ of shape $[D, D]$ are All-Reduced across all data-parallel ranks before the optimizer step.

All-Gather (AG). An All-Gather node `AG` concatenates or aggregates tensor shards across a parallel group to reconstruct a full tensor. For example, in TP, partial outputs along a hidden dimension may be gathered to form a full $[B, S, D]$ tensor.

In the diagrams, these communication nodes are treated as pure operators on tensors; their backward behavior (e.g. gradient flow through `AR` and `AG`) is implicit in the symmetry of the operations.

4.5 Reading the Detailed MHA and MLP Figures

With the conventions above, the large MHA and MLP forward/backward figures can be read as follows:

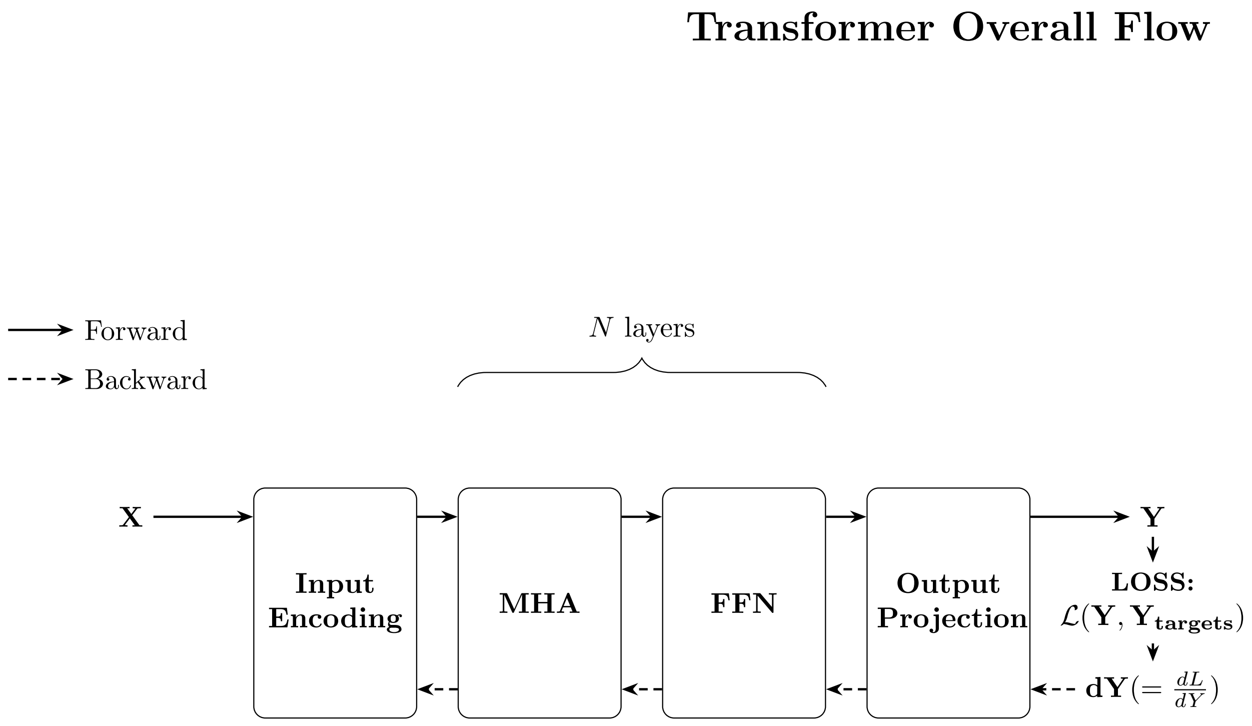
- **Edges** indicate the flow of tensors (activations or gradients), annotated with shapes like $[B, S, D]$ or $[B, N_H, S, D_h]$.
- **Forward nodes** (`S`, `SM`, `DO`, `LN`, `matmul`, `reshape`, `transpose`) compute local functions $f(\mathbf{x}_1, \ldots, \mathbf{x}_k)$.
- **Backward nodes** (`dS`, `dSM`, `dDO`, `dLN`, `dMatmul`) implement the corresponding backward operators $\mathrm{d}f(\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathrm{d}\mathbf{y})$, producing gradients for all inputs.
- **Broadcast labels** such as $\mathrm{BC}_{B,S}(\mathbf{b}_0)$ indicate that a bias vector is conceptually expanded to match a higher-rank tensor before addition.
- **Communication nodes** (`AR`, `AG`) indicate where collective operations occur in TP, DP, or DP+TP settings, and their shape annotations show the logical dimensions involved.

These conventions allow the reader to understand the data and gradient flows at a glance, without being distracted by low-level indexing, while still being precise enough to derive the underlying equations when needed.

5 Single-Node Transformer: Forward and Backward

This section presents the full Transformer layer running on a single node (no parallelism). We emphasize tensor shapes and data flow for both forward and backward passes.

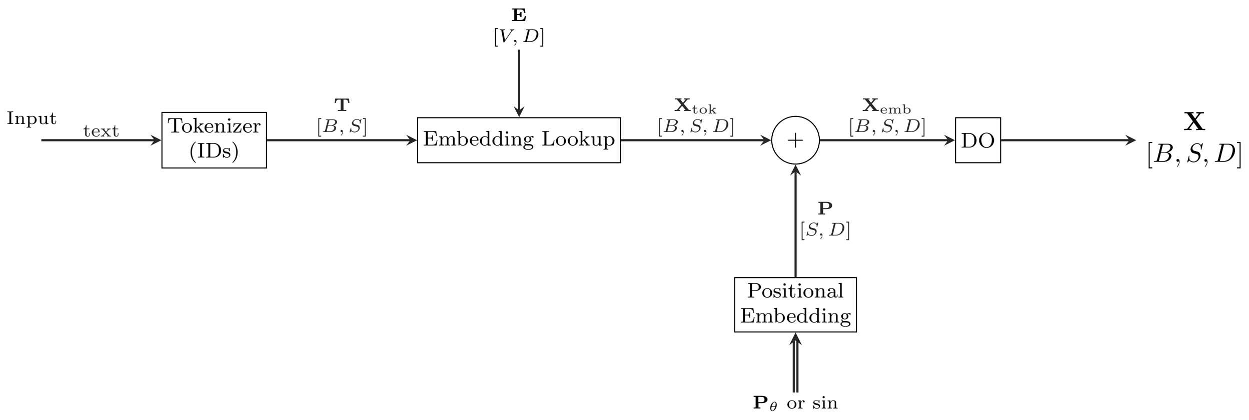
5.1 Overall Transformer Layer Flow



5.2 Input Embedding Layer

5.2.1 Forward Pass

Input → Embedding → LN (Input to MHA)

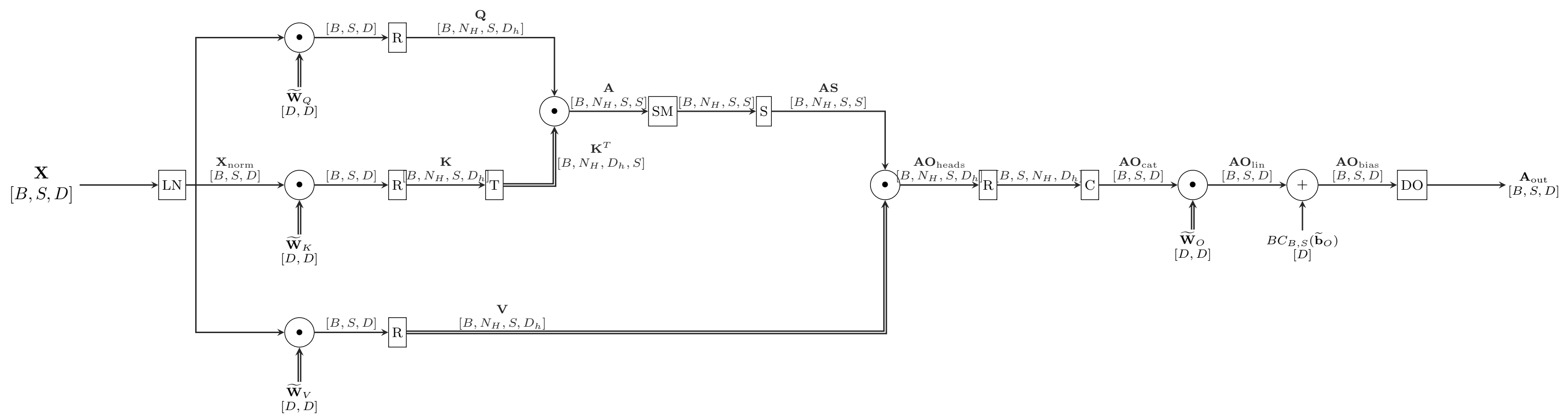


5.3 Multi-Head Attention (MHA)

Multi-Head Attention enables the model to jointly attend to information from different representation sub-spaces.

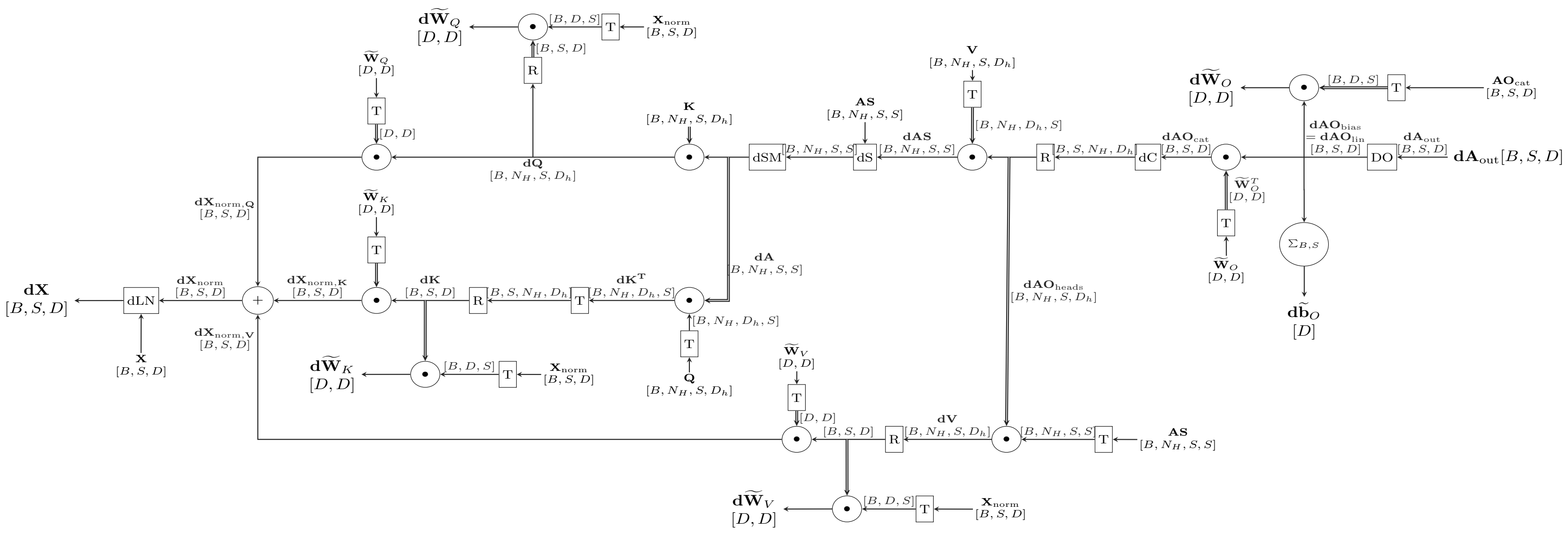
5.3.1 Forward Pass

Multi-Head Attention Forward Pass



5.3.2 Backward Pass

Multi-Head Attention Backward Pass

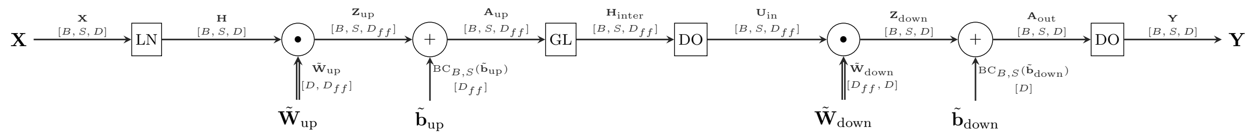


$\mathbf{d}\widetilde{\mathbf{W}}_Q$ $[D, D]$
$[B, D, S]$
$\mathbf{X}_{\text{norm}}$ $[B, S, D]$
$[B, S, D]$
R
$\widetilde{\mathbf{W}}_Q$ $[D, D]$
T
$[D, D]$
$\mathbf{dQ}$ $[B, N_H, S, D_h]$
$\mathbf{K}$ $[B, N_H, S, D_h]$
dSM
$[B, N_H, S, S]$
$\mathbf{AS}$ $[B, N_H, S, S]$
dS
$\mathbf{dAS}$ $[B, N_H, S, S]$
$\mathbf{V}$ $[B, N_H, S, D_h]$
$[B, N_H, D_h, S]$
$[B, S, N_H, D_h]$
dC
$\mathbf{dAO}_{\text{cat}}$ $[B, S, D]$
$\mathbf{d}\widetilde{\mathbf{W}}_O$ $[D, D]$
$[B, D, S]$
$\mathbf{AO}_{\text{cat}}$ $[B, S, D]$
$\mathbf{dAO}_{\text{bias}}$ $= \mathbf{dAO}_{\text{lin}}$ $[B, S, D]$
$\mathbf{dA}_{\text{out}}$ $[B, S, D]$
DO
$\mathbf{dA}_{\text{out}}[B, S, D]$
$\widetilde{\mathbf{W}}_O^T$ $[D, D]$
$\widetilde{\mathbf{W}}_O$ $[D, D]$
$\Sigma_{B,S}$
$\mathbf{d}\widetilde{\mathbf{b}}_O$ $[D]$
$\mathbf{dX}_{\text{norm},\mathbf{Q}}$ $[B, S, D]$
$\widetilde{\mathbf{W}}_K$ $[D, D]$
$\mathbf{dA}$ $[B, N_H, S, S]$
$\mathbf{dAO}_{\text{heads}}$ $[B, N_H, S, D_h]$
$\mathbf{dX}$ $[B, S, D]$
dLN
$\mathbf{dX}_{\text{norm}}$ $[B, S, D]$
$\mathbf{dX}_{\text{norm},\mathbf{K}}$ $[B, S, D]$
$\mathbf{dK}$ $[B, S, D]$
$[B, S, N_H, D_h]$
$\mathbf{dK}^{\mathbf{T}}$ $[B, N_H, D_h, S]$
$[B, N_H, D_h, S]$
$\mathbf{Q}$ $[B, N_H, S, D_h]$
$\mathbf{X}$ $[B, S, D]$
$\mathbf{dX}_{\text{norm},\mathbf{V}}$ $[B, S, D]$
$\mathbf{d}\widetilde{\mathbf{W}}_K$ $[D, D]$
$[B, D, S]$
$\mathbf{X}_{\text{norm}}$ $[B, S, D]$
$\widetilde{\mathbf{W}}_V$ $[D, D]$
$[D, D]$
$[B, S, D]$
$\mathbf{dV}$ $[B, N_H, S, D_h]$
$[B, N_H, S, S]$
$\mathbf{AS}$ $[B, N_H, S, S]$
$\mathbf{d}\widetilde{\mathbf{W}}_V$ $[D, D]$
$[B, D, S]$
$\mathbf{X}_{\text{norm}}$ $[B, S, D]$

5.4 Feed-Forward Network (MLP / FFN)

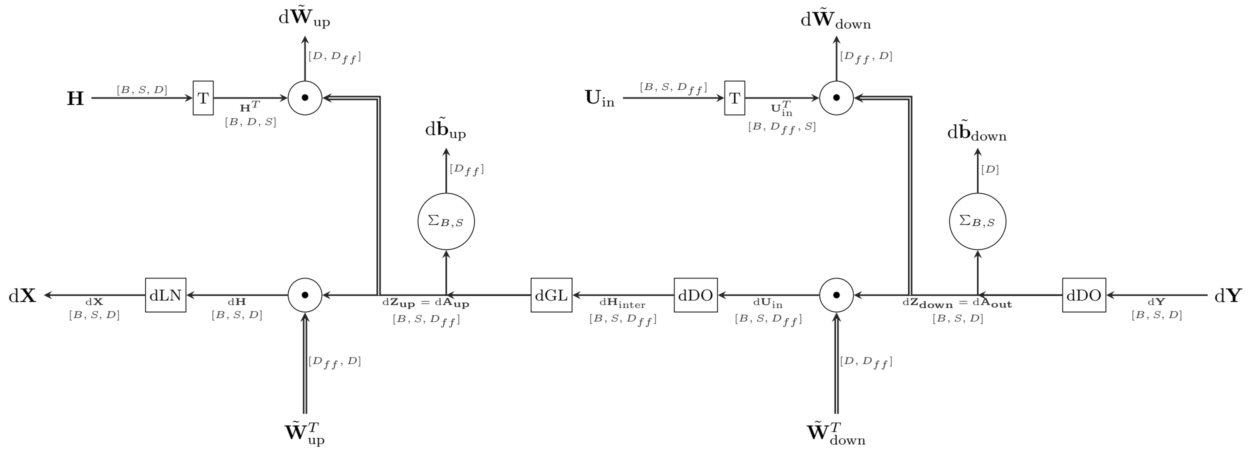
5.4.1 Forward Pass

MLP Forward Pass


$\mathbf{X}$
$\mathbf{X}$ $[B, S, D]$
LN
$\mathbf{H}$ $[B, S, D]$
$\odot$
$\tilde{\mathbf{W}}_{\text{up}}$ $[D, D_{ff}]$
$\tilde{\mathbf{W}}_{\text{up}}$
$\mathbf{Z}_{\text{up}}$ $[B, S, D_{ff}]$
$+$
$\text{BC}_{B,S}(\tilde{\mathbf{b}}_{\text{up}})$ $[D_{ff}]$
$\tilde{\mathbf{b}}_{\text{up}}$
$\mathbf{A}_{\text{up}}$ $[B, S, D_{ff}]$
GL
$\mathbf{H}_{\text{inter}}$ $[B, S, D_{ff}]$
DO
$\mathbf{U}_{\text{in}}$ $[B, S, D_{ff}]$
$\odot$
$\tilde{\mathbf{W}}_{\text{down}}$ $[D_{ff}, D]$
$\tilde{\mathbf{W}}_{\text{down}}$
$\mathbf{Z}_{\text{down}}$ $[B, S, D]$
$+$
$\text{BC}_{B,S}(\tilde{\mathbf{b}}_{\text{down}})$ $[D]$
$\tilde{\mathbf{b}}_{\text{down}}$
$\mathbf{A}_{\text{out}}$ $[B, S, D]$
DO
$\mathbf{Y}$ $[B, S, D]$
$\mathbf{Y}$

5.4.2 Backward Pass

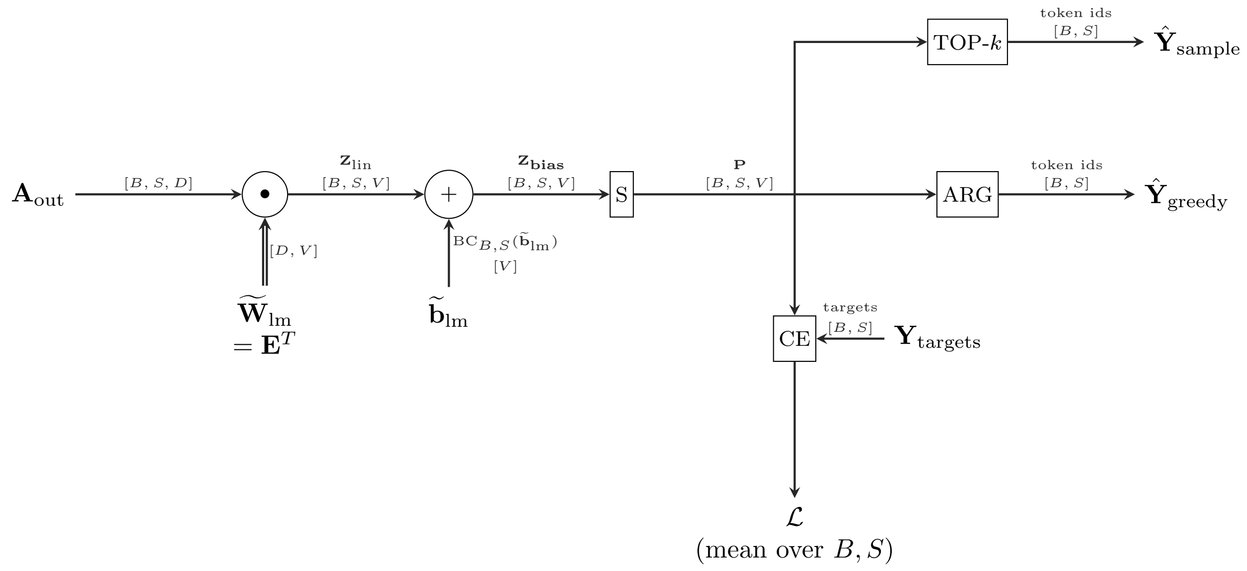
MLP Backward Pass


$d\tilde{\mathbf{W}}_{\text{up}}$
$[D, D_{ff}]$
$\mathbf{H}$
$[B, S, D]$
T
$\mathbf{H}^T$
$[B, D, S]$
$d\tilde{\mathbf{b}}_{\text{up}}$
$[D_{ff}]$
$\Sigma_{B,S}$
$d\mathbf{X}$
$d\mathbf{X}$
$[B, S, D]$
dLN
$d\mathbf{H}$
$[B, S, D]$
$d\mathbf{Z}_{\mathbf{up}} = d\mathbf{A}_{\mathbf{up}}$
$[B, S, D_{ff}]$
$[D_{ff}, D]$
$\tilde{\mathbf{W}}_{\text{up}}^T$
dGL
$d\mathbf{H}_{\text{inter}}$
$[B, S, D_{ff}]$
dDO
$d\mathbf{U}_{\text{in}}$
$[B, S, D_{ff}]$
$d\tilde{\mathbf{W}}_{\text{down}}$
$[D_{ff}, D]$
$\mathbf{U}_{\text{in}}$
$[B, S, D_{ff}]$
T
$\mathbf{U}_{\text{in}}^T$
$[B, D_{ff}, S]$
$d\tilde{\mathbf{b}}_{\text{down}}$
$[D]$
$\Sigma_{B,S}$
$d\mathbf{Z}_{\mathbf{down}} = d\mathbf{A}_{\mathbf{out}}$
$[B, S, D]$
dDO
$d\mathbf{Y}$
$[B, S, D]$
$d\mathbf{Y}$
$[D, D_{ff}]$
$\tilde{\mathbf{W}}_{\text{down}}^T$

5.5 Output Projection and Loss

5.5.1 Forward Pass (Logits, Softmax, Loss)

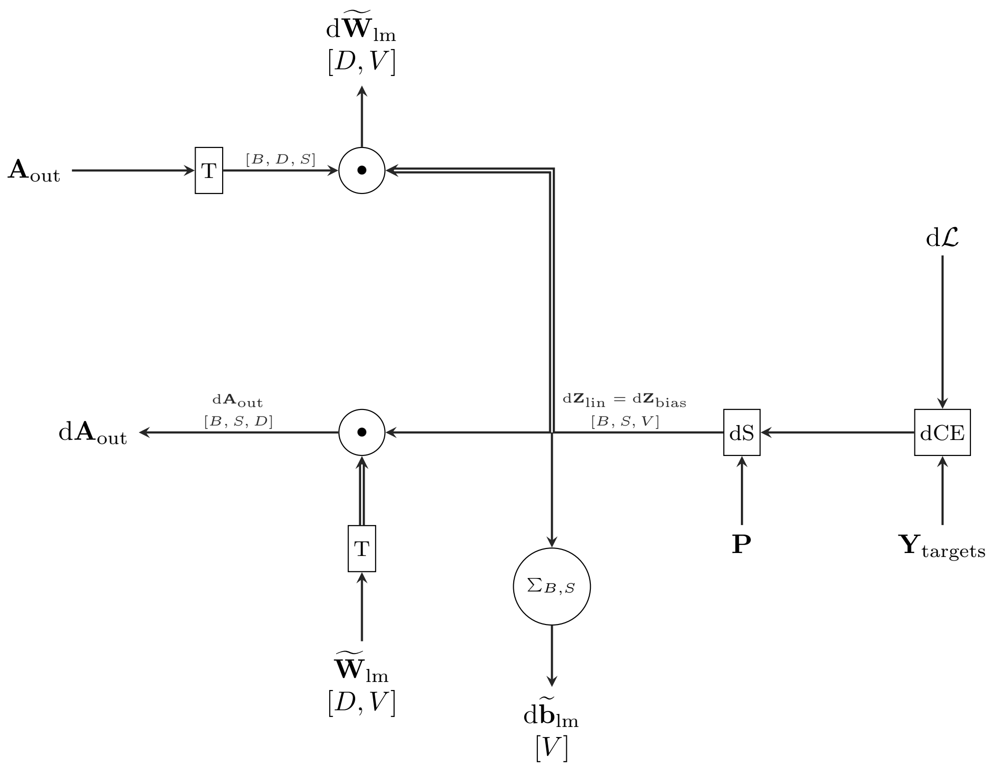
Token Generation & Loss (Forward)



$\mathbf{A}_{\text{out}}$
$[B, S, D]$
$\odot$
$[D, V]$
$\widetilde{\mathbf{W}}_{\text{lm}} = \mathbf{E}^T$
$\mathbf{z}_{\text{lin}}$
$[B, S, V]$
$+$
$\text{BC}_{B,S}(\widetilde{\mathbf{b}}_{\text{lm}})$
$[V]$
$\widetilde{\mathbf{b}}_{\text{lm}}$
$\mathbf{z}_{\mathbf{bias}}$
$[B, S, V]$
S
$\mathbf{P}$
$[B, S, V]$
TOP-k
token ids
$[B, S]$
$\hat{\mathbf{Y}}_{\text{sample}}$
ARG
token ids
$[B, S]$
$\hat{\mathbf{Y}}_{\text{greedy}}$
CE
targets
$[B, S]$
$\mathbf{Y}_{\text{targets}}$
$\mathcal{L}$
(mean over B, S)

5.5.2 Backward Pass

Token Generation & Loss — Backward (Corrected)



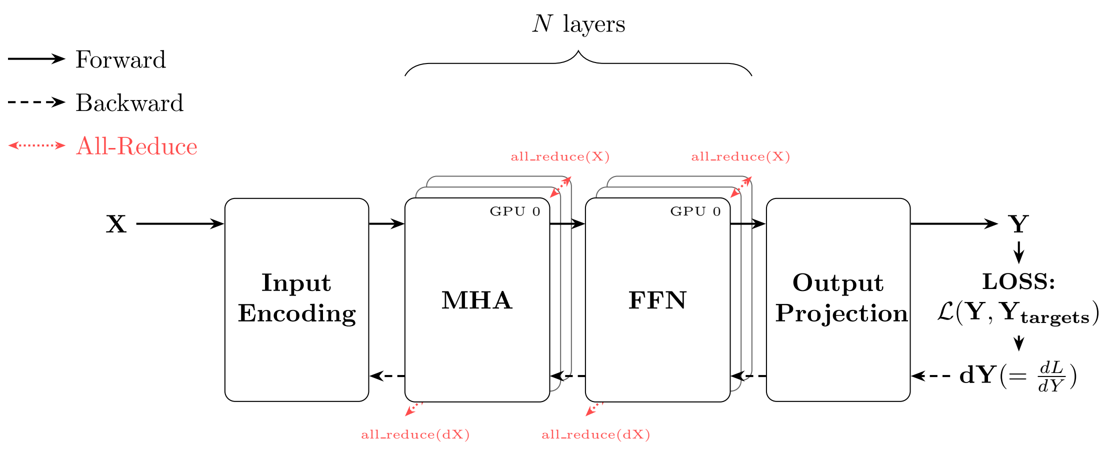
$\mathrm{d}\widetilde{\mathbf{W}}_{\mathrm{lm}}$
$[D, V]$
$\mathbf{A}_{\mathrm{out}}$
T
$[B, D, S]$
$\mathrm{d}\mathcal{L}$
$\mathrm{d}\mathbf{A}_{\mathrm{out}}$
$[B, S, D]$
$\mathrm{d}\mathbf{Z}_{\mathrm{lin}} = \mathrm{d}\mathbf{Z}_{\mathrm{bias}}$
$[B, S, V]$
dS
dCE
$\mathrm{d}\mathbf{A}_{\mathrm{out}}$
T
$\Sigma_{B,S}$
$\mathbf{P}$
$\mathbf{Y}_{\mathrm{targets}}$
$\widetilde{\mathbf{W}}_{\mathrm{lm}}$
$[D, V]$
$\mathrm{d}\widetilde{\mathbf{b}}_{\mathrm{lm}}$
$[V]$

6 Tensor Parallelism (TP)

In tensor parallelism, weight matrices are partitioned across multiple devices along certain dimensions. Each device computes on its own shard, and collective operations (e.g., All-Reduce, All-Gather) synchronize intermediate results.

6.1 TP Overview and End-to-End Flow

Transformer Overall Flow (TP with 3 GPUs)

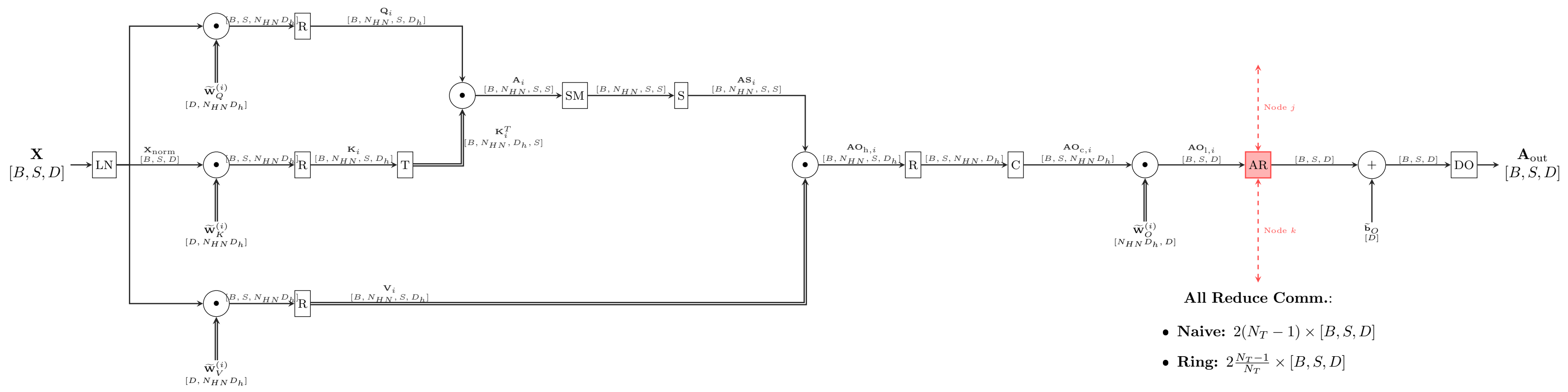


Tensor Parallelism:
- Each GPU processes a shard of the weight matrix
- All-Reduce synchronizes partial results
- Forward: after row-parallel ops
- Backward: after column-parallel ops

6.2 MHA with Tensor Parallelism

6.2.1 Forward Pass

Multi-Head Attention Forward Pass (Node i)



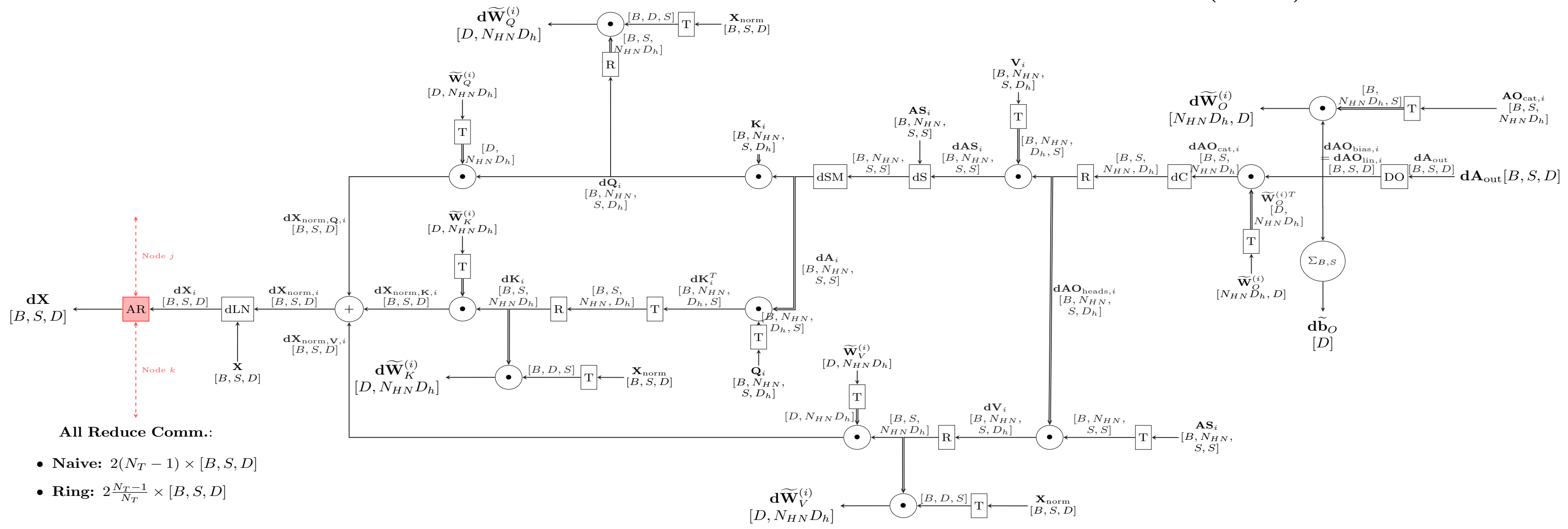
$\mathbf{X}$ $[B, S, D]$
LN
$\mathbf{X}_{\text{norm}}$ $[B, S, D]$
$[B, S, N_{HN} D_h]$
R
$\mathbf{Q}_i$ $[B, N_{HN}, S, D_h]$
$\widetilde{\mathbf{W}}_Q^{(i)}$ $[D, N_{HN} D_h]$
$\mathbf{K}_i$ $[B, N_{HN}, S, D_h]$
T
$\mathbf{K}_i^T$ $[B, N_{HN}, D_h, S]$
$\widetilde{\mathbf{W}}_K^{(i)}$ $[D, N_{HN} D_h]$
$\mathbf{V}_i$ $[B, N_{HN}, S, D_h]$
$\widetilde{\mathbf{W}}_V^{(i)}$ $[D, N_{HN} D_h]$
$\mathbf{A}_i$ $[B, N_{HN}, S, S]$
SM
$[B, N_{HN}, S, S]$
S
$\mathbf{AS}_i$ $[B, N_{HN}, S, S]$
$\mathbf{AO}_{\text{h},i}$ $[B, N_{HN}, S, D_h]$
$[B, S, N_{HN}, D_h]$
C
$\mathbf{AO}_{\text{c},i}$ $[B, S, N_{HN} D_h]$
$\widetilde{\mathbf{W}}_O^{(i)}$ $[N_{HN} D_h, D]$
$\mathbf{AO}_{\text{l},i}$ $[B, S, D]$
AR
Node j
Node k
$[B, S, D]$
$+$
$\mathbf{b}_O$ $[D]$
DO
$\mathbf{A}_{\text{out}}$ $[B, S, D]$

All Reduce Comm.:

- **Naive:** $2(N_T - 1) \times [B, S, D]$
- **Ring:** $2\frac{N_T - 1}{N_T} \times [B, S, D]$

6.2.2 Backward Pass

Multi-Head Attention Backward Pass (Node i)



$\mathbf{dA}_{\text{out}}[B,S,D]$
DO
$\mathbf{dA}_{\text{out}}$ $[B,S,D]$
$\mathbf{dAO}_{\text{bias},i} = \mathbf{dAO}_{\text{lin},i}$ $[B,S,D]$
$\Sigma_{B,S}$
$\mathbf{d}\widetilde{\mathbf{b}}_O$ $[D]$
$\mathbf{AO}_{\text{cat},i}$ $[B,S,N_{HN}D_h]$
T
$[B,N_{HN}D_h,S]$
$\mathbf{d}\widetilde{\mathbf{W}}_O^{(i)}$ $[N_{HN}D_h,D]$
$\widetilde{\mathbf{W}}_O^{(i)T}$ $[D,N_{HN}D_h]$
$\widetilde{\mathbf{W}}_O^{(i)}$ $[N_{HN}D_h,D]$
$\mathbf{dAO}_{\text{cat},i}$ $[B,S,N_{HN}D_h]$
dC
$[B,S,N_{HN},D_h]$
R
$\mathbf{dAO}_{\text{heads},i}$ $[B,N_{HN},S,D_h]$
$\mathbf{V}_i$ $[B,N_{HN},S,D_h]$
$[B,N_{HN},D_h,S]$
$\mathbf{dAS}_i$ $[B,N_{HN},S,S]$
$\mathbf{AS}_i$ $[B,N_{HN},S,S]$
dS
$[B,N_{HN},S,S]$
dSM
$\mathbf{dA}_i$ $[B,N_{HN},S,S]$
$\mathbf{K}_i$ $[B,N_{HN},S,D_h]$
$\mathbf{dQ}_i$ $[B,N_{HN},S,D_h]$
$\widetilde{\mathbf{W}}_Q^{(i)}$ $[D,N_{HN}D_h]$
$[D,N_{HN}D_h]$
$[B,S,N_{HN}D_h]$
$[B,D,S]$
$\mathbf{X}_{\text{norm}}$ $[B,S,D]$
$\mathbf{d}\widetilde{\mathbf{W}}_Q^{(i)}$ $[D,N_{HN}D_h]$
$\mathbf{Q}_i$ $[B,N_{HN},S,D_h]$
$[B,N_{HN},D_h,S]$
$\mathbf{dK}_i^T$ $[B,N_{HN},D_h,S]$
$[B,S,N_{HN},D_h]$
$\mathbf{dK}_i$ $[B,S,N_{HN}D_h]$
$\widetilde{\mathbf{W}}_K^{(i)}$ $[D,N_{HN}D_h]$
$\mathbf{d}\widetilde{\mathbf{W}}_K^{(i)}$ $[D,N_{HN}D_h]$
$\mathbf{dX}_{\text{norm},\mathbf{Q},i}$ $[B,S,D]$
$\mathbf{dX}_{\text{norm},\mathbf{K},i}$ $[B,S,D]$
$\mathbf{dX}_{\text{norm},\mathbf{V},i}$ $[B,S,D]$
$\mathbf{AS}_i$ $[B,N_{HN},S,S]$
$[B,N_{HN},S,S]$
$\mathbf{dV}_i$ $[B,N_{HN},S,D_h]$
$[B,S,N_{HN}D_h]$
$\widetilde{\mathbf{W}}_V^{(i)}$ $[D,N_{HN}D_h]$
$\mathbf{d}\widetilde{\mathbf{W}}_V^{(i)}$ $[D,N_{HN}D_h]$
$\mathbf{dX}_{\text{norm},i}$ $[B,S,D]$
dLN
$\mathbf{X}$ $[B,S,D]$
$\mathbf{dX}_i$ $[B,S,D]$
AR
Node j
Node k
$\mathbf{dX}$ $[B,S,D]$

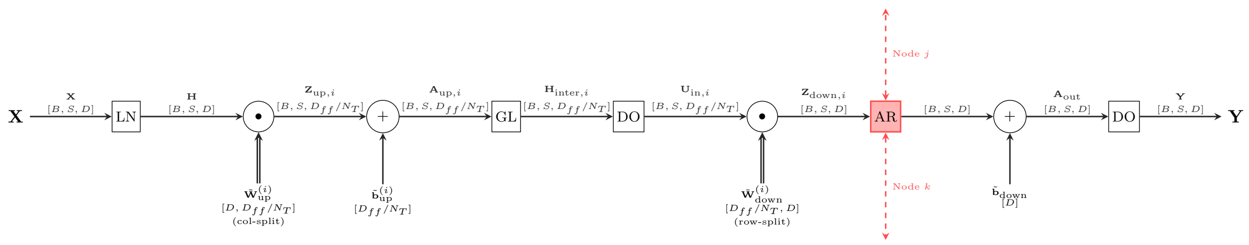
All Reduce Comm.:

- **Naive:** $2(N_T-1)\times[B,S,D]$
- **Ring:** $2\frac{N_T-1}{N_T}\times[B,S,D]$

6.3 MLP with Tensor Parallelism

6.3.1 Forward Pass

MLP Forward Pass (Node i)

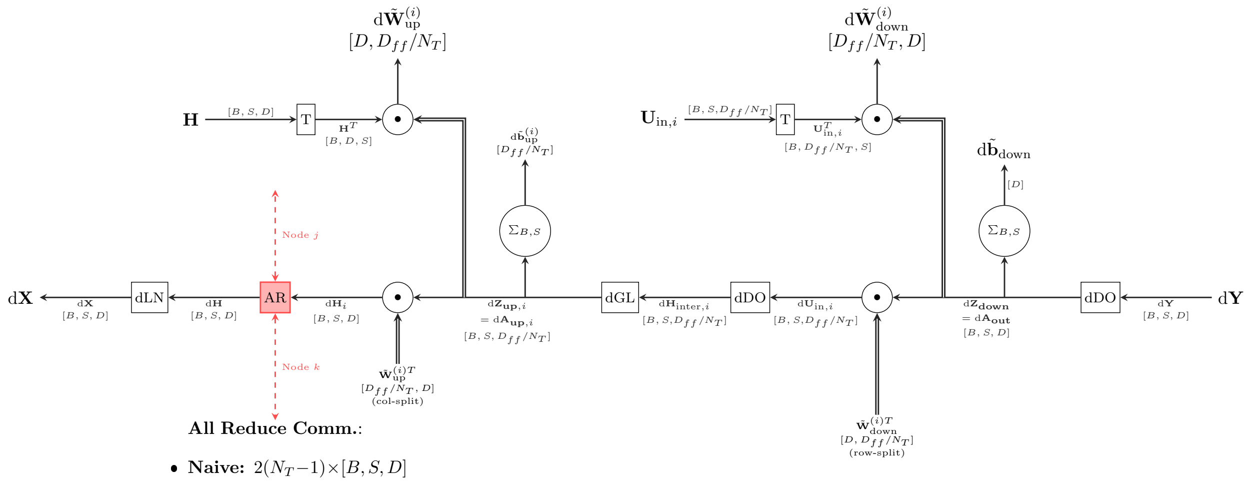

$\mathbf{X}$
$\mathbf{X}$ $[B, S, D]$
LN
$\mathbf{H}$ $[B, S, D]$
$\tilde{\mathbf{W}}_{\text{up}}^{(i)}$ $[D, D_{ff}/N_T]$ (col-split)
$\mathbf{Z}_{\text{up},i}$ $[B, S, D_{ff}/N_T]$
$\tilde{\mathbf{b}}_{\text{up}}^{(i)}$ $[D_{ff}/N_T]$
$\mathbf{A}_{\text{up},i}$ $[B, S, D_{ff}/N_T]$
GL
$\mathbf{H}_{\text{inter},i}$ $[B, S, D_{ff}/N_T]$
DO
$\mathbf{U}_{\text{in},i}$ $[B, S, D_{ff}/N_T]$
$\tilde{\mathbf{W}}_{\text{down}}^{(i)}$ $[D_{ff}/N_T, D]$ (row-split)
$\mathbf{Z}_{\text{down},i}$ $[B, S, D]$
AR
Node j
Node k
$[B, S, D]$
$\tilde{\mathbf{b}}_{\text{down}}$ $[D]$
$\mathbf{A}_{\text{out}}$ $[B, S, D]$
DO
$\mathbf{Y}$ $[B, S, D]$
$\mathbf{Y}$

All Reduce Comm.:

- **Naive:** $2(N_T-1)\times[B, S, D]$
- **Ring:** $2\frac{N_T-1}{N_T}\times[B, S, D]$

6.3.2 Backward Pass

MLP Backward Pass (Node i)


$\mathrm{d}\tilde{\mathbf{W}}_{\text{up}}^{(i)}$
$[D, D_{ff}/N_T]$
$\mathbf{H}$
$[B, S, D]$
T
$\mathbf{H}^T$
$[B, D, S]$
$\mathrm{d}\tilde{\mathbf{b}}_{\text{up}}^{(i)}$
$[D_{ff}/N_T]$
$\Sigma_{B,S}$
Node j
$\mathrm{d}\mathbf{X}$
$\mathrm{d}\mathbf{X}$
$[B, S, D]$
dLN
$\mathrm{d}\mathbf{H}$
$[B, S, D]$
AR
$\mathrm{d}\mathbf{H}_i$
$[B, S, D]$
$\mathrm{d}\mathbf{Z}_{\mathbf{up},i}$
$= \mathrm{d}\mathbf{A}_{\mathbf{up},i}$
$[B, S, D_{ff}/N_T]$
dGL
Node k
$\tilde{\mathbf{W}}_{\text{up}}^{(i)T}$
$[D_{ff}/N_T, D]$
(col-split)
$\mathrm{d}\tilde{\mathbf{W}}_{\text{down}}^{(i)}$
$[D_{ff}/N_T, D]$
$\mathbf{U}_{\text{in},i}$
$[B, S, D_{ff}/N_T]$
T
$\mathbf{U}_{\text{in},i}^T$
$[B, D_{ff}/N_T, S]$
$\mathrm{d}\tilde{\mathbf{b}}_{\text{down}}$
$[D]$
$\Sigma_{B,S}$
$\mathrm{d}\mathbf{H}_{\text{inter},i}$
$[B, S, D_{ff}/N_T]$
dDO
$\mathrm{d}\mathbf{U}_{\text{in},i}$
$[B, S, D_{ff}/N_T]$
$\mathrm{d}\mathbf{Z}_{\mathbf{down}}$
$= \mathrm{d}\mathbf{A}_{\mathbf{out}}$
$[B, S, D]$
dDO
$\mathrm{d}\mathbf{Y}$
$[B, S, D]$
$\mathrm{d}\mathbf{Y}$
$\tilde{\mathbf{W}}_{\text{down}}^{(i)T}$
$[D, D_{ff}/N_T]$
(row-split)

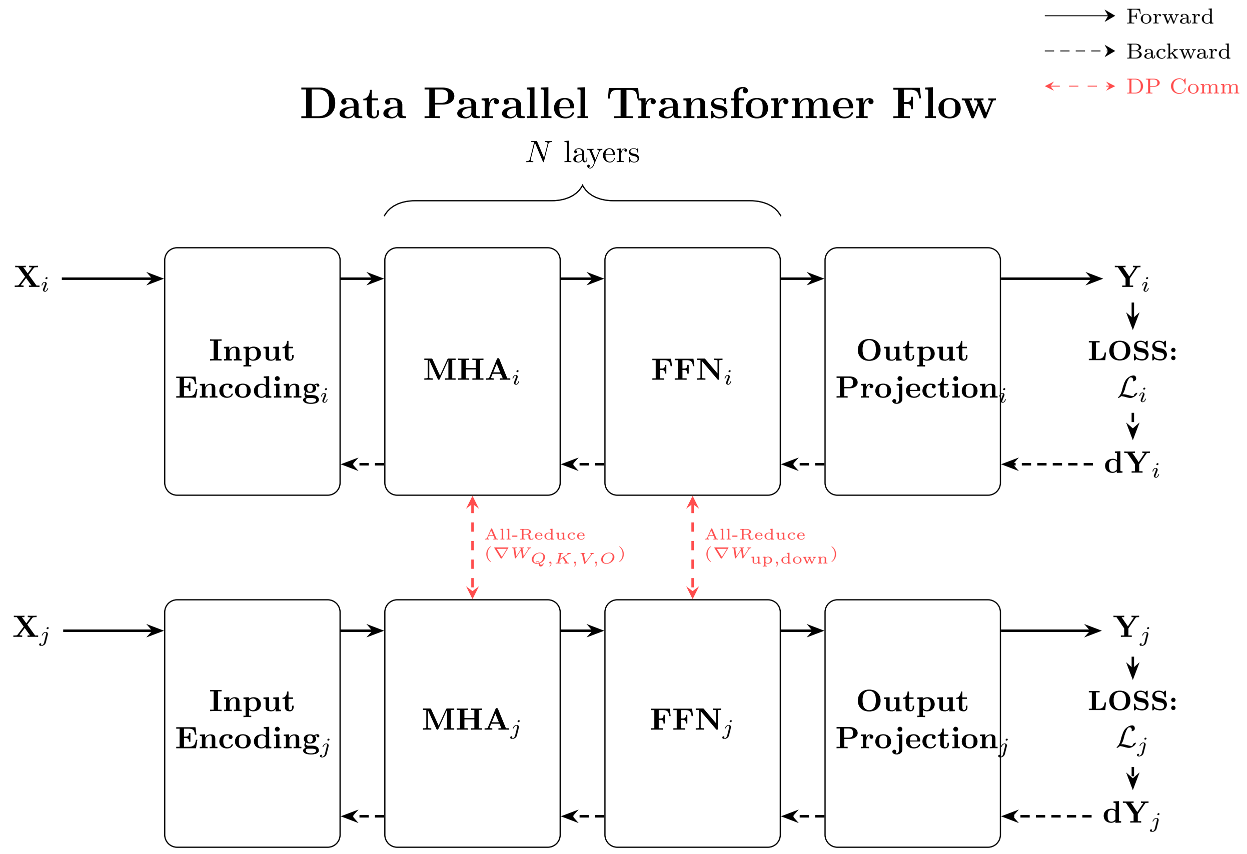
All Reduce Comm.:

- **Naive:** $2(N_T-1)\times[B,S,D]$
- **Ring:** $2\frac{N_T-1}{N_T}\times[B,S,D]$

7 Data Parallelism (DP)

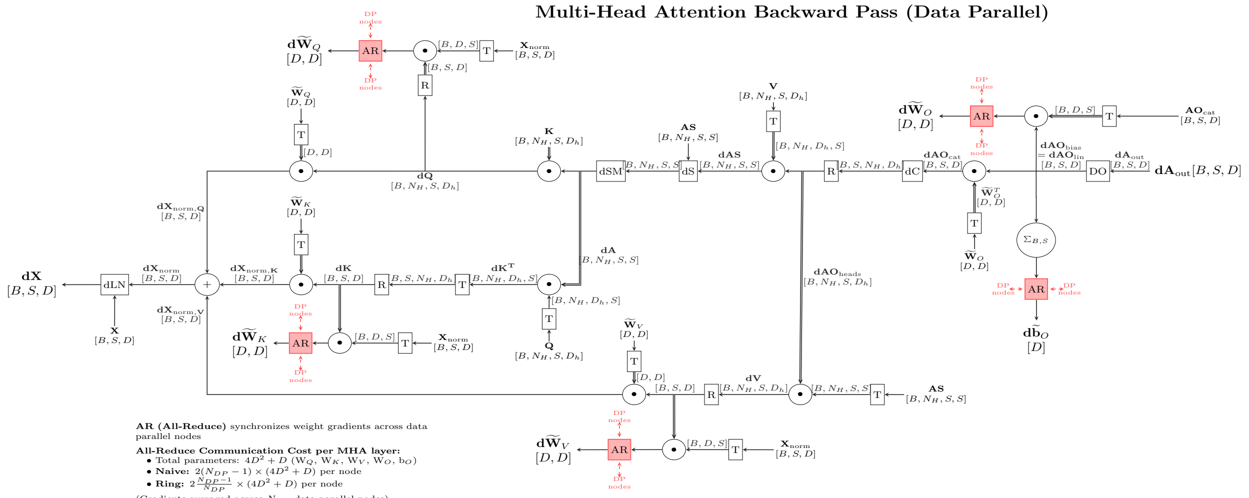
In data parallelism, each replica holds a full copy of the model, but processes a different subset of the batch. Gradients are synchronized across replicas via All-Reduce.

7.1 DP Overview and Transformer Flow



7.2 MHA Backward under DP

Multi-Head Attention Backward Pass (Data Parallel)



$d\widetilde{\mathbf{W}}_Q$ $[D, D]$
AR
DP nodes
$[B, D, S]$
T
$\mathbf{X}_{\text{norm}}$ $[B, S, D]$
R
$[B, S, D]$
$\widetilde{\mathbf{W}}_Q$ $[D, D]$
$[D, D]$
$\mathbf{dQ}$ $[B, N_H, S, D_h]$
$\mathbf{K}$ $[B, N_H, S, D_h]$
dSM
$[B, N_H, S, S]$
$\mathbf{AS}$ $[B, N_H, S, S]$
dS
$\mathbf{dAS}$ $[B, N_H, S, S]$
$\mathbf{V}$ $[B, N_H, S, D_h]$
$[B, N_H, D_h, S]$
$[B, S, N_H, D_h]$
dC
$\mathbf{dAO}_{\text{cat}}$ $[B, S, D]$
$\widetilde{\mathbf{W}}_O^T$ $[D, D]$
$\widetilde{\mathbf{W}}_O$ $[D, D]$
$d\widetilde{\mathbf{W}}_O$ $[D, D]$
$\mathbf{AO}_{\text{cat}}$ $[B, S, D]$
$\mathbf{dAO}_{\text{bias}} = \mathbf{dAO}_{\text{lin}}$ $[B, S, D]$
$\mathbf{dA}_{\text{out}}$ $[B, S, D]$
DO
$\mathbf{dA}_{\text{out}}[B, S, D]$
$\Sigma_{B,S}$
$d\widetilde{\mathbf{b}}_O$ $[D]$
$\mathbf{dAO}_{\text{heads}}$ $[B, N_H, S, D_h]$
$\mathbf{dA}$ $[B, N_H, S, S]$
$\mathbf{dX}_{\text{norm},\mathbf{Q}}$ $[B, S, D]$
$\widetilde{\mathbf{W}}_K$ $[D, D]$
$\mathbf{dX}$ $[B, S, D]$
dLN
$\mathbf{dX}_{\text{norm}}$ $[B, S, D]$
$+$
$\mathbf{dX}_{\text{norm},\mathbf{K}}$ $[B, S, D]$
$\mathbf{dK}$ $[B, S, D]$
$[B, S, N_H, D_h]$
$\mathbf{dK}^{\mathbf{T}}$ $[B, N_H, D_h, S]$
$[B, N_H, D_h, S]$
$\mathbf{Q}$ $[B, N_H, S, D_h]$
$\mathbf{X}$ $[B, S, D]$
$\mathbf{dX}_{\text{norm},\mathbf{V}}$ $[B, S, D]$
$d\widetilde{\mathbf{W}}_K$ $[D, D]$
$\widetilde{\mathbf{W}}_V$ $[D, D]$
$\mathbf{dV}$ $[B, N_H, S, D_h]$
$[B, N_H, S, S]$
$\mathbf{AS}$ $[B, N_H, S, S]$
$d\widetilde{\mathbf{W}}_V$ $[D, D]$

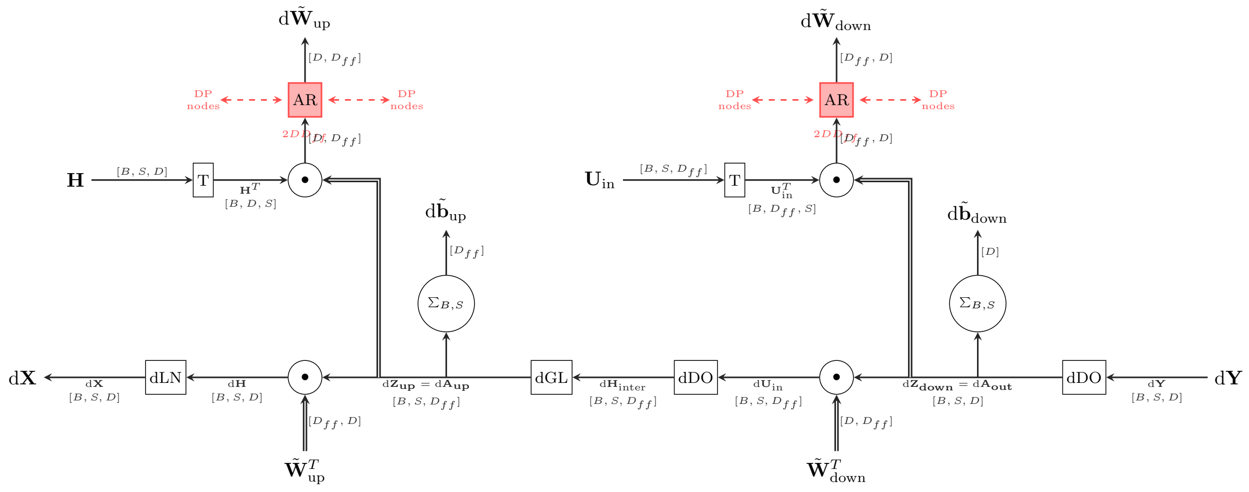
AR (All-Reduce) synchronizes weight gradients across data parallel nodes

All-Reduce Communication Cost per MHA layer:
- Total parameters: $4D^2 + D$ (W_Q, W_K, W_V, W_O, b_O)
- **Naive:** $2(N_{DP} - 1) \times (4D^2 + D)$ per node
- **Ring:** $2\frac{N_{DP}-1}{N_{DP}} \times (4D^2 + D)$ per node

(Gradients averaged across N_{DP} data parallel nodes)

7.3 MLP Backward under DP

MLP Backward Pass (Data Parallel)



$\mathrm{d}\tilde{\mathbf{W}}_{\text{up}}$
$[D, D_{ff}]$
DP nodes
AR
DP nodes
$2DD_{ff}$
$[D, D_{ff}]$
$\mathbf{H}$
$[B, S, D]$
T
$\mathbf{H}^T$
$[B, D, S]$
$\mathrm{d}\tilde{\mathbf{b}}_{\text{up}}$
$[D_{ff}]$
$\Sigma_{B,S}$
$\mathrm{d}\mathbf{X}$
dLN
$\mathrm{d}\mathbf{X}$ $[B, S, D]$
$\mathrm{d}\mathbf{H}$ $[B, S, D]$
$\mathrm{d}\mathbf{Z_{up}} = \mathrm{d}\mathbf{A_{up}}$ $[B, S, D_{ff}]$
$[D_{ff}, D]$
$\tilde{\mathbf{W}}^T_{\text{up}}$
dGL
$\mathrm{d}\mathbf{H}_{\text{inter}}$ $[B, S, D_{ff}]$
dDO
$\mathrm{d}\tilde{\mathbf{W}}_{\text{down}}$
$[D_{ff}, D]$
DP nodes
AR
DP nodes
$2DD_{ff}$
$[D_{ff}, D]$
$\mathbf{U}_{\text{in}}$
$[B, S, D_{ff}]$
T
$\mathbf{U}^T_{\text{in}}$
$[B, D_{ff}, S]$
$\mathrm{d}\tilde{\mathbf{b}}_{\text{down}}$
$[D]$
$\Sigma_{B,S}$
$\mathrm{d}\mathbf{U}_{\text{in}}$ $[B, S, D_{ff}]$
$\mathrm{d}\mathbf{Z_{down}} = \mathrm{d}\mathbf{A_{out}}$ $[B, S, D]$
dDO
$\mathrm{d}\mathbf{Y}$ $[B, S, D]$
$\mathrm{d}\mathbf{Y}$
$[D, D_{ff}]$
$\tilde{\mathbf{W}}^T_{\text{down}}$

AR (All-Reduce) synchronizes weight gradients across data parallel nodes

MLP All-Reduce Cost: $\sim 2DD_{ff}$ parameters (W_{up}, W_{down})

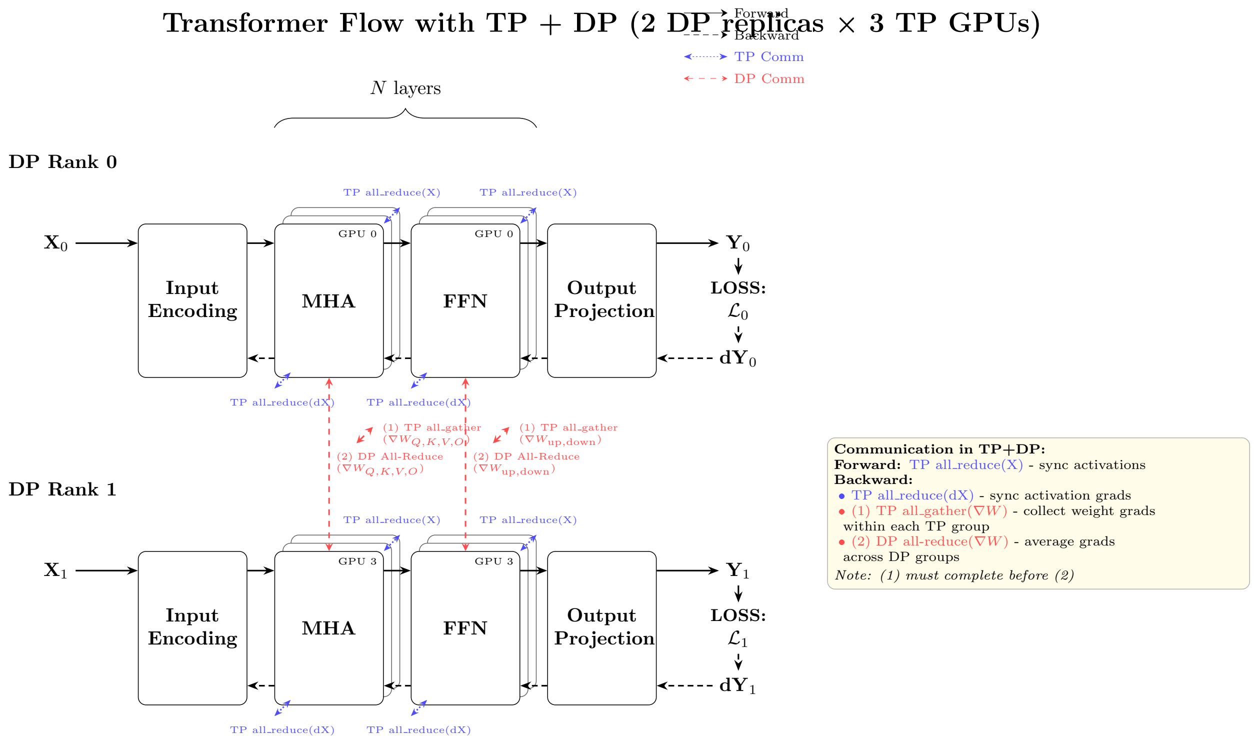
- **Naive:** $2(N_{DP} - 1) \times 2DD_{ff}$ per node
- **Ring:** $2\frac{N_{DP}-1}{N_{DP}} \times 2DD_{ff}$ per node

(Gradients averaged across N_{DP} data parallel nodes)

8 Hybrid Data + Tensor Parallelism (DP + TP)

We combine tensor parallelism within each node (or group of devices) with data parallelism across groups. This section explains how the two forms of parallelism interact in both forward and backward passes.

8.1 DP+TP Overview and Communication Patterns



Communication in TP+DP:
Forward: TP all_reduce(X) - sync activations
Backward:
- TP all_reduce(dX) - sync activation grads
- (1) TP all_gather(∇W) - collect weight grads within each TP group
- (2) DP all-reduce(∇W) - average grads across DP groups

Note: (1) must complete before (2)

9 Summary and Practical Takeaways

We summarize the main ideas:

- How a Transformer layer operates as a composition of embedding, MHA, MLP, and output projection blocks.
- How tensor shapes evolve through forward and backward passes.
- How single-node execution extends to tensor parallelism, data parallelism, and their combination.

We also highlight how these diagrams can be used as a reference when designing or debugging large-scale Transformer training and inference systems.