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

Before we dive into detailed forward and backward diagrams for each Transformer block, we review the basic concepts of loss functions, gradients, and backpropagation. We focus on an abstract operator view, because the diagrams in later sections emphasize tensor flow along edges rather than explicit Jacobian matrices.

3.1 Loss Functions and Gradients

Training a neural network is formulated as the minimization of a scalar *loss function* $L(\theta)$ over parameters θ. For a batch of input–target pairs $(\mathbf{X}, \mathbf{Y}_{\text{targets}})$, we compute predictions $\mathbf{Y} = f_\theta(\mathbf{X})$ and a scalar loss

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

The gradient of L with respect to a parameter vector θ is

$$\nabla_\theta L = \begin{bmatrix} \dfrac{\partial L}{\partial \theta_1} \\ \dfrac{\partial L}{\partial \theta_2} \\ \vdots \end{bmatrix},$$

and we use the notation $\mathrm{d}\theta$ and $\mathrm{d}\mathbf{X}$ for such gradients. For example

$$\mathrm{d}\mathbf{W} = \frac{\partial L}{\partial \mathbf{W}}, \qquad \mathrm{d}\mathbf{X} = \frac{\partial L}{\partial \mathbf{X}}.$$

3.2 Chain Rule 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 $J_f(\mathbf{x})$ denote the Jacobian of f at $\mathbf{x}$. If L depends on $\mathbf{y}$, the chain rule gives

$$\mathrm{d}\mathbf{x} = \frac{\partial L}{\partial \mathbf{x}} = J_f(\mathbf{x})^\mathsf{T} \frac{\partial L}{\partial \mathbf{y}} = J_f(\mathbf{x})^\mathsf{T}\, \mathrm{d}\mathbf{y}.$$

In the diagrams we do not materialize the Jacobian. Instead we introduce an *abstract backward operator* df and write

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

with the understanding that

$$df(\mathbf{x}, \mathrm{d}\mathbf{y}) \;\equiv\; J_f(\mathbf{x})^\mathsf{T}\, \mathrm{d}\mathbf{y}.$$

Thus, conceptually, each backward node in the graph implements the mapping

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

Multiple inputs. More generally, many nodes have several inputs:

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

Given the upstream gradient $\mathrm{d}\mathbf{y}$, the chain rule yields gradients with respect to all inputs:

$$\mathrm{d}\mathbf{x}_i = J_{f,\mathbf{x}_i}(\mathbf{x}_1, \ldots, \mathbf{x}_k)^\mathsf{T}\, \mathrm{d}\mathbf{y}, \qquad i = 1, \ldots, k,$$

where $J_{f,\mathbf{x}_i}$ is the Jacobian of f with respect to $\mathbf{x}_i$.

We bundle this into a single backward operator

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

In the diagrams, this is precisely what nodes such as dS, dSM, dDO, and dLN represent: given the forward inputs (or cached forward state) and the upstream gradient, they produce gradients for *all* inputs of the corresponding forward node.

3.3 Computation Graph View

A full Transformer layer can be viewed as a composition

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

where $\mathbf{X}_0$ is the input and $\mathbf{X}_L$ is the output before the loss. Backpropagation proceeds by visiting nodes in reverse topological order and applying the corresponding backward operator:

$$\mathrm{d}\mathbf{X}_\ell = df_\ell(\mathbf{X}_\ell, \mathrm{d}\mathbf{X}_{\ell+1}), \qquad \ell = L-1, \ldots, 0.$$

Matrix multiplication nodes, softmax nodes, dropout nodes, layer normalization nodes, and communication nodes (All-Reduce, All-Gather) are all special cases of this pattern, each with its own concrete backward operator df.

3.4 Parameter Gradients and Updates

Parameters such as weight matrices and bias vectors are also inputs to some node in the graph. For a matmul

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

we can view the forward as a function of two inputs, $f(\mathbf{X}, \mathbf{W}) = \mathbf{Y}$. The corresponding backward operator produces both activation and parameter gradients:

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

These parameter gradients are then used by an optimizer (SGD, Adam, and so on) to update the parameters, for example with stochastic gradient descent

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

In the figures, we explicitly draw the edges corresponding to $\mathrm{d}\mathbf{W}$ and $\mathrm{d}\mathbf{b}$ for key operators (MHA, MLP, output projection). The detailed formulas for each node (e.g. softmax, scale/mask, layer normalization) are encapsulated by their corresponding backward operators and described in the next section.

4 Graphical Notation and Figure Conventions

All figures in this document are drawn as computational graphs. Before we dive into the layer-wise diagrams, we clarify what nodes and edges represent, and we summarize the forward and backward behavior of each operator.

4.1 Graph View: Nodes, Edges, and Shapes

Each diagram can be interpreted as a directed graph:

- **Nodes** denote operations or transformations applied to tensors. Some nodes perform arithmetic (e.g., matrix multiplication or addition), others only change layout or perform communication.
- **Edges** denote tensors flowing between nodes. An edge is usually labeled with:
  - a symbolic name such as $\mathbf{X}$, $\mathbf{H}$, $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, $\mathbf{AS}$, $\mathrm{d}\mathbf{X}$, and
  - an optional **shape annotation** in bracket form, for example $[B, S, D]$ or $[B, N_H, S, D_h]$.

Whenever an edge is annotated with $[B, S, D]$, this is shorthand for a tensor of shape $\mathbb{R}^{B\times S\times D}$. Likewise, $[B, N_H, S, D_h]$ corresponds to $\mathbb{R}^{B\times N_H\times S\times D_h}$, and so on.

Forward and backward passes are usually drawn as *separate* figures (for example, one for MHA forward and another for MHA backward). In the high-level overall-flow figures, forward and backward paths share the same diagram; in that case we distinguish them by arrow style, as described in Section 4.4.

4.2 Tensor Shapes and Indices

We use the following global symbols and dimensions:

- B: global batch size.
- S: sequence length (number of tokens).
- D: model (hidden) dimension.
- D_{ff}: intermediate MLP dimension.
- N_H: number of attention heads.
- D_h: per-head dimension, typically $D_h = D/N_H$.
- N_T: tensor-parallel degree (number of TP shards).
- N_D: data-parallel degree (number of DP replicas).

Typical global shapes:

- $\mathbf{X} \in \mathbb{R}^{B\times S\times D}$: input or hidden states (annotated as $[B, S, D]$).
- $\mathbf{H} \in \mathbb{R}^{B\times S\times D}$: normalized or intermediate hidden states.
- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{B\times N_H\times S\times D_h}$: query, key, value tensors after projection (annotated as $[B, N_H, S, D_h]$).
- $\mathbf{AS} \in \mathbb{R}^{B\times N_H\times S\times S}$: attention scores or probabilities (annotated as $[B, N_H, S, S]$).
- $\mathbf{Y} \in \mathbb{R}^{B\times S\times D}$: output of a Transformer block or layer.

In parallel settings we also use *local* views, where the batch or hidden dimension is partitioned:

- Under data parallelism (DP), a single replica processes roughly $[B/N_D, S, D]$.
- Under tensor parallelism (TP), a single shard sees hidden slices such as $[B, S, D/N_T]$ or $[B, N_H, S, D_h/N_T]$.

Gradients are denoted with a leading d, e.g. $\mathrm{d}\mathbf{X}$, $\mathrm{d}\mathbf{W}$, $\mathbf{dA}_{\text{out}}$. The corresponding edges in the diagrams use the same symbols together with bracketed shape annotations.

4.3 Operator Dictionary: Forward and Backward

In the detailed diagrams we draw separate nodes for the forward and backward versions of an operator. If a forward node computes

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

then the corresponding backward node, labeled with a leading "d" (e.g. dS, dSM, dDO, dLN), represents the abstract backward operator

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

as introduced in Section 3. In other words, each dNode takes the upstream gradient $\mathrm{d}\mathbf{y}$ and the necessary cached forward inputs, and produces gradients for all inputs of the corresponding forward node.

4.3.1 Matrix Multiplication (Matmul)

Symbol

- **Node icon:** ⊙ (small circle with a dot inside)
- **Incoming edges:**
  - Single-line arrow: first operand (usually activations).
  - Double-line arrow: second operand (weights or transpose).

Forward For simplicity, consider

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

where $\mathbf{X} \in \mathbb{R}^{B \times d_{\text{in}}}$, $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, and $\mathbf{Y} \in \mathbb{R}^{B \times d_{\text{out}}}$. In practice we use batched versions in the diagrams (e.g. shapes $[B, S, D]$, $[D, D]$, or $[B, N_H, S, D_h]$), but the rule is the same.

Backward Given the upstream gradient $\mathrm{d}\mathbf{Y} = \partial L / \partial \mathbf{Y}$, the local gradients are

$$\mathrm{d}\mathbf{X} \;=\; \mathrm{d}\mathbf{Y}\,\mathbf{W}^\top, \qquad \mathrm{d}\mathbf{W} \;=\; \mathbf{X}^\top\,\mathrm{d}\mathbf{Y}.$$

Thus each matmul node in the backward diagrams produces both an activation gradient (e.g. $\mathrm{d}\mathbf{X}$) and a parameter gradient (e.g. $\mathrm{d}\widetilde{\mathbf{W}}_Q$).

4.3.2 Bitwise / Elementwise Addition

Symbol

- **Node icon:** $\oplus$ (small circle with a plus sign)

Forward For tensors of the same shape,

$$\mathbf{Y} = \mathbf{A} + \mathbf{B},$$

we add them elementwise: $\text{shape}(\mathbf{Y}) = \text{shape}(\mathbf{A}) = \text{shape}(\mathbf{B})$.

Backward Given $\mathrm{d}\mathbf{Y}$,

$$\mathrm{d}\mathbf{A} = \mathrm{d}\mathbf{Y}, \qquad \mathrm{d}\mathbf{B} = \mathrm{d}\mathbf{Y}.$$

When one of the inputs is the broadcast of a bias (e.g. $\mathrm{BC}_{B,S}(b_0)$), the gradient with respect to the bias is obtained by summing $\mathrm{d}\mathbf{Y}$ over the broadcasted dimensions (see BC below).

4.3.3 Nonlinear and Pointwise Operations (GLU / GELU / ReLU / DO)

Symbol

- **Node icon (generic op):** $\boxed{\text{OP}}$ (rectangular node with a label)

Forward A scalar nonlinearity $g(\cdot)$ is applied elementwise:

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

for example GELU, ReLU, or GLU-style gates in the MLP block.

For dropout we use a node labeled **DO** and write

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

where $\mathbf{m} \in \{0, 1\}^{\text{shape}(\mathbf{X})}$ is a binary mask cached from the forward pass.

Backward (dGL, dGELU, dReLU, dDO) In the backward diagrams we use separate nodes **dGL**, **dGELU**, **dReLU**, and **dDO** for the corresponding gradient computations. Each such node implements the local mapping

$$(\mathrm{d}\mathbf{Y}, \mathbf{X}, \text{cache}) \;\longmapsto\; \mathrm{d}\mathbf{X}.$$

Given $\mathrm{d}\mathbf{Y}$:

- For a generic nonlinearity g (nodes dGL, dGELU, dReLU):

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

  where $\odot$ denotes elementwise multiplication and $\mathbf{X}$ (and possibly $\mathbf{Y}$) is taken from the forward cache.

- For dropout (node **dDO**), using the cached mask $\mathbf{m}$:

$$\mathrm{d}\mathbf{X} \;=\; \mathbf{m} \odot \mathrm{d}\mathbf{Y}.$$

Thus dDO is the operator that takes the upstream gradient on the dropped-out activations and re-applies the forward mask to obtain the gradient with respect to the pre-dropout tensor.

4.3.4 Scale/Mask Node (SM)

Symbol and Labels In the MHA diagrams we use a node labeled **SM** for *scale/mask* applied to attention scores, and a corresponding backward node labeled **dSM**.

Forward (SM) Given raw attention scores $\mathbf{A}$ (typically from $\mathbf{QK}^\top$), the SM node computes

$$\mathbf{Z} \;=\; \mathrm{SM}(\mathbf{A}) \;=\; \alpha\,\mathbf{A} + \mathbf{M},$$

where $\alpha = 1/\sqrt{D_h}$ is a scalar scaling factor and $\mathbf{M}$ encodes the attention mask (e.g. large negative values on disallowed positions). The shape of $\mathbf{Z}$ is the same as $\mathbf{A}$, typically $[B, N_H, S, S]$. The forward pass caches α and, implicitly, the mask pattern.

Backward (dSM) The node **dSM** implements the local mapping

$$(\mathrm{d}\mathbf{Z}, \alpha, \mathbf{M}) \;\longmapsto\; \mathrm{d}\mathbf{A}.$$

Since $\mathbf{M}$ is treated as a fixed (non-trainable) mask, the gradient with respect to $\mathbf{M}$ is zero, and we obtain

$$\mathrm{d}\mathbf{A} \;=\; \alpha\,\mathrm{d}\mathbf{Z}, \qquad \mathrm{d}\mathbf{M} = 0.$$

In other words, dSM simply rescales the upstream gradient by the same factor used in the forward pass and does not propagate gradients into the mask.

4.3.5 Softmax Node (S)

Forward (S) For a fixed batch index, head index, and query position, let $\mathbf{z} \in \mathbb{R}^S$ be the score vector over keys. The node **S** computes

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

This is applied across all batches and heads, so shapes such as $[B, N_H, S, S]$ are preserved. The forward pass typically caches $\mathbf{p}$ (or $\mathbf{z}$).

Backward (dS) The backward node **dS** is the concrete instance of the abstract operator df for $f =$ softmax. It implements

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

which, in terms of the Jacobian of softmax, is

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

In practice this is computed using the standard softmax backward formula; the diagrams treat dS as a single node whose input edges carry $\mathbf{p}$ (or $\mathbf{z}$) and $\mathrm{d}\mathbf{p}$, and whose output edge carries $\mathrm{d}\mathbf{z}$.

4.3.6 Layer Normalization (LN, dLN)

Symbols and Labels Layer normalization forward nodes are labeled **LN**, and their backward counterparts are labeled **dLN** in the diagrams.

Forward (LN) Layer normalization maps $\mathbf{X} \in \mathbb{R}^{B\times S\times D}$ to $\mathbf{H} \in \mathbb{R}^{B\times S\times D}$ by normalizing each position (across the D dimension) and applying learned scale and shift parameters $\gamma, \beta \in \mathbb{R}^D$. The forward pass also computes and caches per-position mean and variance.

Backward (dLN) The node **dLN** implements the local mapping

$$(\mathrm{d}\mathbf{H}, \mathbf{X}, \gamma, \mathrm{mean}, \mathrm{var}) \longmapsto (\mathrm{d}\mathbf{X}, \mathrm{d}\gamma, \mathrm{d}\beta).$$

Given the upstream gradient $\mathrm{d}\mathbf{H}$ and the cached statistics from LN, standard layer-normalization backward formulas are applied to produce:

- $\mathrm{d}\mathbf{X}$: gradient with respect to the normalized inputs,
- $\mathrm{d}\gamma$: gradient of the scale parameter,
- $\mathrm{d}\beta$: gradient of the shift parameter.

We do not reproduce the full algebraic expressions here, but the dLN node should be understood as the unique operator that recovers these three gradients from $\mathrm{d}\mathbf{H}$ and the forward cache.

4.3.7 Reshape and Transpose (R, T)

Symbols

- **R**: reshape or dimension reordering.
- **T**: transpose of certain axes.

Forward Typical examples in the MHA diagrams include:

- reshaping between $[B, S, N_H, D_h]$ and $[B, N_H, S, D_h]$,
- transposing between $[B, N_H, S, D_h]$ and $[B, N_H, D_h, S]$,
- transposing $[B, N_H, S, S]$ along its last two axes.

These nodes do not change the underlying data, only how dimensions are arranged.

Backward The backward rule simply inverts the layout change: the gradient with respect to the input is obtained by reshaping or transposing $\mathrm{d}\mathbf{Y}$ with the inverse mapping used in the forward pass.

4.3.8 Broadcast ($\mathbf{BC}_{B,S}(b_0)$)

Notation and Role In the diagrams we represent bias broadcast using expressions such as $\mathrm{BC}_{B,S}(b_0)$.

Forward Let $b_0 \in \mathbb{R}^D$ be a bias vector. The notation $\mathrm{BC}_{B,S}(b_0)$ means that b_0 is logically broadcast along the batch and sequence dimensions to shape $[B, S, D]$, so that we can form

$$\mathbf{Y} = \mathbf{X} + \mathrm{BC}_{B,S}(b_0), \qquad \mathbf{X} \in [B, S, D].$$

Conceptually this is a view/layout operation; the actual additions are performed by the subsequent bitwise adder node.

Backward Given $\mathrm{d}\mathbf{Y}$:

$$\mathrm{d}\mathbf{X} = \mathrm{d}\mathbf{Y}, \qquad \mathrm{d}b_0 = \sum_{b=1}^{B}\sum_{s=1}^{S} \mathrm{d}\mathbf{Y}_{b,s,:}.$$

Thus BC itself has no FLOPs; it only defines how gradients are accumulated over broadcasted dimensions.

4.3.9 Communication Nodes (AR, AG)

Symbols

- **AR**: $\boxed{\text{AR}}$ (All-Reduce)
- **AG**: $\boxed{\text{AG}}$ (All-Gather)

Forward

- **All-Reduce (AR)**: each participant starts with a local tensor $\mathbf{X}_{\text{local}}$ (e.g. a gradient shard). AR computes a reduction (usually sum or mean) over all participants and returns the reduced tensor to each of them.
- **All-Gather (AG)**: each participant holds a slice of a larger tensor (e.g. a hidden-dimension shard). AG concatenates all slices along the partitioned dimension and makes the full tensor available on every participant.

Edge annotations such as $[B/N_D, S, D]$ or $[B, S, D/N_T]$ denote the payload shape per node or shard.

Backward The gradient behavior mirrors the forward operation:

- For All-Gather, the backward pass scatters the gradient $\mathrm{d}\mathbf{Y}$ back to the corresponding local slices, producing $\mathrm{d}\mathbf{X}_{\text{local}}$ on each shard.
- For All-Reduce with a sum, each participant receives the same $\mathrm{d}\mathbf{Y}$; the local gradient equals this value. If the forward used a mean, an additional scaling factor (e.g. dividing by the number of participants) is applied.

4.4 Arrow Styles

We use the following arrow styles consistently across the figures:

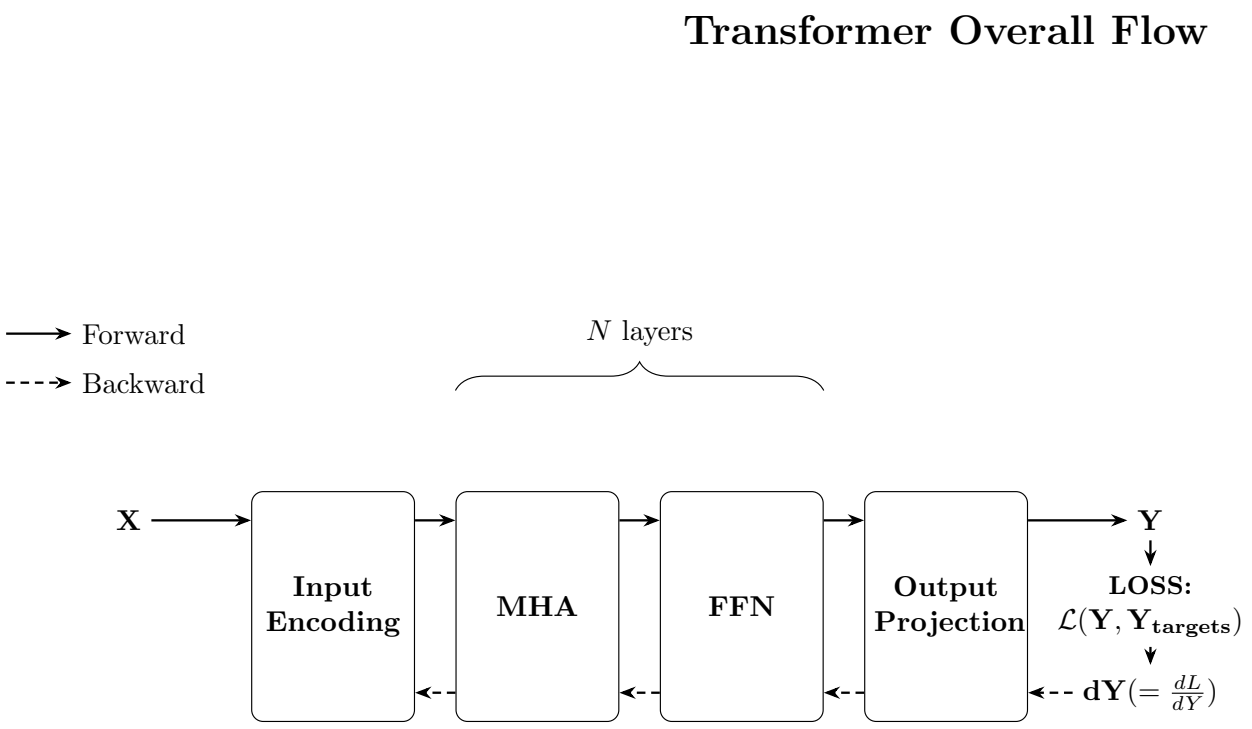
- **Single-line solid arrows**: main data flow into a node, typically the first operand of a matrix multiplication or the activation being transformed, e.g. $\longrightarrow$.
- **Double-line solid arrows**: second operand of a matrix multiplication, usually a weight matrix or a transposed tensor. As a small icon: $\Longrightarrow$. In the detailed MHA and MLP diagrams, every matmul node therefore has one single-line and one double-line incoming edge.
- **Dashed arrows**: used *only* in the high-level overall flow figures to indicate backward flow (gradients) along the same path as the forward activations, e.g. $\dashrightarrow$. In the detailed per-block diagrams, forward and backward passes are drawn as separate figures, so dashed arrows are not reused there.

With this operator dictionary in mind, the subsequent single-node, TP, DP, and DP+TP diagrams can be read as explicit execution blueprints: each node encodes both its forward computation and its local backpropagation rule, and each edge carries a tensor with a clearly annotated shape.

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.

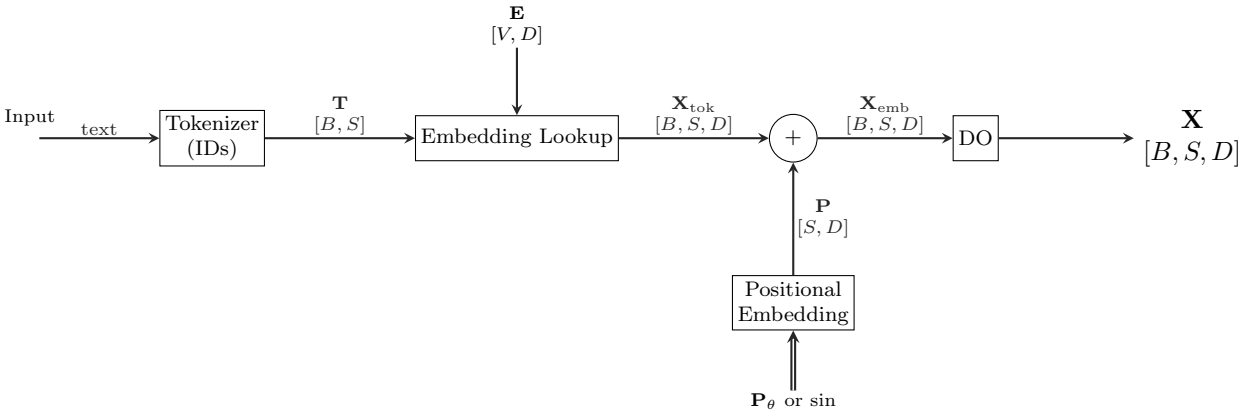
5.1 Overall Transformer Layer Flow



5.2 Input Embedding Layer

5.2.1 Forward Pass

Input → Embedding → LN (Input to MHA)

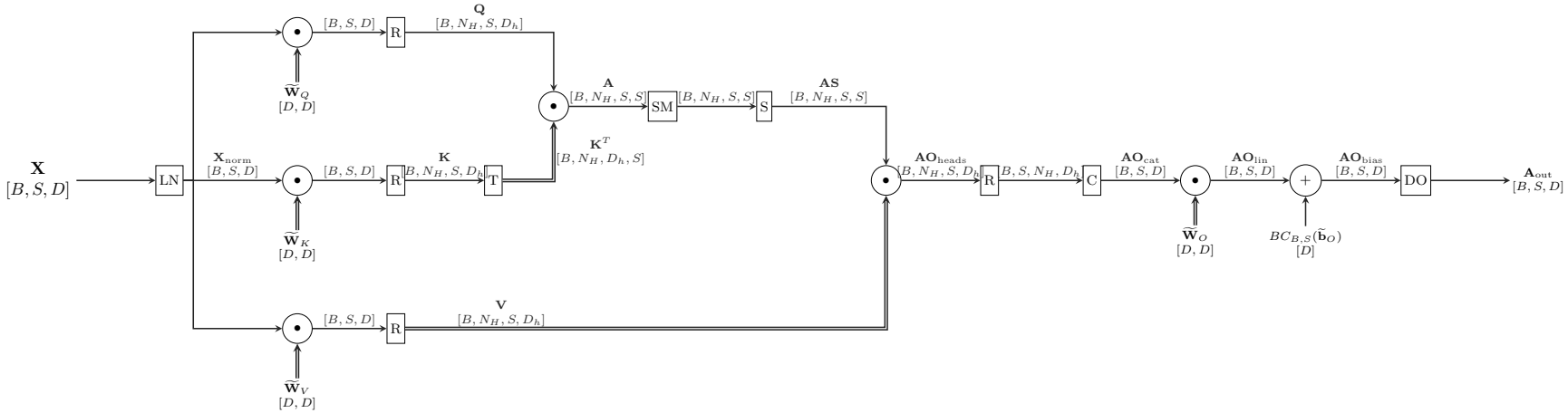


5.3 Multi-Head Attention (MHA)

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

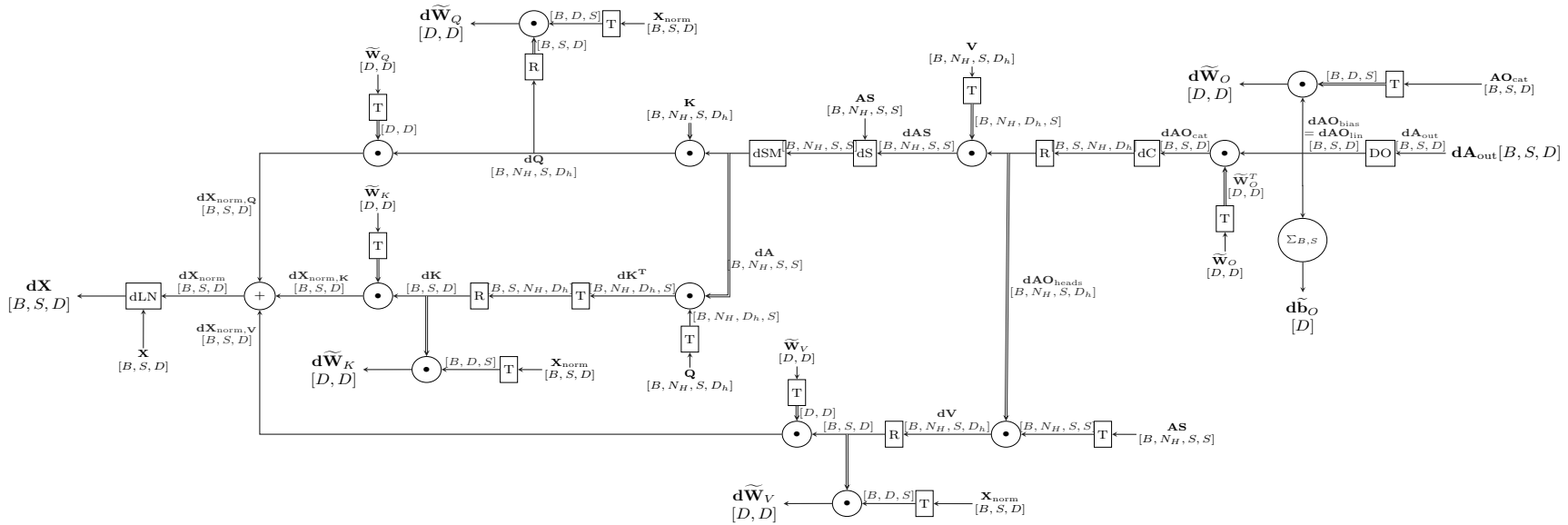
5.3.1 Forward Pass

Multi-Head Attention Forward Pass



5.3.2 Backward Pass

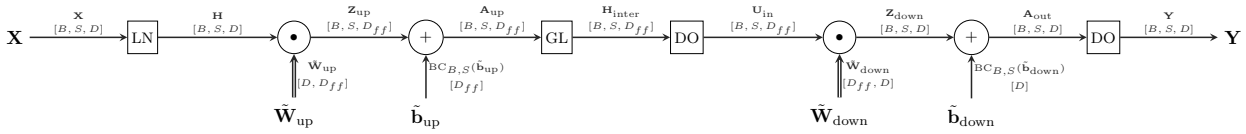
Multi-Head Attention Backward Pass



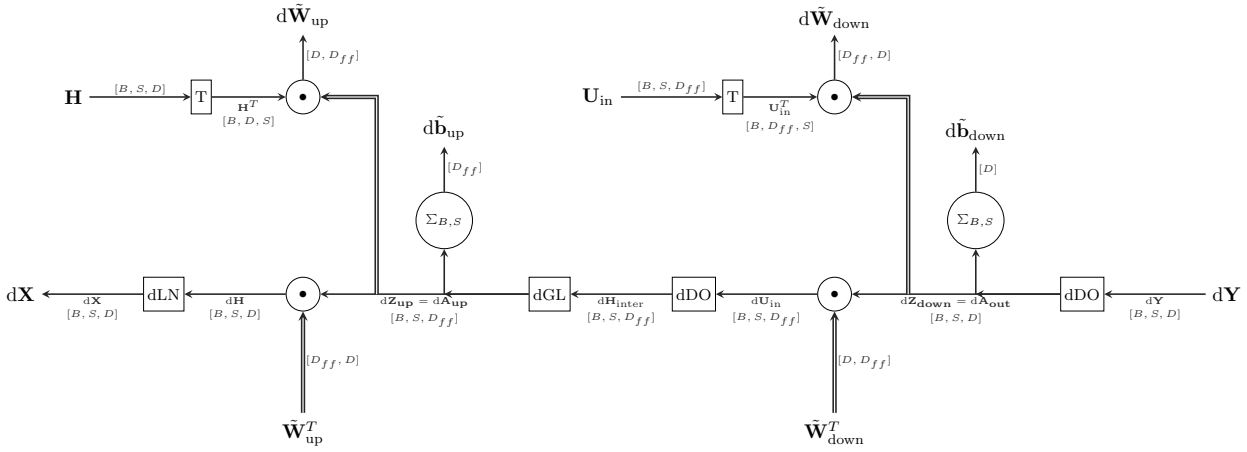
5.4 Feed-Forward Network (MLP / FFN)

5.4.1 Forward Pass

MLP Forward Pass

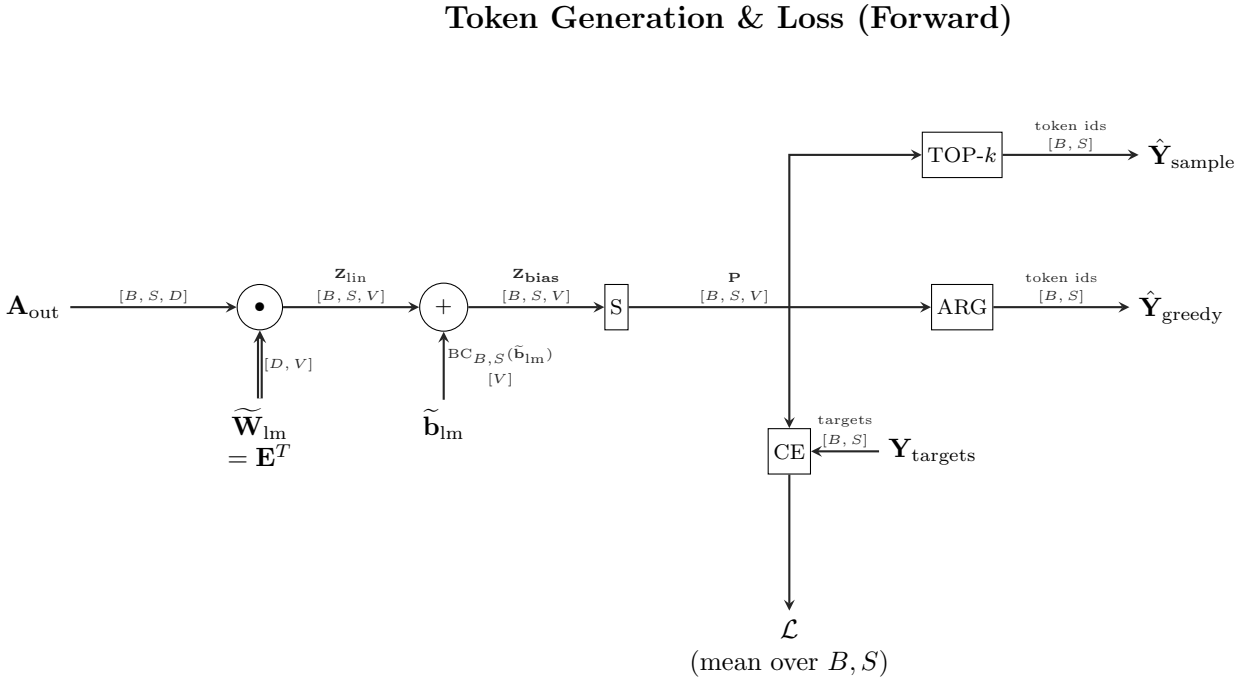


5.4.2 Backward Pass



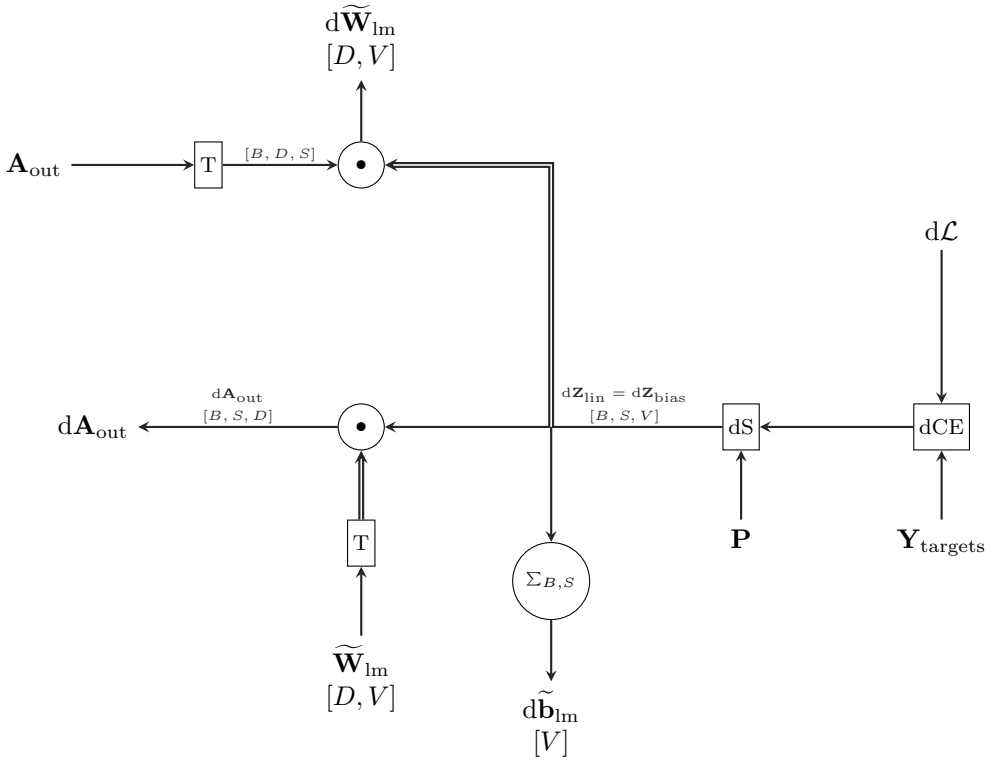
5.5 Output Projection and Loss

5.5.1 Forward Pass (Logits, Softmax, Loss)



5.5.2 Backward Pass

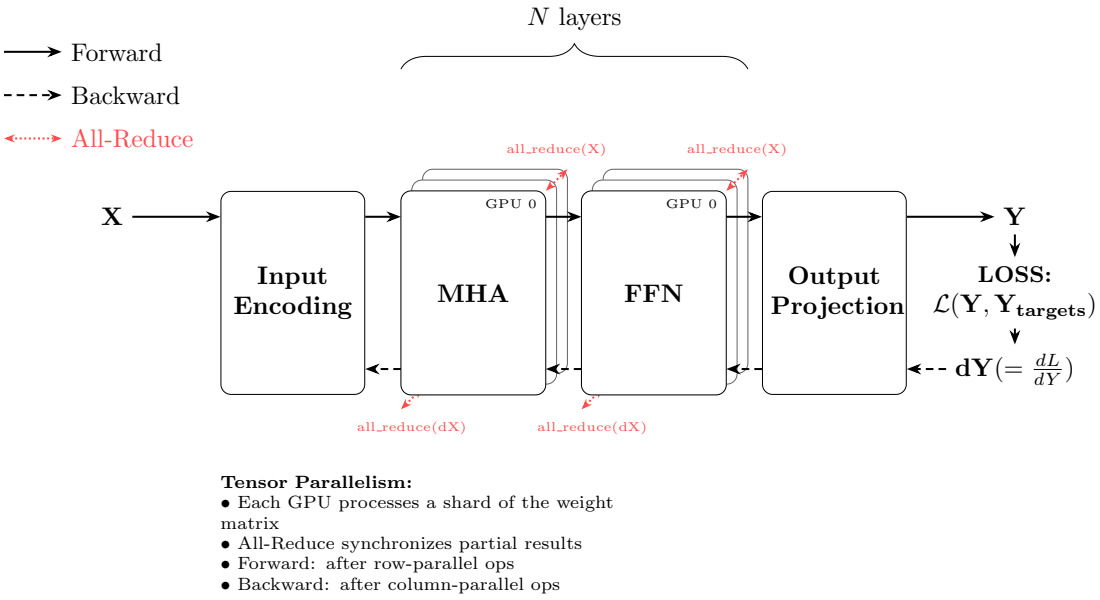
Token Generation & Loss — Backward (Corrected)



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.

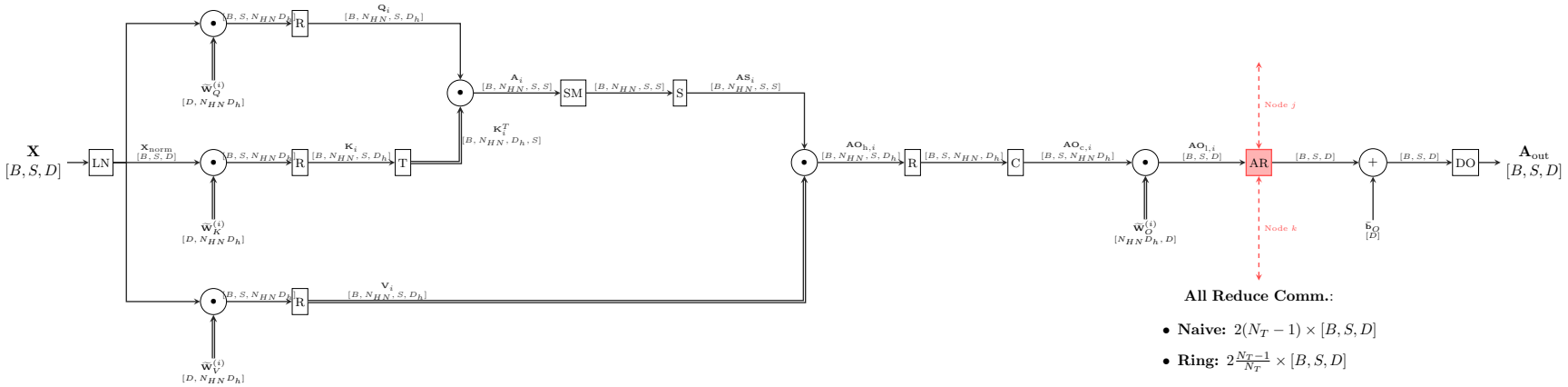
6.1 TP Overview and End-to-End Flow



6.2 MHA with Tensor Parallelism

6.2.1 Forward Pass

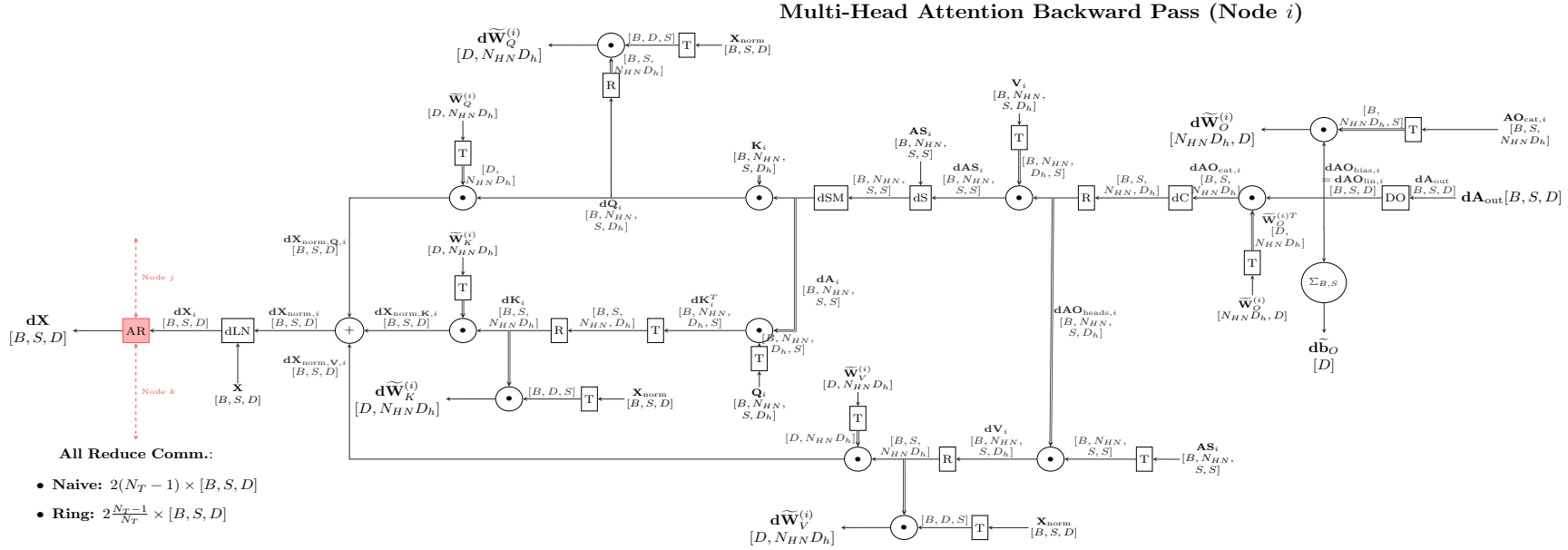
Multi-Head Attention Forward Pass (Node i)



All Reduce Comm.:

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

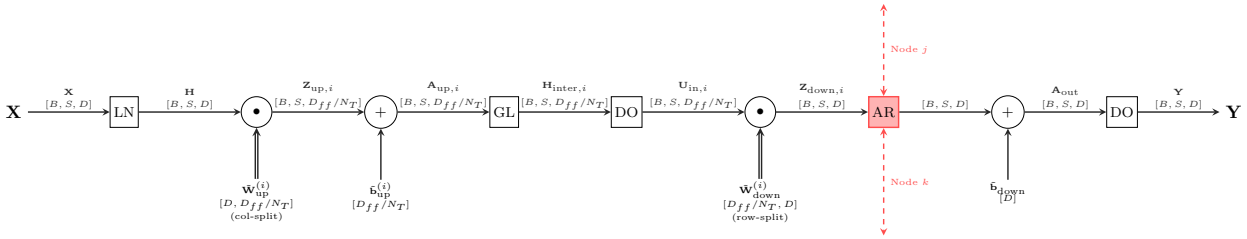
6.2.2 Backward Pass


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

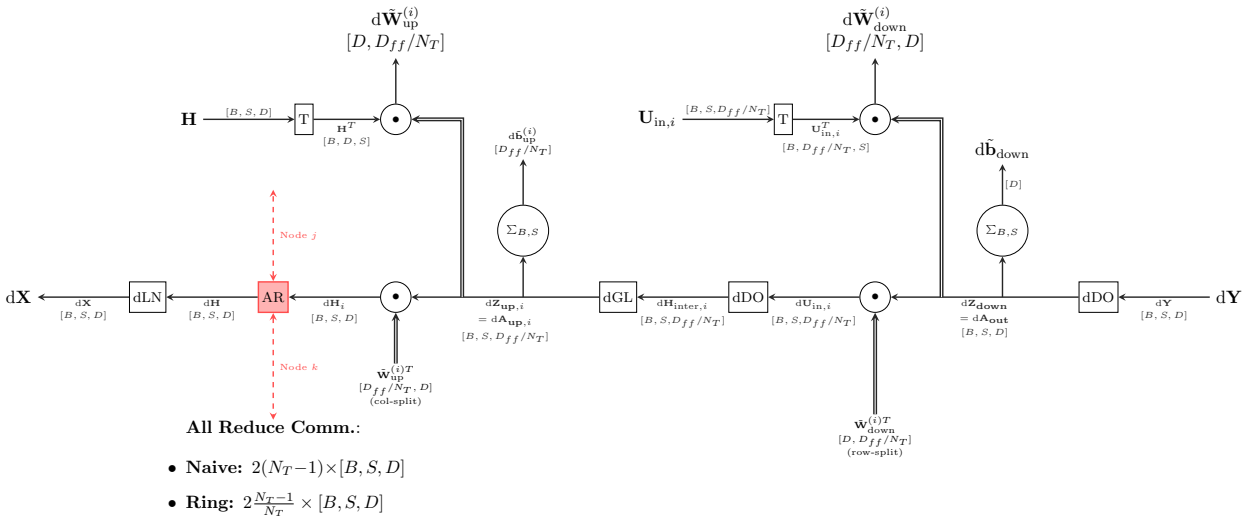
MLP Forward Pass (Node i)



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



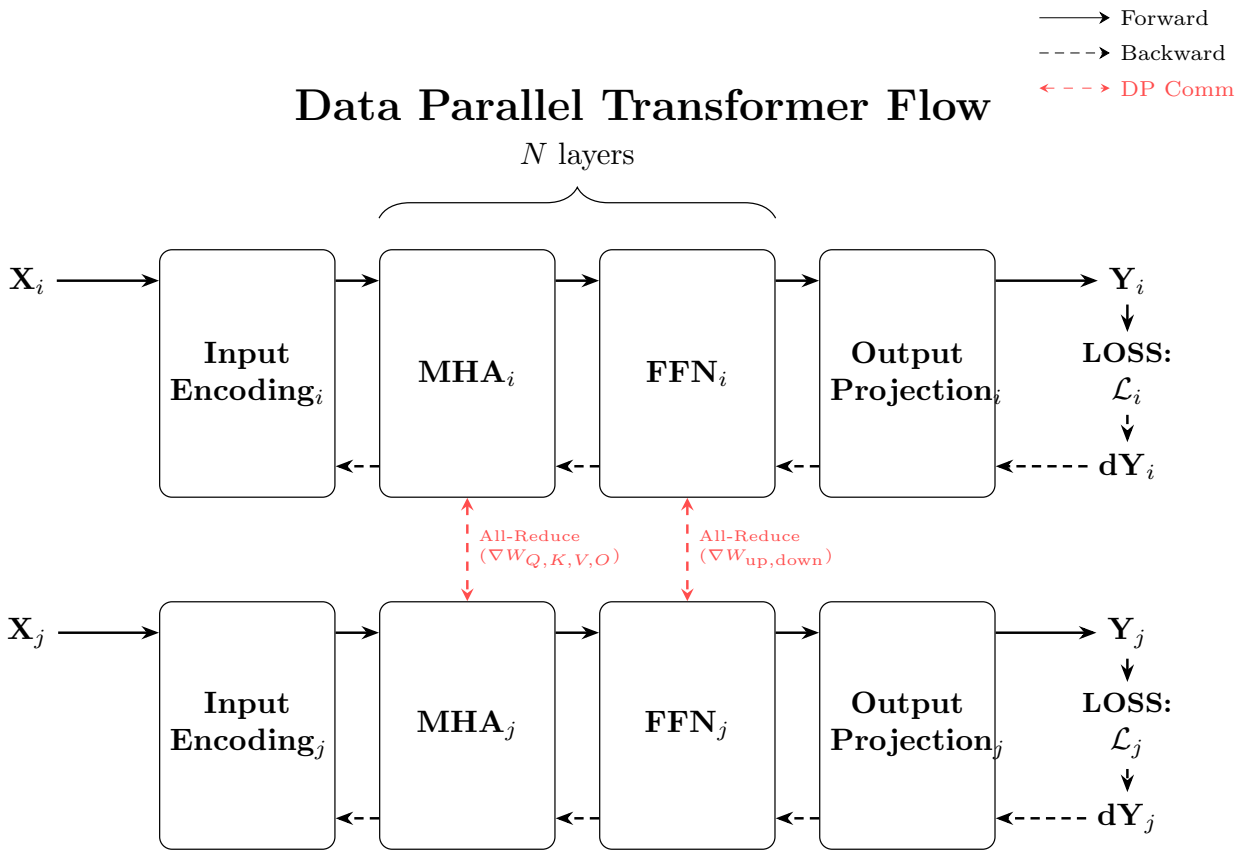
All Reduce Comm.:

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

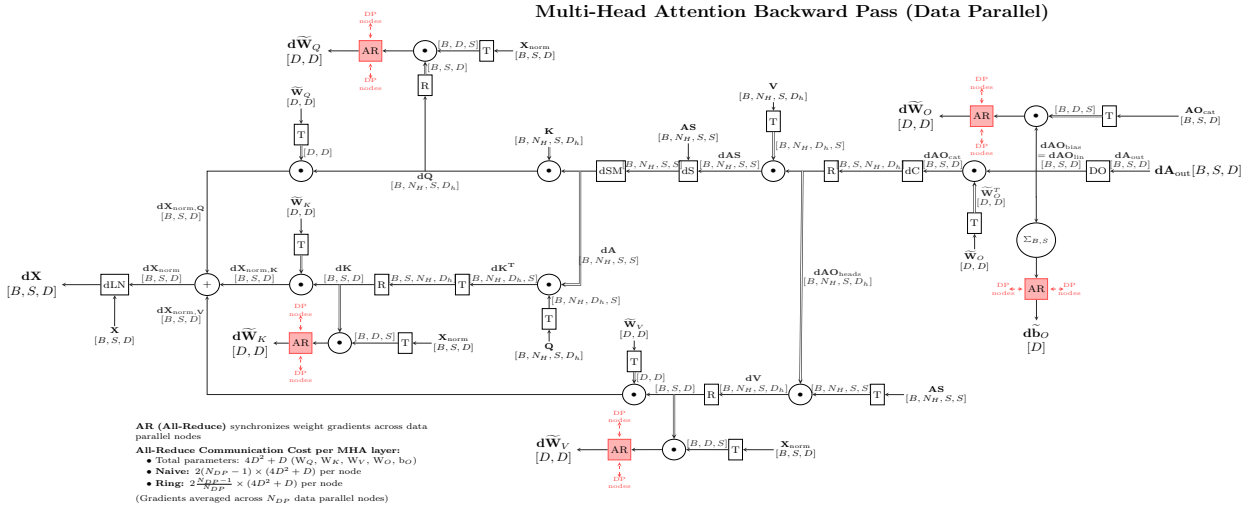
7 Data Parallelism (DP)

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.

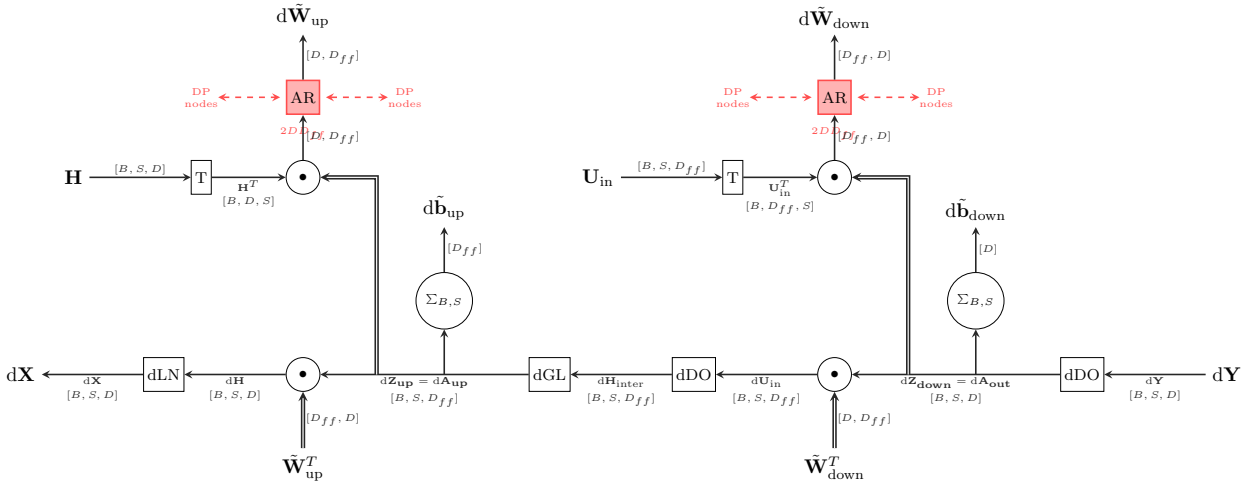
7.1 DP Overview and Transformer Flow



7.2 MHA Backward under DP



7.3 MLP Backward under DP



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

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

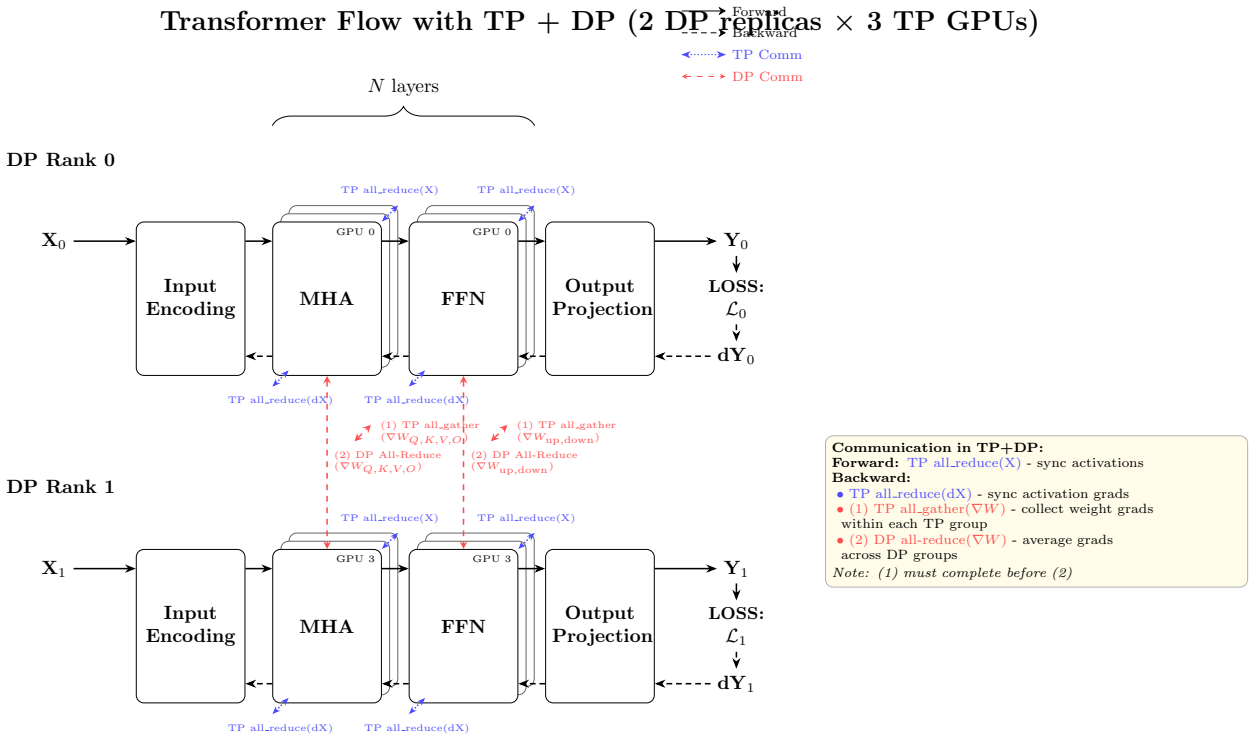
- **Naive:** $2(N_{DP} - 1) \times 2DD_{ff}$ per node
- **Ring:** $2\frac{N_{DP}-1}{N_{DP}} \times 2DD_{ff}$ per node

(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



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.