Original Layer Normalization Model

The conventional Layer Normalization model computes the mean and variance for each input sample as follows

$$\mu_i = \frac{1}{d_{model}} \cdot \sum_{j=1}^{d_{model}} x_{i,j},\ \sigma_i^2 = \frac{1}{d_{model}} \cdot \sum_{j=1}^{d_{model}} \left(x_{i,j} - \mu_i\right)^2$$

This approach involves division and square root operations, which introduce significant computational complexity and latency.

To address this, an approximation model is adopted to reduce both computational load and processing delay.

Pairwise model to calculate variance and mean using IP Catalog.

In the conventional calculation process, no optimization was applied,

and the computations were carried out sequentially.

Module name	Bit	IPs used	Latency
addsub_add(sub)	16bit	2 * 63 = **126**	**1**
addsub_add(sub)	32bit	3 * 63 = **189**	**1**
mult_gen_mult	32bit	5 * 63= **315**	**4**

Suppose IP Catalog when implement Pairwise

63EA : 32 + 16 + 8 + 4 + 2 + 1 (Pairwise Structure)

Pairwise_module

each stage : 11 Latency
Total stage : 64 → 32 → 16 → 8 → 4 → 2 → 1

Total Latency : 66 Latency

As a result, each stage introduced significant latency(suppose IP catalog),

leading to challenges when comparing performance or results.

To address this, we precomputed terms that can be calculated in advance,

such as $\frac{(n_1)(n_2)}{n_1+n_2}$ and $\boldsymbol{\delta = x_1 - x_2}$ in order to optimize the overall delay.

With this optimization, we minimized the latency required for the full computation. During the pairwise reduction process from **64 → 32 → 16 → 8 → 4 → 2 → 1,**

the term $\frac{(n_1)(n_2)}{n_1+n_2}$ can be simplified as a power-of-two shift operation in each stage

: $\frac{1}{2}$ (64→32) → **1** (32→16) → **2** (16→8) → **4** (8→4) → 8 (4→2) → 16 (2→1).

Hence, only the multiplication of δ and the shifted value requires an actual multiplication operation. The final intermediate variance (intVar) is computed by sequentially accumulating

$$intVar = intVar_1 + intVar_2 + \frac{\delta(n_1+n_2)(n_3+n_4)}{n_1+n_2+n_3+n_4}.$$

Consequently, the initial 64 → 32 stage requires a latency of 5 cycles,

and each subsequent stage (32→16, 16→8, ..., 2→1) adds 2 cycles.

The total latency is thus reduced to **15 cycles.**

Using equation about $\sigma_x^2[15:0] = E[x^2][31:0] - E[x]^2[31:0]$

To implement the mean and variance calculations efficiently, an **adder tree structure** was designed using Vivado IP cores.

Adder Tree Depth : 63 intermediate additions (32 + 16 + 8 + 4 + 2 + 1)

- **Mean_Var Module Structure**

Latency per stage

: 10 cycles (6 cycles for multiplier IP + 4 cycles for adder IP) + 1 cycle for final output stage

Total Latency : 11 cycles for full reduction (from 64 inputs to 1 output)

Compare the Pairwise model and Our approx model

Same as the output of Q8.8 format, as they are mathematically equivalent.

Latency

Pairwise model	**15 clk**
Mean_var model	**11 clk**

IP

Pairwise model

Module name	Bit	IPs used	Latency
addsub_add(sub)	16bit	2 * 63 = **126**	**1**
addsub_add(sub)	32bit	2 * 63 = **126**	**1**
mult_gen_mult	32bit	63 = **63**	**4**

Mean_var model

Module name	Bit	IPs used	Latency
addsub_add(sub)	16bit	63(Tree) = **63**	**1**
addsub_add(sub)	32bit	63(Tree) + 1(sub) = **64**	**1**
mult_gen_mult	32bit	64(x^2) + 1(μ^2) = **65**	**4**

Reduce the Latency : 15 clk → 11 clk