

Gemma3, Architecture and Mathematical Foundations

Saman Pour

December 16, 2025

Most of the time, we hear about new model releases, new innovations in them or new way of passing around the data; but in reality very few people know what really happens inside a model.

Everyone starts to refer very general architectural images created years back. The classic can be the widely used Figure 1 from "Attention Is All You Need" [1] paper.

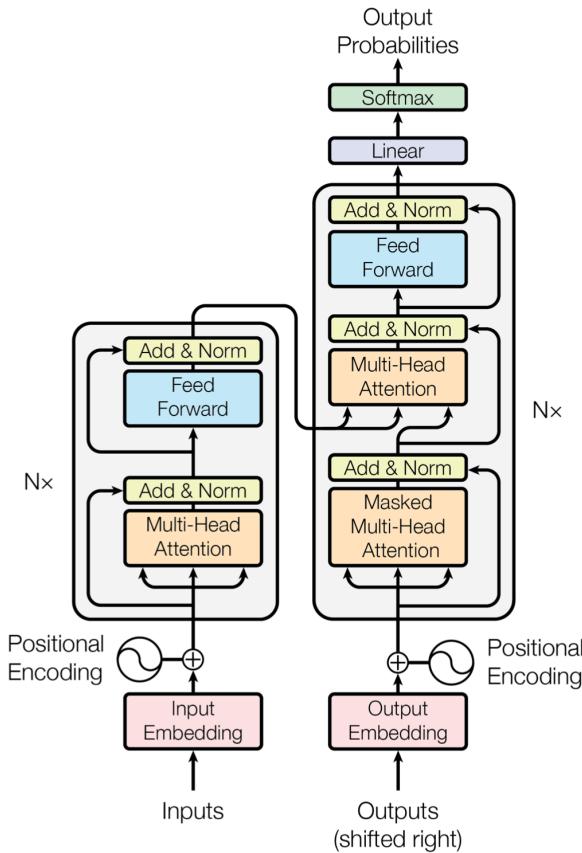


Figure 1: The Transformer - model architecture

Through my work in the domain and studies I've found that the general architecture won't cut it. We need mathematical representation. In addition to that, we need reference implementation that we can use to conduct numerics debugging. A small numerical discrepancy in the fourth decimal point, can easily turn into catastrophe in output generation when repeated through tens of layers in the decoder.

You might say; hey, just the mathematical description and representation is enough!

I would say the accurate implementation matters too. For example in RMSNorm, Llama does `x.to(float16) * w` whilst Gemma does `(x.to(float) * w).to(float16)`. The former rounds the results, while the later performs the calculation in 32 bit precision first and then rounds it. This slight difference in implementation makes model be better in higher context lengths.

And that's my motivation for creating meticulously numerical checked implementation and share it along reports like this full of mathematical background of the subject.

I hope you enjoy this report as much as I enjoyed writing it.

1 Architecture

I think the best approach to review the architecture of an LLM model is from top to bottom. The core of Gemma3 is decoder architecture, sharing it in fig 2.

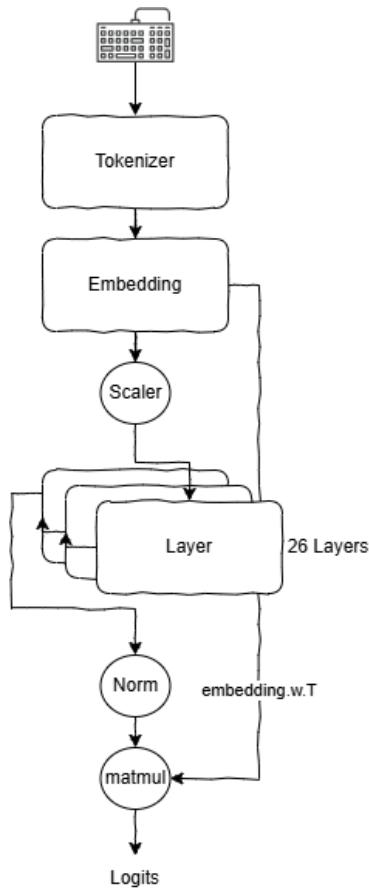


Figure 2: Gemma3 - high level architecture

1.1 Tokenizer

For tokenizer it's using SentencePieceProcessor tokenizer with vocab size of 262144. Nothing out of ordinary in this space, tokenizer model is shipped with the original model package and we can just load it with the same library.

1.2 Embedding

The model has hidden or embedding size of 1152. We're using nn.Embedding implementation to load embedding weights and apply embedding to the input ids.

After embedding, model does apply an scaling function:

$$H^{(0)} = E[x] \in \mathbb{R}^{T \times d}$$

$$H^{(0)} \leftarrow \sqrt{d} \cdot H^{(0)}$$

This multiplying by $\sqrt{d_{\text{model}}}$ prevents embeddings from being numerically too small relative to QKV projections.

As for the head, or where we want to turn embeddings into result ids, we use the same embedding weight transposed.

2 Gemma-3 Transformer Layer

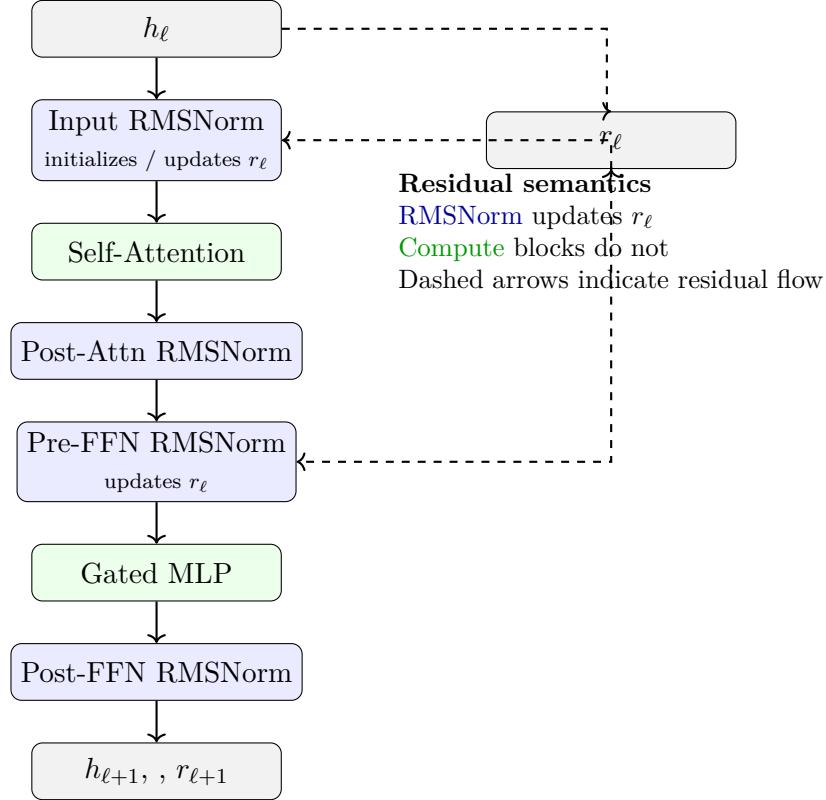


Figure 3: Gemma-3 Transformer layer with explicit residual handling. The residual stream is initialized at the first RMSNorm, updated only at specific RMSNorm boundaries, and returned alongside the hidden state.

Each Gemma-3 layer follows a pre-normalized Transformer design with multiple RMSNorm stages and an explicit residual stream. Let

$$h_\ell \in \mathbb{R}^{T \times d_{\text{model}}}$$

denote the input to layer ℓ , and let r_ℓ denote the residual accumulator.

2.1 Input Normalization and Residual Initialization

At the first layer, the residual is initialized directly from the input:

$$r_\ell = h_\ell.$$

The hidden state is then normalized using Gemma RMSNorm:

$$\tilde{h}_\ell = \text{RMSNorm}_{\text{in}}(h_\ell).$$

For subsequent layers, residual accumulation and normalization are fused:

$$(\tilde{h}_\ell, r_\ell) = \text{RMSNorm}_{\text{in}}(h_\ell + r_{\ell-1}),$$

ensuring that normalization always operates on the accumulated signal.

2.2 Self-Attention Block

The normalized hidden state is passed through multi-head self-attention:

$$a_\ell = \text{Attention}(\tilde{h}_\ell),$$

followed by a post-attention RMS normalization:

$$\hat{h}_\ell = \text{RMSNorm}_{\text{attn}}(a_\ell).$$

Unlike classic Transformer blocks, the attention output is not immediately added to the residual. Instead, residual updates are deferred and handled by subsequent normalization layers.

2.3 Feed-Forward Block

Before entering the MLP, the hidden state is again normalized while updating the residual:

$$(\bar{h}_\ell, r_\ell) = \text{RMSNorm}_{\text{pre-fnn}}(\hat{h}_\ell + r_\ell).$$

The Gemma MLP is a gated feed-forward network:

$$\text{MLP}(x) = W_{\text{down}} \left(\text{GELU}_{\tanh}(W_{\text{gate}}x) \odot (W_{\text{up}}x) \right).$$

The MLP output is then normalized once more:

$$h_{\ell+1} = \text{RMSNorm}_{\text{post-fnn}}(\text{MLP}(\bar{h}_\ell)).$$

The layer returns both the transformed hidden state and the updated residual:

$$(h_{\ell+1}, r_\ell).$$

2.4 Key Architectural Characteristics

- Multiple RMSNorm stages are used to tightly control activation scale.
- Residual accumulation is explicitly separated from normalized activations.
- Attention and MLP blocks do not directly add to the residual, reducing variance explosion.
- All normalization layers use Gemma RMSNorm with $(1 + \gamma)$ scaling.

3 Gemma MLP

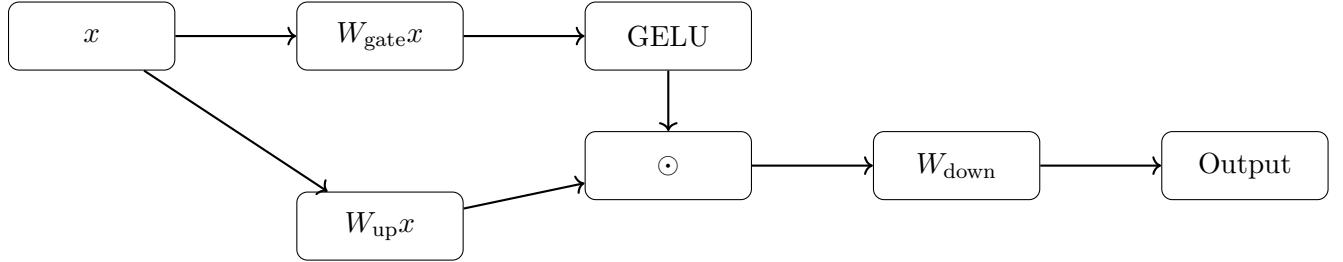


Figure 4: Gated MLP used in Gemma. The gate and up projections are fused in implementation but shown separately for clarity.

The Gemma MLP is a gated feed-forward network applied independently at each token position. For an input activation

$$x \in \mathbb{R}^{d_{\text{model}}},$$

the MLP computes

$$\text{MLP}(x) = W_{\text{down}} \left(\text{GELU}_{\tanh}(W_{\text{gate}} x) \odot (W_{\text{up}} x) \right),$$

where

$$W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \quad W_{\text{down}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}.$$

In implementation, the gate and up projections are fused into a single matrix multiplication:

$$[g(x), u(x)] = \begin{bmatrix} W_{\text{gate}} \\ W_{\text{up}} \end{bmatrix} x,$$

$$\text{MLP}(x) = W_{\text{down}}(\text{GELU}_{\tanh}(g(x)) \odot u(x)),$$

reducing memory reads and improving throughput.

The activation function is the tanh-based approximation of GELU,

$$\text{GELU}_{\tanh}(z) = \frac{1}{2}z \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} \left(z + 0.044715 z^3 \right) \right) \right),$$

which provides a smooth gating signal compared to other activation functions.

Compared to a standard two-layer feed-forward network,

$$\text{FFN}(x) = W_2 \sigma(W_1 x).$$

the gated structure introduces multiplicative interactions that allow the model to dynamically control feature flow. This increases expressivity without additional depth and is a key contributor to Gemma’s strong performance per parameter.

The MLP is used inside a pre-normalized residual block:

$$h_{\ell+1} = h_\ell + \text{MLP}(\text{RMSNorm}(h_\ell)),$$

ensuring stable optimization in deep Transformer stacks.

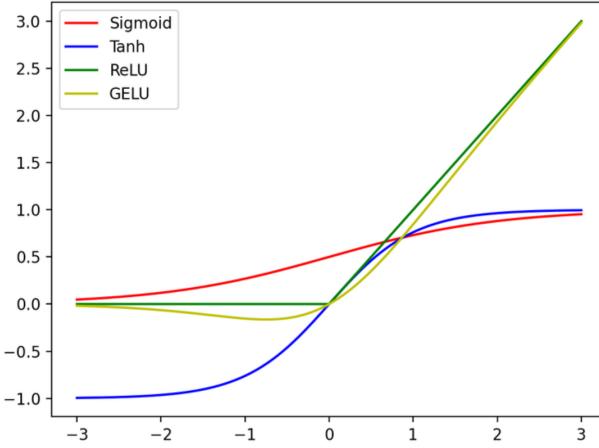


Figure 5: Gelu and other activation functions

4 Gemma RMSNorm

Let $x \in \mathbb{R}^d$ denote the input activation. Gemma uses a modified RMSNorm defined as

$$\text{RMS}(x) = \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \varepsilon}, \quad \hat{x} = \frac{x}{\text{RMS}(x)}.$$

The normalized activation is then scaled as

$$\text{GemmaRMSNorm}(x) = \hat{x} \odot (1 + \gamma),$$

where $\gamma \in \mathbb{R}^d$ is a learned parameter initialized at zero.

This differs from standard RMSNorm, which applies

$$\text{RMSNorm}(x) = \hat{x} \odot \gamma,$$

typically with γ initialized to ones. The Gemma formulation ensures that the initial transformation is the identity,

$$\text{GemmaRMSNorm}(x) \approx x \quad \text{at initialization,}$$

improving training stability.

Additionally, Gemma applies scaling in full precision before casting back to the original dtype:

$$x_{\text{out}} = (\hat{x} \odot (1 + \gamma))_{\text{float}} \xrightarrow{\text{cast}} \text{orig_dtype},$$

which differs from implementations that cast first and then apply scaling. This ordering reduces numerical error in low-precision (e.g., FP16) training and inference.

When residual connections are present, normalization is applied after residual accumulation:

$$x \leftarrow x + r, \quad r \leftarrow x, \quad x_{\text{out}} = \text{GemmaRMSNorm}(x),$$

decoupling residual state from the normalized activation.

References

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.