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 reallity 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.
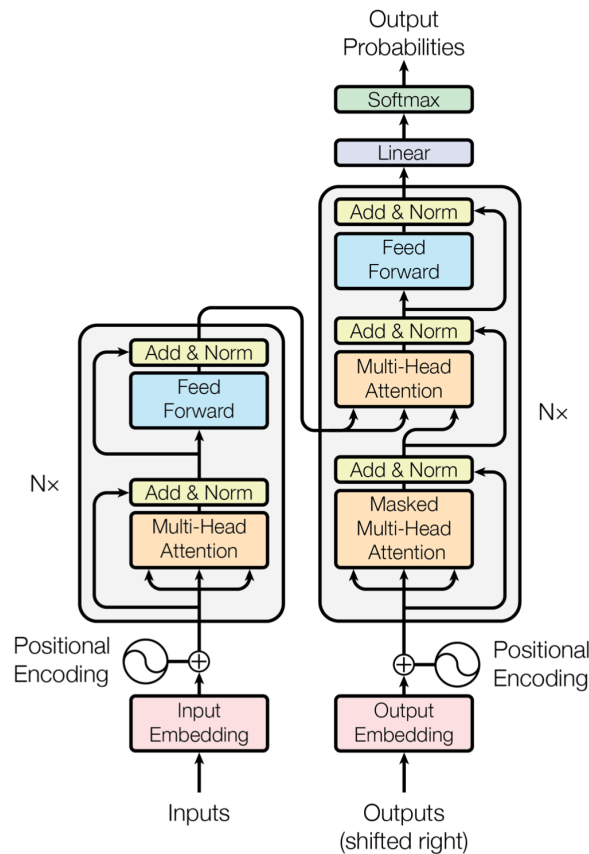

Figure 1: The Transformer - model architecture

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 discrepency in the fourth decimal point, can easily turn into catastrophy in output generation when repeated through tens of layers in the decoder.

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

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 lenghts.

And that's my motivation for creating metuculusly 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.
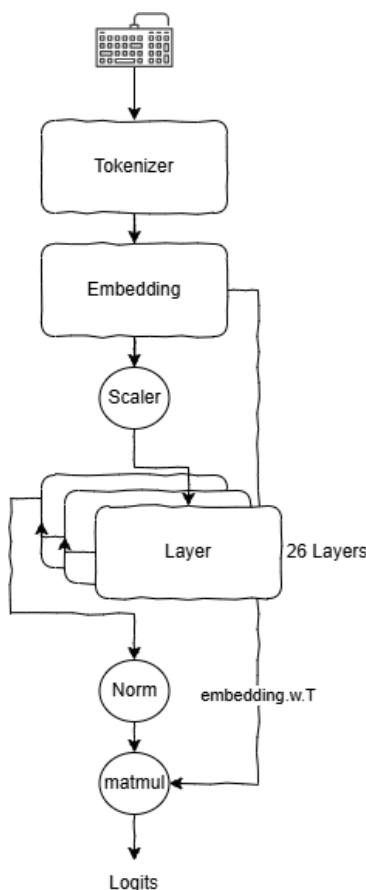

Figure 2: Gemma3 - high level architecture

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 embeding 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 RMSNorm

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

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

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:

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

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.