Gemma3, Architecture and Mathematical Foundations

Saman Keon

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.

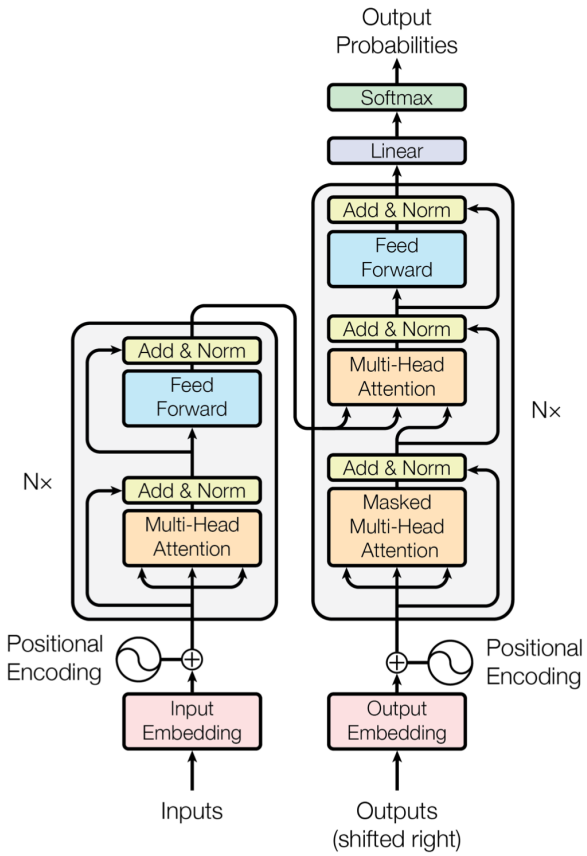

Figure 1: The Transformer - model architecture

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 discripency 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.

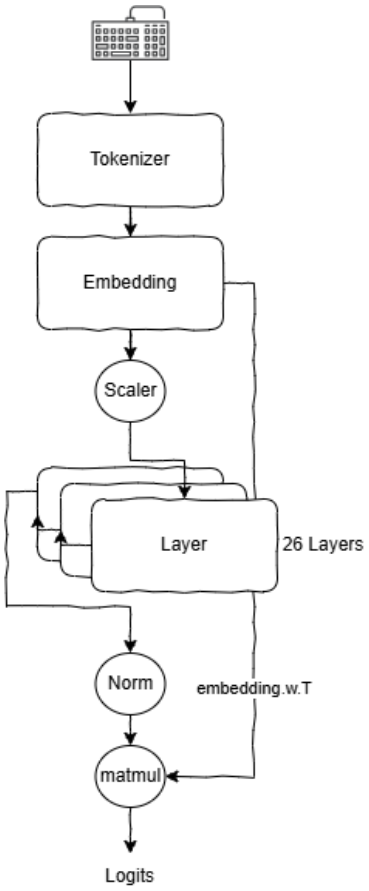

Figure 2: Gemma3 - high level architecture

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 Transformer Layer

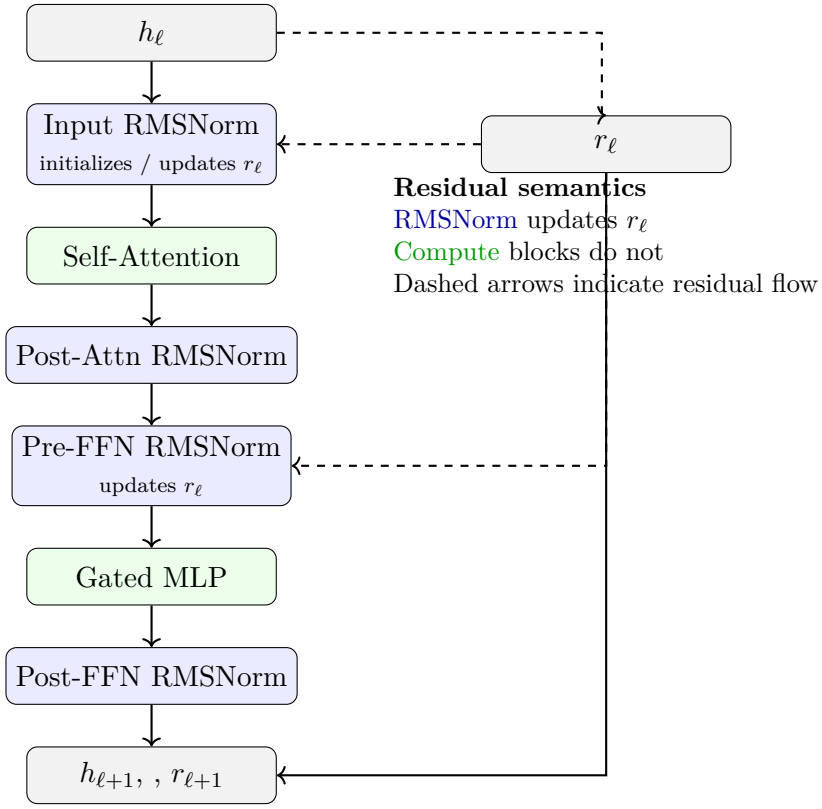

Figure 3: Gemma-3 Transformer layer with explicit residual handling. The residual stream is initialized at the first RMSNorm, updated only at specific RMSNorm boundaries, and returned alongside the hidden state.

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:

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

For subsequent layers, residual accumulation and normalization are fused:

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

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:

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

followed by a post-attention RMS normalization:

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

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:

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

The Gemma MLP is a gated feed-forward network:

$$\mathrm{MLP}(x) = W_{\mathrm{down}}\Big(\mathrm{GELU}_{\mathrm{tanh}}(W_{\mathrm{gate}}x) \odot (W_{\mathrm{up}}x)\Big).$$

The MLP output is then normalized once more:

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

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 Self-Attention

Gemma-3 employs a *multi-query self-attention* (MQA) mechanism with per-head normalization and rotary positional embeddings. Let

$$x \in \mathbb{R}^{B \times T \times d_{\text{model}}}$$

denote the input hidden states for batch size B and sequence length T.

3.1 Multi-Query Attention (MQA)

In Gemma-3, the number of query heads exceeds the number of key–value heads:

$$H_Q > 1, \qquad H_{KV} = 1.$$

All query heads attend to a single shared key–value representation. This formulation is known as *multi-query attention* (MQA) and reduces memory bandwidth and KV-cache size during autoregressive decoding.

MQA vs. GQA vs. MHA. The attention mechanism in Gemma-3 belongs to the family of query–key head sharing schemes. These variants differ in the number of key–value heads relative to query heads:

$$H_Q \geq H_{KV} \geq 1.$$

- **Multi-Head Attention (MHA):**

  $$H_{KV} = H_Q.$$

  Each query head has its own independent key and value heads. This maximizes expressivity but incurs the highest memory and bandwidth cost.

- **Grouped-Query Attention (GQA):**

  $$1 < H_{KV} < H_Q.$$

  Query heads are partitioned into groups, with each group sharing a key–value head. GQA trades a small loss in expressivity for reduced KV-cache size.

- **Multi-Query Attention (MQA):**

  $$H_{KV} = 1.$$

  All query heads attend to a single shared key–value representation. This minimizes KV-cache memory and improves decoding throughput, making it well-suited for long-context and inference-focused models.

In Gemma-3, the choice $H_{KV} = 1$ reflects a design trade-off favoring inference efficiency over maximal per-head expressivity, with minimal observed impact on model quality at sufficient scale.

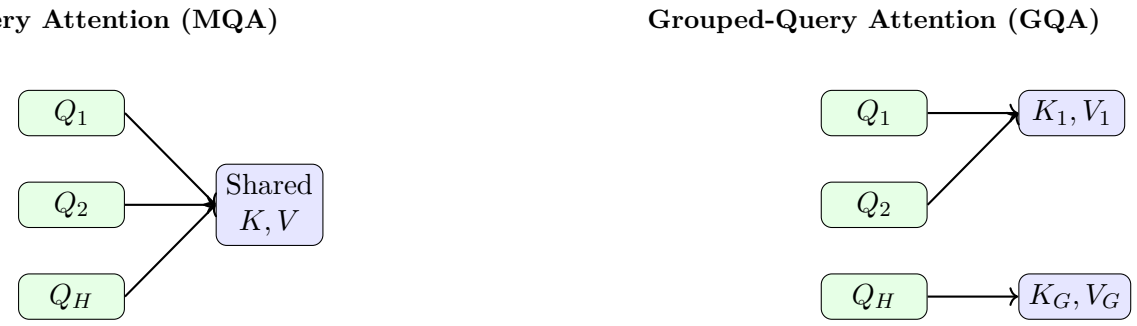

Figure 4: Comparison of Multi-Query Attention (MQA) and Grouped-Query Attention (GQA). In MQA, all query heads share a single key–value pair. In GQA, query heads are partitioned into groups, each with its own shared key–value head.

3.2 Linear Projections and Normalization

Queries, keys, and values are computed as

$$\begin{aligned} Q &= W_Q x, \\ K &= W_K x, \\ V &= W_V x, \end{aligned}$$

with

$$W_Q \in \mathbb{R}^{(H_Q d_h) \times d_{\text{model}}}, \quad W_K, W_V \in \mathbb{R}^{d_h \times d_{\text{model}}},$$

where d_h is the head dimension.

Keys are normalized using RMSNorm:

$$K \leftarrow \text{RMSNorm}(K),$$

while queries are reshaped into heads and normalized per head:

$$Q \in \mathbb{R}^{B \times H_Q \times T \times d_h}, \qquad Q \leftarrow \text{RMSNorm}(Q).$$

Values are projected but not normalized.

3.3 Key–Value Head Expansion

Since $H_{KV} = 1$, the key and value tensors are broadcast across query heads:

$$K, V \in \mathbb{R}^{B \times H_Q \times T \times d_h},$$

ensuring that all query heads attend to the same key–value representations.

3.4 Rotary Positional Embeddings

Rotary positional embeddings are applied to queries and keys:

$$(Q, K) \leftarrow \text{RoPE}(Q, K; \theta),$$

where the base frequency θ alternates by layer index:

- **Global attention layers**: $\theta = 10^6$,
- **Local (sliding) attention layers**: $\theta = 10^4$.

This interleaving enables a hybrid of long-range and local contextual modeling.

3.5 Attention Weights and Masking

Scaled dot-product attention is computed as

$$A = \frac{QK^\top}{\sqrt{d_h}},$$

followed by a causal mask:

$$A_{ij} = \begin{cases} A_{ij}, & j \le i, \\ -\infty, & j > i. \end{cases}$$

The attention weights are obtained via

$$W = \mathrm{softmax}(A).$$

3.6 Context Aggregation and Output Projection

The context vectors are computed as

$$C = WV \in \mathbb{R}^{B \times H_Q \times T \times d_h}.$$

These are concatenated and projected back to the model dimension:

$$\mathrm{Attention}(x) = W_O \ \mathrm{concat}_h(C), \qquad W_O \in \mathbb{R}^{d_{\text{model}} \times (H_Q d_h)}.$$

3.7 Summary

- Gemma-3 uses **multi-query attention** with a single shared key–value head.
- Per-head RMSNorm stabilizes attention logits.
- Alternating RoPE frequencies enable local and global attention.
- Residual connections are managed outside the attention module.

4 Rotary Positional Embeddings (RoPE)

Gemma-3 encodes positional information using *rotary positional embeddings* (RoPE), which inject relative position information directly into the query and key vectors via rotation in feature space. Unlike additive positional embeddings, RoPE preserves relative distance information through inner products.

Let

$$Q, K \in \mathbb{R}^{B \times H \times T \times d_h}$$

denote the query and key tensors, where d_h is even.

4.1 Frequency Construction

The head dimension is split into two halves:

$$d_h = 2d', \qquad Q = [Q^{(1)}, Q^{(2)}], \quad K = [K^{(1)}, K^{(2)}],$$

with $Q^{(i)}, K^{(i)} \in \mathbb{R}^{B\times H\times T\times d'}$.

For each dimension index $i \in \{0, \dots, d'-1\}$, the inverse frequency is defined as

$$\omega_i = \frac{1}{\theta^{2i/d_h}},$$

where θ is the RoPE base frequency (e.g., $\theta = 10^6$ for global attention).

For sequence position $t \in \{0, \dots, T-1\}$, the rotation angle is

$$\phi_{t,i} = t \cdot \omega_i.$$

4.2 Rotary Transformation

For each position t and head, RoPE applies a complex-valued rotation to pairs of features:

$$Q^{(1)}_{t,i} \leftarrow Q^{(1)}_{t,i} \cos\phi_{t,i} - Q^{(2)}_{t,i} \sin\phi_{t,i},$$
$$Q^{(2)}_{t,i} \leftarrow Q^{(1)}_{t,i} \sin\phi_{t,i} + Q^{(2)}_{t,i} \cos\phi_{t,i},$$

and analogously for K.

This operation can be interpreted as a complex multiplication:

$$(Q^{(1)} + iQ^{(2)}) \cdot (\cos\phi + i\sin\phi),$$

applied independently for each position and head.

4.3 Application in Attention

The rotary embedding is applied *before* computing attention scores:

$$(Q, K) \leftarrow \mathrm{RoPE}(Q, K; \theta), \qquad A = \frac{QK^\top}{\sqrt{d_h}}.$$

Because both queries and keys are rotated by the same position-dependent transformation, the resulting dot product depends only on *relative* positions:

$$\langle Q_t, K_s \rangle = \langle Q_{t-s}, K_0 \rangle,$$

up to a fixed phase shift. This property enables extrapolation to longer contexts than seen during training.

4.4 Design Choices in Gemma-3

Gemma-3 alternates RoPE configurations across layers:

- **Global attention layers**: large base frequency ($\theta = 10^6$) to support long-range dependencies.
- **Local (sliding) attention layers**: smaller base frequency ($\theta = 10^4$) to emphasize short-range structure.

This hybrid strategy balances global context modeling with strong local inductive bias.

4.5 Summary

- RoPE encodes position via rotation, not addition.
- Relative positional information is preserved in dot products.
- No learned positional parameters are required.
- Large base frequencies enable long-context extrapolation.

5 MLP

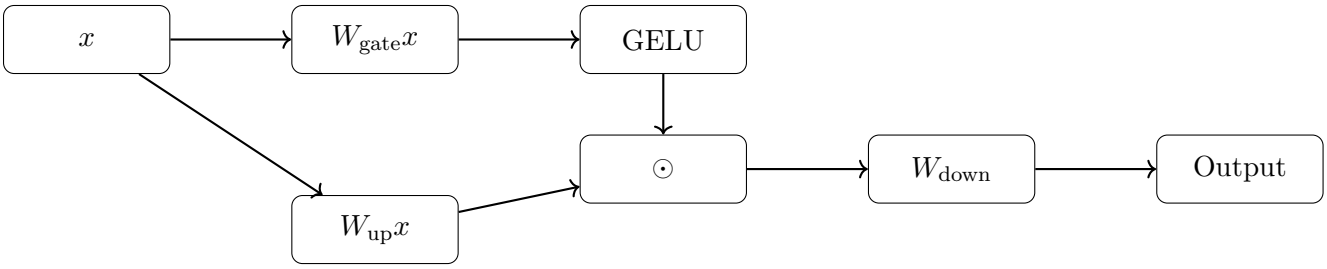

Figure 5: Gated MLP used in Gemma. The gate and up projections are fused in implementation but shown separately for clarity.

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

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

where

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

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

$$[g(x),\, u(x)] = \begin{bmatrix} W_{\text{gate}} \\ W_{\text{up}} \end{bmatrix} x,$$
$$\text{MLP}(x) = W_{\text{down}}\big(\text{GELU}_{\text{tanh}}(g(x)) \odot u(x)\big),$$

reducing memory reads and improving throughput.

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

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

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

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

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

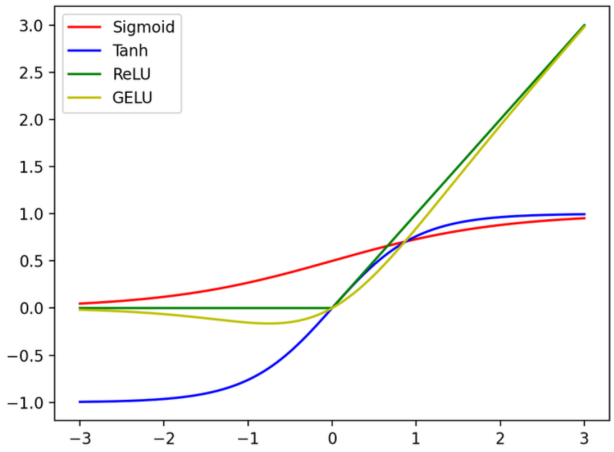

Figure 6: Gelu and other activation functions

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:

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

ensuring stable optimization in deep Transformer stacks.

6 Gemma RMSNorm

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

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

The normalized activation is then scaled as

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

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

This differs from standard RMSNorm, which applies

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

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

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

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.

7 Conclusion

The codebase accompanying this report is completely written by human. Similarly I started writing this technical document all manually, without help of AI. But soon learned that writing the LaTeX formulas can be daunting and error prone. Not only that, if I wanted a diagram rich document, I needed to ask an AI to help; else diagrams would be mediocre at best. So, sorry that this work is not purely human's work; though formulas and diagrams ideas came from me.

My next endeavor would be to implement Gemma with multi modal capabilities. I may come back and add some sections here, or publish a separate report. Will think about it.

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.