**Gemma Model**

Gemma3Model(  
 (tok\_emb): Embedding(50257, 640)  
 (blocks): ModuleList(  
 (0-17): 18 x TransformerBlock(  
 (att): GroupedQueryAttention(  
 (W\_query): Linear(in\_features=640, out\_features=1024, bias=False)  
 (W\_key): Linear(in\_features=640, out\_features=256, bias=False)  
 (W\_value): Linear(in\_features=640, out\_features=256, bias=False)  
 (out\_proj): Linear(in\_features=1024, out\_features=640, bias=False)  
 (q\_norm): RMSNorm()  
 (k\_norm): RMSNorm()  
 )  
 (ff): FeedForward(  
 (fc1): Linear(in\_features=640, out\_features=2048, bias=False)  
 (fc2): Linear(in\_features=640, out\_features=2048, bias=False)  
 (fc3): Linear(in\_features=2048, out\_features=640, bias=False)  
 )  
 (input\_layernorm): RMSNorm()  
 (post\_attention\_layernorm): RMSNorm()  
 (pre\_feedforward\_layernorm): RMSNorm()  
 (post\_feedforward\_layernorm): RMSNorm()  
 )  
 )  
 (final\_norm): RMSNorm()  
 (out\_head): Linear(in\_features=640, out\_features=50257, bias=False)  
)

awesome — we’ll zoom into **TransformerBlock.forward** and run a tiny numeric example so you can see **exact numbers** flowing through the **attention part** (the heaviest bit). I’ll keep it in **rows** (no tables), super literal.

For this demo I’ll use a tiny setup so math is readable:

* batch **B=1**, tokens **T=3**, model dim **D=4**
* **1 attention head** (so head\_dim = 2)
* attention type = "full\_attention" (uses the **global causal mask** and **global RoPE**)
* we’ll pick simple weights
* input to this block (called **x**) is:
  + t0: [1, 0, 0, 0]
  + t1: [0, 1, 0, 0]
  + t2: [0, 0, 1, 0]

**Step 0: shortcut & input layernorm (RMSNorm)**

* shortcut = x (we’ll add back later)
* RMSNorm per token (divide each token by its RMS).  
  RMS of a 4-d one-hot vector is sqrt(1/4)=0.5.

input\_layernorm(x) → call this **x\_norm**:

* t0: [1,0,0,0] / 0.5 → [2.0, 0.0, 0.0, 0.0]
* t1: [0,1,0,0] / 0.5 → [0.0, 2.0, 0.0, 0.0]
* t2: [0,0,1,0] / 0.5 → [0.0, 0.0, 2.0, 0.0]

**Step 1: linear projections to Q, K, V**

(use simple weight matrices)

* **Wq (4×2)** =  
  [[1,0], [0,1], [1,1], [0,0]]
* **Wk (4×2)** =  
  [[1,0], [0,1], [1,0], [0,0]]
* **Wv (4×2)** =  
  [[0,1], [1,0], [0,1], [0,0]]

Compute:

Q\_raw = x\_norm @ Wq

* t0: [2,0,0,0] @ Wq → [2, 0]
* t1: [0,2,0,0] @ Wq → [0, 2]
* t2: [0,0,2,0] @ Wq → [2, 2]

K\_raw = x\_norm @ Wk

* t0: [2,0,0,0] @ Wk → [2, 0]
* t1: [0,2,0,0] @ Wk → [0, 2]
* t2: [0,0,2,0] @ Wk → [2, 0]

V = x\_norm @ Wv

* t0: [2,0,0,0] @ Wv → [0, 2]
* t1: [0,2,0,0] @ Wv → [2, 0]
* t2: [0,0,2,0] @ Wv → [0, 2]

**Step 2: RoPE on Q and K**

Keep it simple: **head\_dim=2**, a single frequency band → angles per position = [0, 1, 2] radians for positions 0,1,2.

So **cos/sin** per position:

* pos0: cos=1.0000, sin=0.0000
* pos1: cos≈0.5403, sin≈0.8415
* pos2: cos≈-0.4161, sin≈0.9093

RoPE formula for a 2D pair (a,b):

* a' = a·cos − b·sin
* b' = a·sin + b·cos

Apply to **Q\_raw**:

* t0 (pos0, Q\_raw=[2,0]): → [2\*1 − 0\*0, 2\*0 + 0\*1] = [2.0000, 0.0000]
* t1 (pos1, Q\_raw=[0,2]): → [0\*0.5403 − 2\*0.8415, 0\*0.8415 + 2\*0.5403] = [-1.6829, 1.0806]
* t2 (pos2, Q\_raw=[2,2]): → [2\*(-0.4161) − 2\*0.9093, 2\*0.9093 + 2\*(-0.4161)] = [-2.6509, 0.9863]

Apply to **K\_raw**:

* t0 (pos0, K\_raw=[2,0]) → [2.0000, 0.0000]
* t1 (pos1, K\_raw=[0,2]) → [-1.6829, 1.0806]
* t2 (pos2, K\_raw=[2,0]) → [ -0.8323, 1.8186] (since b=0 at pos2)

Call these **Q** and **K**.

**Step 3: scale Q and build causal mask**

Scale Q by 1/sqrt(head\_dim)=1/sqrt(2)=0.7071:

* t0: [2.0000, 0.0000] \* 0.7071 → [1.4142, 0.0000]
* t1: [-1.6829, 1.0806] \* 0.7071 → [-1.1898, 0.7636]
* t2: [-2.6509, 0.9863] \* 0.7071 → [-1.8744, 0.6975]

Causal mask (no looking ahead):

mask True for j>i → mask (0,1), (0,2), (1,2).

**Step 4: attention scores S = QKᵀ, mask, softmax**

Dot products S[i,j] = Q[i] · K[j]:

Row i=0 (Q0=[1.4142,0]):

* S00 = [1.4142,0]·[2.0000,0.0000] = 2.8284
* S01 = dot with K1 = [1.4142,0]·[-1.6829,1.0806] = -2.3819
* S02 = dot with K2 = [1.4142,0]·[-0.8323,1.8186] = -1.1775

Row i=1 (Q1=[-1.1898,0.7636]):

* S10 = dot with K0 = -2.3796
* S11 = dot with K1 = ≈ 2.8277
* S12 = dot with K2 = ≈ 2.3807

Row i=2 (Q2=[-1.8744,0.6975]):

* S20 = dot with K0 = -3.7488
* S21 = dot with K1 = ≈ 3.9070
* S22 = dot with K2 = ≈ 2.8270

Apply causal mask (set masked to −∞):

* row0 → [ 2.8284, -inf, -inf ]
* row1 → [ -2.3796, 2.8277, -inf ]
* row2 → [ -3.7488, 3.9070, 2.8270 ] (no future to mask on last row)

Softmax per row → attention weights **A**:

* row0 softmax → [1.0000, 0.0000, 0.0000]
* row1 softmax → [0.0055, 0.9945, 0.0000]
* row2 softmax → [0.00035, 0.7467, 0.2530] (approx)

**Step 5: context = A @ V**

Recall V:

* v0: [0, 2]
* v1: [2, 0]
* v2: [0, 2]

Compute C = A @ V:

* t0: 1.0000\*v0 + 0\*v1 + 0\*v2 = [0.0000, 2.0000]
* t1: 0.0055\*v0 + 0.9945\*v1 + 0\*v2 = [1.9890, 0.0110]
* t2: 0.00035\*v0 + 0.7467\*v1 + 0.2530\*v2  
  = [1.4934, 0.5067]

**Step 6: output projection back to model dim**

Use simple **out\_proj W\_o (2×4)**:

* W\_o =  
  [[0.5, 0.0, 0.0, 0.0],  
  [0.0, 0.5, 0.0, 0.0]]

Attn output (**x\_attn\_raw** = C @ W\_o):

* t0: [0,2] @ W\_o → [0.0000, 1.0000, 0.0000, 0.0000]
* t1: [1.989, 0.011] @ W\_o → [0.9945, 0.0055, 0.0000, 0.0000]
* t2: [1.4934,0.5067] @ W\_o → [0.7467, 0.2533, 0.0000, 0.0000]

**Step 7: post-attention RMSNorm, then residual add**

Post-attention RMSNorm (normalize each row):

* t0: [0,1,0,0] → RMS=sqrt(1/4)=0.5 → normed: [0,2,0,0]
* t1: [0.9945, 0.0055, 0,0] → normed ≈ [2.0000, 0.0111, 0,0]
* t2: [0.7467, 0.2533, 0,0] → normed ≈ [1.8930, 0.6425, 0,0]

Residual add: **x\_after\_attn = shortcut + post\_norm**

* shortcut was the original block input:
  + t0: [1,0,0,0]
  + t1: [0,1,0,0]
  + t2: [0,0,1,0]

Add them:

* t0: [1,0,0,0] + [0,2,0,0] → [1.0000, 2.0000, 0.0000, 0.0000]
* t1: [0,1,0,0] + [2.0000, 0.0111, 0,0] → [2.0000, 1.0111, 0.0000, 0.0000]
* t2: [0,0,1,0] + [1.8930, 0.6425, 0,0] → [1.8930, 0.6425, 1.0000, 0.0000]

👉 This vector now goes into the **feed-forward** part of the block (pre-FF RMSNorm → two linears fc1/fc2 → GELU gate → fc3 → post-FF RMSNorm → residual add). The FFN step is the same idea: linear algebra with elementwise nonlinearity and another residual. If you want, I can run the **exact same row-by-row numbers** through a tiny FFN (with simple weights) next.

perfect — continuing exactly where we left off (the **FFN half** of one TransformerBlock) with the **same three token rows**:

* after attention + residual, we had:  
  t0 = [1.0000, 2.0000, 0.0000, 0.0000]  
  t1 = [2.0000, 1.0111, 0.0000, 0.0000]  
  t2 = [1.8930, 0.6425, 1.0000, 0.0000]

I’ll use a tiny, easy FFN so you can see the arithmetic without drowning in big matrices:

* **Pre-FFN RMSNorm** (per token).
* **fc1** = 0.5×identity (so x\_fc1 = 0.5·x\_norm)
* **fc2** = 0.25×identity (so x\_fc2 = 0.25·x\_norm)
* **gate** = GELU(x\_fc1) ⊙ x\_fc2 (elementwise)
* **fc3** = identity (so FF\_out = gate)
* **Post-FFN RMSNorm** then **residual add**.

Numbers below are rounded so you can follow (≈ means “about”).

**1) Pre-FFN RMSNorm (normalize each token vector)**

RMS = sqrt(mean of squares of the 4 dims)

t0 input [1, 2, 0, 0]

• sumsq = 1² + 2² = 5 → mean = 5/4 = 1.25 → RMS = √1.25 ≈ 1.1180

• normalized (**X2n t0**) = [1/1.1180, 2/1.1180, 0, 0] ≈ [0.8944, 1.7889, 0, 0]

t1 input [2, 1.0111, 0, 0]

• sumsq ≈ 4 + 1.0223 = 5.0223 → mean ≈ 1.2556 → RMS ≈ 1.1214

• **X2n t1** ≈ [1.784, 0.902, 0, 0]

t2 input [1.8930, 0.6425, 1.0000, 0]

• sumsq ≈ 3.584 + 0.413 + 1 = 4.997 → mean ≈ 1.2493 → RMS ≈ 1.1178

• **X2n t2** ≈ [1.693, 0.575, 0.895, 0]

**2) Parallel linear layers (fc1, fc2)**

(we chose fc1 = 0.5·I, fc2 = 0.25·I to keep math clear)

t0:

• x\_fc1 = 0.5·X2n ≈ [0.4472, 0.8945, 0, 0]

• x\_fc2 = 0.25·X2n ≈ [0.2236, 0.4472, 0, 0]

t1:

• x\_fc1 ≈ [0.892, 0.451, 0, 0]

• x\_fc2 ≈ [0.446, 0.2255, 0, 0]

t2:

• x\_fc1 ≈ [0.8465, 0.2875, 0.4475, 0]

• x\_fc2 ≈ [0.423, 0.1438, 0.2238, 0]

**3) Gate = GELU(x\_fc1) ⊙ x\_fc2 (elementwise)**

(Using tanh-approx GELU; I’m rounding to 3 decimals so it reads well.)

t0:

• GELU(0.4472) ≈ 0.301, GELU(0.8945) ≈ 0.727

• gate = [0.301·0.224, 0.727·0.447, 0, 0] ≈ [0.067, 0.326, 0, 0]

t1:

• GELU(0.892) ≈ 0.725, GELU(0.451) ≈ 0.304

• gate ≈ [0.725·0.446, 0.304·0.226, 0, 0] ≈ [0.323, 0.069, 0, 0]

t2:

• GELU(0.8465) ≈ 0.678, GELU(0.2875) ≈ 0.176, GELU(0.4475) ≈ 0.301

• gate ≈ [0.678·0.423, 0.176·0.144, 0.301·0.224, 0] ≈ [0.287, 0.025, 0.067, 0]

**4) fc3 back to model dim**

(we chose fc3 = identity → FF\_out = gate)

t0 FF\_out ≈ [0.067, 0.326, 0, 0]

t1 FF\_out ≈ [0.323, 0.069, 0, 0]

t2 FF\_out ≈ [0.287, 0.025, 0.067, 0]

**5) Post-FFN RMSNorm + residual add**

Normalize FF\_out per token, then add back the input to FFN (which was [1,2,0,0], [2,1.0111,0,0], [1.8930,0.6425,1.0000,0] from the attention residual).

t0:

• FF\_out sumsq ≈ 0.067² + 0.326² ≈ 0.1105 → mean ≈ 0.0276 → RMS ≈ 0.166

• post-FF norm ≈ [0.067/0.166, 0.326/0.166, 0, 0] ≈ [0.404, 1.959, 0, 0]

• **final after FFN** = attention-residual + post-FF-norm

= [1, 2, 0, 0] + [0.404, 1.959, 0, 0] ≈ **[1.404, 3.959, 0, 0]**

t1:

• sumsq ≈ 0.323² + 0.069² ≈ 0.1087 → mean ≈ 0.0272 → RMS ≈ 0.165

• post-FF norm ≈ [0.323/0.165, 0.069/0.165, 0, 0] ≈ [1.959, 0.416, 0, 0]

• **final after FFN** = [2.0000, 1.0111, 0, 0] + [1.959, 0.416, 0, 0] ≈ **[3.959, 1.427, 0, 0]**

t2:

• sumsq ≈ 0.287² + 0.025² + 0.067² ≈ 0.0875 → mean ≈ 0.0219 → RMS ≈ 0.148

• post-FF norm ≈ [0.287/0.148, 0.025/0.148, 0.067/0.148, 0] ≈ [1.940, 0.171, 0.456, 0]

• **final after FFN** = [1.8930, 0.6425, 1.0000, 0] + [1.940, 0.171, 0.456, 0] ≈ **[3.833, 0.8135, 1.456, 0]**

**What leaves this block (to the next block)**

t0 → [1.404, 3.959, 0, 0]

t1 → [3.959, 1.427, 0, 0]

t2 → [3.833, 0.8135, 1.456, 0]

reat — we’ll finish the pass with **final norm → logits → probabilities → loss**, using the **same three rows** we got at the end of the block’s FFN:

* t0 = [1.404, 3.959, 0.000, 0.000]
* t1 = [3.959, 1.427, 0.000, 0.000]
* t2 = [3.833, 0.8135, 1.456, 0.000]

We’ll map to a tiny vocab in this order: **[dog, is, chasing, a, ball]**.

Output head weights (4×5) (same for all tokens):

* W\_out row0: [ 0.2, 0.0, -0.1, 0.3, 0.1]
* W\_out row1: [-0.1, 0.2, 0.3, 0.0, 0.1]
* W\_out row2: [ 0.0, -0.2, 0.1, 0.2, -0.1]
* W\_out row3: [ 0.3, 0.1, 0.0, -0.2, 0.2]

**final RMSNorm (per token)**

RMS = sqrt(mean of squares of the 4 dims). Then divide the vector by that RMS.

t0: squares = 1.404² + 3.959² = 1.9712 + 15.6737 = 17.6449

* mean = 17.6449 / 4 = 4.4112 → RMS ≈ 2.101
* normalized x\_f0 = [1.404/2.101, 3.959/2.101, 0, 0] = [0.669, 1.885, 0, 0]

t1: squares = 3.959² + 1.427² = 15.6737 + 2.0363 = 17.7100

* mean = 4.4275 → RMS ≈ 2.104
* normalized x\_f1 = [3.959/2.104, 1.427/2.104, 0, 0] = [1.883, 0.678, 0, 0]

t2: squares = 3.833² + 0.8135² + 1.456² = 14.6919 + 0.6618 + 2.1199 = 17.4736

* mean = 4.3684 → RMS ≈ 2.090
* normalized x\_f2 = [3.833/2.090, 0.8135/2.090, 1.456/2.090, 0] = [1.834, 0.389, 0.697, 0]

**logits = x\_f @ W\_out (dot product with each column)**

(Remember vocab order: dog, is, chasing, a, ball)

**t0 (x\_f0 = [0.669, 1.885, 0, 0])**

* dog: 0.669*0.2 + 1.885*(-0.1) = 0.1338 − 0.1885 = **−0.0547**
* is: 0.669*0.0 + 1.885*0.2 = 0 + 0.377 = **0.3770**
* chasing: 0.669\*(−0.1) + 1.885\*0.3 = −0.0669 + 0.5655 = **0.4986**
* a: 0.669*0.3 + 1.885*0.0 = **0.2007**
* ball:0.669*0.1 + 1.885*0.1 = 0.0669 + 0.1885 = **0.2554**→ logits\_t0 = [-0.0547, 0.3770, 0.4986, 0.2007, 0.2554]  
  → target next token at t0 is **“is”**

**t1 (x\_f1 = [1.883, 0.678, 0, 0])**

* dog: 1.883*0.2 + 0.678*(−0.1) = 0.3766 − 0.0678 = **0.3088**
* is: 1.883*0.0 + 0.678*0.2 = **0.1356**
* chasing: 1.883\*(−0.1) + 0.678\*0.3 = −0.1883 + 0.2034 = **0.0151**
* a: 1.883*0.3 + 0.678*0.0 = **0.5649**
* ball:1.883*0.1 + 0.678*0.1 = 0.1883 + 0.0678 = **0.2561**→ logits\_t1 = [0.3088, 0.1356, 0.0151, 0.5649, 0.2561]  
  → target next token at t1 is **“chasing”**

**t2 (x\_f2 = [1.834, 0.389, 0.697, 0])**

* dog: 1.834*0.2 + 0.389*(−0.1) + 0.697\*0.0 = 0.3668 − 0.0389 = **0.3279**
* is: 0 + 0.389*0.2 + 0.697*(−0.2) = 0.0778 − 0.1394 = **−0.0616**
* chasing: 1.834\*(−0.1) + 0.389*0.3 + 0.697*0.1 = −0.1834 + 0.1167 + 0.0697 = **0.0030**
* a: 1.834*0.3 + 0.697*0.2 = 0.5502 + 0.1394 = **0.6896**
* ball: 1.834*0.1 + 0.389*0.1 + 0.697\*(−0.1) = 0.1834 + 0.0389 − 0.0697 = **0.1526**→ logits\_t2 = [0.3279, −0.0616, 0.0030, 0.6896, 0.1526]  
  → target next token at t2 is **“a”**

**softmax → probabilities → per-token loss (cross-entropy)**

Softmax is exp(logit − max) / sum(exp(...)).

Loss at t = −log p(target\_next\_token).

**t0 (target “is”)**

* subtract max (0.4986): diffs = [−0.5533, −0.1216, 0, −0.2979, −0.2432]
* exp = [0.575, 0.885, 1.000, 0.742, 0.784] → sum = 3.986
* p(“is”) = 0.885 / 3.986 = **0.222**
* loss₀ = −ln(0.222) = **1.504**

**t1 (target “chasing”)**

* subtract max (0.5649): diffs = [−0.2561, −0.4293, −0.5498, 0, −0.3088]
* exp = [0.774, 0.651, 0.577, 1.000, 0.734] → sum = 3.736
* p(“chasing”) = 0.577 / 3.736 = **0.154**
* loss₁ = −ln(0.154) = **1.871**

**t2 (target “a”)**

* subtract max (0.6896): diffs = [−0.3617, −0.7512, −0.6866, 0, −0.5370]
* exp = [0.696, 0.472, 0.503, 1.000, 0.584] → sum = 3.255
* p(“a”) = 1.000 / 3.255 = **0.307**
* loss₂ = −ln(0.307) = **1.179**

**final result for this tiny example**

* per-position probs for the true next token:  
  t0: 0.222 (target = “is”)  
  t1: 0.154 (target = “chasing”)  
  t2: 0.307 (target = “a”)
* per-position losses:  
  t0: 1.504  
  t1: 1.871  
  t2: 1.179
* average loss over these 3 steps ≈ **(1.504 + 1.871 + 1.179) / 3 = 1.518**

That’s the full path, end-to-end:

**embeddings → (RMSNorm → attention → RMSNorm + residual → FFN → RMSNorm + residual) → final RMSNorm → logits → softmax → loss.**