

Transformer & LLM Architecture: Attention, RoPE & KV-Cache Math

Scaled dot-product attention, Multi-Query / Grouped-Query Attention (GQA), RoPE, FlashAttention, and VRAM sizing.

EXECUTIVE SUMMARY

Mathematical and systems-level breakdown of modern frontier Large Language Models: exact equations for Scaled Dot-Product Attention, Rotary Position Embeddings (RoPE), SwiGLU feed-forward networks, RMSNorm, and KV-Cache memory bandwidth calculations.

KEY EQUATIONS & INVARIANTS

* Scaled Dot-Product Attention:

LaTeX: $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{top}} + M}{\sqrt{d_k}}\right)V$

Note: Scaling by $1/\sqrt{d_k}$ keeps dot-product variance at 1 so softmax gradients do not vanish.

* KV-Cache Memory Footprint (Bytes per Token):

LaTeX: $\text{KV}_{\text{bytes/token}} = 2 \cdot n_{\text{layers}} \cdot n_{\text{kv_heads}} \cdot d_{\text{head}} \cdot \text{bytes}_{\text{precision}}$

Note: Factor of 2 accounts for Key and Value tensors; GQA reduces $n_{\text{kv_heads}}$ by 4x-8x.

* SwiGLU Feed-Forward Activation:

LaTeX: $\text{SwiGLU}(x) = \text{Swish}_{\beta}(xW_g) \odot xW_u$

Note: Gated linear unit with Swish activation used in Llama-3, Qwen, and Gemma architectures.

1. Multi-Head (MHA) vs. Grouped-Query (GQA) vs. Multi-Head Latent Attention (MLA)

Autoregressive decoding is memory-bandwidth bound because every generated token loads the full model weights and the historical KV-Cache from HBM into SRAM. Grouped-Query Attention shares Key/Value projections across groups of Query heads.

Mechanism | KV Heads | KV-Cache Size | Representative Models

Multi-Head Attention (MHA) | $n_{\text{kv}} = n_{\text{heads}}$ | 1.0x (Baseline) | GPT-3, Llama-1
 Grouped-Query Attention (GQA) | $n_{\text{kv}} = n_{\text{heads}} / g$ (e.g. 8) | $1/g$ (4x - 8x smaller)
 | Llama-3, Mistral, Qwen-2.5
 Multi-Query Attention (MQA) | $n_{\text{kv}} = 1$ | $1 / n_{\text{heads}}$ | PaLM, Falcon
 Multi-Head Latent Attention (MLA) | Low-rank compressed c_{KV} | $\sim 5x - 10x$ smaller | DeepSeek-V2 / V3 / R1

[PYTHON] PyTorch Grouped-Query Attention (GQA) with Causal Masking

```
import math
import torch
import torch.nn.functional as F

def grouped_query_attention(
    q: torch.Tensor, # [B, H_q, T, D]
    k: torch.Tensor, # [B, H_kv, T, D]
    v: torch.Tensor, # [B, H_kv, T, D]
) -> torch.Tensor:
    b, h_q, t, d = q.shape
    h_kv = k.shape[1]
    groups = h_q // h_kv
    if groups > 1:
        k = k.repeat_interleave(groups, dim=1)
        v = v.repeat_interleave(groups, dim=1)
```

```
scores = torch.matmul(q, k.transpose(-2, -1)) / math.sqrt(d)
mask = torch.triu(torch.ones(t, t, device=q.device, dtype=torch.bool), diagonal=1)
scores = scores.masked_fill(mask, float('-inf'))
attn = F.softmax(scores, dim=-1)
return torch.matmul(attn, v)
```

2. LLM Inference & Training VRAM Sizing Rules

For inference in BF16/FP16, allocate $2 \times P$ bytes for weights plus ~20% activation/KV overhead. For full-parameter AdamW mixed-precision training, allocate $16 \times P$ bytes (2B FP16 weights + 2B FP16 gradients + 4B FP32 master weights + 8B FP32 Adam m, v states).

COMMON PRODUCTION & INTERVIEW PITFALLS

- Applying Softmax in FP16 without subtracting the row-wise maximum causes `exp(x)` overflow (`NaN` logits); always upcast softmax accumulation to FP32.
- Forgetting to scale the residual stream projection weights by $1/\sqrt{2}$ N_{layers} at initialization leads to exploding activations in deep transformers.

FREQUENTLY ASKED QUESTIONS

Q: How does FlashAttention reduce memory from $O(N^2)$ to $O(N)$?

A: FlashAttention tiles Q, K, and V blocks into fast GPU on-chip SRAM and uses the online softmax rescaling trick (tracking running max m and normalizer l) so the full $N \times N$ attention matrix is never materialized in HBM.