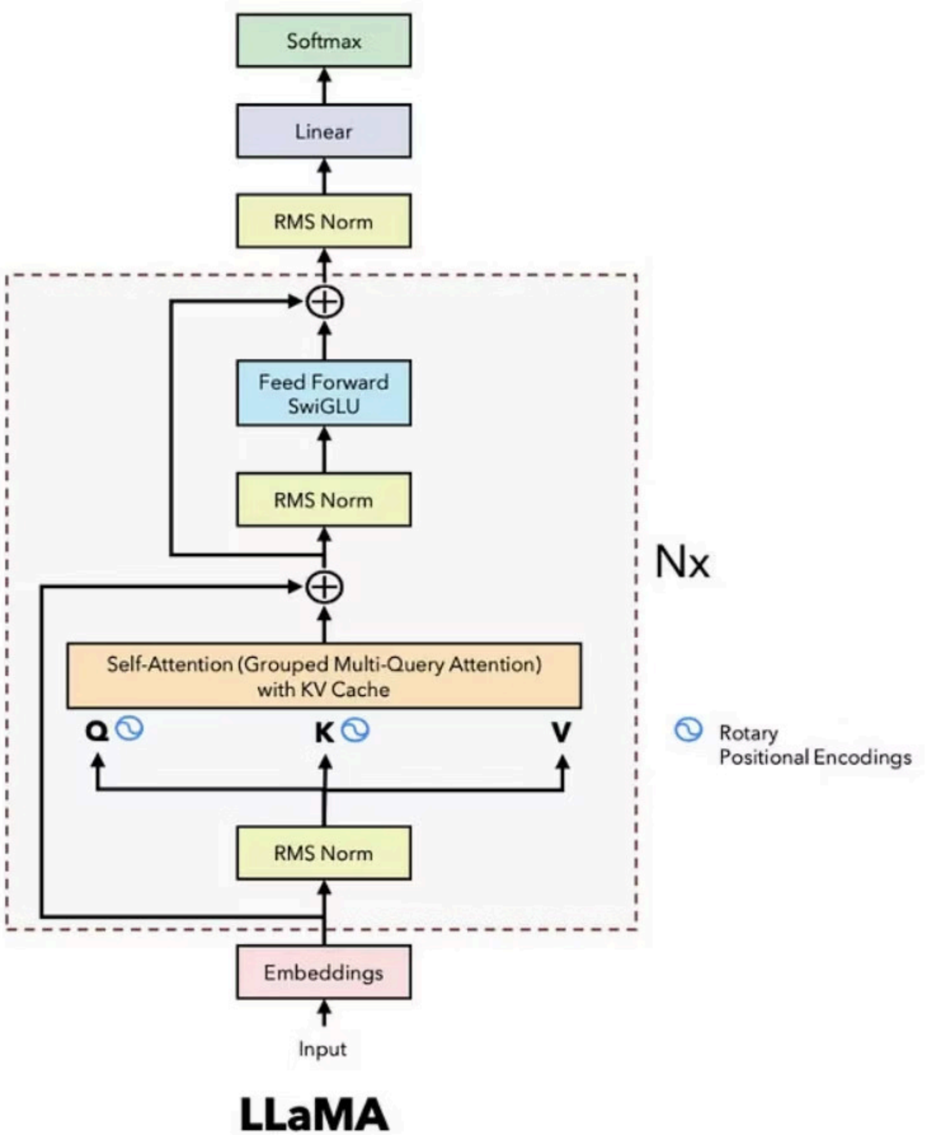
Transformer

Architecture Deep Dive

Core Components & Implementation

Self-Attention, FFN, and Output Layers

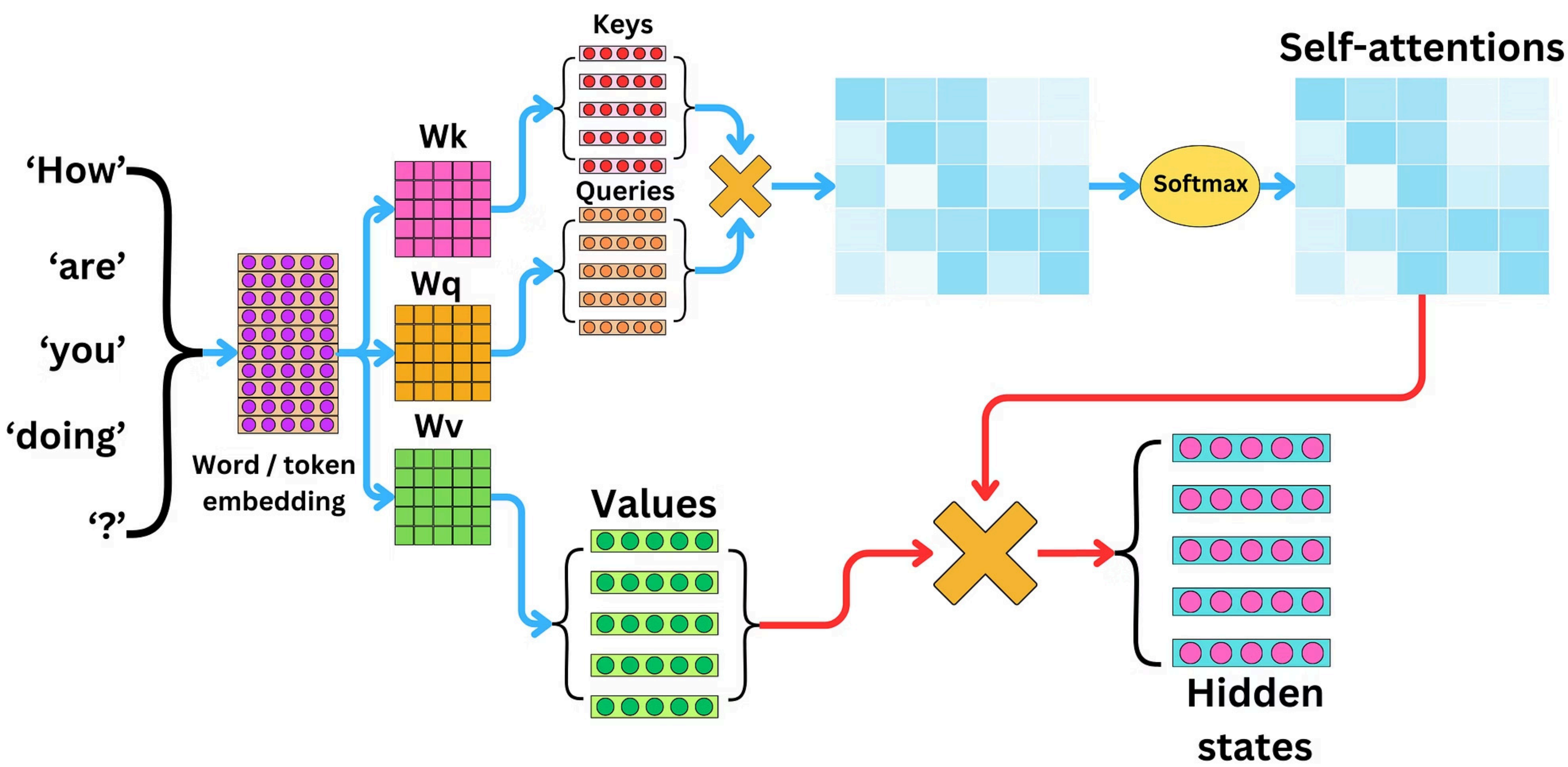
Architecture Overview



Grouped Multi Query Attention (GQA)

Contextualizing tokens through attention mechanisms

Normal Self Attention - Transformer



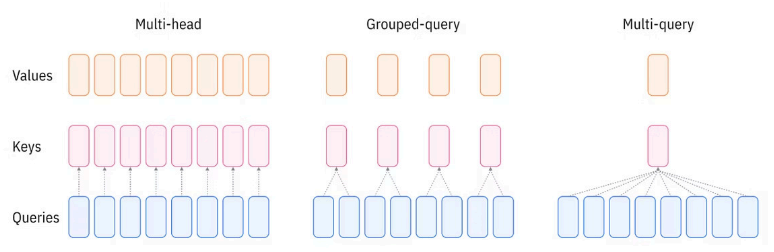
Q, K, V Projections

Linear transformations create query, key, and value representations:

```
Q = X (W_Q)        # Query projection
K = X (W_K)        # Key projection
V = X (W_V)        # Value projection
```

Grouped-Query Attention: 32 Q heads, 4 KV heads

KV heads are **repeated** to match Q heads for efficiency



Attention Computation

Core Attention Formula

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

RoPE Integration: Applied to Q and K before attention computation

Output Projection: Final linear layer combines attention outputs

```
Initializing SelfAttention:
Model dimension: 4096
Number of query heads: 32
Number of key/value heads: 4
```

Output of GQA

```
SelfAttention Forward Pass:
Input shape: torch.Size([1, 10, 4096])
After linear projections - Q: torch.Size([1, 10, 4096]), K: torch.Size([1, 10, 512]), V: torch.Size([1, 10, 512])
After reshaping - Q: torch.Size([1, 10, 32, 128]), K: torch.Size([1, 10, 4, 128]), V: torch.Size([1, 10, 4, 128])
```

```
Input tensor shape: torch.Size([1, 10, 4, 128])
Frequencies tensor shape: torch.Size([10, 64])
Input as complex numbers shape: torch.Size([1, 10, 4, 64])
Expanded frequencies shape: torch.Size([1, 10, 1, 64])
After RoPE - Q: torch.Size([1, 10, 32, 128]), K: torch.Size([1, 10, 4, 128]), V: torch.Size([1, 10, 4, 128])
Cache shapes - K: torch.Size([1, 10, 4, 128]), V: torch.Size([1, 10, 4, 128])
```

```
Repeating KV heads:
Input shape: torch.Size([1, 10, 4, 128])
Output shape after repeating: torch.Size([1, 10, 32, 128])
After transpose - Q: torch.Size([1, 32, 10, 128]), K: torch.Size([1, 32, 10, 128]), V: torch.Size([1, 32, 10, 128])
Attention scores shape: torch.Size([1, 32, 10, 10])
After attention shape: torch.Size([1, 32, 10, 128])
Before final projection: torch.Size([1, 10, 4096])
Final output shape: torch.Size([1, 10, 4096])
After attention shape: torch.Size([1, 10, 4096])
After first residual connection: torch.Size([1, 10, 4096])
```

Self - Attention:

- **Mechanism**: Each token attends to all previous tokens using **dot-product attention**.
- **Computation**: Requires forming full Q, K, V matrices for **all tokens**, followed by dot products between Q and K for each position.
- **Drawback**: Computationally expensive — especially during inference — since we need to recompute attention over the entire sequence for each new token.

Attention With KV - Cache:

- **Optimization**: Store previously computed K and V values in a **KV-Cache**.
- **How it works**:
  - Only compute Q for the **new input token**.
  - Reuse stored K and V from earlier tokens to compute attention.
- **Benefit**: Drastically reduces computation — only **one forward pass per token**.
- **Trade-off**: Introduces a **memory bandwidth bottleneck**, since K/V vectors for long sequences must be **fetched repeatedly** from memory (which is slower than GPU compute).

Grouped Multi Query Attention

- **Design**: Use **more query heads** (e.g., 32) than key/value heads (e.g., 4).
- **Implementation**: K and V are **shared across multiple Q heads**, reducing the memory and compute load.
- **Benefit**:
  - Lower memory usage than full multi-head attention.
  - Faster inference, especially with KV caching.
- **Downside**:
  - Slight degradation in attention granularity.
  - But overall performance and scalability improve — especially in large models.

Feed-Forward Network

Non-linear transformations for richer representations

SwiGLU Activation

Three linear layers: W_1, W_2, W_3

Up_projection = (2/3)(4*d_model)

$$SwiGLU = W_2(SiLu(W_1x) \odot (W_3x))$$

```
Initializing FeedForward Network:
Input dimension: 4096
Initial hidden dimension: 10922
Final hidden dimension (rounded): 11008
W1 shape: 4096 -> 11008
W2 shape: 11008 -> 4096
W3 shape: 4096 -> 11008
```

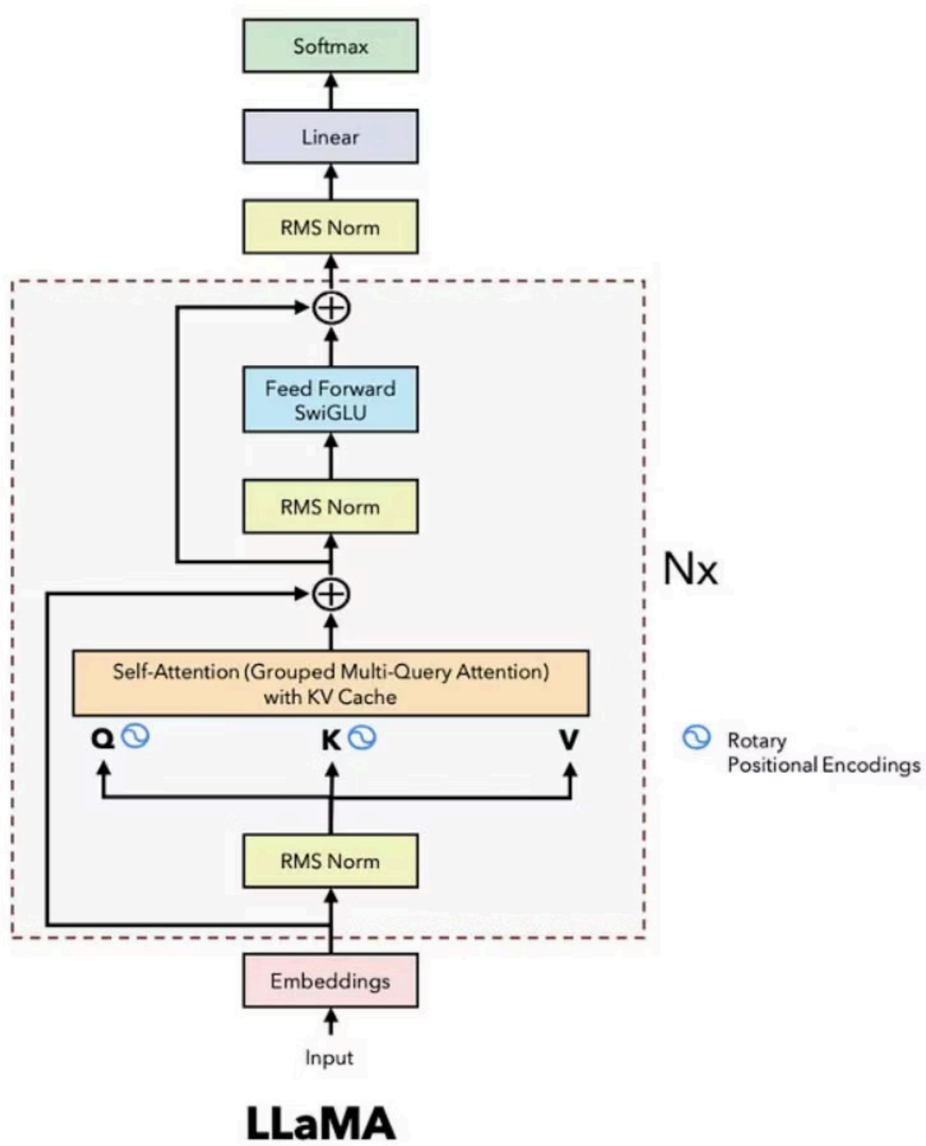
Dimensional Flow:

- Input: d_model dimensions
- Up-projection: 2.66x expansion (W_1, W_3)
- Down-projection: Back to d_model (W_2)

Purpose: Enables complex, non-linear feature interactions

Encoder Block

Stacked 32 times to build the full transformer



Logits to Probabilities

Linear transformation: Hidden state → vocabulary logits

$$P(token) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Weight Tying: Shared weights between token embedding and output projection

Training Loss: Cross-entropy between predicted and target distributions

Key Architectural Principles

32:4

Grouped-Query Attention

Efficient attention with fewer KV heads

2.66x

FFN Expansion

Dimensional up-projection for capacity

32

Block Stacking

Repeated encoder blocks for depth

> **Each component serves a distinct purpose:** attention for context, FFN for transformation, residual connections for gradient flow, and normalization for stability.