Table of Contents

1. Advanced Report on DeepSeek R1 Architecture
    1. Inside One Transformer Block (Mixture of Experts)
        - RMSNorm Layer
        - Multi-Head Latent Attention (MLA)
        - Residual Connection
        - Feed-Forward Network (FFN) or Mixture-of-Experts (MoE)
    2. Overall DeepSeek R1 Architecture
        - Router Network
        - Mixture of Experts (MoE)
        - Expert Aggregator
        - KV Cache
        - Latent Vector Compression
        - Attention Processing → Model Output
    3. Putting It All Together
2. Introduction to Qwen Large Language Model
    1. Overview
    2. Why Qwen Matters
    3. Embedding Layer
    4. Rotary Positional Embedding (RoPE)
    5. Attention Mechanism
        - LogN-Scaling
        - Window Attention
    6. Feed-Forward Network (SwiGLU)
    7. RMSNorm
    8. Bias in QKV Projections
    9. Dynamic NTK-Aware Interpolation
    10. Output Projection
    11. Summary

Advanced Report on DeepSeek R1 Architecture

Below is a multi-part technical report that describes **DeepSeek R1** in detail. The first section focuses on the internal structure of a **single Transformer Block** as employed in the Mixture-of-Experts (MoE) mechanism. The second section gives a **layer-by-layer breakdown** of the entire **DeepSeek R1 architecture**, showing how all components (Router, Experts, Multi-Head Latent Attention, etc.) come together. Numerical “dry-run” mini-examples are provided throughout so you can see how the mathematics might unfold with small input sizes.

---

1. Inside One Transformer Block (Mixture of Experts)

Each Transformer block in the Mixture of Experts arrangement has the following sublayers:

1. **RMSNorm**
2. **Attention (Multi-Head Latent Attention, MLA)**
3. **RMSNorm**
4. **Feed-Forward Network (FFN)** or **MoE** feed-forward (DeepSeekMoE)

All of these include residual connections around the attention and feed-forward sublayers. Below is the rationale and math for each piece.

1.1 RMSNorm Layer

Purpose

- Normalizes the hidden states using Root Mean Square (RMS) normalization rather than LayerNorm.
- Maintains numerical stability and helps with training dynamics.

RMSNorm Formula

For a hidden vector *h* ∈ $\mathbb{R}^D$, RMSNorm can be written as:

$$\text{RMSNorm}(\mathbf{h}) \;=\; \frac{\mathbf{h}}{\sqrt{\frac{1}{D}\sum_{i=1}^{D} h_i^2 + \epsilon}} \;\odot\; \boldsymbol{\gamma}$$

where:

- D is the dimension of *h*.
- ε is a small constant to avoid division by zero.
- ⊙ denotes element-wise multiplication.
- γ ∈ $\mathbb{R}^D$ is a trainable scale parameter.

Mini-Example

Suppose D=4 and you have one token's hidden state:

$$\mathbf{h} = [2,\, -2,\, 1,\, 1].$$

Then

$$\frac{1}{D}\sum_{i=1}^{4} h_i^2 = \frac{1}{4}(2^2 + (-2)^2 + 1^2 + 1^2) = \frac{1}{4}(4 + 4 + 1 + 1) = 2.5.$$

$$\sqrt{2.5 + \epsilon} \approx \sqrt{2.5} = 1.58 \quad (\text{assuming } \epsilon \text{ is very small}).$$

If γ = [1, 1, 1, 1] (for simplicity), then

$$\text{RMSNorm}(\mathbf{h}) = \frac{[\,2,\, -2,\, 1,\, 1\,]}{1.58} \approx [\,1.27,\, -1.27,\, 0.63,\, 0.63\,].$$

1.2 Multi-Head Latent Attention (MLA)

Purpose

- Allows each token's representation to attend to other tokens' representations via multiple "heads."
- Uses query (Q), key (K), and value (V) projections.
- The "latent attention" extension can incorporate specialized latent vectors or gating (like RoPE, positional embeddings, or other advanced transformations).

Mathematics

In a standard multi-head attention, each token embedding h_t ∈ $\mathbb{R}^D$ is projected into:

$$\mathbf{q}_t = W^Q\,\mathbf{h}_t, \quad \mathbf{k}_t = W^K\,\mathbf{h}_t, \quad \mathbf{v}_t = W^V\,\mathbf{h}_t,$$

where W^Q, W^K, W^V ∈ $\mathbb{R}^{D\times D}$. Then attention for a single head is:

$$\mathrm{Attn}(\mathbf{q}_t, \{\mathbf{k}_j\}, \{\mathbf{v}_j\}) = \sum_{j=1}^{T} \mathrm{softmax}\Big(\frac{\mathbf{q}_t \cdot \mathbf{k}_j}{\sqrt{d_k}}\Big)\, \mathbf{v}_j,$$

where d_k = D / (number of heads). In multi-head attention, we do this for each head, then concatenate:

$$\mathbf{o}_t = \big[\mathrm{Attn}^1; \mathrm{Attn}^2; \ldots; \mathrm{Attn}^H\big]\, W^O$$

(H heads, plus a final projection W^O).

In **Multi-Head Latent Attention**, we may also incorporate:

- **RoPE** (Rotary Positional Embeddings) to transform K_t and Q_t.
- **Latent vectors** c_t^Q, c_t^K (extra "latent" queries/keys) that can add further context.

The net result is an updated hidden state u_t, which merges standard attention with specialized latent transformations.

Mini-Example

A toy multi-head attention with:

- D=4.
- 2 heads (H=2) ⇒ d_k=2.
- 2 tokens (t ∈ {1, 2}).

1. **Projecting inputs**
   Let W^Q, W^K, W^V each be 4×4 identity (for demonstration). For token 1, h_1=[1,2,3,4]. Then

   $$\mathbf{q}_1 = [1,2,3,4], \quad \mathbf{k}_1 = [1,2,3,4], \quad \mathbf{v}_1 = [1,2,3,4].$$

   For token 2, h_2=[2,0,1,0]. Then

   $$\mathbf{q}_2 = [2,0,1,0], \quad \mathbf{k}_2 = [2,0,1,0], \quad \mathbf{v}_2 = [2,0,1,0].$$

2. **Compute attention scores** (per head).
   Head 1 might use the first half of each vector, etc. Head 2 uses the second half.
3. **Softmax** over token dimension to gather values from each token.

Eventually, we get u_t for each token after merging heads (plus any latent transformations).

1.3 Residual Connection

After the attention block, we add the original hidden state back:

$$\mathbf{h}_t^{(\text{after attn})} = \mathbf{h}_t + \mathbf{u}_t.$$

This helps gradient flow and stabilizes training.

1.4 Feed-Forward Network (FFN) or Mixture-of-Experts (MoE)

Purpose

- Applies a position-wise nonlinear transformation to each token's embedding.
- In a conventional Transformer, this is typically a 2-layer MLP.
- In **DeepSeekMoE**, it becomes a mixture-of-experts block, where the router chooses which feed-forward "expert(s)" to apply.

Conventional FFN

$$\mathbf{z}_t = \sigma(W_1\,\mathbf{h}_t + \mathbf{b}_1), \quad \mathbf{f}_t = W_2\,\mathbf{z}_t + \mathbf{b}_2,$$

where σ is a nonlinear activation (e.g. GELU).

MoE FFN

Instead of one FFN, we have multiple "experts" (FFN_1, FFN_2, …, FFN_{N_r}). A router produces gating weights g_t to select top-k experts:

$$\mathbf{f}_t = \sum_{r \in \text{Top-}k} g_{t,r}\,\text{FFN}_r(\mathbf{h}_t).$$

DeepSeek uses a large set of experts but activates only a small subset per token.

Mini-Example

- 3 experts E_1, E_2, E_3.
- Gating vector g_t = [0.7, 0.3, 0.0].
- Output = $0.7 \cdot E_1(h_t) + 0.3 \cdot E_2(h_t)$.

Residual Connection

We again add the output of the FFN/MoE back to h_t:

$$\mathbf{h}'_t = \mathbf{h}_t^{(\text{after attn})} + \mathbf{f}_t.$$

---

2. Overall DeepSeek R1 Architecture

Below is a broader schematic:

1. **Input Data** flows into the **Router Network**.
2. The router decides which experts in the **Mixture-of-Experts (MoE)** block activate.
3. Only the active experts handle each token's forward pass.
4. The outputs from experts are merged in the **Expert Aggregator**.
5. Then **Multi-Head Latent Attention** with a **KV cache** (for up to 128k tokens) plus latent compression.
6. The resulting representations pass to **Attention Processing** and final heads for the model output.

Below is a layer-by-layer breakdown.

2.1 Router Network

Purpose

- Takes token representations as input.
- Computes a routing distribution so each token is sent to a small subset of experts.
- Greatly reduces compute cost because only a fraction of the total parameters is active per token.

Routing Math

If $h_t \in \mathbb{R}^D$, the router can compute:

$$\mathbf{r}_t = W^{(\text{router})}\,\mathbf{h}_t + \mathbf{b}^{(\text{router})} \;\in\; \mathbb{R}^{N_r}.$$

We convert r_t into gating vector $g_t \in \mathbb{R}^{N_r}$. Often with **softmax top-k**:

1. p_t = softmax(r_t).
2. Sort p_t to find top-k.

3. Zero out all but top-k.

Example

- N_r=3.
- r_t=[3.4, 1.1, 2.2].
- softmax(r_t)≈[0.70, 0.06, 0.24].
- If k=2, g_t=[0.70, 0.0, 0.24].
- Experts 1 and 3 are active.

2.2 Mixture of Experts (MoE)

Purpose

- Houses many feed-forward sub-networks (experts).
- Each expert can specialize in different parts of representation space.
- Only top-k experts run for each token; all are trained collectively.

Inside each “MoE block,” we have:

$$\mathbf{f}_t = \sum_{r \in \text{Top-}k} g_{t,r}\, \mathrm{FFN}_r(\mathbf{h}_t).$$

2.3 Expert Aggregator

Purpose

- Collects outputs of whichever experts were active and merges them back into a single hidden vector per token.
- May include load-balancing constraints.

After the aggregator, you have the next hidden layer representation h_t'.

2.4 KV Cache

Purpose

- Stores key–value projections for previously processed tokens, avoiding redundant computations.
- Enables context lengths up to 128k tokens.

Example

When processing a new token at position t:

$$\mathbf{k}_t = W^K\,\mathbf{h}_t, \quad \mathbf{v}_t = W^V\,\mathbf{h}_t.$$

We append k_t, v_t to the cache for subsequent attention:

$$\mathrm{Attn}(\mathbf{q}_t,\, [\mathbf{k}_1, \ldots, \mathbf{k}_t],\, [\mathbf{v}_1, \ldots, \mathbf{v}_t]).$$

2.5 Latent Vector Compression

Purpose

- Compresses or projects the keys/values into a lower-dimensional space.
- Maintains representational capacity while controlling memory usage for very long contexts.

Possible Math

$$\tilde{\mathbf{k}}_t = W_k^{(\text{compress})}\,\mathbf{k}_t, \quad \tilde{\mathbf{v}}_t = W_v^{(\text{compress})}\,\mathbf{v}_t,$$

reducing dimension from D to D′.

2.6 Attention Processing → Model Output

After the MoE, caching, and compression steps, further multi-head attention blocks and final layers produce the **model output** (e.g. next-token logits). In practice:

1. Tokens go through the MoE block (router + experts).
2. Expert aggregator merges outputs.
3. Multi-Head Latent Attention uses the KV cache.
4. Then final heads or further Transformer blocks produce the final layer.

A single Transformer block in **DeepSeek R1** looks like:

$$\mathbf{h}_t \xrightarrow{\text{RMSNorm}} (\text{Multi-Head Latent Attention}) \xrightarrow{+\mathbf{h}_t} \xrightarrow{\text{RMSNorm}} (\text{MoE FFN}) \xrightarrow{+\text{previous}} \mathbf{h}'_t.$$

This repeats L times to form a deep stack.

1.3 Putting It All Together

DeepSeek R1 is a large-scale Transformer-type model that combines:

1. **Mixture of Experts** for the feed-forward portion of each Transformer block, drastically increasing total parameters while limiting active parameters per token.
2. **Multi-Head Latent Attention** plus **KV caching** (up to 128k tokens) for extremely long sequences.
3. **Router Network** to dynamically select experts (top-k gating).
4. **RMSNorm** layers for stable training.
5. **Residual connections** around both attention and feed-forward sublayers.

The overall flow is:

1. **Input Data** → (Token Embeddings + Positional Embeddings)
2. **Router Network** → pick top-k experts
3. **Active Experts (MoE)** → feed-forward sub-networks
4. **Expert Aggregator** → merges the outputs
5. **KV Cache** → store k, v states for long contexts
6. **Latent Vector Compression** → keep memory usage feasible
7. **Attention** → standard multi-head or latent extension
8. **Attention Processing** → final or subsequent Transformer blocks
9. **Model Output** → e.g. next-token probabilities

At each Transformer block:

$$\mathbf{h}_t \;\to\; \text{RMSNorm} \;\to\; \text{MultiHeadLatentAttention} \;\to\; \mathbf{h}_t + \ldots \;\to\; \text{RMSNorm} \;\to\; \text{MoE FFN} \;\to\; \mathbf{h}_t + \ldots \;=\; \mathbf{h}'_t$$

Introduction to Qwen Large Language Model

Overview

Qwen is a state-of-the-art large language model (LLM) series developed by Alibaba, designed to handle diverse tasks ranging from natural language understanding to specialized domains like coding and mathematics. The model series includes base pretrained models (Qwen), aligned chat models (Qwen-Chat), and domain-specific variants (Code-Qwen, Math-Qwen). Key innovations in its architecture enable efficient training, long-context processing, and robust performance across benchmarks.

Why Qwen Matters

1. **Versatility**: Supports multilingual tasks (Chinese/English), tool usage, code interpretation, and mathematical reasoning.
2. **Efficiency**: Implements optimizations like NTK-aware interpolation and windowed attention to reduce computational costs.
3. **Scalability**: Available in 1.8B, 7B, and 14B parameter sizes, balancing performance and resource constraints.
4. **Innovation**: Integrates advanced techniques like Rotary Positional Embeddings (RoPE) and SwiGLU activation, outperforming similarly sized open-source models.

---

1. Embedding Layer

Why Needed

Transformers cannot process raw token IDs directly. Discrete tokens must be mapped to continuous vectors to enable mathematical operations and capture semantic relationships.

How It Happens

- Uses an embedding matrix $E \in \mathbb{R}^{V \times d}$ where V = vocabulary size, d = hidden dimension.
- Each token ID i is converted to a vector via Embedding(i) = E[i].

Mathematics

$$\text{Input: Token ID } i \quad \Rightarrow \quad \text{Output: } E[i] \in \mathbb{R}^d$$

Example

Token IDs: [5, 3]

Embedding Matrix (hypothetical slice):

```
E = [
  [ 0.1  -0.2   0.4 ]
   ...
  [-0.3   0.5   0.7 ]
   ...
]
```

Output:

```
[
  [ 0.1  -0.2   0.4 ],
  [-0.3   0.5   0.7 ]
]
```

2. Rotary Positional Embedding (RoPE)

Why Needed

Transformers are permutation-invariant. Positional encoding is required to inject sequential order information.

How It Happens

- Applies rotation matrices to query/key vectors based on their positions.
- Uses complex numbers to encode relative positions into attention scores.

Mathematics

For position m, angle $\theta_j = 10000^{-2j/d}$:

$$q'_m = q_m \odot e^{i\,m\,\theta}, \quad k'_n = k_n \odot e^{i\,n\,\theta}$$

Dot product becomes:

$$q'_m \cdot k'_n = \mathrm{Re}\left[\sum_j q_{m,j} k_{n,j} e^{i(m-n)\theta_j}\right]$$

Example

Query: q = [1.0, 2.0], **Key**: k = [3.0, 4.0], **Positions**: m=1, n=2, θ = π/4

Rotated Vectors:

```
q' ≈ [-0.707,  2.121],   k' ≈ [-4.0,  3.0]
```

Dot Product: 9.191

---

3. Attention Mechanism

a. LogN-Scaling

Why Needed

Prevents attention scores from growing excessively with sequence length.

How It Happens

Scales dot products by log N, where N = context length.

Mathematics:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}} \cdot \log N\right)V$$

Example

For Q = [1, 2], K = [3, 4], N=4, d=2:

$$\mathrm{Scaled\ Score} = \frac{(1 \cdot 3 + 2 \cdot 4)}{\sqrt{2}} \cdot \log 4 \approx 16.97$$

b. Window Attention

Why Needed

Reduces computational complexity for long sequences.

How It Happens

Restricts attention to a fixed window around each token.

Mathematics:

$$\text{Attention}(Q_i, K_{i-w:i+w}, V_{i-w:i+w})$$

where w = window size (e.g., 512).

Example

For token i=50, window size w=2:
Attend to tokens 48, 49, 50, 51, 52.

---

4. Feed-Forward Network (SwiGLU)

Why Needed

Standard feed-forward layers lack dynamic gating. SwiGLU improves feature interaction.

How It Happens

- Combines Swish activation (Swish(x) = x · σ(βx)) with gating.
- Uses element-wise multiplication for conditional computation.

Mathematics:

$$\text{SwiGLU}(x) = \text{Swish}(xW) \odot (xV)$$

Example

Input: x = [2.0, -1.0], W = [1, 0], V = [0.5, 0], β=1:

$$\text{Swish}(xW) = [2\sigma(2), -\sigma(-1)] \approx [1.76, -0.27]$$

$$\text{Output} = [1.76 \cdot 1.0, -0.27 \cdot 0.5] = [1.76, -0.135]$$

---

5. RMSNorm

Why Needed

Standard LayerNorm introduces centering, which can be detrimental in some cases.

How It Happens

- Normalizes by root mean square (RMS) without subtracting the mean.
- Computes x′ = x / √(mean(x²) + ε).

Example

Input: x = [1.0, 2.0, 3.0], ε=1e-6:

$$\text{RMS} = \sqrt{\frac{1^2 + 2^2 + 3^2}{3}} \approx 2.16 \quad \Rightarrow \quad x' = [0.46, 0.92, 1.38]$$

---

6. Bias in QKV Projections

Why Needed

Allows fine-grained control over initial attention computations.

How It Happens

Adds learnable biases to query/key/value projections:

$$Q = xW_Q + b_Q, \quad K = xW_K + b_K, \quad V = xW_V + b_V$$

Example

Input: x = [0.5, -0.3], W_Q = [1, -1], b_Q = [0.1, 0.2]:

```
Q = [ 0.5*1 + (-0.3)*(-1) + 0.1,  0.5*(-1) + (-0.3)*0 + 0.2 ]
Q ≈ [0.9,  -0.3]
```

---

7. Dynamic NTK-Aware Interpolation

Why Needed

Extends context length without retraining by preserving high-frequency positional information.

How It Happens

- Adjusts RoPE base frequency θ_j dynamically:

$$\theta'_j = \theta_j \cdot (1 + \gamma \log N)$$

- Uses smaller windows in lower layers (e.g., 512 tokens) and larger windows in higher layers (e.g., 2048 tokens).

Example

Original RoPE: θ_j = 0.01
Extended Context (N=8192):

$$\theta'_j = 0.01 \cdot (1 + 0.1 \cdot \log 8192) \approx 0.018$$

---

8. Output Projection

Why Needed

Converts final hidden states into vocabulary-space logits for token prediction.

How It Happens

Applies a linear transformation:

$$\text{logits} = hW_{\text{out}} + b_{\text{out}}$$

where $W_{out} \in \mathbb{R}^{d \times V}$.

Example

Input: h = [0.8, -0.5], W_{out} = [1, -1], b_{out} = [0.1, 0.3]:

```
logits = [ 0.8*1 + (-0.5)*(-1) + 0.1,   0.8*(-1) + (-0.5)*2 + 0.3 ]
       = [ 1.4,  -1.5 ]
```

Summary

Qwen's architecture integrates **rotary positional embeddings** for sequence awareness, **SwiGLU** for dynamic gating, **RMSNorm** for stable training, and **NTK-aware interpolation** for efficient long-context processing. Each component addresses specific limitations of vanilla Transformers while maintaining computational efficiency.