

AI

AI VIET NAM
@aivietnam.edu.vn

Special Topic

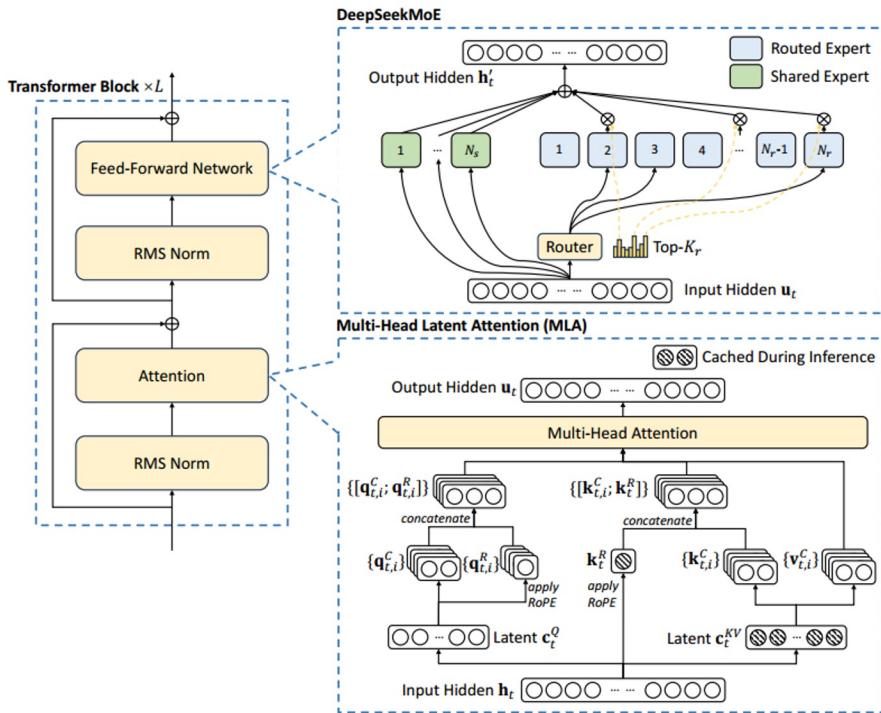


TA Hùng An

Nội dung

1. Giới thiệu DeepSeekV2
2. DeepSeekV3 - Bước đệm hoàn hảo
3. DeepSeekR1 - Công nghệ đột phá
4. Cơ hội áp dụng DeepSeekR1

1 - Giới thiệu DeepSeekV2



Kiến trúc mạng DeepSeekV2

Nhiều khối Transformer Block liên tiếp với một số cải tiến:

- DeepSeekMoE thay cho FFN thông thường
- RMS Norm thay cho Layer Norm
- Multi-Head Latent Attention thay cho Multi-Head Attention thông thường

Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

1 - Giới thiệu DeepSeekV2

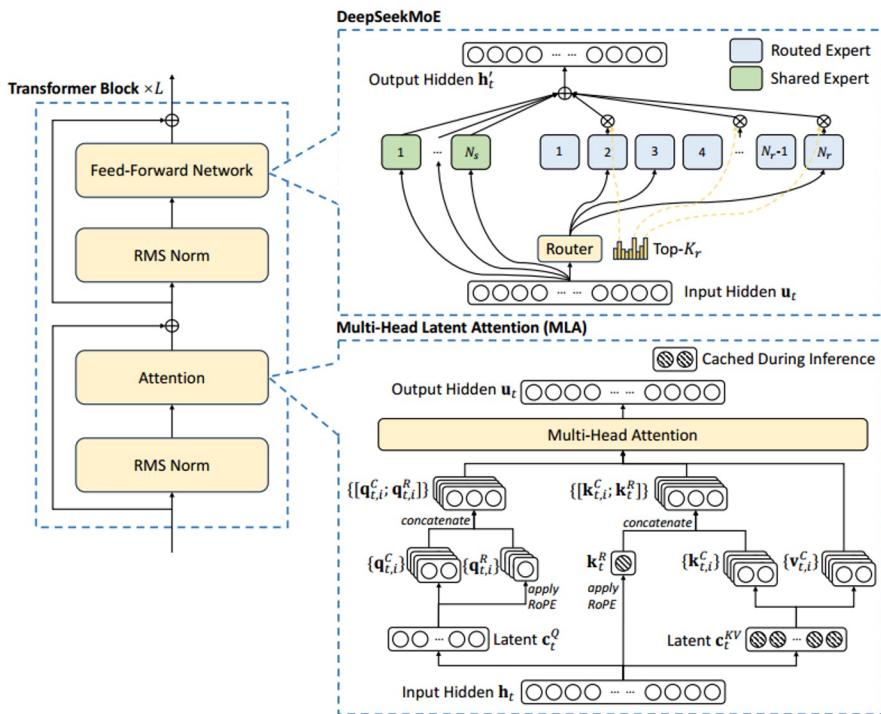


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

RMS Norm thay cho Layer Norm

1. So Sánh về mặt công thức

- Layer Norm trên vector $\mathbf{x} \in \mathbb{R}^d$ thường được viết:

$$\text{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta,$$

trong đó:

- $\mu = \frac{1}{d} \sum_{i=1}^d x_i$ là trung bình của các thành phần.
- $\sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2$ là phương sai.
- ϵ đảm bảo ổn định số học (tránh chia cho 0).
- γ, β là tham số học được (scale và shift).

- RMS Norm chỉ thực hiện chuẩn hoá theo norm (không “trung bình hoá”), thường được viết:

$$\text{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \epsilon}} \cdot \gamma,$$

trong đó không có bước trừ μ . Thay vì lấy trung bình và độ lệch chuẩn, RMS Norm chỉ dùng root mean square (căn bậc hai của trung bình bình phương).

Ý nghĩa cốt lõi:

- Layer Norm đưa dữ liệu về trung bình 0, phương sai 1 (rồi nhân/buộc thêm γ, β).
- RMS Norm chỉ đưa độ lớn (norm) của vector về 1 (sau đó nhân thêm γ), không nhất thiết đưa trung bình về 0.

1 - Giới thiệu DeepSeekV2

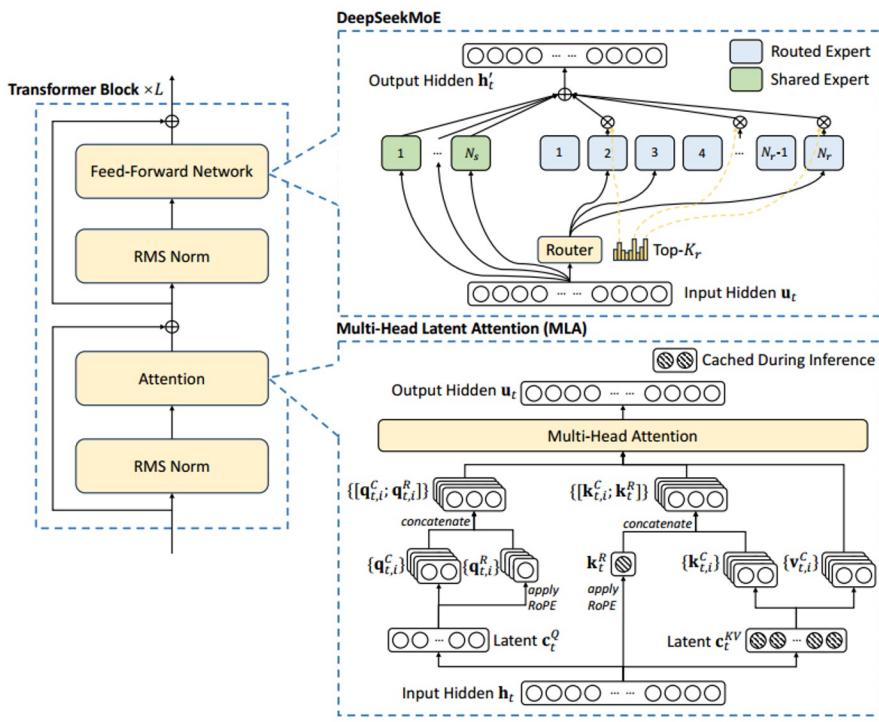


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

RMS Norm thay cho Layer Norm

2. So Sánh về mặt kiến trúc và huấn luyện

1. Giảm phép tính "trữ trung bình"

- RMS Norm bỏ qua bước trừ μ . Điều này giúp:
 - Giảm một số phép toán, tăng tốc đôi chút khi huấn luyện mô hình rất lớn (mặc dù mức độ cải thiện hiệu năng có thể không quá lớn).
 - Loại bỏ một nguồn sai số số học (floating point error) khi tính trung bình, nhất là với float16/bfloat16.

2. Độ ổn định huấn luyện với mô hình lớn

- Một số công trình (chẳng hạn như GPT-NeoX, Llama, v.v.) nhận thấy RMS Norm có thể đem lại độ ổn định cao hơn khi huấn luyện mạng lớn (Large Language Models).
- Trong Layer Norm, việc ép trung bình về 0 có thể đôi lúc làm "trôi" kích hoạt (activation drift), nhất là trong giai đoạn khởi tạo hoặc khi gradient rất lớn. RMS Norm có ít nguy cơ ấy hơn.

3. Không thay đổi trung bình

- Vì không trừ μ , RMS Norm giữ nguyên thiên hướng (bias) tổng thể của vector x . Đôi khi trong các cơ chế attention phức tạp, việc "bảo toàn" một phần thông tin tuyệt đối (thay vì đưa về 0) giúp mô hình phân tách tín hiệu tốt hơn.

4. Ít tham số hơn Layer Norm (nếu bỏ β)

- Thông thường, RMS Norm chỉ có tham số γ , không có β . Dù ít hơn một chút, nhưng đối với mạng rất lớn thì bất kì việc tiết kiệm nào cũng có lợi phần nào.

5. Triển khai (implementation) đơn giản

- Công thức RMS Norm khá gọn: chỉ cần tính $\sqrt{\frac{1}{d} \sum x_i^2 + \epsilon}$ để chia. Dễ vector hoá GPU/TPU.

1 - Giới thiệu DeepSeekV2

Chứng minh Layer Norm luôn ra trung bình 0 và phương sai 1

Cho một vector kích hoạt $\mathbf{x} \in \mathbb{R}^d$. LayerNorm (bản cơ bản, chưa nhân γ hay cộng β) thường được định nghĩa:

$$\mathbf{y} = \text{LN}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}},$$

với

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i, \quad \sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2, \quad \varepsilon \text{ là số rất nhỏ để tránh chia } 0.$$

(a) Trung bình bằng 0

Xét $\mathbf{y} = (y_1, \dots, y_d)$. Ta có:

$$y_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}.$$

Trung bình các thành phần của \mathbf{y} là:

$$\frac{1}{d} \sum_{i=1}^d y_i = \frac{1}{d} \sum_{i=1}^d \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} = \frac{1}{\sqrt{\sigma^2 + \varepsilon}} \cdot \left(\frac{1}{d} \sum_{i=1}^d (x_i - \mu) \right).$$

Nhưng $\frac{1}{d} \sum_{i=1}^d (x_i - \mu) = 0$ (vì μ chính là trung bình của \mathbf{x}).

Vậy:

$$\frac{1}{d} \sum_{i=1}^d y_i = 0.$$

(b) Phương sai bằng 1

Phương sai của \mathbf{y} (nếu lấy theo định nghĩa "trung bình bình phương lệch") là:

$$\frac{1}{d} \sum_{i=1}^d (y_i - \bar{y})^2 \quad \text{với} \quad \bar{y} = \frac{1}{d} \sum_{i=1}^d y_i = 0.$$

Nên:

$$\frac{1}{d} \sum_{i=1}^d y_i^2 = \frac{1}{d} \sum_{i=1}^d \left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} \right)^2 = \frac{1}{\sigma^2 + \varepsilon} \cdot \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2.$$

Theo định nghĩa, $\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2 = \sigma^2$. Vậy:

$$\frac{1}{d} \sum_{i=1}^d y_i^2 = \frac{\sigma^2}{\sigma^2 + \varepsilon}.$$

Trong trường hợp ε rất nhỏ, đại lượng này xấp xỉ 1.

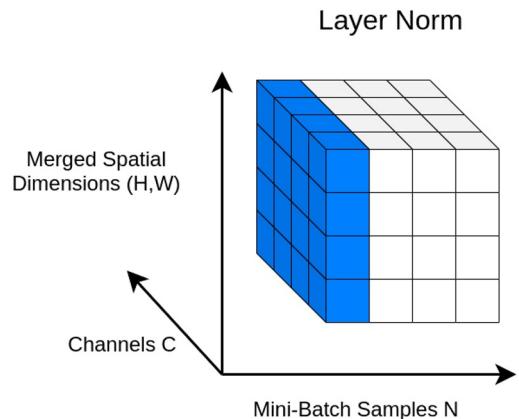
(Bản chất, nếu tính "chuẩn" thì $\text{Var}(\mathbf{y}) \approx 1$ vì chúng ta chia chính xác cho σ^2 thay vì $\sigma^2 + \varepsilon$. ε chỉ dùng để tránh chia 0.)

Kết luận: Khi bỏ qua γ, β , LayerNorm đưa vector \mathbf{x} về trung bình 0 và phương sai xấp xỉ 1.

1 - Giới thiệu DeepSeekV2

Hiện tượng “activation drift” của LayerNorm

Activation drift - chỉ việc phân bố các kích hoạt có thể bị dịch chuyển hoặc “trôi” trong quá trình học do LayerNorm luôn ép dữ liệu về trung bình 0, phương sai 1 ở mỗi lớp.



1. Cốt lõi:

LayerNorm luôn trừ trung bình μ của \mathbf{x} . Điều đó có nghĩa bất kì độ lệch (offset) tổng thể nào mà mô hình sinh ra đều bị khử.

- Hệ quả: mạng không thể “giữ lại” offset toàn cục trong lớp đó; thay vào đó, mọi offset phải học cách “chèn” qua β (trong LayerNorm) hoặc các bias ở lớp sau.

2. Quá trình học có thể dẫn đến bù trừ:

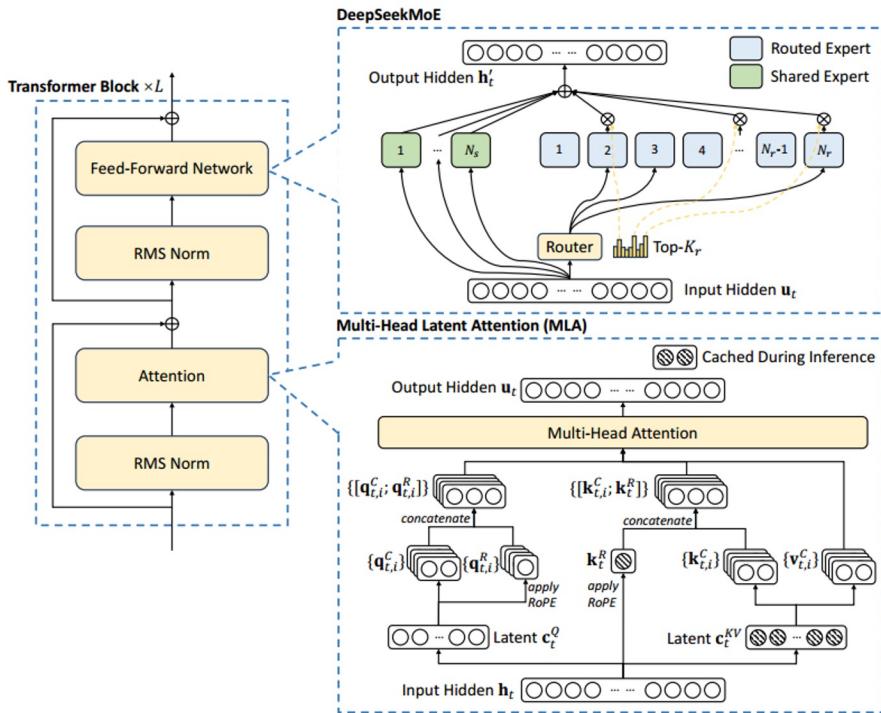
- Nếu mô hình (trong một lớp nào đó) sinh ra một phân bố \mathbf{x} có trung bình lớn (hoặc nhỏ), LayerNorm ngay lập tức đưa trung bình về 0.
- Lặp lại qua nhiều batch, giá trị trung bình và độ lệch chuẩn “thực” của \mathbf{x} có thể thay đổi linh hoạt (vì mạng có thể “cố” thay đổi đầu ra để tối ưu loss), nhưng rồi LayerNorm vẫn ép tất cả về 0 và 1 (sau cùng vẫn nhân γ , cộng β , v.v.).
- Điều này được gọi là “drift” vì mô hình có thể “lang thang” trong không gian tham số để tìm cách duy trì hoặc bù đắp cho những offset bị LayerNorm khử.

3. Ảnh hưởng thực tế:

- Ở mức độ vừa phải, LayerNorm giúp mô hình ổn định, tránh gradient explode/vanish.
- Tuy nhiên, trong các mạng rất sâu (Deep Transformers), việc lặp đi lặp lại thao tác “trừ trung bình” và “chia phương sai” cũng có thể gây ra một số hiện tượng “che khuất” (mask) một số tín hiệu liên quan đến tổng thể của \mathbf{x} .
- Một số nghiên cứu (đặc biệt ở các mô hình rất lớn) cho thấy bỏ trừ trung bình (chuyển sang RMS Norm) có thể tránh được phần nào hiện tượng drift này và cải thiện ổn định huấn luyện.



1 - Giới thiệu DeepSeekV2



Multi-Head Latent Attention
Low-Rank Key-Value Joint Compression
để giảm kích thước KV cache.

Decoupled Rotary Position Embedding (RoPE) để giữ thông tin vị trí hiệu quả.

Attention Computation theo cơ chế
Transformer tiêu chuẩn.

Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

1 - Giới thiệu DeepSeekV2

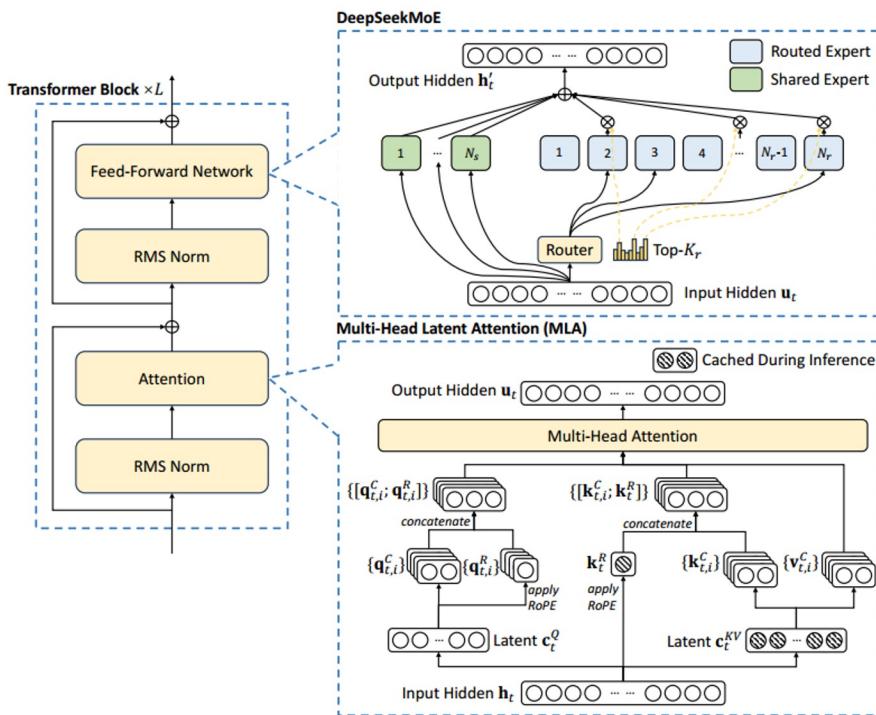


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

Multi-Head Latent Attention

1. Tạo Key (K), Value (V), và Query (Q):

- W_q, W_k, W_v dùng để chiếu đầu vào thành Query, Key, Value.
- Các giá trị này sau đó được chia thành nhiều đầu (multi-head).

2. Low-Rank Compression:

- W_{d_kv} : Giảm chiều K và V xuống thành một vector tiềm ẩn.
- W_{u_k}, W_{u_v} : Dùng để tái tạo K và V từ vector tiềm ẩn.
- W_{d_q}, W_{u_q} : Tương tự nhưng áp dụng cho Query.

3. Tính Attention:

- q_{final} và k_{final} được tạo bằng cách nối giá trị nén và không nén của Query và Key.
- Sử dụng attention softmax để tính toán giá trị trọng số.

4. Tạo đầu ra:

- Attention weight áp dụng lên Value.
- Kết quả được đưa qua w_o để trả về kích thước ban đầu.

1 - Giới thiệu DeepSeekV2

Multi-Head Latent Attention

```

def forward(self, x, mask=None):
    batch_size, seq_len, _ = x.size()

    # Compute Queries, Keys, Values
    q = self.W_q(x).view(batch_size, seq_len, self.num_heads, self.d_k)
    k = self.W_k(x).view(batch_size, seq_len, self.num_heads, self.d_k)
    v = self.W_v(x).view(batch_size, seq_len, self.num_heads, self.d_v)

    # Low-rank compression for KV
    c_kv = self.W_d_kv(x) # Compress
    k_compressed = self.W_u_k(c_kv).view(batch_size, seq_len, self.num_heads, self.d_k) # Reconstruct K
    v_compressed = self.W_u_v(c_kv).view(batch_size, seq_len, self.num_heads, self.d_v) # Reconstruct V

    # Low-rank compression for Q
    c_q = self.W_d_q(x) # Compress
    q_compressed = self.W_u_q(c_q).view(batch_size, seq_len, self.num_heads, self.d_k) # Reconstruct Q

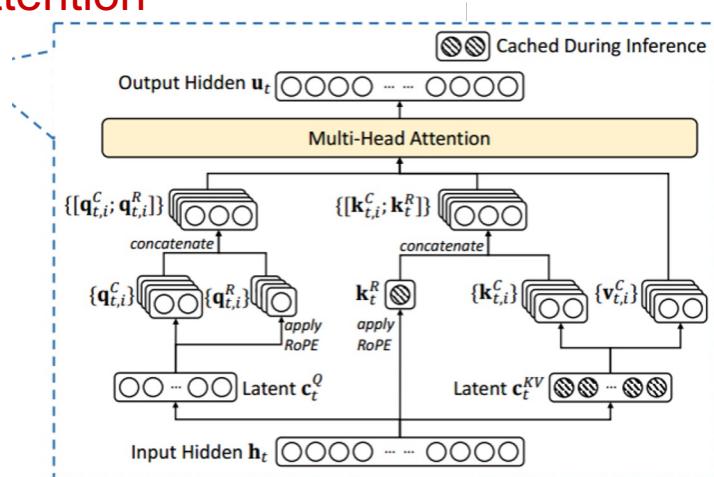
    # Concatenate decoupled queries and keys
    q_final = torch.cat([q_compressed, q], dim=-1) # [B, S, H, 2*d_k]
    k_final = torch.cat([k_compressed, k], dim=-1) # [B, S, H, 2*d_k]

    # Compute attention scores
    scores = torch.matmul(q_final, k_final.transpose(-2, -1)) / (self.d_k ** 0.5)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, float('-inf'))
    attn = F.softmax(scores, dim=-1)

    # Compute output
    out = torch.matmul(attn, v_compressed)
    out = out.transpose(1, 2).contiguous().view(batch_size, seq_len, self.d_model)
    out = self.W_o(out)

    return out

```



$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; \dots; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \text{RoPE}(W^{QR}\mathbf{c}_t^Q),$$

$$\mathbf{k}_t^R = \text{RoPE}(W^{KR}\mathbf{h}_t),$$

$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],$$

$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R],$$

$$\mathbf{o}_{t,i} = \sum_{j=1}^t \text{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right) \mathbf{v}_{j,i}^C$$

$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \dots; \mathbf{o}_{t,n_h}],$$

1 - Giới thiệu DeepSeekV2

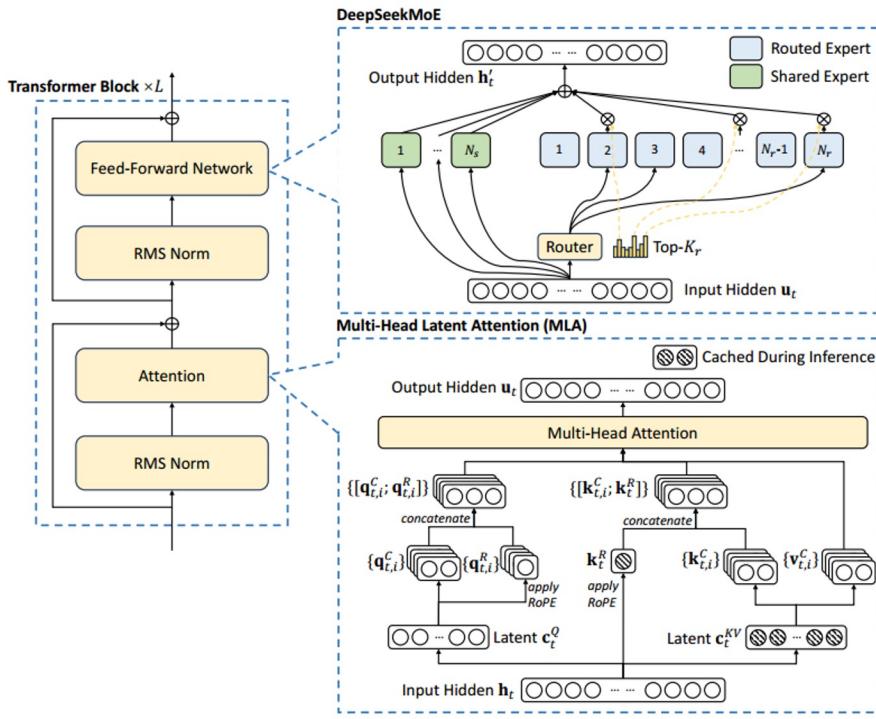


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

DeepSeekMoE

1. Cách thức hoạt động

Mỗi token đầu vào u_t sẽ đi qua một bộ định tuyến (Router) để chọn ra các chuyên gia tốt nhất để xử lý token đó. Công thức tính toán FFN (Feed-Forward Network) đầu ra của token thứ t là:

$$h'_t = u_t + \sum_{i=1}^{N_s} FFN_i^{(s)}(u_t) + \sum_{i=1}^{N_r} g_{i,t} FFN_i^{(r)}(u_t)$$

Trong đó:

- N_s là số lượng chuyên gia được chia sẻ (Shared Experts).
- N_r là số lượng chuyên gia định tuyến (Routed Experts).
- $g_{i,t}$ là trọng số của chuyên gia được chọn cho token t , được tính bằng:

$$g_{i,t} = \begin{cases} s_{i,t}, & \text{nếu } s_{i,t} \in \text{Top-}K_r \{s_{j,t} | 1 \leq j \leq N_r\} \\ 0, & \text{ngược lại} \end{cases}$$

Với $s_{i,t}$ được tính bằng:

$$s_{i,t} = \text{Softmax}_i(u_t^T e_i)$$

- e_i là vector trung tâm của chuyên gia i .
- K_r là số lượng chuyên gia được kích hoạt cho mỗi token.
- Chỉ một số chuyên gia được kích hoạt thay vì toàn bộ, giúp tiết kiệm chi phí tính toán

1 - Giới thiệu DeepSeekV2

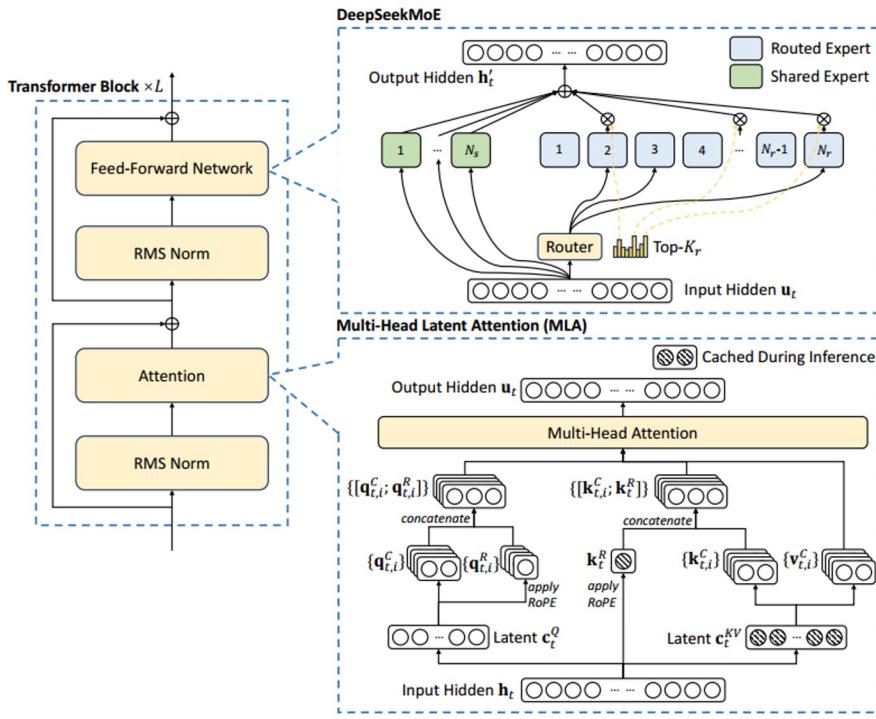


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

DeepSeekMoE

2. Device-Limited Routing và Auxiliary Loss

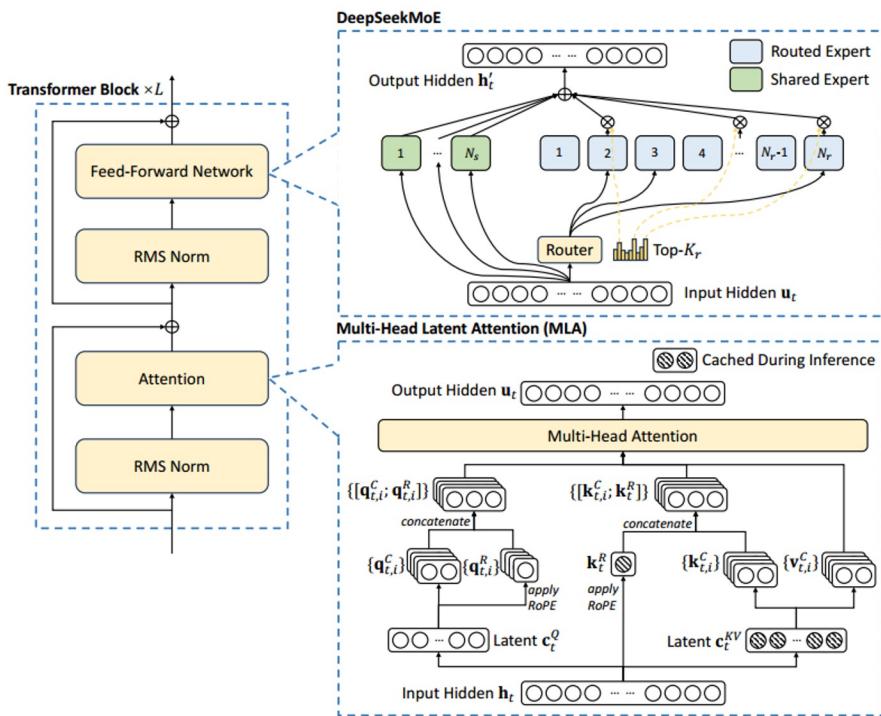
2.2. Cơ chế định tuyến chuyên gia giới hạn thiết bị (Device-Limited Routing)

- Khi sử dụng Expert Parallelism, các chuyên gia sẽ được phân bố trên nhiều thiết bị khác nhau.
- Nếu một token cần truy cập quá nhiều thiết bị, chi phí liên lạc giữa các thiết bị sẽ tăng cao.
- DeepSeekMoE giới hạn số lượng thiết bị tối đa mà một token có thể truy cập, giảm bớt chi phí giao tiếp mà vẫn giữ hiệu suất cao .

2.3. Cơ chế cân bằng tài nguyên Auxiliary Loss

- Vấn đề mất cân bằng tài: Một số chuyên gia có thể bị sử dụng quá mức trong khi những chuyên gia khác lại không được sử dụng.
- DeepSeekMoE sử dụng ba dạng Auxiliary Loss để đảm bảo cân bằng tài:
 1. **Expert-Level Balance Loss:** Đảm bảo rằng tất cả các chuyên gia đều được huấn luyện đầy đủ.
 2. **Device-Level Balance Loss:** Giảm sự chênh lệch về việc sử dụng chuyên gia giữa các thiết bị.
 3. **Communication Balance Loss:** Giảm tải việc truyền dữ liệu giữa các thiết bị .

1 - Giới thiệu DeepSeekV2



DeepSeekMoE

2. Token-Dropping Strategy

DeepSeekMoE sử dụng kỹ thuật Token Dropping, nơi một số token ít quan trọng sẽ bị loại bỏ trong quá trình huấn luyện.

=> Giúp tập trung vào các token quan trọng hơn và tăng tốc độ huấn luyện

Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

1 - Giới thiệu DeepSeekV2

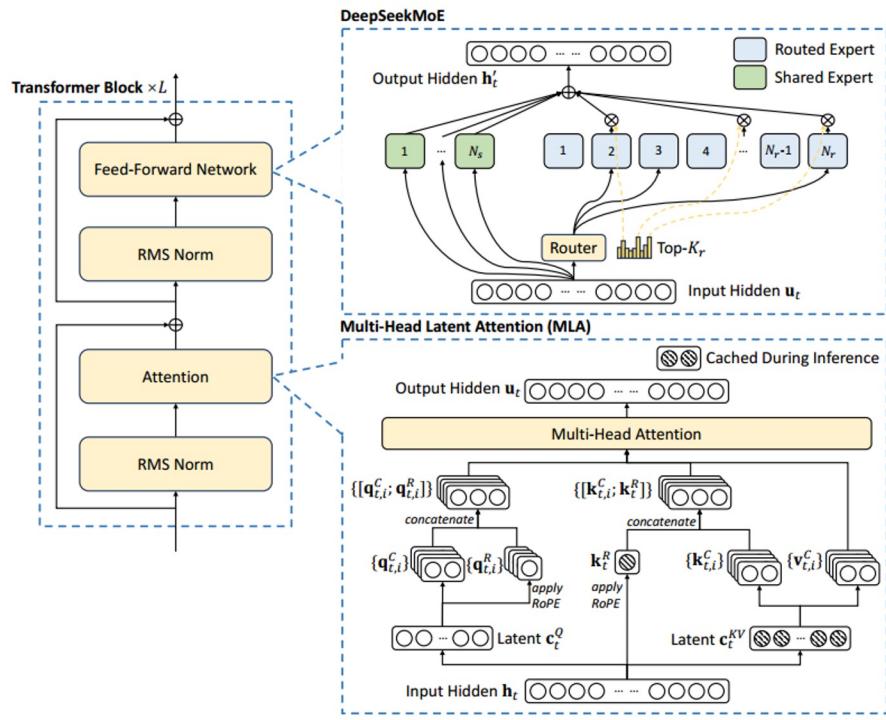


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

DeepSeekMoE

```

def forward(self, x):
    batch_size, seq_len, d_model = x.shape

    # Compute routing scores
    routing_logits = self.router(x) # (B, S, num_experts)
    routing_weights = F.softmax(routing_logits, dim=-1) # Softmax to get probabilities

    # Select top-k experts per token
    topk_weights, topk_indices = torch.topk(routing_weights, self.top_k, dim=-1) # (B, S, top_k)

    # Normalize top-k weights
    topk_weights = topk_weights / topk_weights.sum(dim=-1, keepdim=True)

    # Compute output from shared experts
    shared_output = torch.stack([expert(x) for expert in self.shared_experts]).sum(dim=0) / self.num_sharedExperts

    # Compute output from routed experts
    routed_output = torch.zeros_like(x)
    for i in range(self.top_k):
        expert_idx = topk_indices[:, :, i]
        expert_weight = topk_weights[:, :, i].unsqueeze(-1)

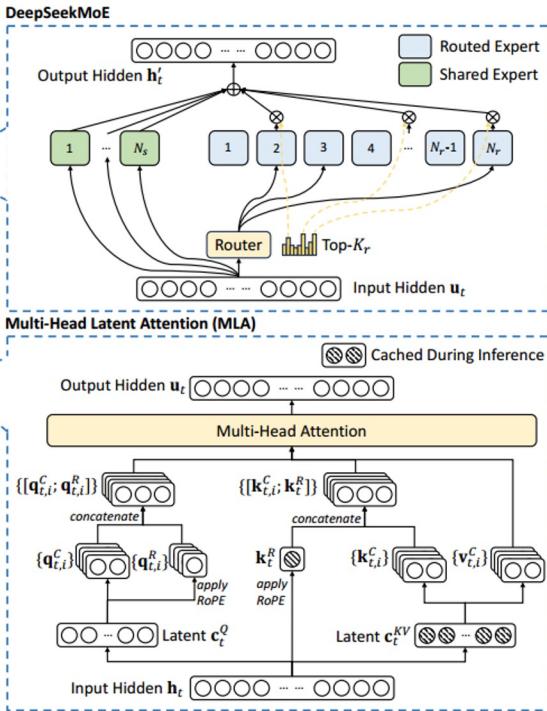
        # Get expert outputs
        expert_outputs = torch.stack([
            self.routed_experts[idx](x[b, t]) for b in range(batch_size) for t, idx in enumerate(expert_idx[b])
        ])
        expert_outputs = expert_outputs.view(batch_size, seq_len, d_model)

        # Weight expert outputs and accumulate
        routed_output += expert_weight * expert_outputs

    # Combine shared and routed expert outputs
    output = shared_output + routed_output
    return output

```

1 - Giới thiệu DeepSeekV2



DeepSeekMoE

```
class MoELayer(nn.Module):
    def __init__(self, d_model, numExperts, numSharedExperts=2, top_k=2):
        super().__init__()
        self.d_model = d_model
        self.numExperts = numExperts
        self.numSharedExperts = numSharedExperts
        self.top_k = top_k

        # Shared Experts (always active for every token)
        self.sharedExperts = nn.ModuleList([nn.Linear(d_model, d_model) for _ in range(numSharedExperts)])

        # Routed Experts (selected dynamically per token)
        self.routedExperts = nn.ModuleList([nn.Linear(d_model, d_model) for _ in range(numExperts)])

        # Router Network
        self.router = nn.Linear(d_model, numExperts)
```

Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

1 - Giới thiệu DeepSeekV2

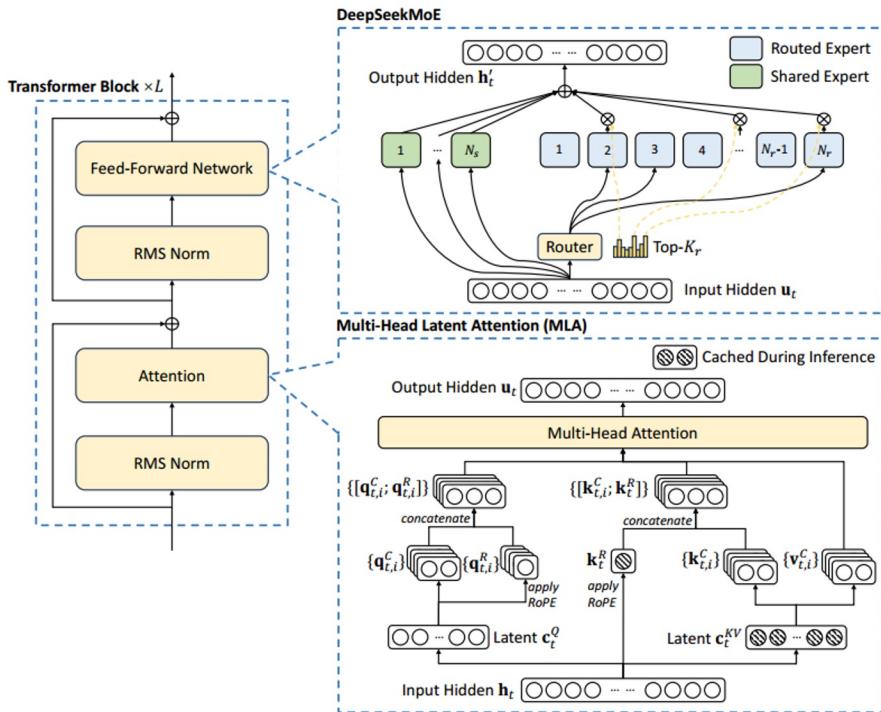


Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

Hiệu suất của DeepSeekV2

- Hiệu suất cao hơn các MoE trước đây
- Giảm 42.5% chi phí huấn luyện so với DeepSeek 67B
- Tăng tốc độ suy luận lên 5.76 lần
- KV Cache giảm đến 93.6%, tối ưu hóa bộ nhớ cho suy luận

Nội dung

1. Giới thiệu DeepSeekV2
2. DeepSeekV3 - Bước đệm hoàn hảo
3. DeepSeekR1 - Công nghệ đột phá
4. Cơ hội áp dụng DeepSeekR1

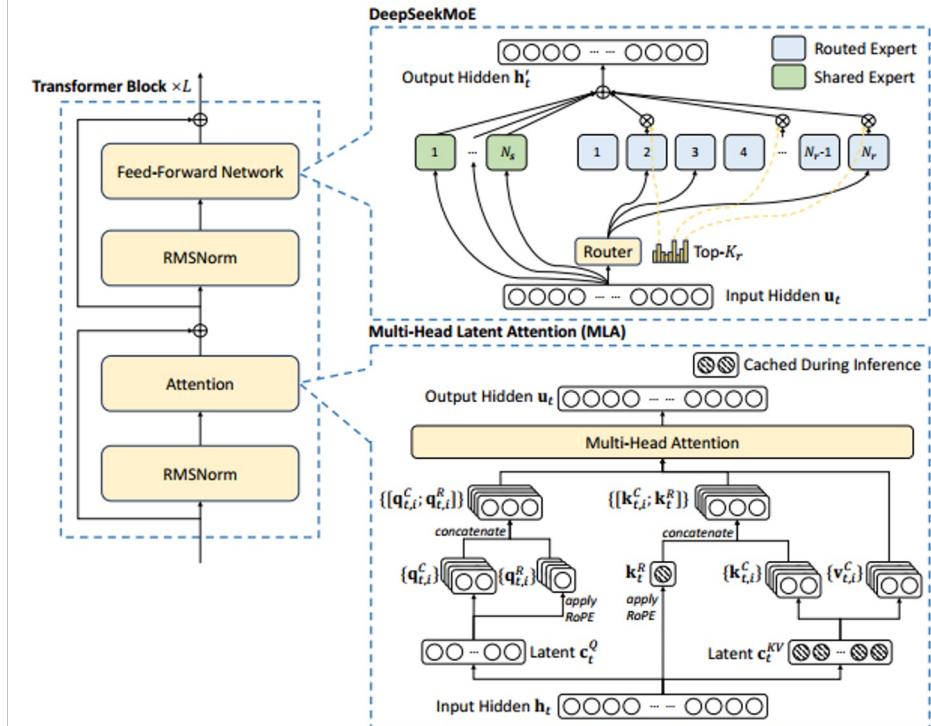


Figure 2 | Illustration of the basic architecture of DeepSeek-V3. Following DeepSeek-V2, we adopt MLA and DeepSeekMoE for efficient inference and economical training.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Sự khác biệt của DeepSeekv3 so với DeepSeekV2

1. DeepSeekMoE

Loại bỏ auxiliary loss trong cân bằng tải: DeepSeek-V3 áp dụng một chiến lược cân bằng tải không cần auxiliary loss, giúp giảm tác động tiêu cực lên hiệu suất mô hình do việc phân bổ tài nguyên.

Kết hợp chuyên gia chia sẻ (Shared Experts) và chuyên gia định tuyến (Routed Experts): Cấu trúc của DeepSeekMoE trong V3 vẫn bao gồm hai loại chuyên gia, nhưng có sự tối ưu hóa trong cách chọn chuyên gia cho mỗi token.

Tối ưu hóa huấn luyện: DeepSeek-V3 thực hiện huấn luyện với mức tài nguyên GPU thấp hơn mà vẫn duy trì hiệu suất cao nhờ kiến trúc MoE được tối ưu hóa.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Sự khác biệt của DeepSeekv3 so với DeepSeekV2

2. Multi Head Latent Attention

Giảm chi phí lưu trữ Key-Value (KV) Cache: DeepSeek-V3 sử dụng một phương pháp nén kết hợp (low-rank joint compression) cho các khóa (keys) và giá trị (values), giúp giảm dung lượng lưu trữ trong quá trình suy luận.

Áp dụng RoPE (Rotary Position Embeddings) cho khóa rời rạc: Trong DeepSeek-V3, phần query, key, và value được chia thành hai phần: phần nén và phần sử dụng RoPE. Điều này giúp cải thiện khả năng lưu trữ ngũ cảnh dài.

Tăng tốc suy luận: Với kỹ thuật tối ưu hóa KV cache, DeepSeek-V3 cải thiện tốc độ sinh token, làm cho quá trình suy luận nhanh hơn so với V2.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Sự khác biệt của DeepSeekv3 so với DeepSeekV2

3. Một số cải tiến khác

Sử dụng FP8 để tối ưu hóa hiệu suất và giảm chi phí huấn luyện.

Multi-Token Prediction (MTP): Thay vì dự đoán từng token một, DeepSeek-V3 dự đoán nhiều token một lúc, giúp tăng tốc suy luận.

Huấn luyện trên tập dữ liệu lớn hơn và đa dạng hơn (~14.8T tokens).

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

Vấn đề của Auxiliary Loss trong MoE truyền thống

- Trong DeepSeek-V2 và các MoE khác như GShard, Switch Transformer, auxiliary loss được dùng để đảm bảo tất cả các chuyên gia được sử dụng đồng đều.
- Tuy nhiên, auxiliary loss quá lớn có thể làm giảm hiệu suất của mô hình, dẫn đến lựa chọn chuyên gia không tối ưu .

Cách DeepSeek-V3 thay thế Auxiliary Loss

DeepSeek-V3 sử dụng "Auxiliary-Loss-Free Load Balancing", thay vì dựa vào auxiliary loss để đảm bảo cân bằng tải, mô hình sử dụng **bias term** b_i để điều chỉnh sự phân bổ của các chuyên gia:

$$s'_{i,t} = s_{i,t} + b_i$$

Bias term này được cập nhật liên tục:

- Nếu một chuyên gia bị quá tải, b_i sẽ **giảm**.
- Nếu một chuyên gia bị sử dụng ít, b_i sẽ **tăng**.

Điều này giúp tối ưu hóa việc phân bổ chuyên gia một cách tự nhiên mà không làm giảm hiệu suất của mô hình .

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

Nhược điểm của Auxiliary Loss

1. Gradient ảnh hưởng tiêu cực đến hiệu suất mô hình
 - Gradient của auxiliary loss tác động vào bộ định tuyến (router), có thể cản trở việc tối ưu hóa trọng số mô hình chính.
 - Điều này có thể làm giảm hiệu suất mô hình trên các bài toán suy luận.
2. Giảm độ chính xác của định tuyến chuyên gia
 - Khi auxiliary loss quá lớn, router bị ép phải chọn các chuyên gia kém hơn chỉ để đảm bảo cân bằng tải.
 - Điều này làm giảm chất lượng của quá trình suy luận.
3. Mất cân bằng trong huấn luyện
 - Nếu một chuyên gia bị sử dụng quá ít, auxiliary loss có thể khiến mô hình ép buộc sử dụng nó nhiều hơn, gây ra sự bất ổn trong huấn luyện.

1. Auxiliary Loss và vấn đề của nó

Auxiliary Loss trong MoE truyền thống được định nghĩa như sau:

$$\mathcal{L}_{aux} = \lambda \sum_{i=1}^{N_r} \left(p_i - \frac{1}{N_r} \right)^2$$

Với:

- p_i là xác suất sử dụng của chuyên gia thứ i (tính trên toàn bộ batch).
- N_r là tổng số chuyên gia được định tuyến.
- λ là hệ số điều chỉnh auxiliary loss.

Mục tiêu của Auxiliary Loss:

- Cân bằng tải giữa các chuyên gia bằng cách đảm bảo rằng **mỗi chuyên gia được sử dụng đồng đều**.
- Điều này giúp tránh tình trạng một số chuyên gia bị quá tải trong khi một số chuyên gia khác ít được sử dụng.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

2. Auxiliary-Loss-Free Load Balancing (ALF-LB)

Trong DeepSeek-V3, auxiliary loss được thay thế bằng bias term b_i , giúp tự điều chỉnh phân bổ chuyên gia mà không ảnh hưởng đến gradient chính của mô hình.

Quá trình định tuyến trong DeepSeek-V3 được tính như sau:

$$s'_{i,t} = s_{i,t} + b_i$$

$$g_{i,t} = \frac{s'_{i,t}}{\sum_{j=1}^{N_r} s'_{j,t}}$$

Với:

- $s_{i,t}$ là xác suất ban đầu của chuyên gia i cho token t .
- b_i là giá trị bias giúp điều chỉnh sự cân bằng chuyên gia.

Cách thức hoạt động của ALF-LB

1. Nếu một chuyên gia bị sử dụng quá ít, giá trị bias b_i **tự động tăng lên** → giúp router chọn nó thường xuyên hơn.
2. Nếu một chuyên gia bị quá tải, b_i **tự động giảm xuống**, giúp giảm tải cho chuyên gia này.
3. Không có gradient phụ thuộc vào auxiliary loss, do đó không ảnh hưởng đến trọng số mô hình chính.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

3. Chứng minh ALF-LB hiệu quả hơn Auxiliary Loss

3.1. Ổn định Gradient

- Auxiliary Loss có gradient tác động đến trọng số router:

$$\frac{\partial \mathcal{L}_{aux}}{\partial s_{i,t}} = 2\lambda \left(p_i - \frac{1}{N_r} \right)$$

Điều này dẫn đến sự can thiệp vào quá trình tối ưu hóa trọng số chính của mô hình.

- ALF-LB không có gradient phụ thuộc vào auxiliary loss, chỉ có bias tự điều chỉnh:

$$\frac{\partial g_{i,t}}{\partial s_{i,t}} = \frac{1}{\sum_{j=1}^{N_r} s'_{j,t}}$$

→ Gradient này chỉ ảnh hưởng đến trọng số của bộ định tuyến, không tác động đến mô hình chính, giúp tăng hiệu suất tối ưu hóa.

3.2. Cân bằng tài tốt hơn

- Auxiliary Loss ép buộc các chuyên gia có xác suất xuất hiện bằng nhau, nhưng không tính đến chất lượng của chuyên gia.
- ALF-LB điều chỉnh bias động để đảm bảo phân bổ chuyên gia một cách tự nhiên và linh hoạt.

Giả sử một chuyên gia bị sử dụng ít:

- Trong Auxiliary Loss: p_i thấp sẽ làm tăng loss, mô hình bị ép chọn chuyên gia này nhiều hơn một cách cưỡng ép.
- Trong ALF-LB: Bias b_i tăng lên một cách tự nhiên, giúp router có xu hướng chọn nó nhiều hơn, nhưng vẫn ưu tiên chuyên gia có chất lượng tốt hơn.

3.3. Hiệu suất huấn luyện

- DeepSeek-V3 báo cáo rằng việc loại bỏ Auxiliary Loss giúp **tăng tốc độ huấn luyện** mà không làm giảm hiệu suất mô hình .
- Không cần lưu trữ auxiliary loss**, giúp tiết kiệm bộ nhớ GPU.
- Không làm giảm hiệu suất suy luận** do mất cân bằng tải, giúp mô hình ổn định hơn trong quá trình huấn luyện.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

Nhược điểm của Auxiliary Loss trong DeepSeekMoE

1. Gradient ảnh hưởng tiêu cực đến hiệu suất mô hình

Auxiliary Loss (\mathcal{L}_{aux}) trong MoE truyền thống được định nghĩa như sau:

$$\mathcal{L}_{aux} = \lambda \sum_{i=1}^{N_r} \left(p_i - \frac{1}{N_r} \right)^2$$

Với:

- p_i là xác suất chọn chuyên gia i trên toàn batch.
- N_r là số lượng Routed Experts.
- λ là hệ số điều chỉnh Auxiliary Loss.

Đạo hàm của Auxiliary Loss theo p_i :

$$\frac{\partial \mathcal{L}_{aux}}{\partial p_i} = 2\lambda \left(p_i - \frac{1}{N_r} \right)$$

Ảnh hưởng tiêu cực của gradient này

1. Làm thay đổi quá trình tối ưu hóa mô hình chính:

- Gradient của Auxiliary Loss được cộng vào gradient tổng của mô hình khi cập nhật trọng số.
- Điều này khiến mô hình không chỉ tối ưu hóa loss chính (ví dụ: cross-entropy loss) mà còn phải điều chỉnh xác suất chọn chuyên gia.
- Kết quả: Auxiliary Loss có thể cản trở việc tối ưu hóa chính, dẫn đến hiệu suất suy luận thấp hơn.

2. Tác động đến Router:

- Bộ định tuyến (Router) ban đầu chọn chuyên gia dựa trên xác suất Softmax:

$$p_i = \frac{e^{s_i}}{\sum_{j=1}^{N_r} e^{s_j}}$$

- Nếu Auxiliary Loss ép buộc p_i về $\frac{1}{N_r}$, mô hình sẽ có xu hướng chọn **tất cả chuyên gia với xác suất gần như bằng nhau**, ngay cả khi có một số chuyên gia hiệu quả hơn những chuyên gia khác.
- Hậu quả:** Làm giảm khả năng chọn chuyên gia tối ưu cho từng token.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss

Nhược điểm của Auxiliary Loss trong DeepSeekMoE

2. Giảm độ chính xác của định tuyến chuyên gia

Mô hình MoE sử dụng softmax-based routing, tức là router chọn chuyên gia dựa trên:

$$p_i = \frac{e^{s_i}}{\sum_{j=1}^{N_r} e^{s_j}}$$

Nhưng khi thêm Auxiliary Loss, công thức cập nhật trở thành:

$$p_i = \frac{e^{s_i - 2\lambda(p_i - \frac{1}{N_r})}}{\sum_{j=1}^{N_r} e^{s_j - 2\lambda(p_j - \frac{1}{N_r})}}$$

Vấn đề:

- Thành phần $-2\lambda(p_i - \frac{1}{N_r})$ làm thay đổi giá trị chọn chuyên gia, khiến router không còn chọn chuyên gia tối ưu mà chọn một cách đồng đều hơn.
- Điều này gây ra sự mất hiệu suất, vì chuyên gia mạnh có thể bị chọn ít hơn chỉ để cân bằng tải.

Ví dụ:

- Nếu một chuyên gia có trọng số tối ưu hơn với $s_i = 10$, còn một chuyên gia khác yếu hơn với $s_j = 5$, thì bình thường:

$$p_i \approx \frac{e^{10}}{e^{10} + e^5} = 0.993$$

Nhưng khi có Auxiliary Loss:

$$p_i \approx \frac{e^{10 - 2\lambda(p_i - \frac{1}{N_r})}}{e^{10 - 2\lambda(p_i - \frac{1}{N_r})} + e^{5 - 2\lambda(p_j - \frac{1}{N_r})}}$$

Nếu λ lớn, p_i giảm xuống một cách không hợp lý, làm router không còn chọn chuyên gia tốt nhất.

- Kết quả: Giảm độ chính xác trong việc chọn chuyên gia, dẫn đến suy luận kém hơn.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Loại bỏ Auxiliary Loss
Nhược điểm của Auxiliary Loss trong DeepSeekMoE

3. Mất cân bằng trong huấn luyện

Trong quá trình huấn luyện, nếu một chuyên gia bị sử dụng quá ít (do giá trị s_i thấp), auxiliary loss sẽ ép buộc mô hình phải chọn chuyên gia đó nhiều hơn.

Để minh họa, giả sử chuyên gia i ít được chọn, ta có:

$$\mathcal{L}_{aux} = \lambda \sum_{i=1}^{N_r} \left(p_i - \frac{1}{N_r} \right)^2$$

Khi p_i quá nhỏ ($p_i \approx 0$), đạo hàm của loss tăng mạnh:

$$\frac{\partial \mathcal{L}_{aux}}{\partial p_i} = 2\lambda \left(0 - \frac{1}{N_r} \right) = -\frac{2\lambda}{N_r}$$

Điều này khiến router cố gắng tăng p_i bằng cách thay đổi trọng số s_i :

$$s_i = s_i + \eta \frac{2\lambda}{N_r}$$

Với η là learning rate.

Tác động tiêu cực:

- Nếu một chuyên gia yếu nhưng bị ép phải được chọn, mô hình có thể học các biểu diễn kém chính xác hơn.
- Nếu auxiliary loss quá lớn, mô hình không thể học được chuyên gia nào thực sự hiệu quả, vì mọi chuyên gia đều phải được sử dụng với xác suất gần như bằng nhau.
- Mất cân bằng trong batch training:
 - Một số batch có thể sử dụng chuyên gia mạnh hơn, nhưng auxiliary loss lại ép buộc sử dụng chuyên gia yếu hơn ở batch tiếp theo → gây nhiễu trong huấn luyện.
 - Hệ quả: Làm chậm quá trình hội tụ và có thể làm giảm độ chính xác cuối cùng của mô hình.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert

Shared Experts (Chuyên gia chia sẻ):

- Luôn hoạt động cho mọi token trong quá trình suy luận và huấn luyện.
- Được thiết kế để nắm bắt các đặc trưng phổ quát mà mọi token đều cần.
- Giúp mô hình giữ sự ổn định trong quá trình huấn luyện, đặc biệt khi các chuyên gia định tuyến có thể thay đổi linh hoạt.

Ví dụ:

- Khi xử lý câu văn dài, các Shared Experts có thể học các cấu trúc ngữ pháp cơ bản hoặc các đặc trưng chung như phát hiện tên thực thể (NER).

Routed Experts (Chuyên gia định tuyến):

- Chỉ được kích hoạt cho một số token nhất định dựa trên cơ chế định tuyến (routing mechanism).
- Chuyên gia này học các đặc trưng chuyên biệt hơn, phục vụ cho từng loại dữ liệu hoặc ngữ cảnh cụ thể.
- Quá trình chọn Routed Experts được thực hiện thông qua bộ định tuyến, giúp tiết kiệm tài nguyên tính toán.

Ví dụ:

- Khi xử lý câu hỏi liên quan đến toán học, Routed Experts có thể học cách giải phương trình hoặc phân tích dữ liệu.

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert

2.1. Kết hợp tối ưu giữa Shared và Routed Experts

Trong DeepSeekMoE V3, sự kết hợp giữa Shared và Routed Experts được thực hiện thông qua công thức:

$$h'_t = u_t + \sum_{i=1}^{N_s} FFN_i^{(s)}(u_t) + \sum_{i=1}^{N_r} g_{i,t} FFN_i^{(r)}(u_t)$$

Trong đó:

- N_s : Số lượng Shared Experts.
- N_r : Số lượng Routed Experts.
- $g_{i,t}$: Trọng số quyết định chuyên gia định tuyến nào sẽ được kích hoạt cho token t .

Điểm tối ưu mới trong V3:

1. Cơ chế định tuyến mềm mại hơn:

- Thay vì sử dụng Softmax như trong V2, V3 sử dụng Sigmoid-based routing kết hợp với bias b_i .
- Điều này giúp giảm sự cứng nhắc trong việc chọn chuyên gia, tăng tính linh hoạt khi lựa chọn Routed Experts.

2. Tự động cân bằng thông minh hơn:

- Shared Experts luôn hoạt động như một nền tảng vững chắc, đảm bảo rằng các đặc trưng phổ quát được giữ nguyên.
- Routed Experts tự điều chỉnh qua cơ chế bias cập nhật động, giúp phân bổ tài nguyên tốt hơn mà không làm giảm hiệu suất của mô hình.

3. Tối ưu chi phí tính toán:

- Với cơ chế chọn lọc thông minh hơn, số lượng chuyên gia định tuyến được kích hoạt cho mỗi token giảm, giúp tiết kiệm tài nguyên tính toán nhưng vẫn đảm bảo hiệu suất.
- Điều này giúp giảm độ phức tạp tính toán tổng thể trong quá trình suy luận.

$$\mathbf{h}'_t = \mathbf{u}_t + \sum_{i=1}^{N_s} FFN_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t} FFN_i^{(r)}(\mathbf{u}_t),$$

$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}},$$

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \text{Topk}(\{s_{j,t} | 1 \leq j \leq N_r\}, K_r), \\ 0, & \text{otherwise,} \end{cases}$$

$$s_{i,t} = \text{Sigmoid}(\mathbf{u}_t^T \mathbf{e}_i),$$

DeepSeekV3

$$\mathbf{h}'_t = \mathbf{u}_t + \sum_{i=1}^{N_s} FFN_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t} FFN_i^{(r)}(\mathbf{u}_t),$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \text{Topk}(\{s_{j,t} | 1 \leq j \leq N_r\}, K_r), \\ 0, & \text{otherwise,} \end{cases}$$

$$s_{i,t} = \text{Softmax}_i(\mathbf{u}_t^T \mathbf{e}_i),$$

DeepSeekV2

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert Chứng minh sigmoid attention hiệu quả hơn softmax attention

1. Giảm sự cạnh tranh giữa các token

- Softmax Attention có tính chất chuẩn hóa hàng (row-wise), nghĩa là tổng các trọng số chú ý luôn bằng 1. Điều này tạo ra sự cạnh tranh giữa các token, tức là khi một token có trọng số lớn hơn, các token khác sẽ bị giảm trọng số.
- Ngược lại, Sigmoid Attention áp dụng hàm sigmoid theo từng phần tử (element-wise), giúp loại bỏ sự cạnh tranh không cần thiết giữa các token. Điều này giúp mô hình không bỏ qua các đặc điểm tiềm năng quan trọng trong dữ liệu.

2. Cải thiện hiệu suất tính toán

- Softmax Attention yêu cầu chuẩn hóa trên mỗi hàng của ma trận trọng số, dẫn đến chi phí tính toán cao hơn.
- Sigmoid Attention chỉ áp dụng sigmoid trên từng phần tử riêng lẻ, giúp tăng tốc độ tính toán và giảm nhu cầu bộ nhớ, đặc biệt là khi triển khai trên GPU.

3. Hiệu quả lấy mẫu (Sample Efficiency) cao hơn

- Bài báo chứng minh rằng Sigmoid Attention có hiệu quả lấy mẫu cao hơn bằng cách liên kết nó với mô hình Mixture-of-Experts (MoE).
- Trong bối cảnh Dense Regime (môi trường có nhiều dữ liệu), các chuyên gia (experts) trong Sigmoid Attention chỉ cần số lượng dữ liệu $O(\epsilon^{-2})$ để đạt được độ chính xác mong muốn, trong khi Softmax Attention cần $O(\exp(\epsilon^{-1}/\tau))$ (nghĩa là tốn nhiều dữ liệu hơn để đạt cùng mức độ chính xác).
- Điều này làm cho Sigmoid Attention có khả năng tổng quát hóa tốt hơn khi làm việc với dữ liệu ít hơn.

4. Giảm hiện tượng "Attention Sink"

- Trong mô hình ngôn ngữ tự hồi quy (auto-regressive language models), Softmax Attention có xu hướng gán trọng số lớn cho token đầu tiên do cơ chế chuẩn hóa của Softmax. Điều này dẫn đến mô hình có thể không phân bổ đủ trọng số cho các token quan trọng khác.
- Sigmoid Attention không gặp vấn đề này do mỗi trọng số được tính độc lập.

5. Thực nghiệm chứng minh hiệu quả cao hơn

- Bài báo đã thực hiện các thí nghiệm trên cả dữ liệu tổng hợp và các bài toán thực tế, bao gồm mô hình ngôn ngữ.
- Kết quả cho thấy Sigmoid Attention có tốc độ huấn luyện và suy luận nhanh hơn Softmax Attention, đồng thời đạt hiệu suất tương đương hoặc tốt hơn trong các nhiệm vụ như ARC Challenge, HellaSwag, PIQA, SciQ, Winogrande.
- Sigmoid Attention đạt tốc độ xử lý cao hơn (~10% so với Softmax Attention).

Paper: [Sigmoid Self-Attention is Better than Softmax Self-Attention: A Mixture-of-Experts Perspective](#)

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert
Chứng minh sigmoid attention hiệu quả hơn softmax attention

3. So sánh trực tiếp

Tiêu chí	Softmax Attention	Sigmoid Attention
Công thức chuẩn hóa	$\text{Softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$	$\sigma \left(\frac{QK^T}{\sqrt{d_k}} \right) V$
Tính cạnh tranh (Competition)	Có: Tổng trọng số bằng 1, tăng trọng số cho token này làm giảm token khác	Không: Trọng số của các token độc lập, không làm giảm nhau
Tính toán	Phức tạp hơn do chuẩn hóa theo hàng (row-wise normalization)	Nhanh hơn vì áp dụng hàm sigmoid từng phần tử (element-wise)
Phân bổ trọng số	Chú ý nhiều vào một số token quan trọng, dễ bỏ qua các token khác	Phân bổ trọng số đều hơn, không bỏ sót thông tin quan trọng
Hiệu suất tính toán	Chậm hơn, tốn tài nguyên tính toán hơn	Nhanh hơn, tiết kiệm tài nguyên hơn, đặc biệt khi sử dụng GPU

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert
Chứng minh sigmoid attention hiệu quả hơn softmax attention

4. Ví dụ minh họa

Giả sử ta có $QK^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, và $\sqrt{d_k} = 1$ để đơn giản.

- Softmax Attention:

$$\text{Softmax} \left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \right) = \begin{bmatrix} \frac{e^1}{e^1+e^2+e^3} & \frac{e^2}{e^1+e^2+e^3} & \frac{e^3}{e^1+e^2+e^3} \\ \frac{e^4}{e^4+e^5+e^6} & \frac{e^5}{e^4+e^5+e^6} & \frac{e^6}{e^4+e^5+e^6} \end{bmatrix}$$

- Sigmoid Attention:

$$\sigma \left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \right) = \begin{bmatrix} \frac{1}{1+e^{-1}} & \frac{1}{1+e^{-2}} & \frac{1}{1+e^{-3}} \\ \frac{1}{1+e^{-4}} & \frac{1}{1+e^{-5}} & \frac{1}{1+e^{-6}} \end{bmatrix}$$

2 - DeepSeekV3 - Bước đệm hoàn hảo

Cải tiến quan trọng: Thay đổi trong cơ chế Share Expert và Route Expert

3. Minh họa quy trình Multi-Token Prediction

So sánh giữa One-Token và Multi-Token Prediction:

Quy trình	One-Token Prediction (Truyền thống)	Multi-Token Prediction (MTP)
Bước 1	Dự đoán token thứ $t + 1$	Dự đoán token từ $t + 1$ đến $t + k$ cùng một lúc
Bước 2	Cập nhật KV cache với token $t + 1$	Cập nhật KV cache sau khi dự đoán đủ k token
Bước 3	Sử dụng token $t + 1$ làm đầu vào để dự đoán $t + 2$	Tiếp tục dự đoán token mới từ $t + k + 1$ đến $t + 2k$
Hiệu suất	Chậm vì cần dự đoán và cập nhật từng token	Nhanh hơn do dự đoán và cập nhật nhiều token cùng lúc

4. Lợi ích của Multi-Token Prediction

1. Tăng tốc độ suy luận:

- Vi mô hình dự đoán nhiều token trong một lần forward pass, tốc độ suy luận có thể tăng lên 2x đến 5x tùy vào số lượng token được dự đoán cùng lúc.

2. Giảm chi phí bộ nhớ:

- Việc giảm số lần cập nhật KV cache giúp tiết kiệm bộ nhớ đáng kể, đặc biệt trong các mô hình lớn như DeepSeek-V3.

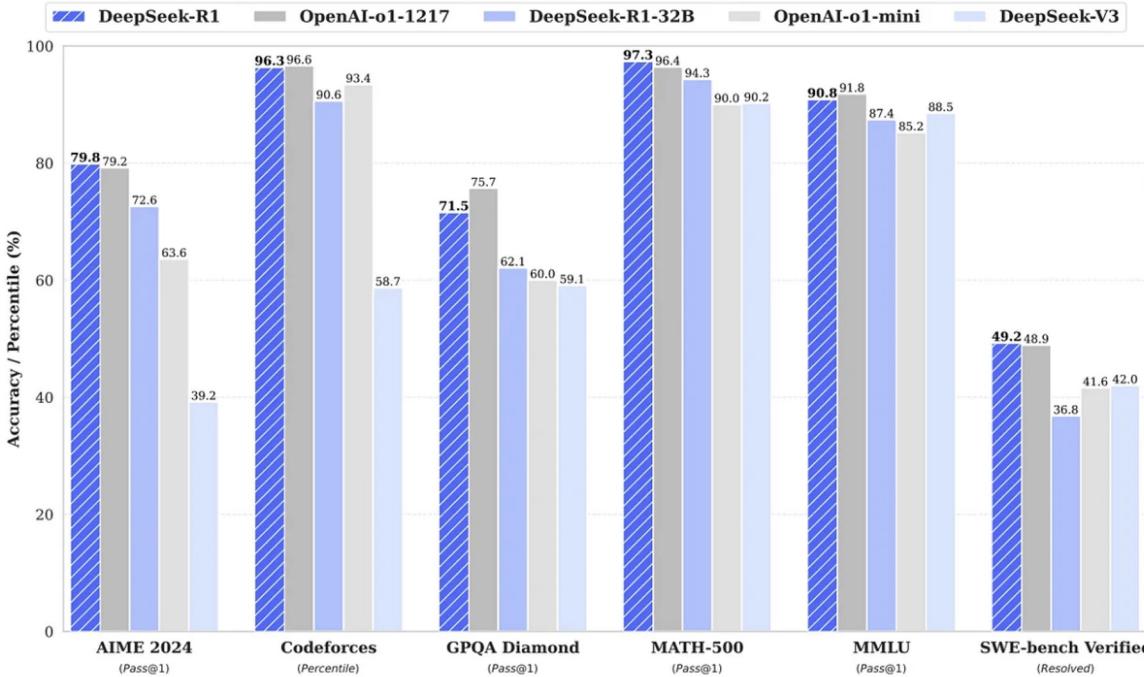
3. Tối ưu hóa hiệu suất mà không ảnh hưởng đến chất lượng:

- Nhờ vào kỹ thuật Rotary Position Embedding (RoPE) và nén Key-Value (KV compression), mô hình vẫn giữ được chất lượng cao khi dự đoán nhiều token cùng lúc.

Nội dung

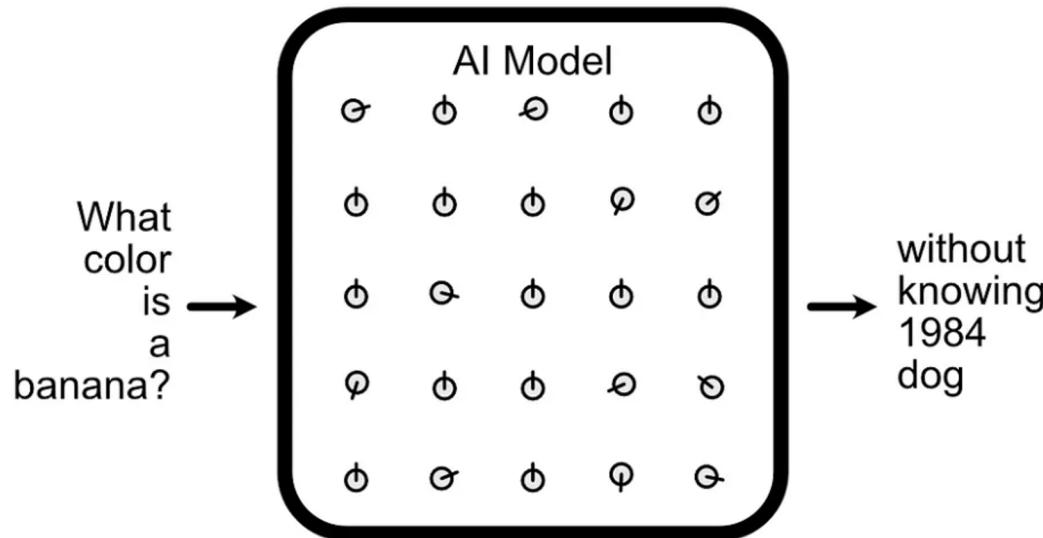
1. Giới thiệu DeepSeekV2
2. DeepSeekV3 - Bước đệm hoàn hảo
- 3. DeepSeekR1 - Công nghệ đột phá**
4. Cơ hội áp dụng DeepSeekR1

1 - Giới thiệu DeepSeekR1



DeepSeek-R1, compared to OpenAI o1 at a variety of tasks. [Source](#)

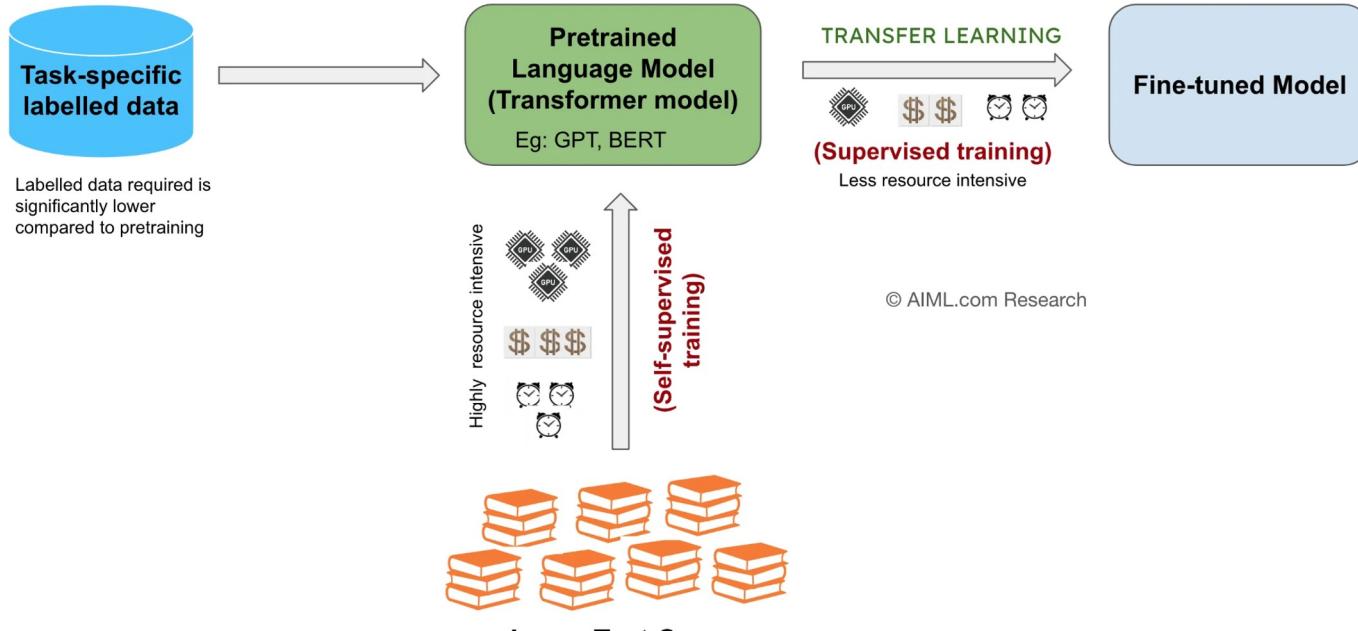
1 - Foundation



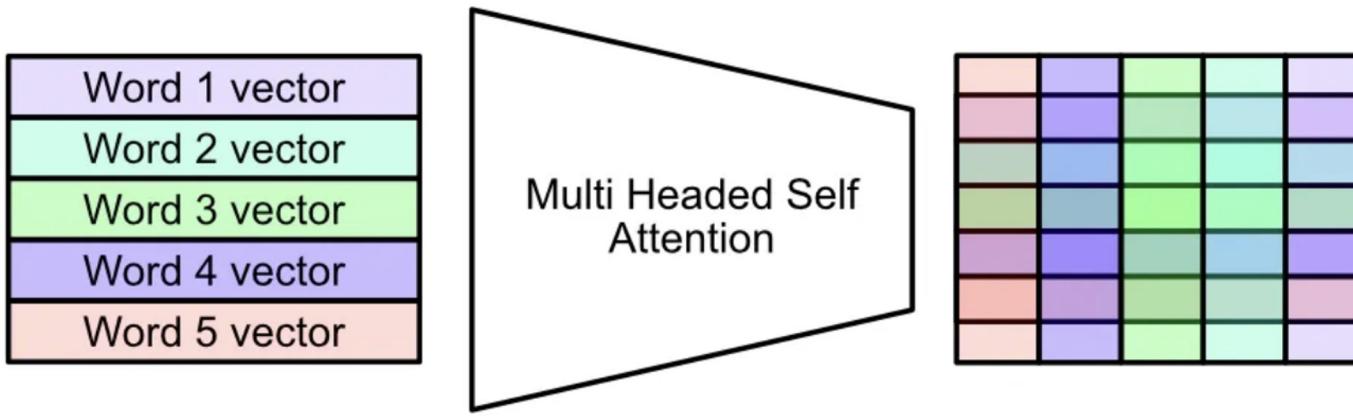
A reasonable way to conceptualize an AI model. A big group of dials that turns some input into an output. At first, these dials are random, and the output is bad.

1 - Foundation

Pretraining, Finetuning and Transfer Learning Using Transformers



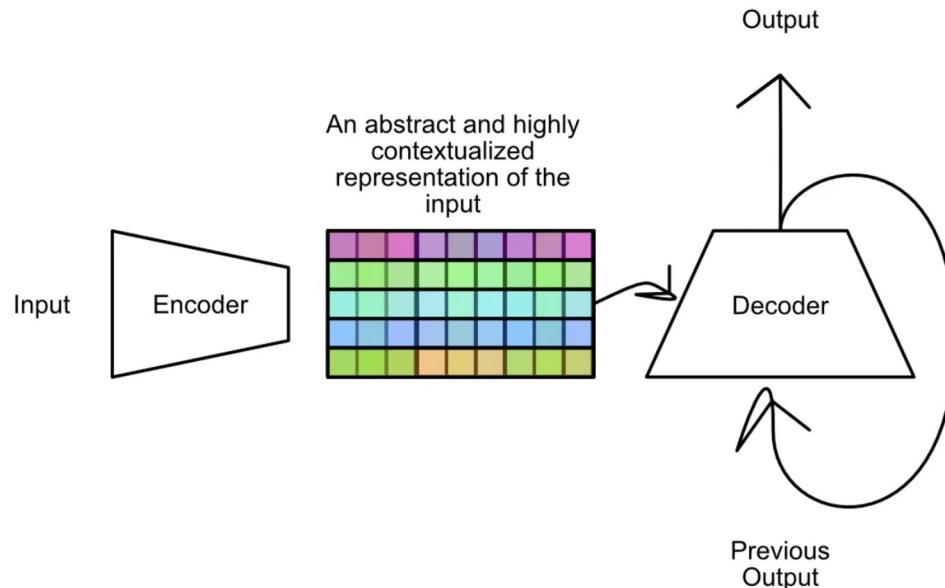
1 - Foundation



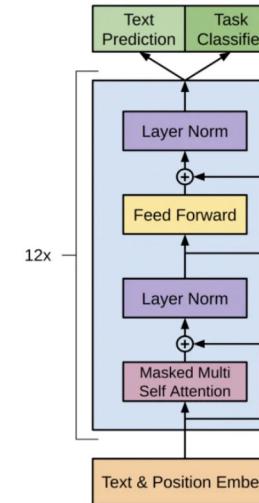
The primary mechanism in the transformer which creates abstract representations of data used to power its complex logical abilities is called “Attention”. The job of attention is to take in a list of atomic information (like a list of vectors representing a sentence of words) and turn that into an abstract and meaning rich representation that represents the whole input. There are other components in the transformer, but this is the primary one.

From [my article on transformers](#).

1 - Foundation



The essential idea of the original transformer. It used attention in clever ways to reason about the input and output separately. From [my article on transformers](#)



The model architecture of GPT. For our purposes the components don't really matter, what does matter is that it's one model which takes in text (at the bottom) and spits out text (at the top). [source](#)

1 - Foundation

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. 

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. 

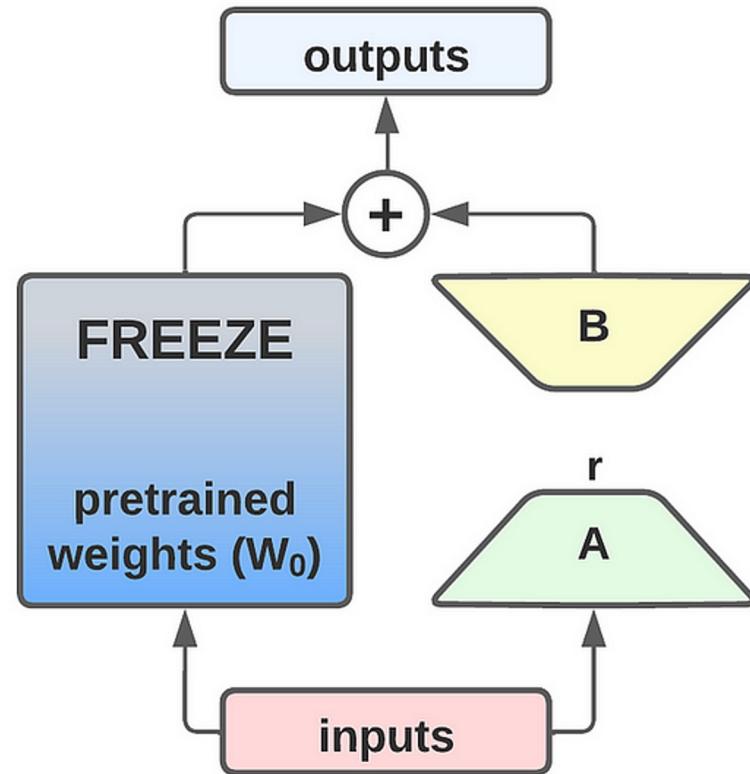
The example on the left failed with standard prompting, even though the model was exactly the same. [From my article on agents. Source](#)

1 - Foundation

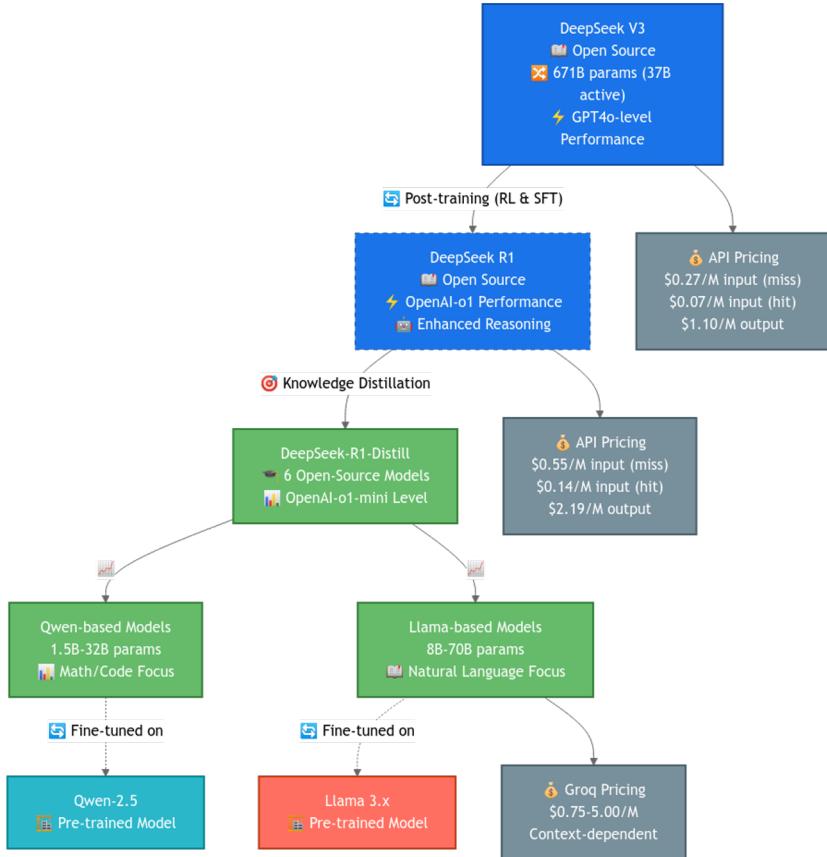
Linear Projection



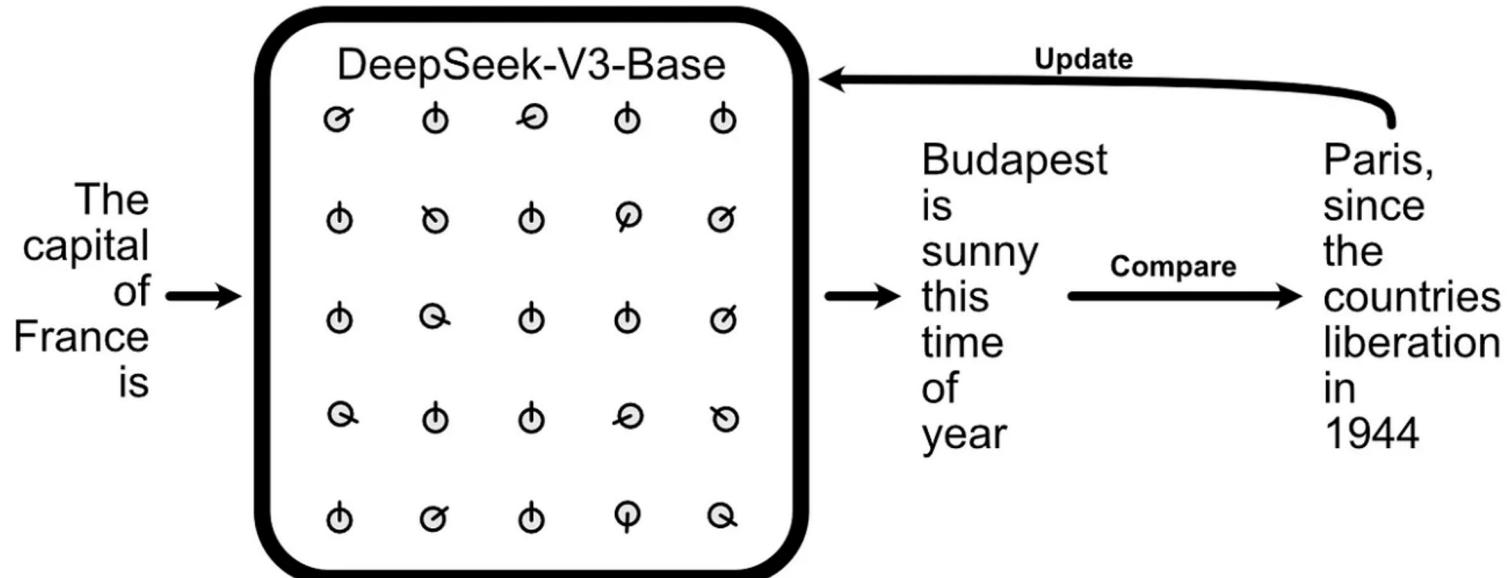
LoRA adaptation



1 - Fundamental Idea

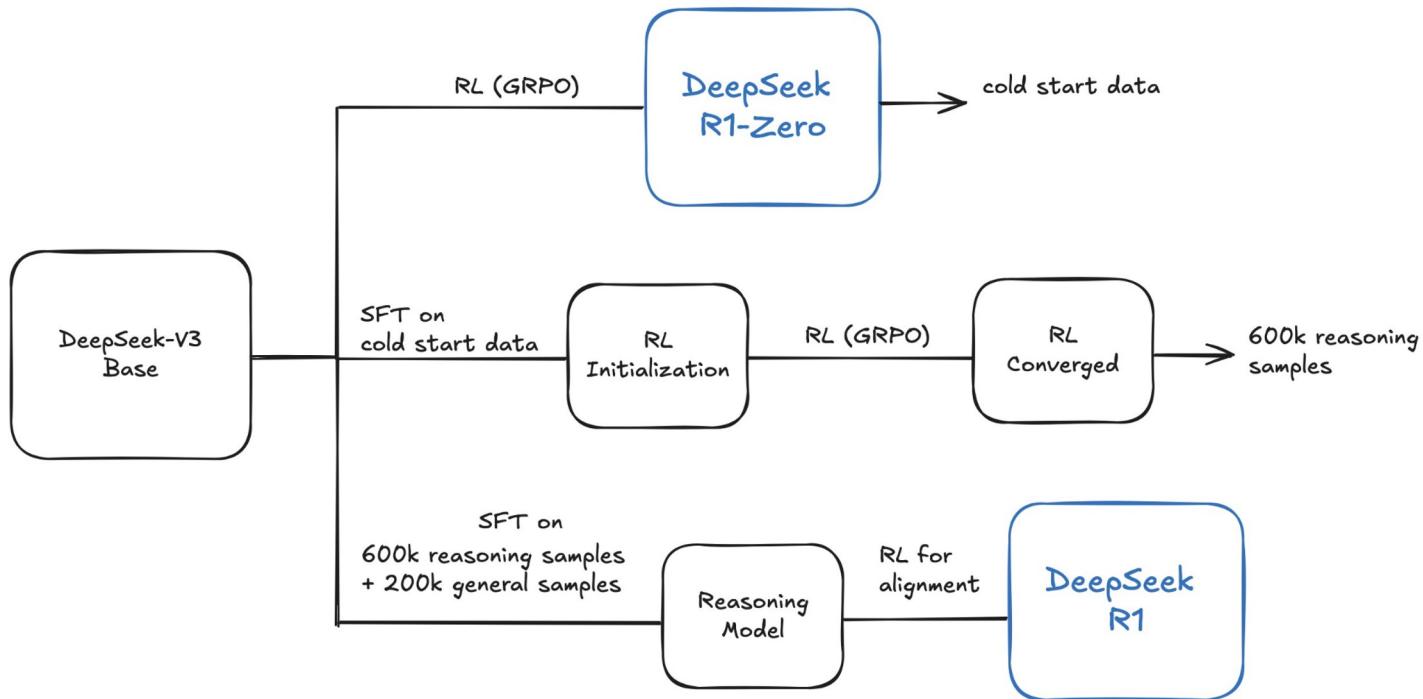


DeepSeek V3 - Base

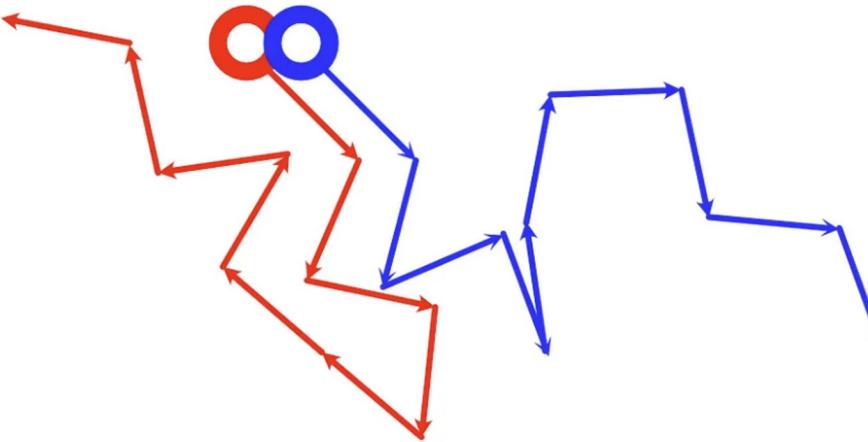


A conceptual diagram of how a transformer style model might be trained.

DeepSeek R1 - Zero



DeepSeek R1



A conceptual diagram of the importance of the initialization of the model. Because the model is being reinforced based on its own output, any slight deviation between models being trained with reinforcement learning can be compounded over time. In mathematics this is oftenen reffered to as a “chaotic system” because the output heavily depends on the input.

GRPO

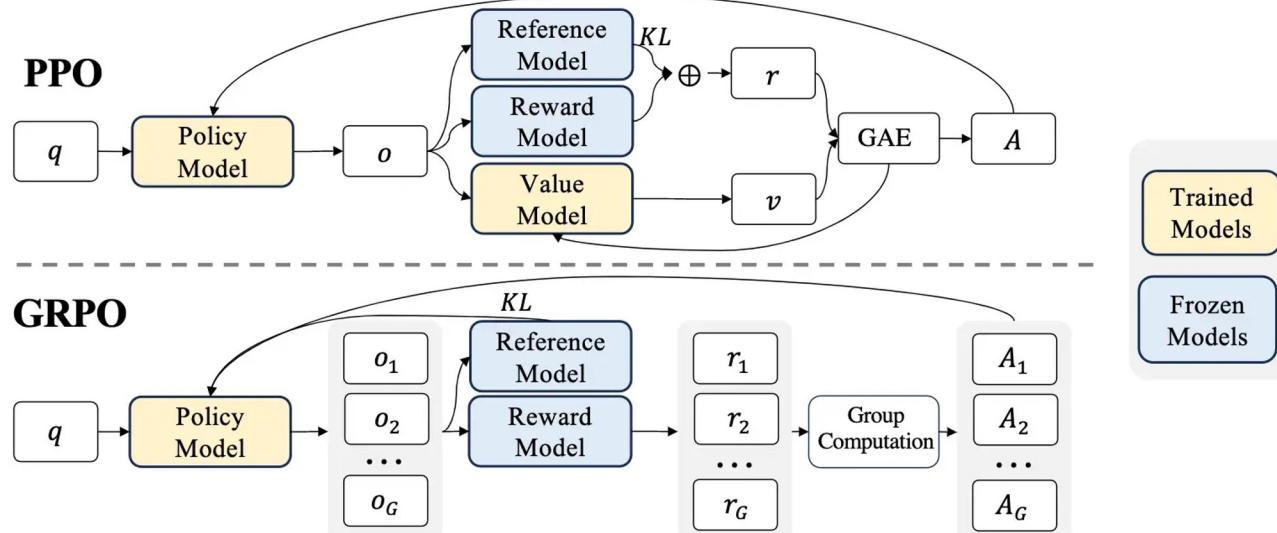
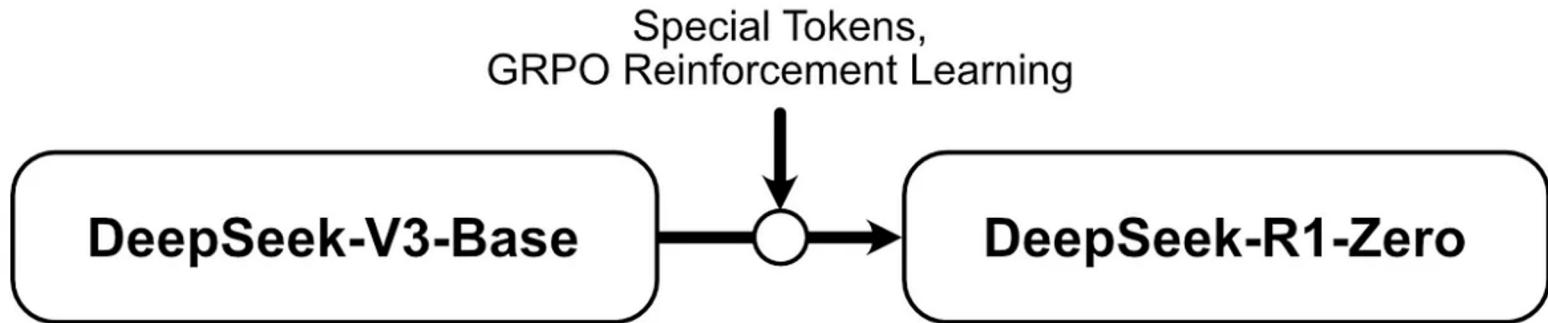


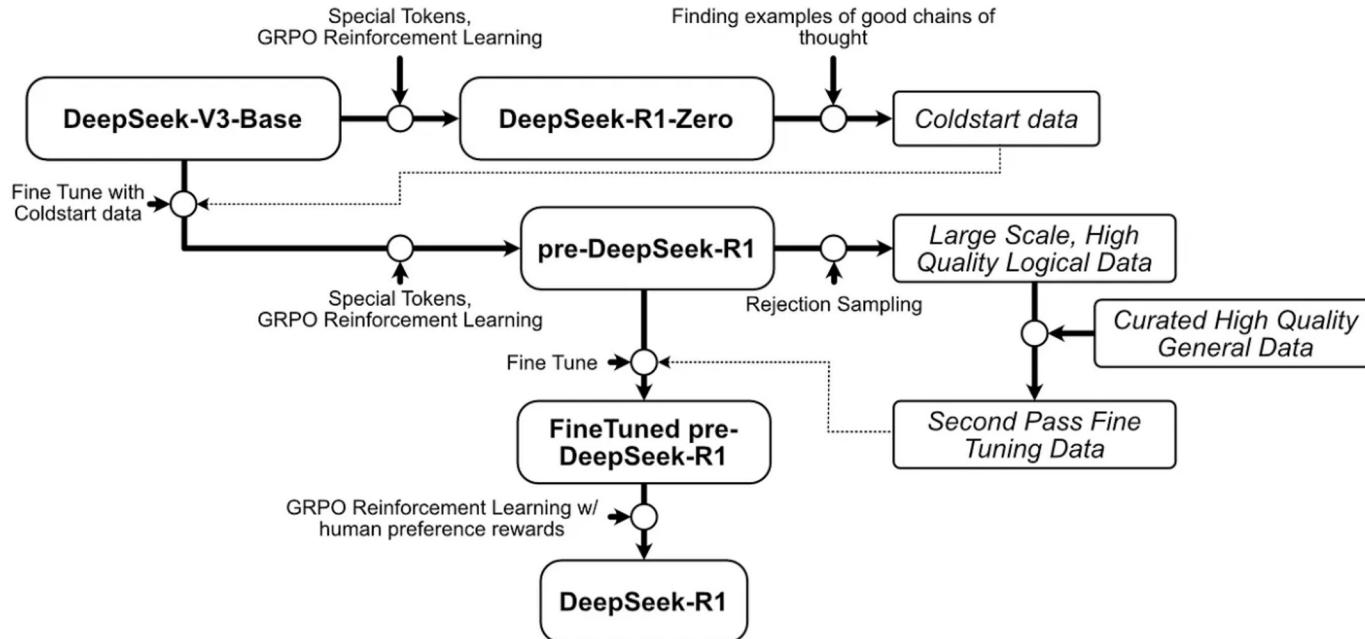
Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

Training Pipeline



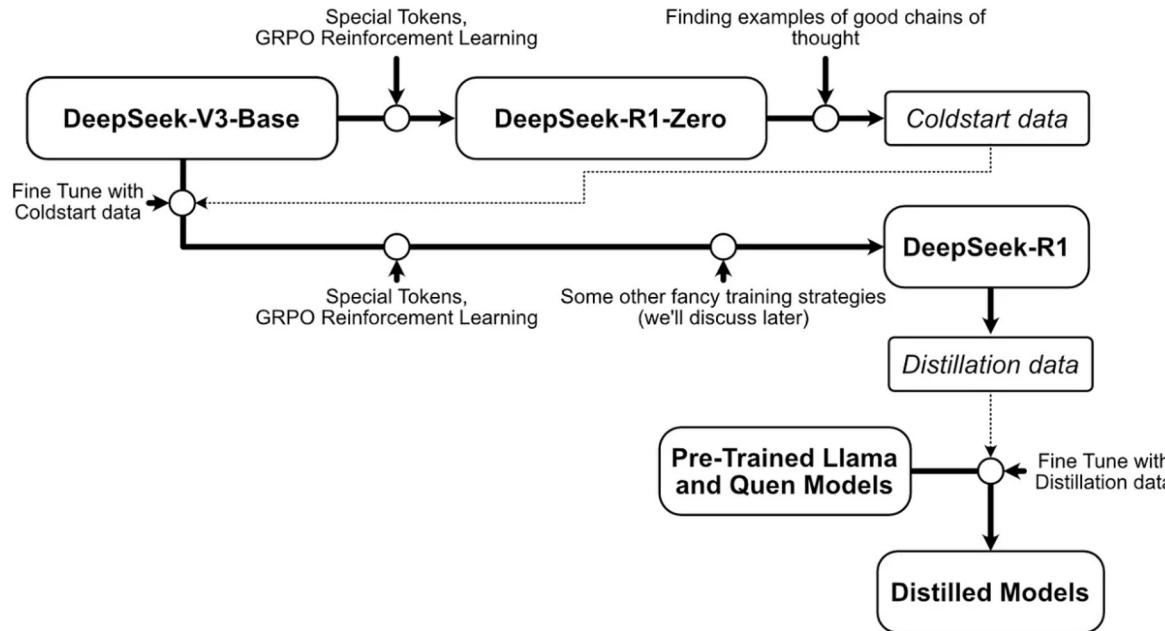
the process used to train DeepSeek-R1-Zero

Training Pipeline



The entire pipeline for training DeepSeek-R1

Training Pipeline



A snapshot of the entire training pipeline in the DeepSeek R1 paper.

Comparison

Aspect	OpenAI GPT	Meta LLaMa	DeepSeek(MoE)
Pre-Training Objective	Causal Language Modeling	Causal Language Modeling	Multi-stage training
Architectural Features	Dense Attention, LayerNorm	RMSNorm, SwiGLU, RoPE	MoE
Fine-Tuning	RLHF(SFT + PPO)	No RLHF	RL + SFT
Training Efficiency	High computational demand	No RLHF	Sparse activation (MoE-based)

Comparison

Feature	OpenAI GPT	Meta LLaMa	DeepSeek(MoE)
Architecture	Transformer (Decoder-only)	Transformer (Decoder-only)	Transformer + Mixture-of-Experts
Activation	Dense	Dense	Sparse
Efficiency	High resource consumption	Optimized for efficiency	Highly efficient
Specialization	Text generation	General purpose	Dynamic expert selection
Code Differences	Standard transformer fine- tuning	Similar to GPT	Requires gating mechanism

Nội dung

1. Giới thiệu DeepSeekV2
2. DeepSeekV3 - Bước đệm hoàn hảo
3. DeepSeekR1 - Công nghệ đột phá
4. Cơ hội áp dụng DeepSeekR1

Cơ hội áp dụng DeepSeekR1

Emerging Reasoning with Reinforcement Learning

Phần này tập trung vào quá trình khám phá việc “bắt chước” quá trình đào tạo DeepSeek-R1. Nó tập trung vào các mô hình nhỏ và dữ liệu hạn chế. Long Chain-of-Thought (CoT) và self-reflection.

Các thí nghiệm sử dụng mô hình **Qwen-2.5-Math-7B** chỉ với các ví dụ MATH 8K. Kết quả mạnh mẽ đáng ngạc nhiên về suy luận toán học phức tạp đã đạt được.

Code: <https://github.com/hkust-nlp/simpleRL-reason> - paper release soon

Cơ hội áp dụng DeepSeekR1

Những thách thức với LLM

1. Mô hình nhỏ có hiệu quả không?

- Các thí nghiệm trước đây chủ yếu dựa trên các mô hình lớn trong môi trường RL quy mô lớn. Tuy nhiên, vẫn chưa rõ liệu các mô hình nhỏ hơn có thể đạt được hành vi tương tự không.

1. Cần bao nhiêu dữ liệu?

- Câu hỏi đặt ra là lượng dữ liệu tối thiểu cần thiết để mô hình học được các mẫu lập luận và phản biện hiệu quả.

1. Kết quả như thế nào khi train mô hình nhỏ với ít dữ liệu?

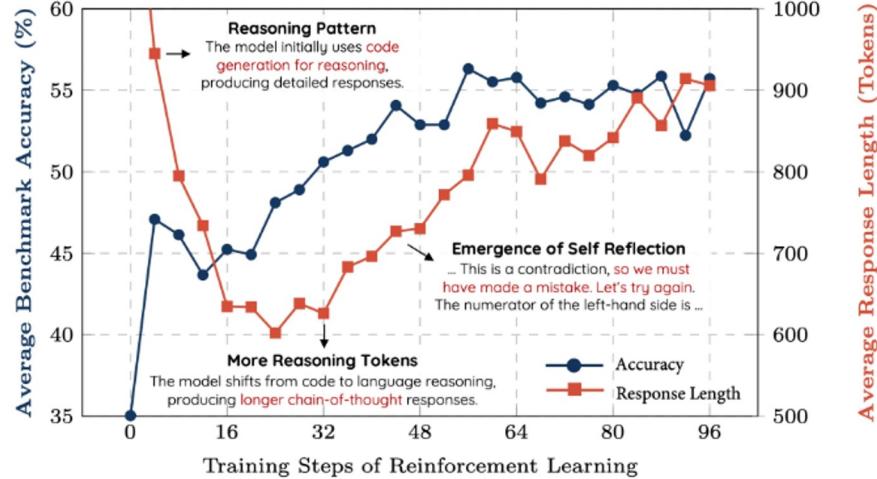
- Cần so sánh các kết quả định lượng từ phương pháp này với các phương pháp khác để đánh giá hiệu quả.

Cơ hội áp dụng DeepSeekR1

Emerging Reasoning with Reinforcement Learning: Contribution

- **Không cần reward model và SFT:** Khác với các phương pháp truyền thống, phương pháp này bỏ qua bước fine-tuning có giám sát (SFT) và reward model, mà chỉ sử dụng 8000 sample MATH để huấn luyện bằng RL.
- **Hiệu quả cao:** Mô hình được huấn luyện theo phương pháp này đạt được kết quả ấn tượng trên các bộ dữ liệu kiểm tra như AIME, AMC và MATH, thậm chí vượt trội hơn so với các mô hình sử dụng dữ liệu lớn hơn nhiều và các thành phần phức tạp hơn.
- **Khả năng khái quát hóa tốt:** Mô hình có khả năng khái quát hóa tốt từ dữ liệu huấn luyện dễ (MATH) sang các bài toán khó hơn (AIME, AMC).
- **Tái tạo các nghiên cứu trước:** Phương pháp này tái tạo thành công các nghiên cứu của DeepSeek-R1-Zero và DeepSeek-R1 trên các mô hình nhỏ hơn và dữ liệu giới hạn hơn.
- **Tính đơn giản:** Phương pháp này đơn giản nhưng hiệu quả, mở ra một hướng đi mới cho việc huấn luyện mô hình reasoning trên các tài nguyên hạn chế.

Emerging Reasoning with RL



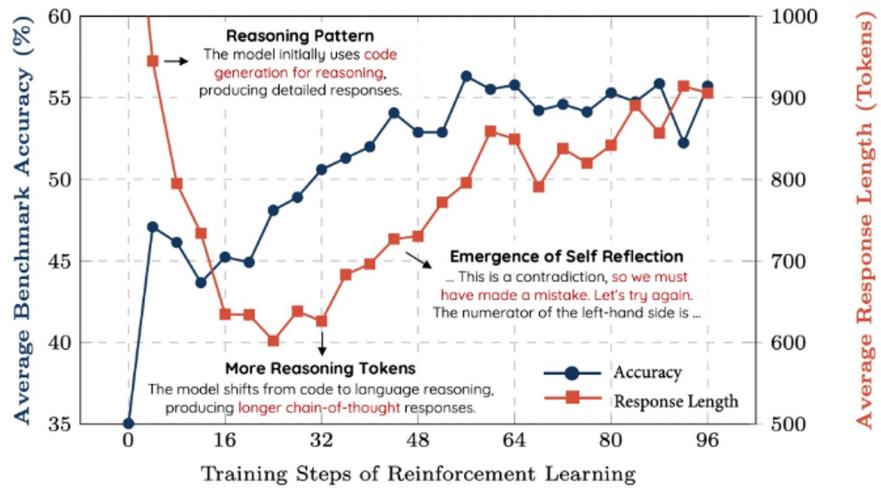
Kết luận:

- RL giúp mô hình chuyển từ việc lập luận bằng code sang ngôn ngữ tự nhiên, làm tăng hiệu quả lập luận và độ chính xác.
- Khả năng tự phản biện là một dấu hiệu quan trọng cho thấy mô hình đã học được cách kiểm tra và sửa lỗi trong lập luận, giúp nâng cao hiệu suất trên các bài toán phức tạp.

Hình này minh họa quá trình huấn luyện mô hình Qwen2.5-SimpleRL-Zero bằng phương pháp Reinforcement Learning (RL) từ mô hình cơ sở Qwen2.5-Math-7B. Biểu đồ thể hiện hai chỉ số chính trong suốt quá trình huấn luyện:

- **Độ chính xác trung bình (Average Benchmark Accuracy)** - Đường màu xanh lam, tỷ lệ phần trăm trên trục y bên trái.
- **Độ dài phản hồi trung bình (Average Response Length)** - Đường màu đỏ, tính bằng số token trên trục y bên phải.

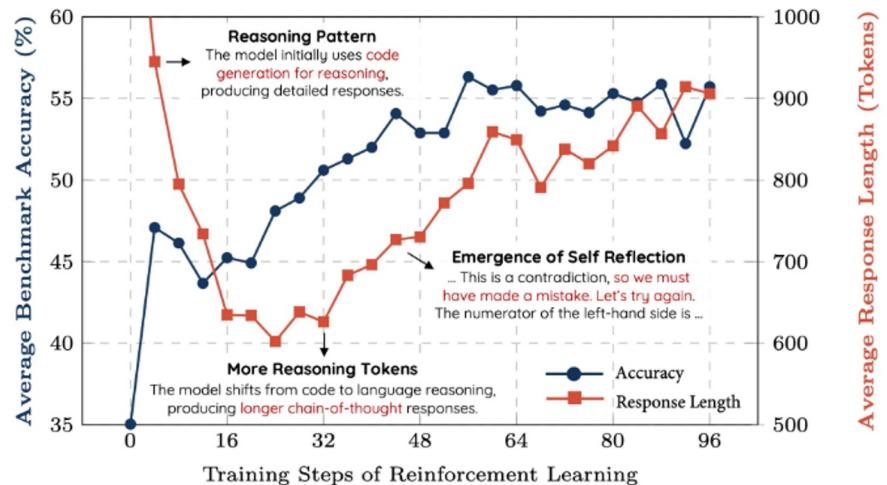
Emerging Reasoning with RL



Giai đoạn đầu (0 - 16 bước huấn luyện):

- Mô hình sử dụng mẫu lập luận bằng code:** Lúc đầu, mô hình có xu hướng tạo ra các đoạn mã để lập luận, dẫn đến phản hồi dài (khoảng 1000 tokens) nhưng độ chính xác không cao (~45%).
- Sự sụt giảm độ dài phản hồi:** Qua một vài bước huấn luyện, RL nhanh chóng điều chỉnh hành vi này, khiến mô hình giảm độ dài phản hồi xuống dưới 600 tokens.

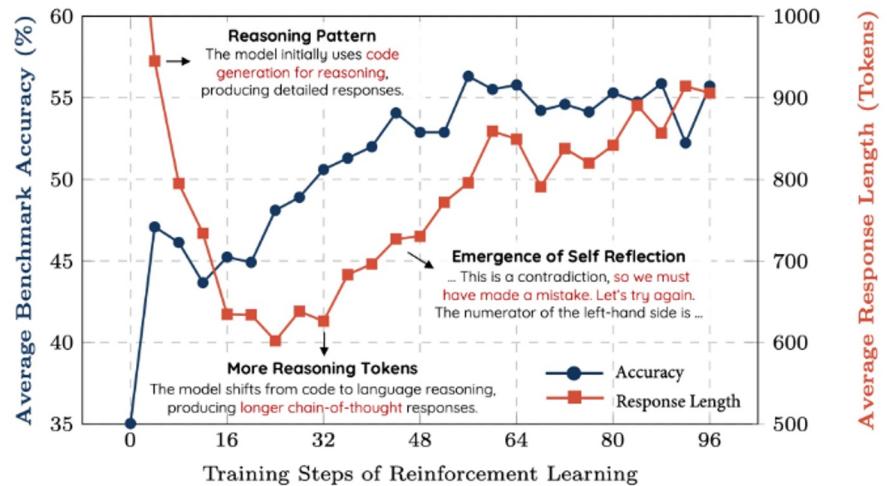
Emerging Reasoning with RL



Giai đoạn chuyển đổi (16 - 48 bước huấn luyện):

- Chuyển từ code sang lập luận ngôn ngữ:** Mô hình bắt đầu tạo ra các phản hồi dưới dạng ngôn ngữ tự nhiên thay vì mã code, đồng thời phản hồi có xu hướng dài hơn nhờ việc áp dụng chiến lược "chain-of-thought" (suy luận từng bước).
- Độ chính xác tăng dần đều lên khoảng 50% trong khi độ dài phản hồi cũng tăng lên hơn 700 tokens.**

Emerging Reasoning with RL



Giai đoạn hoàn thiện (64 - 96 bước huấn luyện):

- **Sự xuất hiện của khả năng tự phản biện (Emergence of Self Reflection):** Mô hình bắt đầu nhận diện và tự sửa lỗi trong quá trình lập luận, ví dụ như "Đây là một mâu thuẫn, chúng ta phải mắc lỗi ở đâu đó". Điều này giúp tăng đáng kể độ chính xác lên gần 55%.
- **Độ dài phản hồi ổn định:** Phản hồi tiếp tục dài hơn, dao động quanh 900 tokens, cho thấy mô hình đã tối ưu hóa việc giải thích chi tiết nhưng vẫn duy trì sự ngắn gọn cần thiết.

Emerging Reasoning with RL

Những kỹ thuật được sử dụng thực nghiệm đánh giá trong giải pháp này

1. **Distillation (Chung cất mô hình)**: Quá trình tối ưu hóa một mô hình lớn thành mô hình nhỏ hơn nhưng vẫn giữ được hiệu suất cao.
2. **MCTS (Monte Carlo Tree Search)**: Thuật toán tìm kiếm cây Monte Carlo, thường được sử dụng trong các bài toán ra quyết định phức tạp.
3. **Process-based reward models**: Hệ thống phần thưởng đánh giá quá trình lập luận thay vì chỉ kết quả cuối cùng.
4. **Reinforcement Learning**: Phương pháp huấn luyện mô hình thông qua việc tối ưu hóa phần thưởng dựa trên phản hồi từ môi trường.

Emerging Reasoning with RL

The Simple RL Recipe

1 PPO Algorithm

Tương tự như DeepSeek R1, công thức RL được giữ cực kỳ đơn giản. Nó không bao gồm các mô hình phần thưởng hoặc các kỹ thuật giống như MCTS.

1 Rule-Based Reward Function

Phần thưởng được chỉ định dựa trên hình thức và tính chính xác. +1 cho câu trả lời cuối cùng đúng. -0.5 cho câu trả lời sai. -1 cho không có câu trả lời cuối cùng.

3 Implementation

Việc thực hiện dựa trên [OpenRLHF](#). Chức năng phần thưởng này giúp mô hình chính sách hội tụ theo hướng phản hồi mong muốn.

Emerging Reasoning with RL

Bảng kết quả (pass@1 accuracy):

Mô hình	AIME 2024	MATH 500	AMC	Minerva Math	Olympia dBench	Trung bình (Avg.)
Qwen2.5-Math-7B-Base	16.7	52.4	52.5	12.9	16.4	30.2
Qwen2.5-Math-7B-Base + 8K MATH SFT	3.3	54.6	22.5	32.7	19.6	26.5
Qwen2.5-Math-7B-Instruct	13.3	79.8	50.6	34.6	40.7	43.8
Llama-3-1.7B-Instruct	16.7	64.6	30.1	35.3	31.9	35.7
rStar-Math-7B	26.7	78.4	47.5	38.0	41.4	46.4
Eurus-2-7B-PRIME	26.7	79.2	57.8	38.8	46.1	48.9
Qwen2.5-7B-SimpleRL-Zero	33.3	77.2	62.5	35.3	37.6	48.0
Qwen2.5-7B-SimpleRL	26.7	82.4	59.4	39.7	43.3	50.9

1. Hiệu suất của mô hình cơ sở (Qwen2.5-Math-7B-Base)

- Đạt độ chính xác trung bình **30.2%**, hiệu suất thấp trên các bài toán phức tạp như **Minerva Math (12.9%)** và **Olympia dBench (16.4%)**.
- Sau khi bổ sung thêm 8K ví dụ từ bộ MATH với **SFT (Supervised Fine-Tuning)**, hiệu suất không được cải thiện rõ rệt (26.5% trung bình), cho thấy SFT đơn thuần không đủ để nâng cao khả năng lập luận toán học.

Emerging Reasoning with RL

Bảng kết quả (pass@1 accuracy):

Mô hình	AIME 2024	MATH 500	AMC	Minerva Math	Olympia dBench	Trung bình (Avg.)
Qwen2.5-Math-7B-Base	16.7	52.4	52.5	12.9	16.4	30.2
Qwen2.5-Math-7B-Base + 8K MATH SFT	3.3	54.6	22.5	32.7	19.6	26.5
Qwen2.5-Math-7B-Instruct	13.3	79.8	50.6	34.6	40.7	43.8
Llama-3-1.7B-Instruct	16.7	64.6	30.1	35.3	31.9	35.7
rStar-Math-7B	26.7	78.4	47.5	38.0	41.4	46.4
Eurus-2-7B-PRIME	26.7	79.2	57.8	38.8	46.1	48.9
Qwen2.5-7B-SimpleRL-Zero	33.3	77.2	62.5	35.3	37.6	48.0
Qwen2.5-7B-SimpleRL	26.7	82.4	59.4	39.7	43.3	50.9

2. Tác động của Reinforcement Learning (RL):

- **Qwen2.5-7B-SimpleRL-Zero** đạt hiệu suất cao hơn đáng kể (**48% trung bình**), đặc biệt trên **AMC (62.5%)** và **MATH 500 (77.2%)**.
- **Qwen2.5-7B-SimpleRL** (phiên bản đầy đủ RL) đạt hiệu suất cao nhất với **50.9%** trung bình, chứng minh RL có hiệu quả vượt trội trong việc cải thiện khả năng lập luận toán học.

Emerging Reasoning with RL

Bảng kết quả (pass@1 accuracy):

Mô hình	AIME 2024	MATH 500	AMC	Minerva Math	Olympia dBench	Trung bình (Avg.)
Qwen2.5-Math-7B-Base	16.7	52.4	52.5	12.9	16.4	30.2
Qwen2.5-Math-7B-Base + 8K MATH SFT	3.3	54.6	22.5	32.7	19.6	26.5
Qwen2.5-Math-7B-Instruct	13.3	79.8	50.6	34.6	40.7	43.8
Llama-3-1.7B-Instruct	16.7	64.6	30.1	35.3	31.9	35.7
rStar-Math-7B	26.7	78.4	47.5	38.0	41.4	46.4
Eurus-2-7B-PRIME	26.7	79.2	57.8	38.8	46.1	48.9
Qwen2.5-7B-SimpleRL-Zero	33.3	77.2	62.5	35.3	37.6	48.0
Qwen2.5-7B-SimpleRL	26.7	82.4	59.4	39.7	43.3	50.9

3. So sánh với các mô hình khác:

- rStar-Math-7B và Eurus-2-7B-PRIME cũng cho kết quả tốt (46.4% và 48.9%), nhưng vẫn thấp hơn so với **Qwen2.5-7B-SimpleRL**.
- Llama-3-1.7B-Instruct có hiệu suất trung bình chỉ 35.7%, cho thấy mô hình nhỏ hơn hoặc không được tối ưu hóa bằng RL sẽ kém hiệu quả hơn trong các bài toán phức tạp.