DeepSeek-V3

March 13
Xupeng Chen

https://arxiv.org/pdf/2412.19437v1

DeepSeek V3

Amazingly **Efficient** And **Effective**

Costs amount to only $5.576M to train. And Very low inference cost

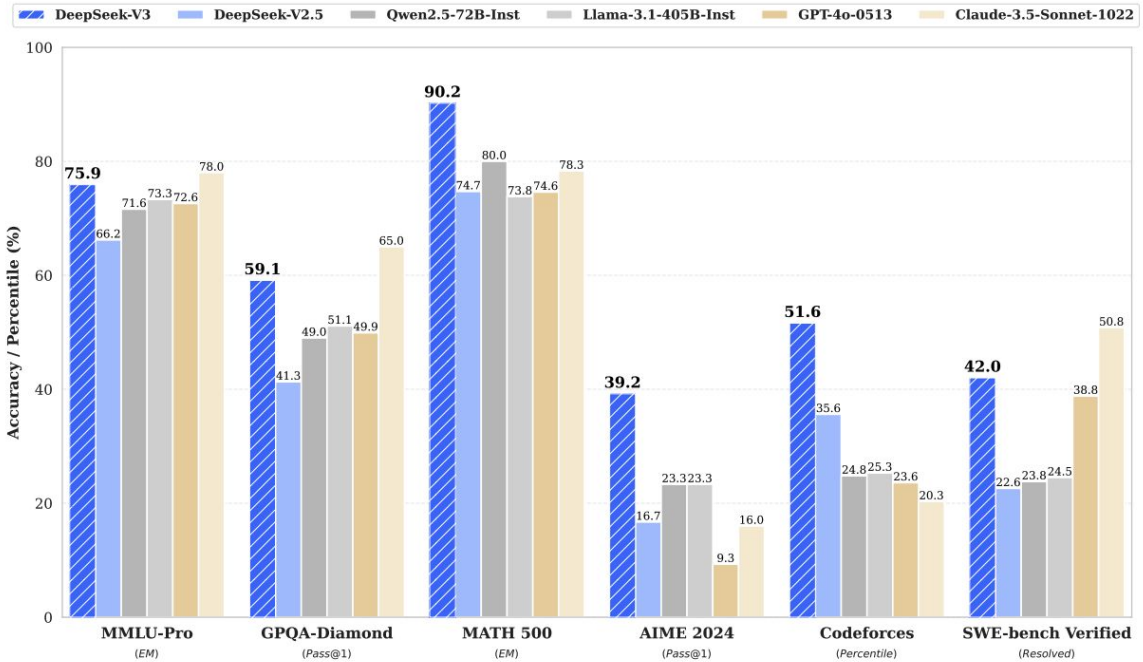
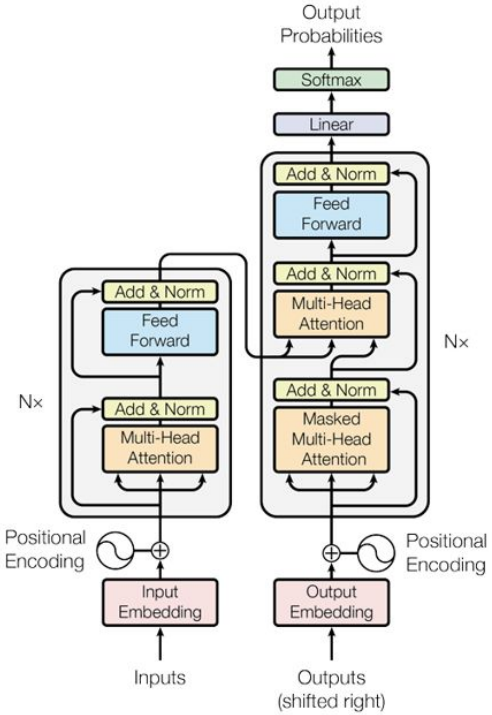


Figure 1 | Benchmark performance of DeepSeek-V3 and its counterparts.

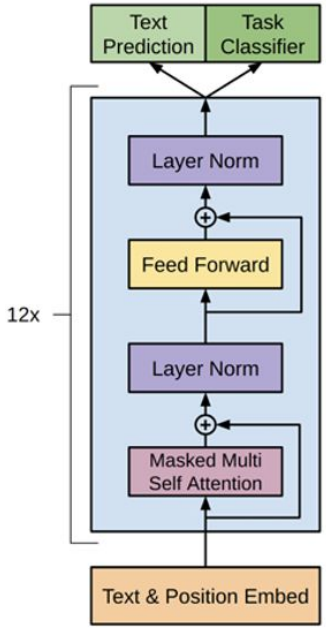
DeepSeek upgrade Algo and HardWare

We will focus on algorithm level innovations

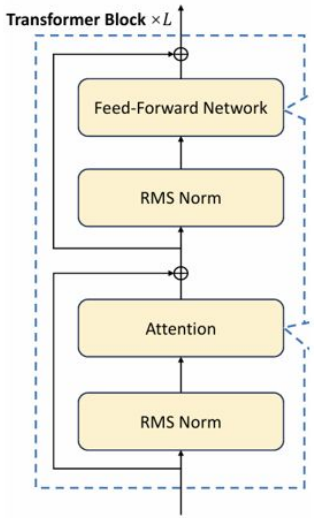
Some (redundant) preliminaries…



(a) encoder-decoder in Transformer.

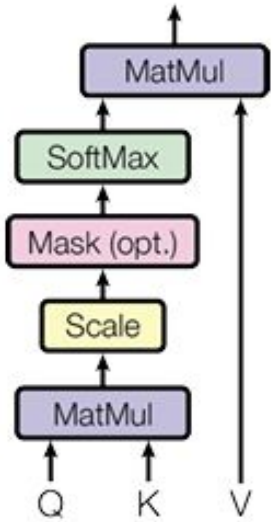


(b) decoder-only Transformer in GPT.

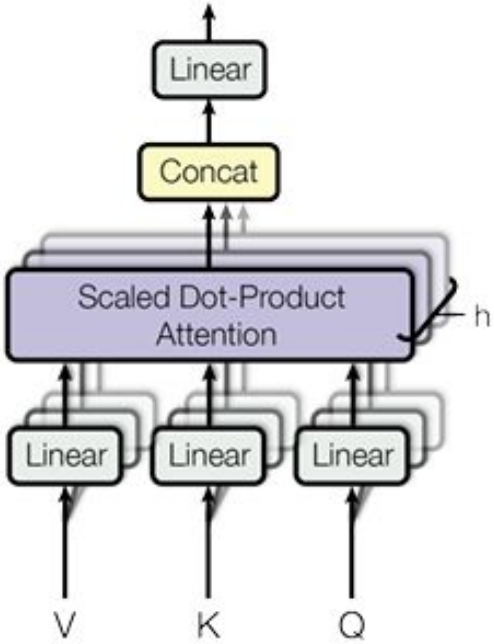


(c) Optimized decoder-only Transformer.

Scaled Dot-Product Attention



Multi-Head Attention



Multi-Head Attention Mechanism Summary

Step 1: Linear projection

Input vector h_t is projected into Query (q_t), Key (k_t), and Value (v_t) vectors:

$$q_t = W^Q h_t, \quad k_t = W^K h_t, \quad v_t = W^V h_t$$

where projection matrices:

$$W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}, \quad q_t, k_t, v_t \in \mathbb{R}^{d_h n_h}$$

Step 2: Split into multiple heads

$$q_t = [q_{t,1}; q_{t,2}; \ldots; q_{t,n_h}], \quad k_t = [k_{t,1}; k_{t,2}; \ldots; k_{t,n_h}], \quad v_t = [v_{t,1}; v_{t,2}; \ldots; v_{t,n_h}]$$

Step 3: Scaled dot-product attention (for each head i):

$$o_{t,i} = \sum_{j=1}^{t} \text{Softmax} j \left(\frac{qt, i^\top k_{j,i}}{\sqrt{d_h}} \right) v_{j,i}$$

Step 4: Concatenate and project back

Concatenate outputs from all heads and apply another linear transformatic

$$u_t = W^O [o_{t,1}; o_{t,2}; \ldots; o_{t,n_h}]$$

Here, W^O maps from dimension $(n_h \cdot d_h)$ back to d.

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

class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, n_heads):
        super().__init__()
        self.n_heads = n_heads
        self.d_k = d_model // n_heads
        self.W_QKV = nn.Linear(d_model, 3 * d_model)
        self.W_O = nn.Linear(d_model, d_model)

    def forward(self, x):
        B, T, _ = x.size()
        q, k, v = self.W_QKV(x).chunk(3, dim=-1)
        q, k, v = [m.view(B, T, self.n_heads, self.d_k)
                   .transpose(1, 2) for m in (q, k, v)]
        attn = (q @ k.transpose(-2, -1)) / math.sqrt(self.d_k)
        attn = F.softmax(attn, dim=-1)
        out = (attn @ v).transpose(1, 2).contiguous()
        out = out.view(B, T, -1)
        return self.W_O(out)
```

Key-Value Cache (KV Cache)

LLM in **training**: parallel attention matrix calculation. In **infer**: one by one

Purpose: Speeds up **autoregressive** decoding by reusing cached keys (K) and values (V) instead of recomputing them.

How:

- Compute only the new query (Q).
- Reuse cached K & V from previous steps.
- Append newly computed K & V to the cache for future use.

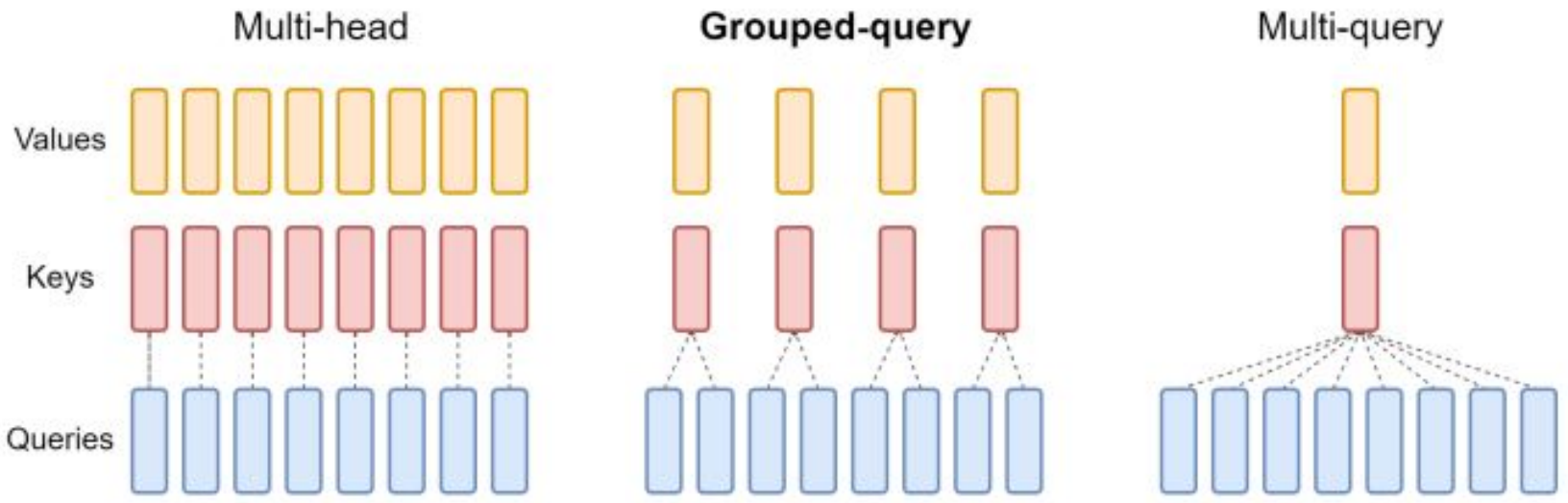
Trade-off: Increased memory usage (batch × seq_len × hidden_size × heads), avoid repeat calculation of K and V.

Solutions: Multi-Query Attention, Grouped-Query Attention reduce memory overhead.

Reduce heads of K and V

MHA, GQA, MQA

GQA widely used, (e.g.: Llama)



RoPE: Rotary Positional Embeddings

Absolute Position Embedding

❌ Poor generalization to unseen sequence lengths

Relative Position Embedding

❌ Extra parameters, computationally expensive

RoPE's Advantages

✅ Generalizes naturally to longer sequences

✅ Parameter-efficient (no extra params)

✅ Stable positional encoding via rotation operations

RoPE: Rotary Positional Embeddings

Motivation: Encode relative positions naturally by rotating vectors.

Represent query/key vectors as complex numbers, rotating them

by angles proportional to their positions

$$f_q(x_m, m) = (W_q x_m) e^{im\theta}, \quad f_k(x_n, n) = (W_k x_n) e^{in\theta}$$

- **Rotation matrix form:**

$$f_q(x_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} W_q x_m$$

- **Relative positional attention:**

$$g(x_m, x_n, m-n) = \mathrm{Re}\left[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}\right]$$

$$q^T k$$
$$(R_q q)^T (R_k k)$$
$$q^T R_q^T R_k k$$
$$q^T (R_q^T R_k) k$$
$$q^T R k$$

1 DeepSeek's new techniques

Multi-head **Latent** Attention: Compress the **KV cache**!

$$q_t = W^Q h_t, \quad k_t = W^K h_t, \quad v_t = W^V h_t$$

where projection matrices:

$$W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}, \quad q_t, k_t, v_t \in \mathbb{R}^{d_h n_h}$$

$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t,$$

$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; ...; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q,$$

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

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

$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV}\mathbf{h}_t,$$

$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; ...; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV},$$

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

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

$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; ...; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV},$$

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

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

Compress attention input h_t to c_t^{KV} through W^{DKV} (down KV). This reduce KV cache!
Could also shrink query.
Need extra treat of RoPE.

MLA continued

$$c_t^{KV} = W^{DKV} h_t$$

$$k_t^C = W^{UK} c_t^{KV} = W^{UK} W^{DKV} h_t$$

$$c_t^Q = W^{DQ} h_t$$

$$q_t^C = W^{UQ} c_t^Q = W^{UQ} W^{DQ} h_t$$

$$(q_t^C)^T k_t^C = (c_t^Q)^T (W^{UQ})^T W^{UK} c_t^{KV}$$

Weight matrix absorbed in implementation
C_t reduce cache storage

Decoupled RoPE: W could not be absorbed since R is inserted.

R depends on q and k position.

$$(q_t^C)^T R k_t^C = (c_t^Q)^T (W^{UQ})^T R W^{UK} c_t^{KV}$$

Decouple RoPE with Latent

Simple Solution:

Add extra heads for RoPE:

q_t^R and k_t^R

$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t,$$
$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; ...; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q,$$
$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; ...; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR}\mathbf{c}_t^Q),$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],$$

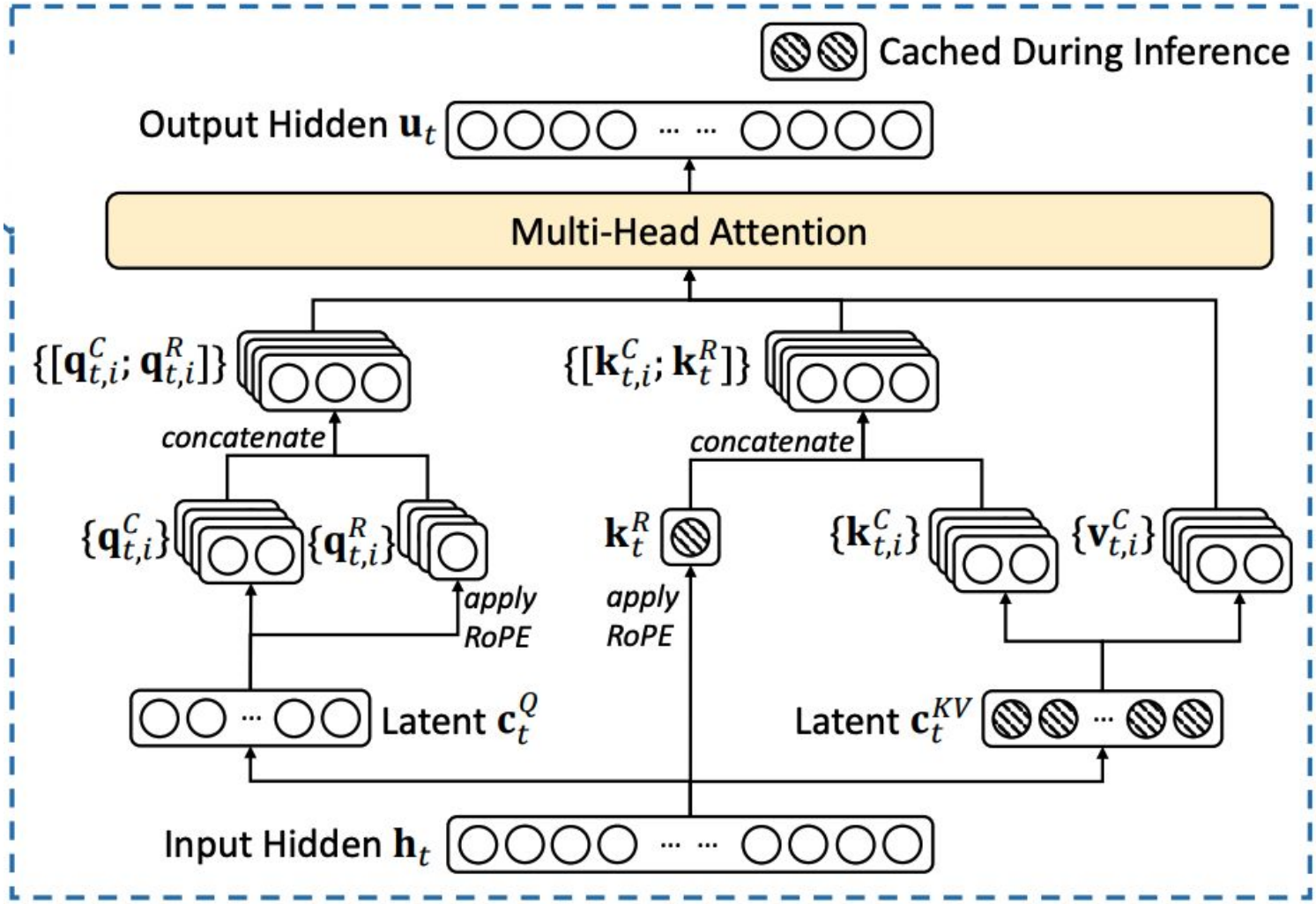
$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV}\mathbf{h}_t,$$
$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; ...; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV},$$

$$\boxed{\mathbf{k}_t^R} = \mathrm{RoPE}(W^{KR}\mathbf{h}_t),$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R],$$

$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; ...; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV},$$

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right)\mathbf{v}_{j,i}^C,$$
$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; ...; \mathbf{o}_{t,n_h}],$$

Multi-Head Latent Attention (MLA)



So MLA is efficient but keeps good capacity

Attention Mechanism	KV Cache per Token (# Element)	Capability
Multi-Head Attention (MHA)	$2n_h d_h l$	Strong
Grouped-Query Attention (GQA)	$2n_g d_h l$	Moderate
Multi-Query Attention (MQA)	$2d_h l$	Weak
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2} d_h l$	Stronger

From DeepSeek V2 https://arxiv.org/pdf/2405.04434

DeepSeek’s Mixture of Experts

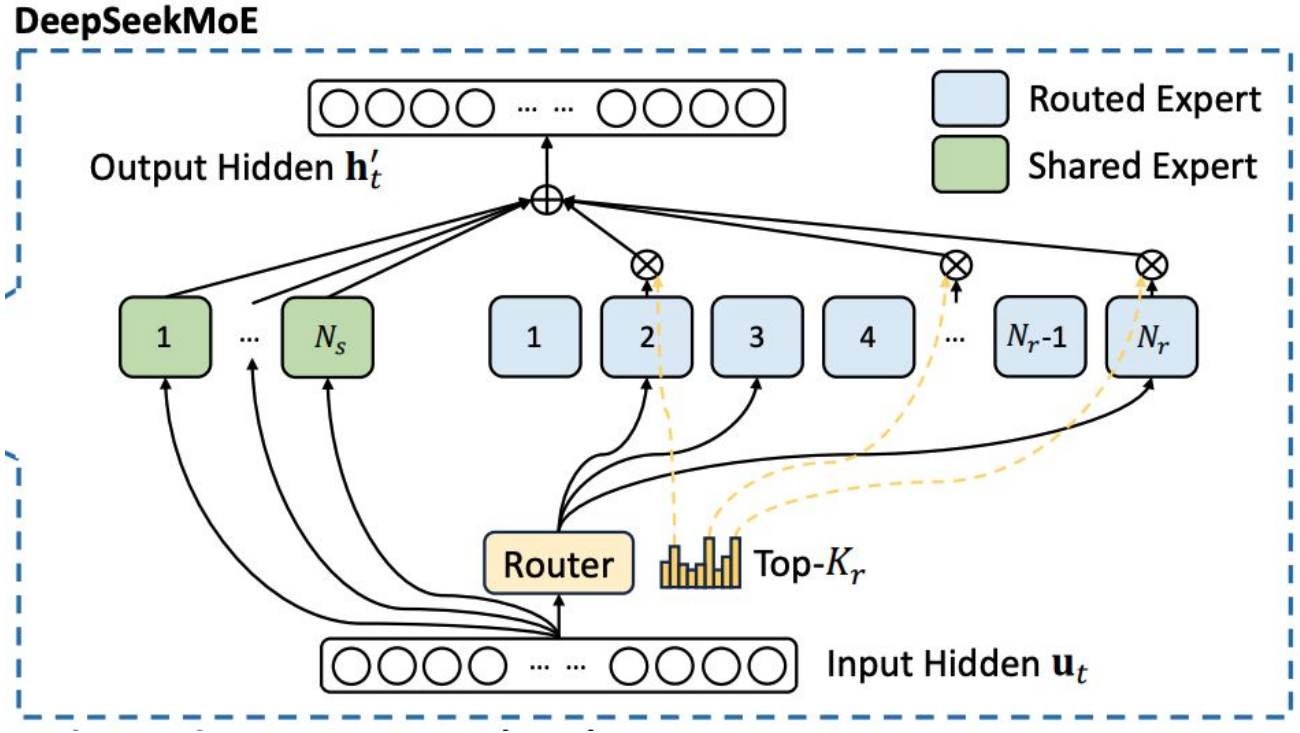
671B total parameters

with **37B** activated

for each token

Great capacity in training

Efficient in Infer



Feed-Forward Network

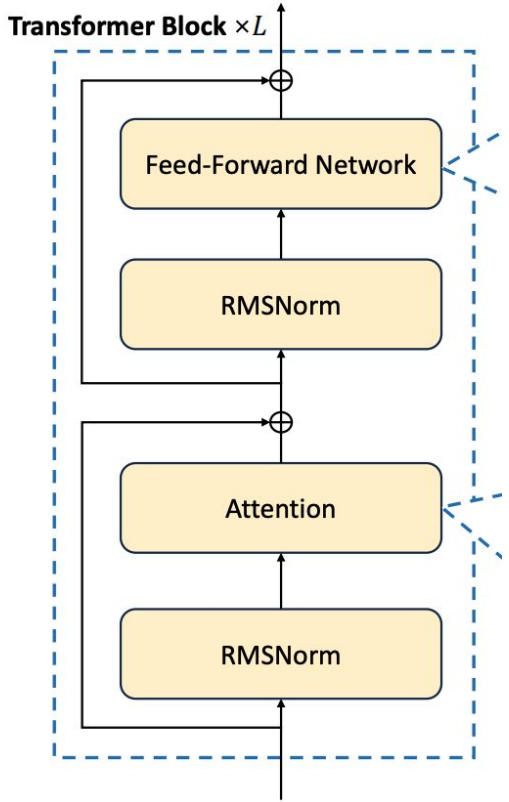
$$\mathbf{h}_t' = \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}(\mathbf{u}_t),$$

Attention: most computation heavy

FFN: most parameters (> 70%), less computational heavy,

Knowledge stored here

Compute somewhere, store elsewhere (Brain?)



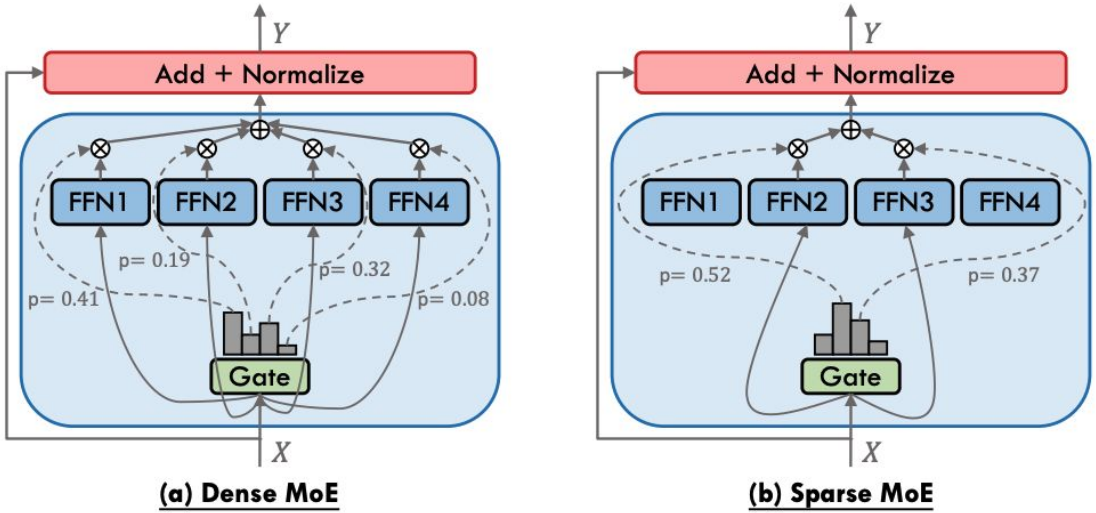
2Mixture of Experts

Experts: split FFN into groups (feature dimension)

Dense MoE: weight the group's output

Sparse MoE: select Top K FFN

Gate could be a MLP



Issues of MoE

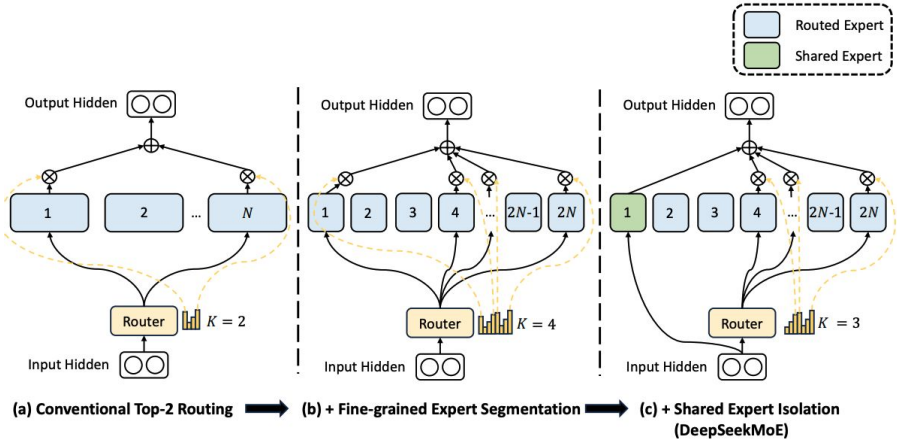
Experts role overlap: redundant

We want different experts have different role: coding, math, daily, etc

Idea: need ensembling of more small models, number and quality tradeoff

DeepSeekMOE:

- Specialist: Split experts to finer granularity
  - Increase FFN numbers
  - Each FFN decrease feat dimension
- Generalist: Make some experts shared
  - Common sense experts
  - Always chosen



From DeepSeekMoE https://arxiv.org/pdf/2401.06066

Load balancing

After initialization

Self-reinforce: Router always assign token to some “better” experts.

the rich get richer and the poor get poorer

Ensemble is meaningless

Loss to control load balancing

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i$$

where f_i is the fraction of tokens dispatched to expert i,

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1}\{\text{argmax}\, p(x) = i\}$$

and P_i is the fraction of the router probability allocated for expert i,

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x).$$

In a very large batch size, make actual allocation and probability assigned to each expert balance. Min achieved when uniformly routed.

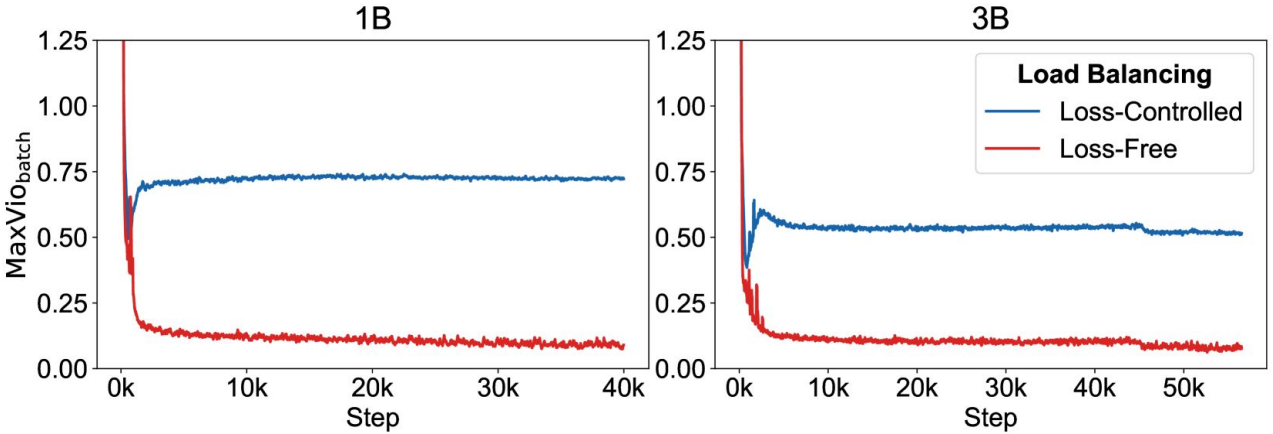
From Switch Transformer https://arxiv.org/pdf/2101.03961

Concern with loss control

Might be wrong??

(0.5, 0.5) (0.5, 0.5) vs (0.1, 0.9) (0.9, 0.1)

It is worse than DeepSeek's loss-free balancing



From DeepSeek auxiliary loss free load balancing https://arxiv.org/html/2408.15664v1

Loss free load balancing

Add loss might influence the model performance.

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

Add a bias term b_i

If in one batch, expert i is overload, decrease the bias term by gamma to reduce the prob it is assigned

u_t: input to FFN

e_i: centroid of expert i

DeepSeek has good load balancing so

They avoid token dropping (skip connection)

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

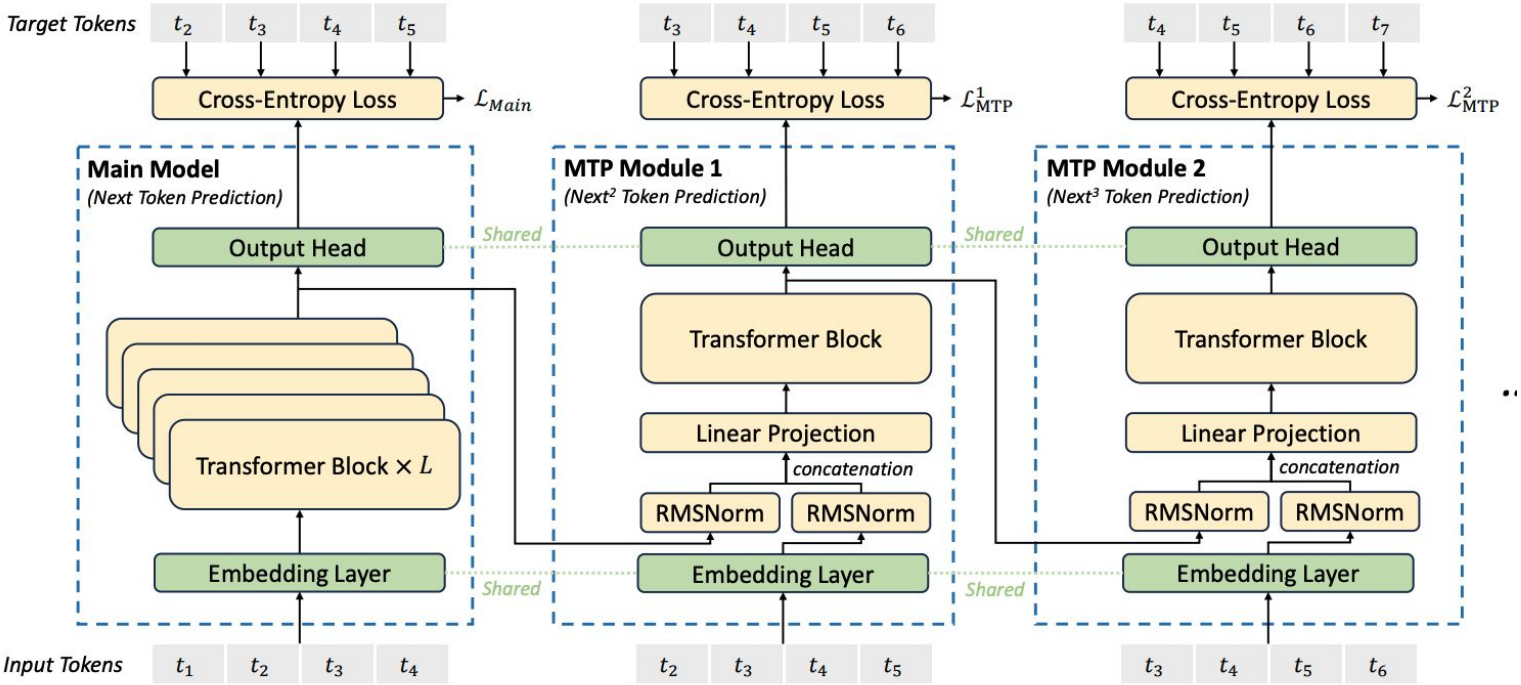
$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}},$$

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

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

3 Multi Token Prediction

Key technique to reduce cost and increase performance



Prior works: how to predict next tokens

LLM Training: Meta MTP: Parallel Heads + Training

LLM Inference → Speculative Decoding

Independent: Google/DeepMind

Self: Medusa (Parallel heads), EAGLE (Causal heads)

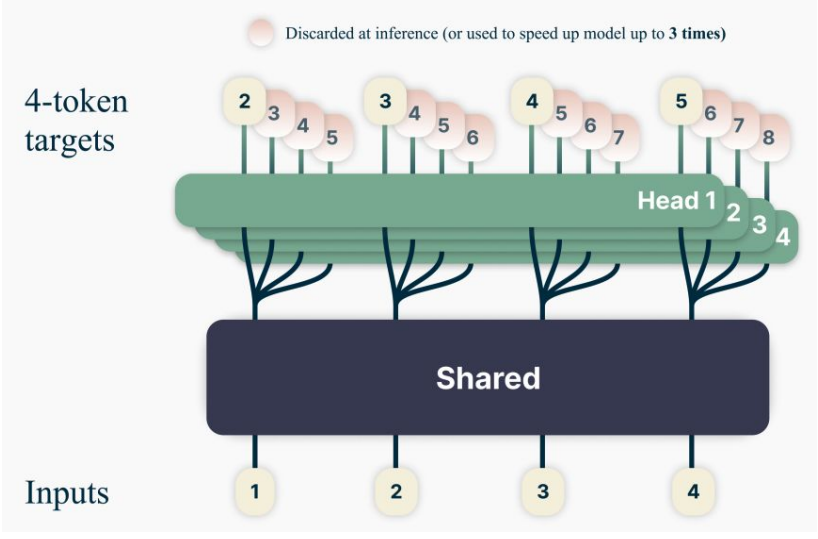
DeepSeek MTP: causal heads for future token prediction

Multi Token Prediction

During training: parallel, next token prediction with teacher forcing

A causal mask make sure only see previous ground truth tokens.

Lacks the **planning** ability → if we predict multi tokens…



Group heads and predict future 4 tokens
Training signals * 4
Planning ability, learn hard transition, useful in inference when no teacher forcing

Meta MTP https://arxiv.org/pdf/2404.19737

MTP helps predict hard transition

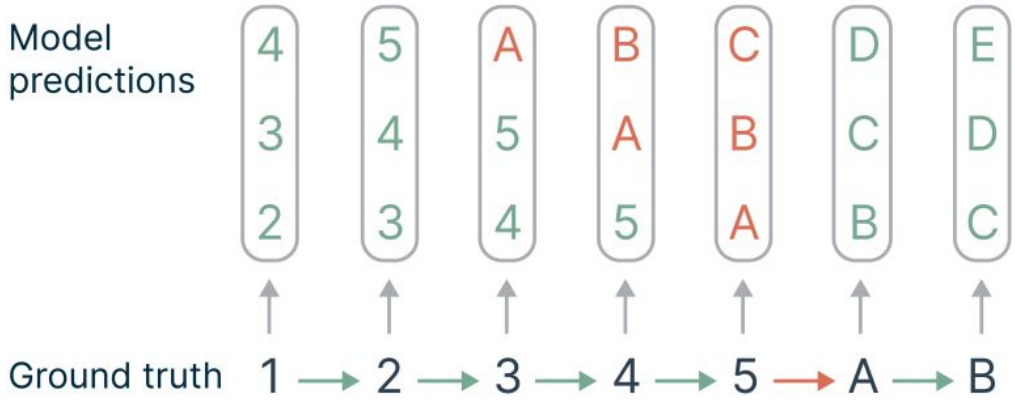
The hard transition is 5→A

In next token prediction case: **1/7** weights are put in hard transition learning

In next 3 tokens prediction: 3→A, 4→A, 4→B, 5→A, 5→B, 5→C. **2/7** weights

Training efficiency

Reasoning ability



LLM Inference: Speculative Decoding

Inference is so **SLOW**!

Big model, large KV cache, high memory consumption, sequential predicting.

Small model, fast but stupid

Speculative Decoding: Ask juniors do some easy task and take over if they are wrong

Step1 Quick Guess: A small model predicts 3-5 tokens.

Step2 Cheap Verification: The big model accepts or corrects them

Speculative Decoding

Step 1: a small 7B LLM proposes

Independent: another small LLM

Self: use part of the heads within big model

Step2: the big 70B LLM verifies

The predicted 5 tokens could be fed into the big LLM and compute final probability parallelly. If it is biggest, accept, if now, big LLM recompute the wrong token.

Could reduce the forward pass rounds, save time! (not saving compute)

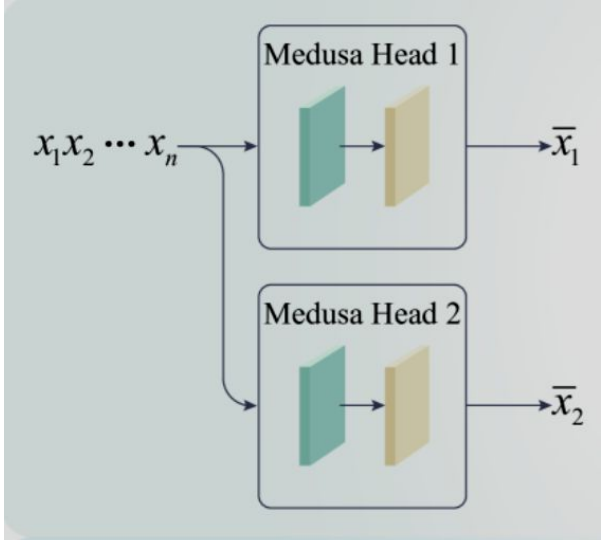
https://arxiv.org/abs/2211.17192

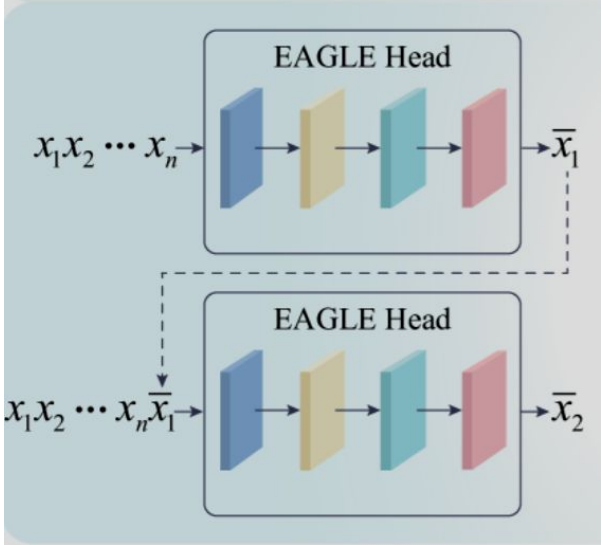
Self Speculative Decoding

Use part of the heads within big model

Parallel heads

Autoregressive(causal) heads (better)





https://arxiv.org/html/2408.08146v1

DeepSeek's MTP

Putting things together:

Predict multiple future tokens, with causal heads. I-th token's feature concat with i+1 token's embedding to predict i+2 token…

During inference, only main model needed. And the feature contains planning information!

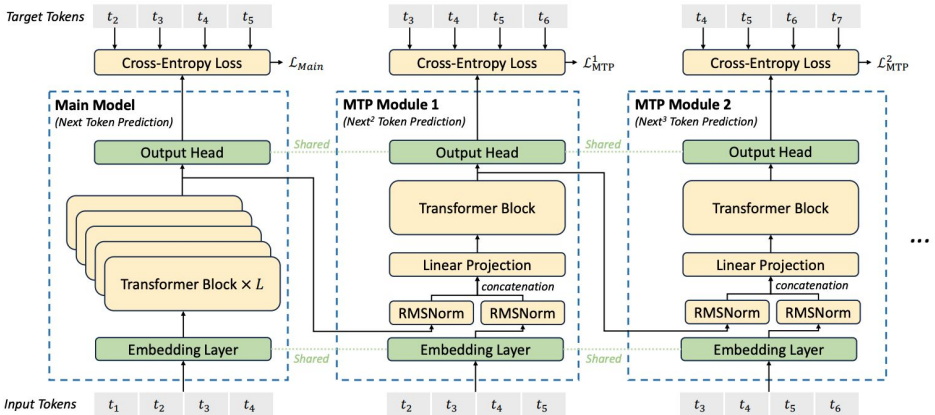
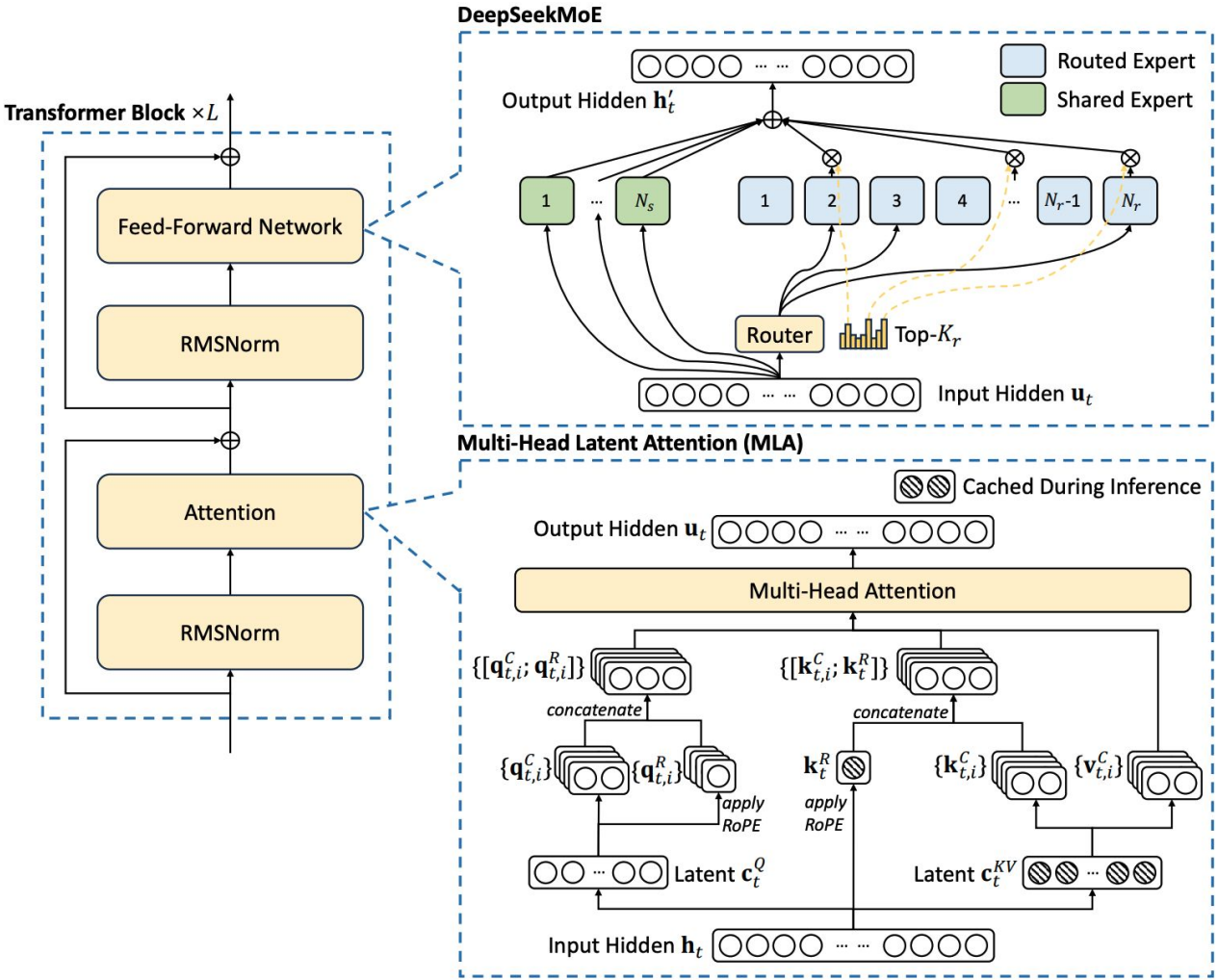


Figure 3 | Illustration of our Multi-Token Prediction (MTP) implementation. We keep the complete causal chain for the prediction of each token at each depth.


Transformer Block ×L
Feed-Forward Network
RMSNorm
Attention
RMSNorm
DeepSeekMoE
Routed Expert
Shared Expert
Output Hidden $\mathbf{h}'_t$
1
...
N_s
1
2
3
4
...
N_r-1
N_r
Router
Top-K_r
Input Hidden $\mathbf{u}_t$
Multi-Head Latent Attention (MLA)
Cached During Inference
Output Hidden $\mathbf{u}_t$
Multi-Head Attention
$\{[\mathbf{q}^C_{t,i}; \mathbf{q}^R_{t,i}]\}$
$\{[\mathbf{k}^C_{t,i}; \mathbf{k}^R_t]\}$
concatenate
concatenate
$\{\mathbf{q}^C_{t,i}\}$
$\{\mathbf{q}^R_{t,i}\}$
apply RoPE
$\mathbf{k}^R_t$
apply RoPE
$\{\mathbf{k}^C_{t,i}\}$
$\{\mathbf{v}^C_{t,i}\}$
Latent $\mathbf{c}^Q_t$
Latent $\mathbf{c}^{KV}_t$
Input Hidden $\mathbf{h}_t$