# MoE&LLM

稀疏结构大模型详解

# Vanilla MoE

Export Network，用于学习不同数据，一个Gating Network用于分配每个Expert的输出权重。对于一个输入样本c，第i个 expert 的输出为 $o_c^i$，Ground truth是$d^c$，则损失函数为：

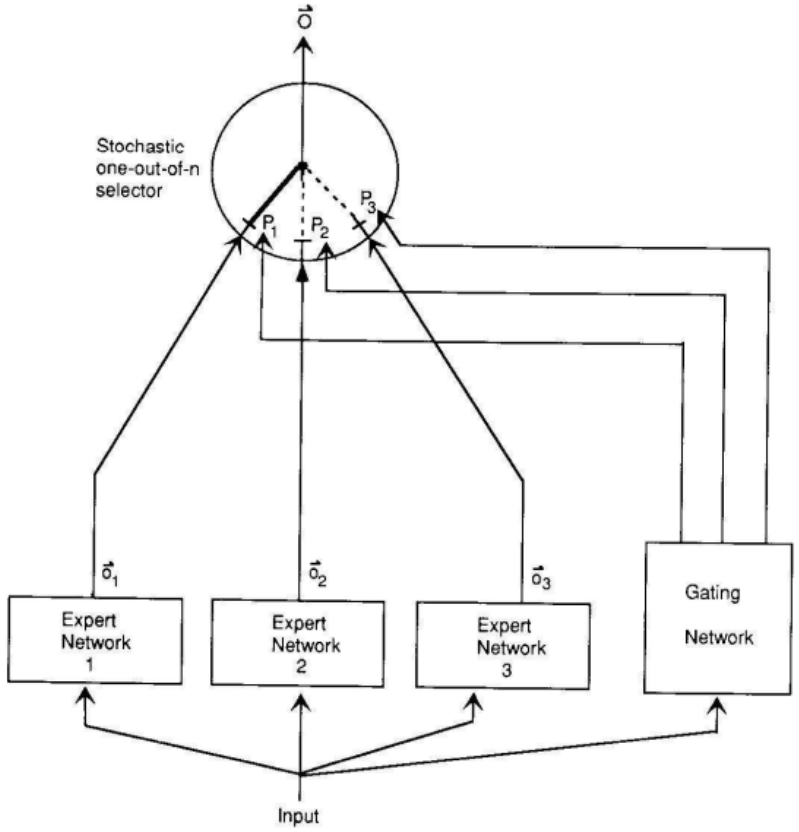
- $E^c = \left|\left|d^c - \sum_i p_i^c o_i^c\right|\right|^2$

其中$p_i^c$为Gating Network 分配给每个 expert 的权重，此时MoE本质上还是在做Ensemble

相当于利用多个Expert共同的获得当前样本的输出。这样的设计方式很自然，但是也会造成不同Expert间相互影响的问题，因此Jacobs, Robert A.，等人在此基础上进行了改进，将损失函数改造为：

- $E^c = \sum_i p_i^c \left|\left|d^c - o_i^c\right|\right|^2$

将$p_i^c$提前，使得每个expert单独计算损失函数，鼓励不同Expert的竞争，**使得每个数据样本尽可能被一个Expert处理**。




Jacobs, Robert A., et al. "Adaptive mixtures of local experts." Neural computation 3.1 (1991): 79-87.

# Sparse MoE

设$G(x)$和 分别是$E_i(x)$分别是 gating network 和第$i$个 expert 的输出，对于在当前position的输入$x$，输出是experts 的加权和：

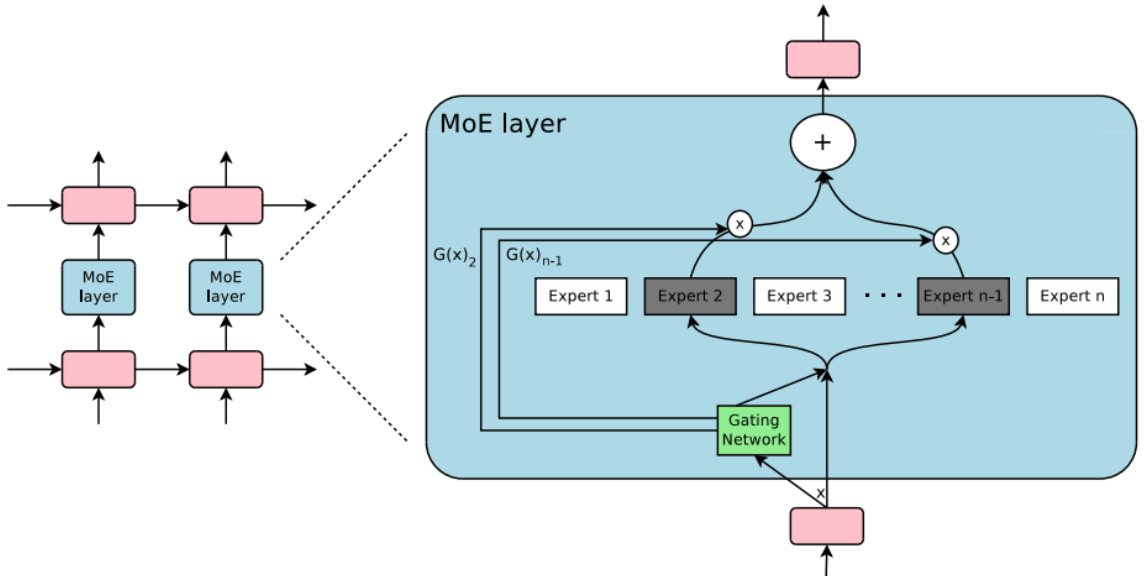
$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

$G(x)$的输出是稀疏的，只有部分的 experts 的权重> 0，其余=0 的 expert 直接不参与计算

$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i, & if\ v_i\ in\ topK\ elements. \\ -\infty, & otherwise. \end{cases}$$




Shazeer, Noam, et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." arXiv preprint arXiv:1701.06538 (2017).

# Expert Balancing

作者在实验中发现，不同 experts 在竞争的过程中，会出现**“赢者通吃”的现象**：前期变现好的 expert 会更容易被 gating network 选择，导致最终只有少数的几个 experts 真正起作用。因此作者额外增加了一个 loss，来缓解这种不平衡现象，公式如下：

$$Importance(X) = \sum_{x \in X} G(x)$$

$$L(X) = \lambda \cdot CV(Importance(X))^2$$

其中 X 代表的是一个batch的样本，把一个batch所有样本的gating weights加起来，然后计算变异系数（ coefficient of variation）。

Shazeer, Noam, et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." arXiv preprint arXiv:1701.06538 (2017).

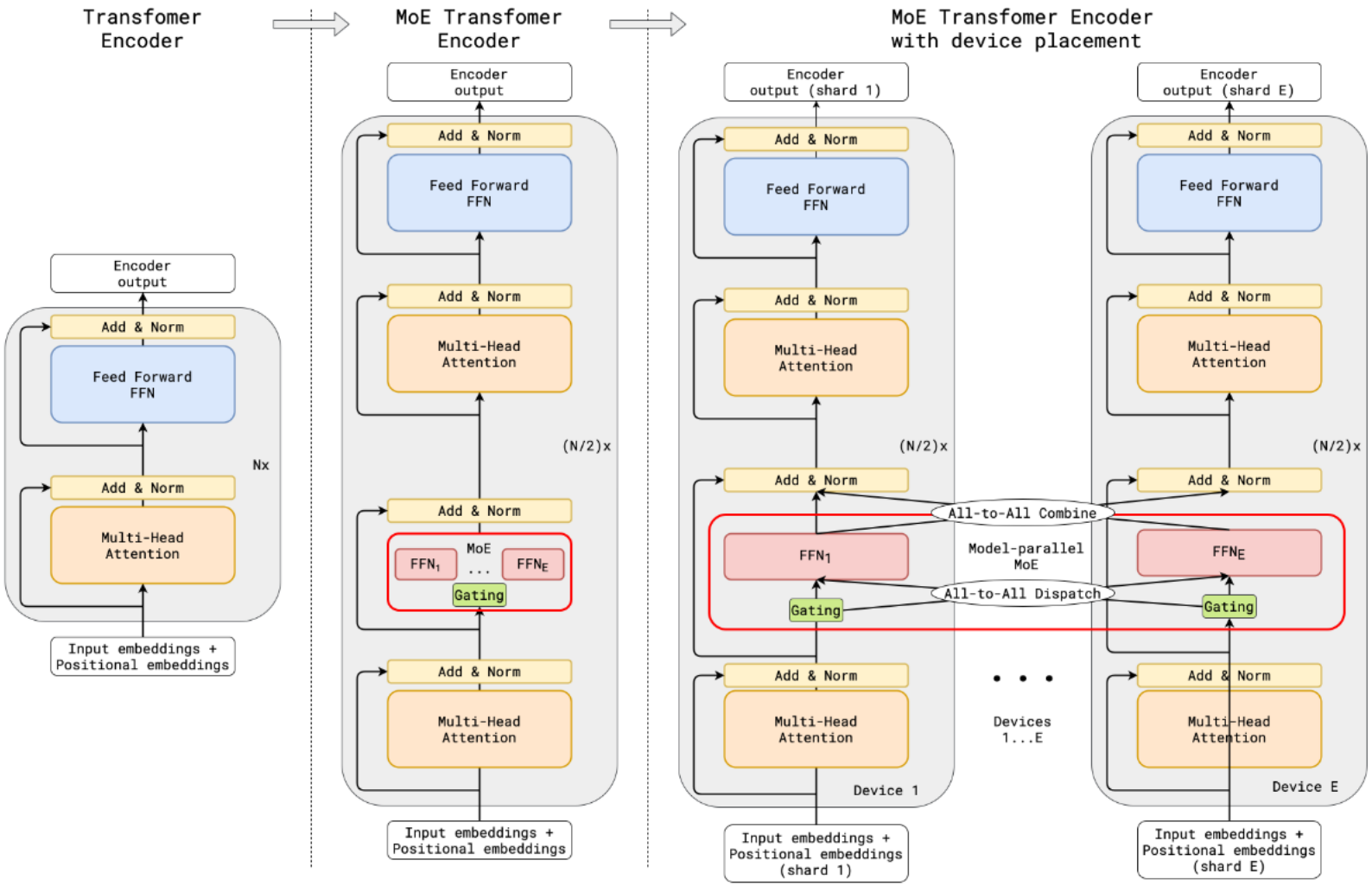
# Transformer MoE

- GShard
- Switch Transformer
- GLaM

# GShard

首个MoE+Transformer模型：

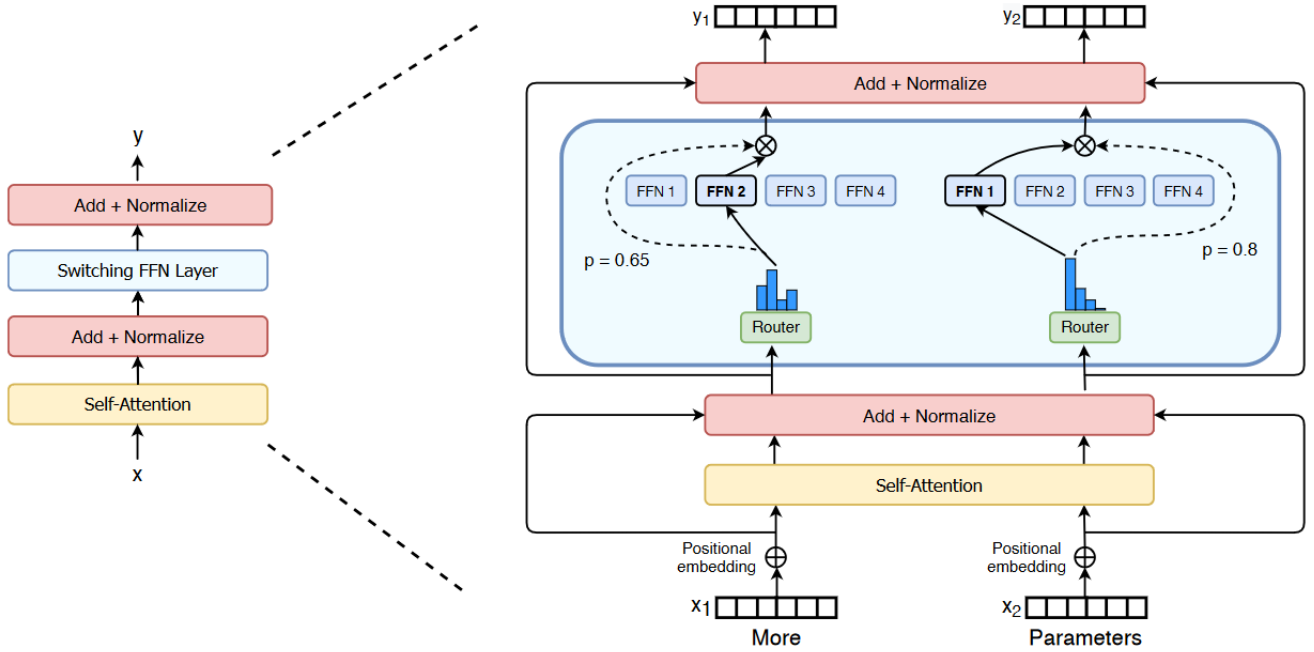
- Transformer的encoder和decoder中，每隔一个（every other）FFN层，替换成position-wise MoE层
- Top-2 gating network




Lepikhin, Dmitry, et al. "Gshard: Scaling giant models with conditional computation and automatic sharding." arXiv preprint arXiv:2006.16668 (2020).

# Switch Transformer

- 简化了MoE的routing算法
- gating network 每次只 route 到 1 个 expert

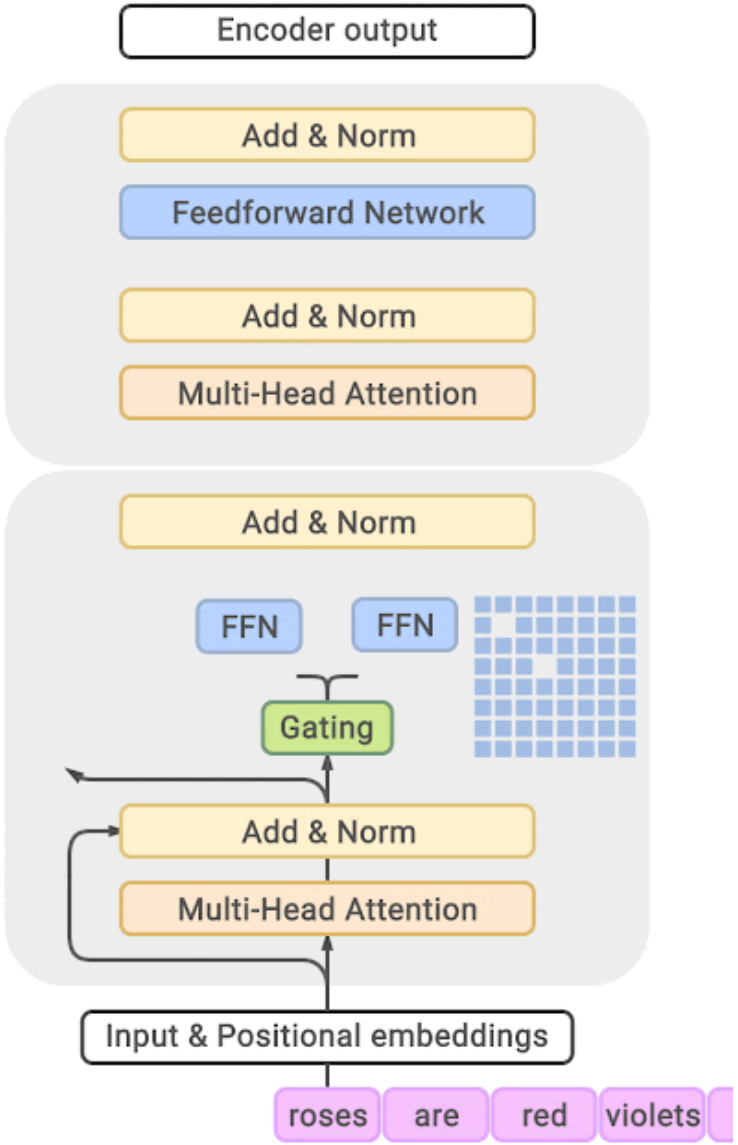



Fedus, William, Barret Zoph, and Noam Shazeer. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity." The Journal of Machine Learning Research 23.1 (2022): 5232-5270.

# GLaM

- Gshard结构
- Scale参数量
- 降低训练推理成本

| Model Name | Model Type | $n_{\text{params}}$ | $n_{\text{act-params}}$ |
|---|---|---|---|
| BERT | Dense Encoder-only | 340M | 340M |
| T5 | Dense Encoder-decoder | 13B | 13B |
| GPT-3 | Dense Decoder-only | 175B | 175B |
| Jurassic-1 | Dense Decoder-only | 178B | 178B |
| Gopher | Dense Decoder-only | 280B | 280B |
| Megatron-530B | Dense Decoder-only | 530B | 530B |
| GShard-M4 | MoE Encoder-decoder | 600B | 1.5B |
| Switch-C | MoE Encoder-decoder | 1.5T | 1.5B |
| GLaM (64B/64E) | MoE Decoder-only | 1.2T | 96.6B |




Du, Nan, et al. “Glam: Efficient scaling of language models with mixture-of-experts.” ICML, 2022.

# MoE结构复杂度

Transformer的MoE会对FFN层进行MoE扩展，但是Transformer结构本身还有Multihead Attention结构，这使得MoE扩展会变成Transformer结构的侵入式改造。

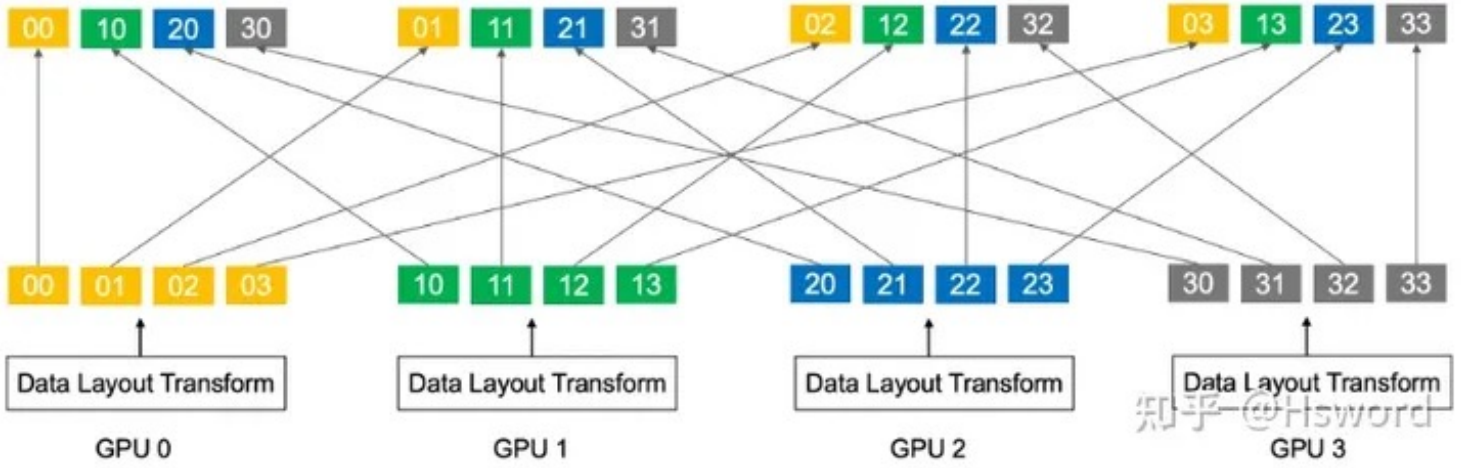
```
class MoE(Cell):

    def __init__(self, hidden_size,
                 ffn_hidden_size,
                 dropout_rate,
                 hidden_act='gelu',
                 param_init_type=mstype.float32,
                 moe_config=default_moe_config,
                 parallel_config=default_moeparallel_config):
        super(MoE, self).__init__()
        if _get_parallel_mode() in (ParallelMode.AUTO_PARALLEL,) and _is_sharding_propagation():
            self.hidden_size = hidden_size
            self.expert_dim = moe_config.expert_num
            self.capacity_factor = moe_config.capacity_factor
            self.aux_loss_factor = moe_config.aux_loss_factor
            self.num_experts_chosen = moe_config.num_experts_chosen
            self.expert_group_size = moe_config.expert_group_size
            self.dp_group = parallel_config.data_parallel
            self.dp = parallel_config.data_parallel
            self.ep = parallel_config.expert_parallel
            self.mp = parallel_config.model_parallel
            self.comp_comm_parallel = moe_config.comp_comm_parallel
            self.comp_comm_parallel_degree = moe_config.comp_comm_parallel_degree
            self.group_wise_a2a = moe_config.group_wise_a2a
            if not (self.mp > 1 and self.dp == self.ep):
                self.group_wise_a2a = False
            from mindformers.modules.transformer import FeedForward

            self.ffn = FeedForward(hidden_size=hidden_size,
                                   ffn_hidden_size=ffn_hidden_size,
                                   dropout_rate=dropout_rate,
                                   hidden_act=hidden_act,
                                   expert_num=self.expert_dim,
                                   expert_group_size=self.expert_group_size,
                                   param_init_type=param_init_type,
                                   parallel_config=parallel_config)
            self.reshape = P.Reshape()
            self.shape = P.Shape()
            self.transpose_2dim = P.Transpose().shard(((self.dp, 1),))
            self.transpose_3dim = P.Transpose().shard(((self.dp, 1, 1),))
            self.transpose_4dim = P.Transpose().shard(((1, self.dp, 1, 1),))
            self.transpose_4dim_dp = P.Transpose().shard(((1, 1, self.dp, 1),))
            self.batch_mm = P.BatchMatMul().shard(((self.dp, 1, 1), (self.dp, 1, 1)))
            self.batch_mm2 = P.BatchMatMul().shard(((self.dp, 1, 1), (self.dp, 1, 1)))
            self.mul = P.Mul()
            self.router = Router(d_model=hidden_size, moe_config=moe_config, routing_policy=None,
                                 training=True, parallel_config=parallel_config)
```

# Expert Balancing

- 物理世界规律造成的不均衡。2-8定律在此，总会有一部分任务或领域占据所有数据的大部分，也一定会有长尾数据，使用等参数量、随机Gating的方式进行强制的均衡分配，实际上也是在伤害模型对现实世界的拟合。
- 神经网络特点决定的赢者通吃。Gating Network可学习会很自然的朝着几个拟合较好的Expert进行数据分配，这一点仍需要大量的尝试和研究，也许可以缓解，也许可以解决。

# 分布式通信问题

MoE结构和普通的Dense模型的差异在于，其需要额外的AllToAll通信，来实现数据的路由(Gating)和结果的回收。而AllToAll通信会跨Node（服务器）、跨pod（路由），进而造成大量的通信阻塞问题



# Mistral Model

## Mistral 7B in short

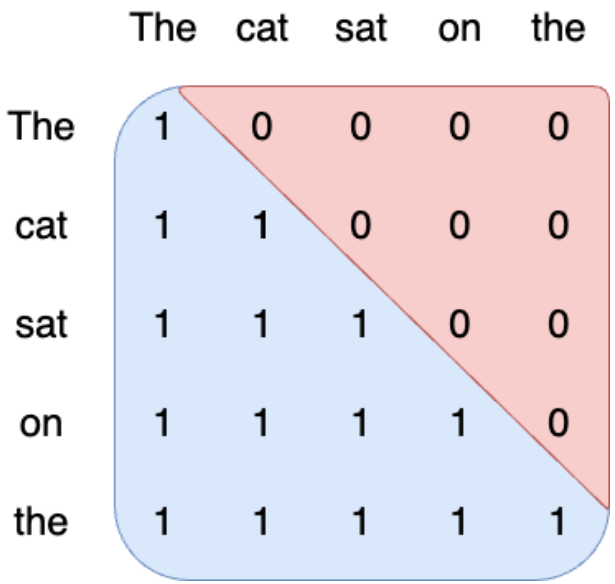
Mistral 7B is a 7.3B parameter model that:

- Outperforms Llama 2 13B on all benchmarks
- Outperforms Llama 1 34B on many benchmarks
- Approaches CodeLlama 7B performance on code, while remaining good at English tasks
- Uses Grouped-query attention (GQA) for faster inference
- Uses Sliding Window Attention (SWA) to handle longer sequences at smaller cost

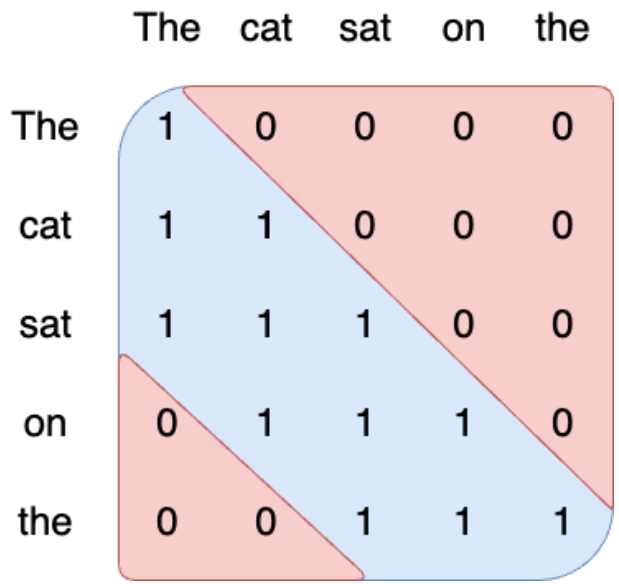
| Model | Modality | MMLU | HellaSwag | WinoGrande | PIQA | Arc-e | Arc-c | NQ | TriviaQA | HumanEval | MBPP | MATH | GSM8K |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LLaMA 2 7B | Pretrained | 44.4% | 77.1% | 69.5% | 77.9% | 68.7% | 43.2% | 24.7% | 63.8% | 11.6% | 26.1% | 3.9% | 16.0% |
| LLaMA 2 13B | Pretrained | 55.6% | **80.7%** | 72.9% | 80.8% | 75.2% | 48.8% | **29.0%** | **69.6%** | 18.9% | 35.4% | 6.0% | 34.3% |
| Code LLaMA 7B | Finetuned | 36.9% | 62.9% | 62.3% | 72.8% | 59.4% | 34.5% | 11.0% | 34.9% | **31.1%** | **52.5%** | 5.2% | 20.8% |
| Mistral 7B | Pretrained | **60.1%** | **81.3%** | **75.3%** | **83.0%** | **80.0%** | **55.5%** | **28.8%** | **69.9%** | **30.5%** | 47.5% | **13.1%** | **52.1%** |

- RoPE
- RMSNorm
- Transformer decoder

# Sliding window attention




**Vanilla attention**

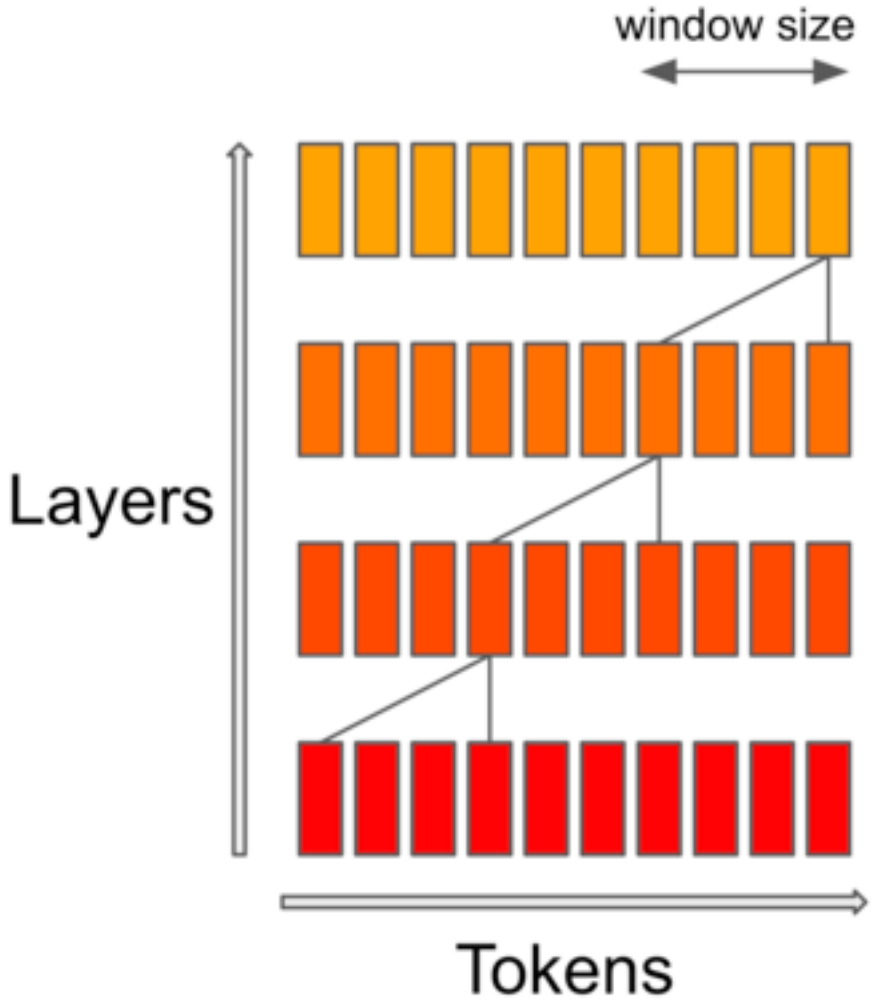



**Sliding window**

The number of operations of attention is quadratic in the sequence length, and the memory pressure is linear in the sequence length. At inference time, this incurs higher latency and smaller throughput due to reduced cache availability. To alleviate this issue, we use a sliding window attention :

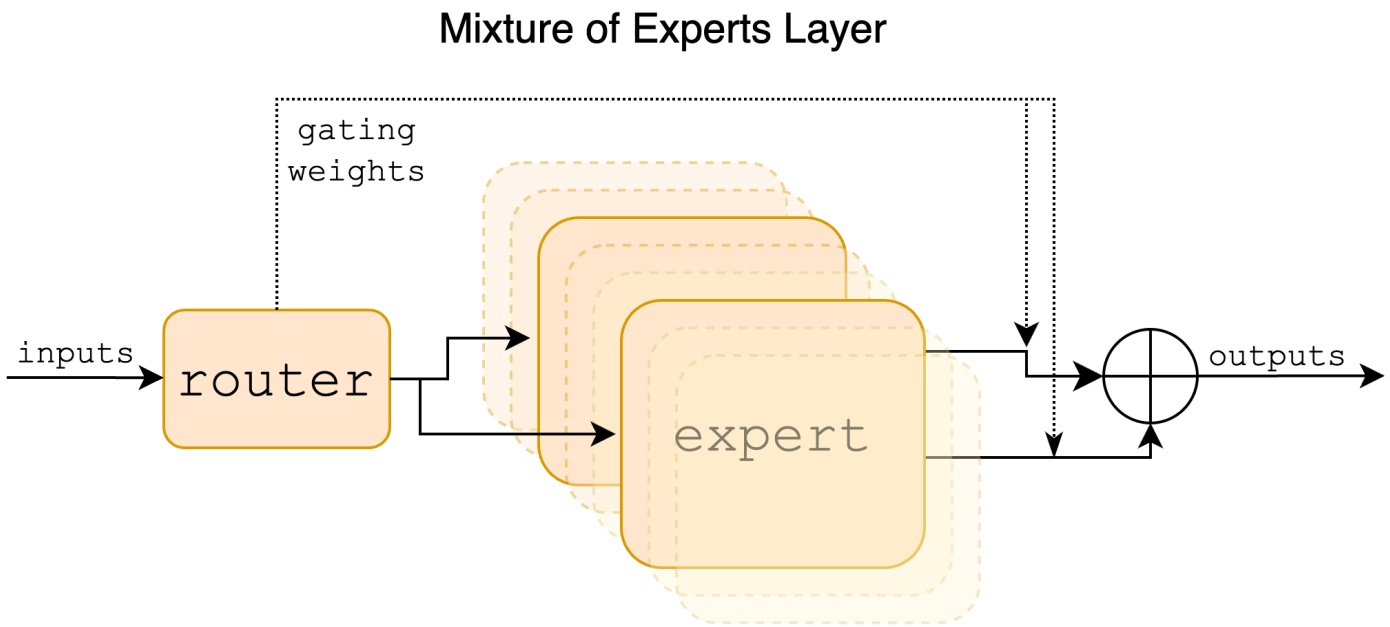
- each token can attend to at most W tokens in the past (here, W=3).

# Sliding window attention




Note that tokens outside the sliding window still influence next word prediction. At each attention layer, information can move forward by W tokens at most: after two attention layers, information can move forward by 2W tokens, etc. For instance in a sequence of length 16K and a sliding window of 4K, after 4 layers, information has propagated to the full sequence length.

# MoE——Mixtral



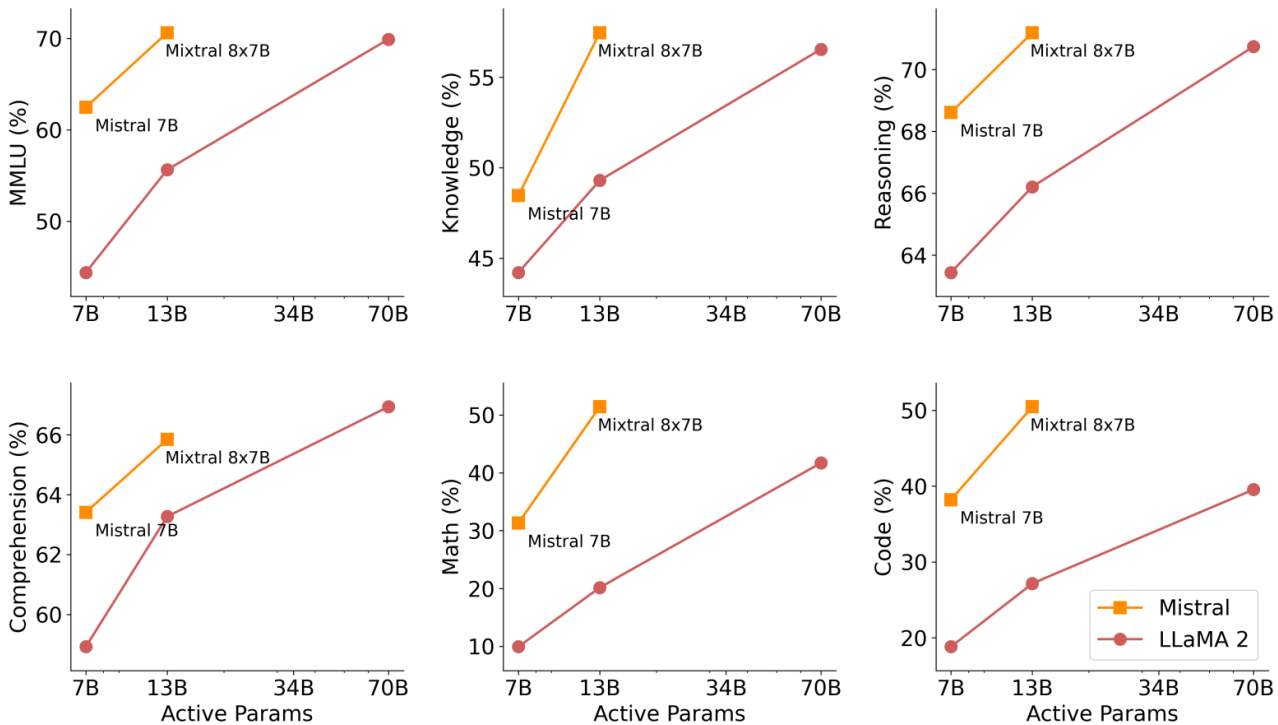

Sparse Mixture of Experts allows one to decouple throughput from memory costs by only activating subsets of the overall model for each token. In this approach, each token is assigned to one or more "experts" -- a separate set of weights -- and only processed by sunch experts. This division happens at feedforward layers of the model. The expert models specialize in different aspects of the data, allowing them to capture complex patterns and make more accurate predictions.

# MoE Layer实现

```python
class MoeLayer(nn.Cell):
    def __init__(self, experts: List[nn.Cell], gate: nn.Cell, moe_args: MoeArgs):
        super().__init__()
        assert len(experts) > 0
        self.experts = nn.CellList(experts)
        self.gate = gate
        self.args = moe_args

    def construct(self, inputs: mindspore.Tensor):
        gate_logits = self.gate(inputs)
        weights, selected_experts = ops.topk(gate_logits, self.args.num_experts_per_tok)
        weights = ops.softmax(weights, axis=1, dtype=mindspore.float32).to(inputs.dtype)
        results = ops.zeros_like(inputs)
        for i, expert in enumerate(self.experts):
            non_zero = ops.nonzero(selected_experts == i)
            if 0 not in non_zero.shape:
                batch_idx, nth_expert = non_zero.tensor_split(2, 1)
                results[batch_idx] = results[batch_idx] + weights[batch_idx, nth_expert, None] * expert(
                    inputs[batch_idx]
                )
        return results
```

| Model | Active Params | MMLU | HellaS | WinoG | PIQA | Arc-e | Arc-c | NQ | TriQA | HumanE | MBPP | Math | GSM8K |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **LLaMA 2 7B** | 7B | 44.4% | 77.1% | 69.5% | 77.9% | 68.7% | 43.2% | 17.5% | 56.6% | 11.6% | 26.1% | 3.9% | 16.0% |
| **LLaMA 2 13B** | 13B | 55.6% | 80.7% | 72.9% | 80.8% | 75.2% | 48.8% | 16.7% | 64.0% | 18.9% | 35.4% | 6.0% | 34.3% |
| **LLaMA 1 33B** | 33B | 56.8% | 83.7% | 76.2% | 82.2% | 79.6% | 54.4% | 24.1% | 68.5% | 25.0% | 40.9% | 8.4% | 44.1% |
| **LLaMA 2 70B** | 70B | 69.9% | **85.4%** | **80.4%** | 82.6% | 79.9% | 56.5% | 25.4% | **73.0%** | 29.3% | 49.8% | 13.8% | 69.6% |
| **Mistral 7B** | 7B | 62.5% | 81.0% | 74.2% | 82.2% | 80.5% | 54.9% | 23.2% | 62.5% | 26.2% | 50.2% | 12.7% | 50.0% |
| **Mixtral 8x7B** | 13B | **70.6%** | 84.4% | 77.2% | **83.6%** | **83.1%** | **59.7%** | **30.6%** | 71.5% | **40.2%** | **60.7%** | **28.4%** | **74.4%** |

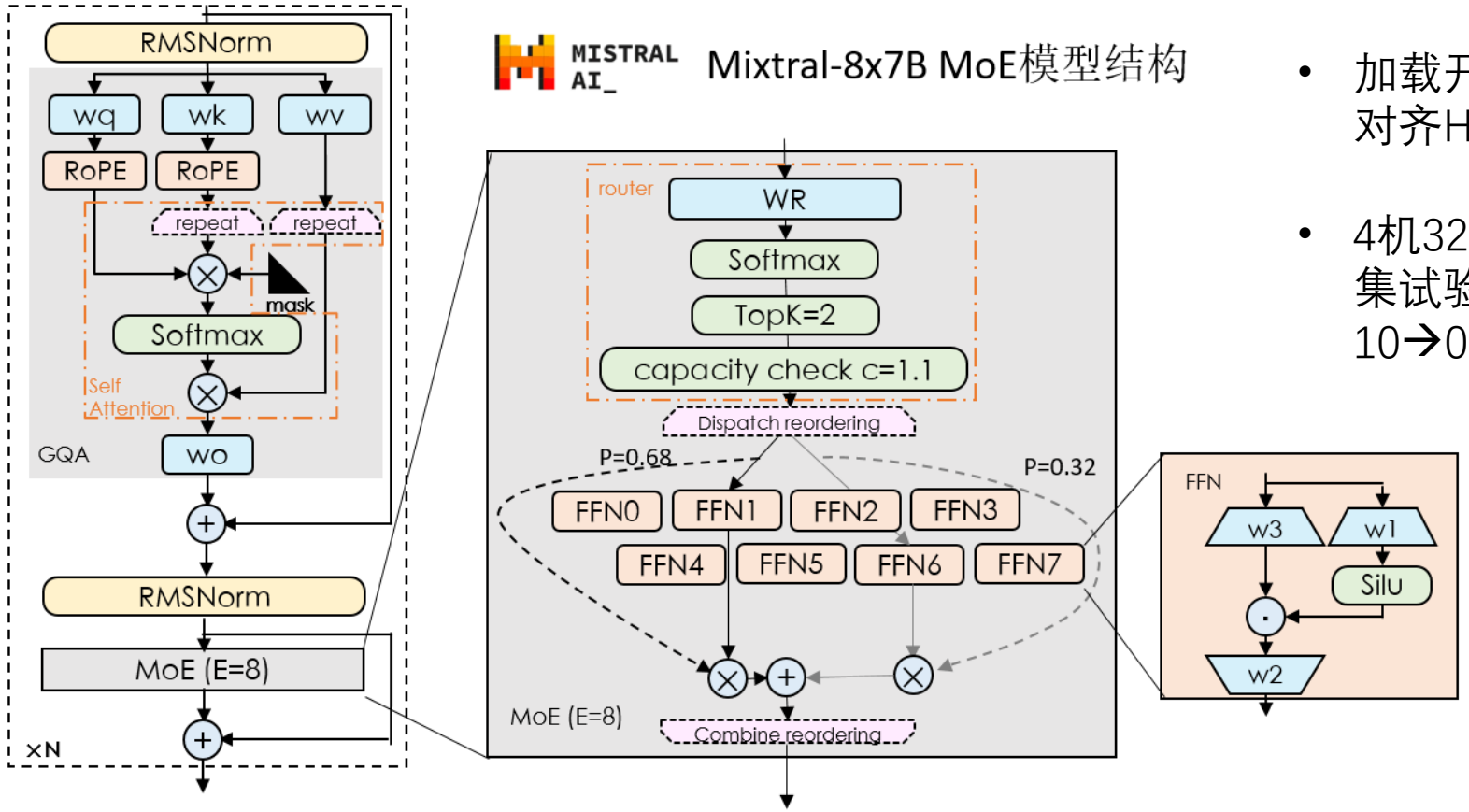
**Table 2: Comparison of Mixtral with Llama.** Mixtral outperforms or matches Llama 2 70B performance on almost all popular benchmarks while using 5x fewer active parameters during inference.



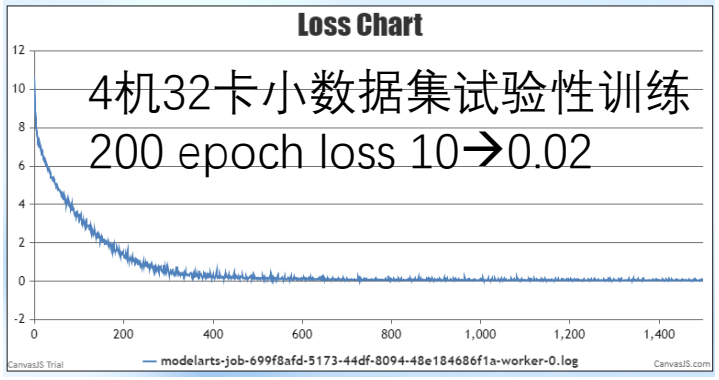

| | LLaMA 2 70B | GPT-3.5 | Mixtral 8x7B |
|---|---|---|---|
| **MMLU** (MCQ in 57 subjects) | 69.9% | 70.0% | **70.6%** |
| **HellaSwag** (10-shot) | **87.1%** | 85.5% | 86.7% |
| **ARC Challenge** (25-shot) | 85.1% | 85.2% | **85.8%** |
| **WinoGrande** (5-shot) | **83.2%** | 81.6% | 81.2% |
| **MBPP** (pass@1) | 49.8% | 52.2% | **60.7%** |
| **GSM-8K** (5-shot) | 53.6% | 57.1% | **58.4%** |
| **MT Bench** (for Instruct Models) | 6.86 | **8.32** | 8.30 |

# 基于MindSpore实现的Mixtral EP-MP等多维混合并行训练

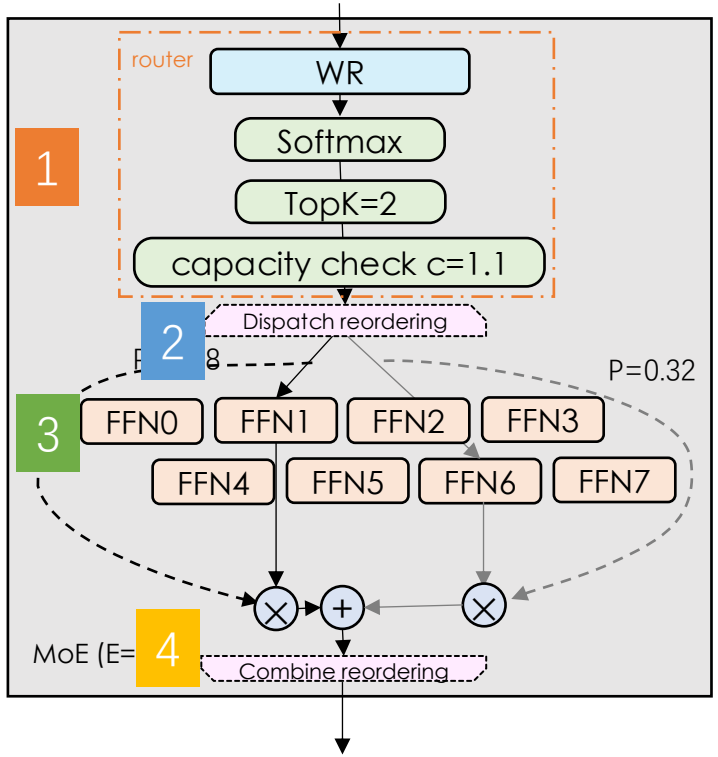
| nlayer | heads | hidden | intermediate | GQA | vocab | MoE size | MoE topk |
|---|---|---|---|---|---|---|---|
| 32 | 32 | 4096 | 14336 | 8 | 32000 | 8 | 2 |




- 基于MindFormers实现Mixtral-8x7B MoE模型。
  - 关键结构: GQA, RoPE, RMSNorm, Silu
  - MoE配置: 8 Experts, TopK=2, capacity c=1.1
- 加载开源的Mixtral权重和tokenizer，推理结果对齐HF.
- 4机32卡EP,PP等多维混合并行，基于自有数据集试验性训练收敛符合预期。200 epoch loss 10→0.02



# MoE的代码实现




```
# calculate router
dispatch_index, combine_index, router_coeff = self.router(input_tensor
                                                                    32, (dp, N, k)fp16 <--

# dispatch
input_tensor_padded = self.concat_3d((self.zeros_3d, input_tensor)) #
expert_input = self.dispatch_gather(input_tensor_padded, dispatch_index, 1)

# ffn
expert_input = self.transpose_4dim_dp1(expert_input,(1,0,2,3)) # (E, d                n, h)
expert_output = self.ffn_infer(expert_input) # (E, dp, n, h) <-- (E, d
expert_output = self.transpose_4dim_dp1(expert_output,(1,0,2,3)) # (dp, E, n, h) <-- (E, dp, n, h)

# combine
expert_output = self.concat((self.zeros, expert_output)) # (dp, E, 1+n
expert_output = self.reshape(expert_output, (self.dp_group, expert_output.shape[1]*expert_output.shape[2], self.hidden_size))
output_tensor = self.combine_gather(expert_output, combine_index, 1) # (dp, N, k, h) <-- (dp, E*(1+n), h), (dp, N, k)
output_tensor = self.mul_router_coeff(output_tensor, self.reshape(router_coeff, (self.dp_group, N, k, 1)) ) # (dp, N, k, h) <-
output_tensor = self.sum_router_coeff(output_tensor, 2) #reduce sum # (dp, N, h) <-- (dp, N, k, h)
output_tensor = self.reshape(output_tensor, input_tensor_shape) # (B*S, h) <-- (dp, N, h)
return output_tensor, 0 # (dp, N, h)
```

1. router 出MoE方案

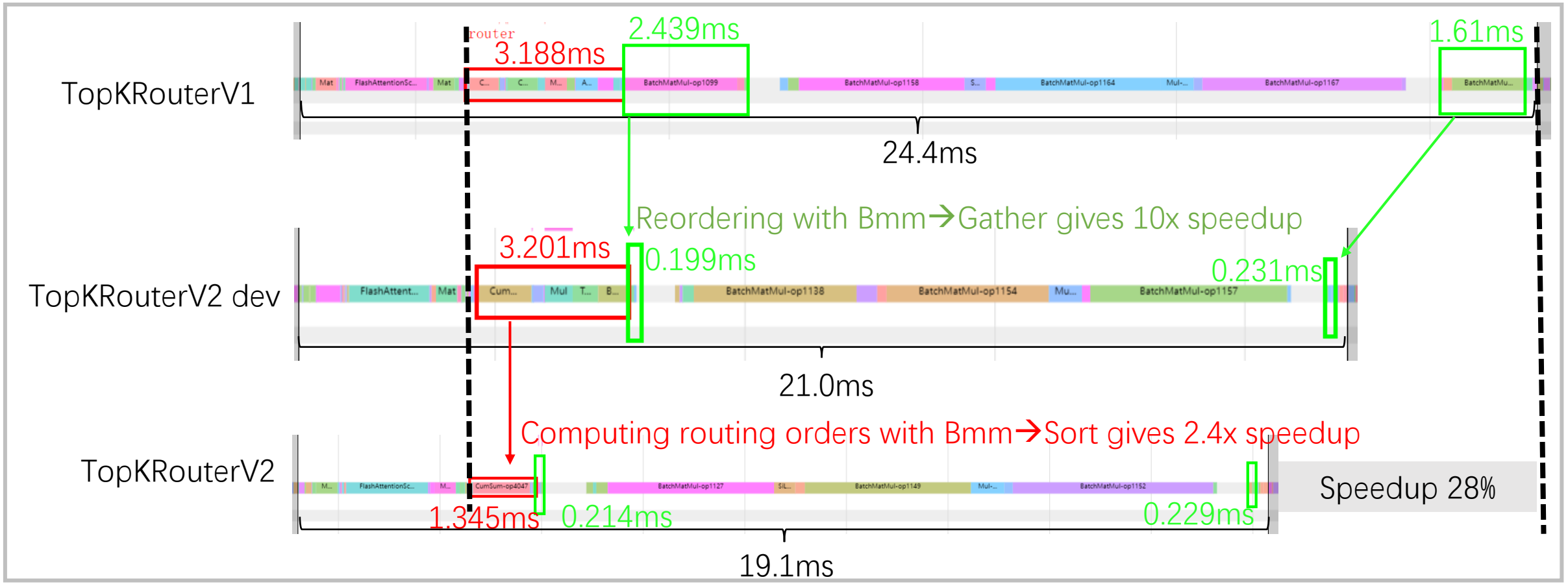
2. dispatch 时间序列→专家序列

3. Batch FFN

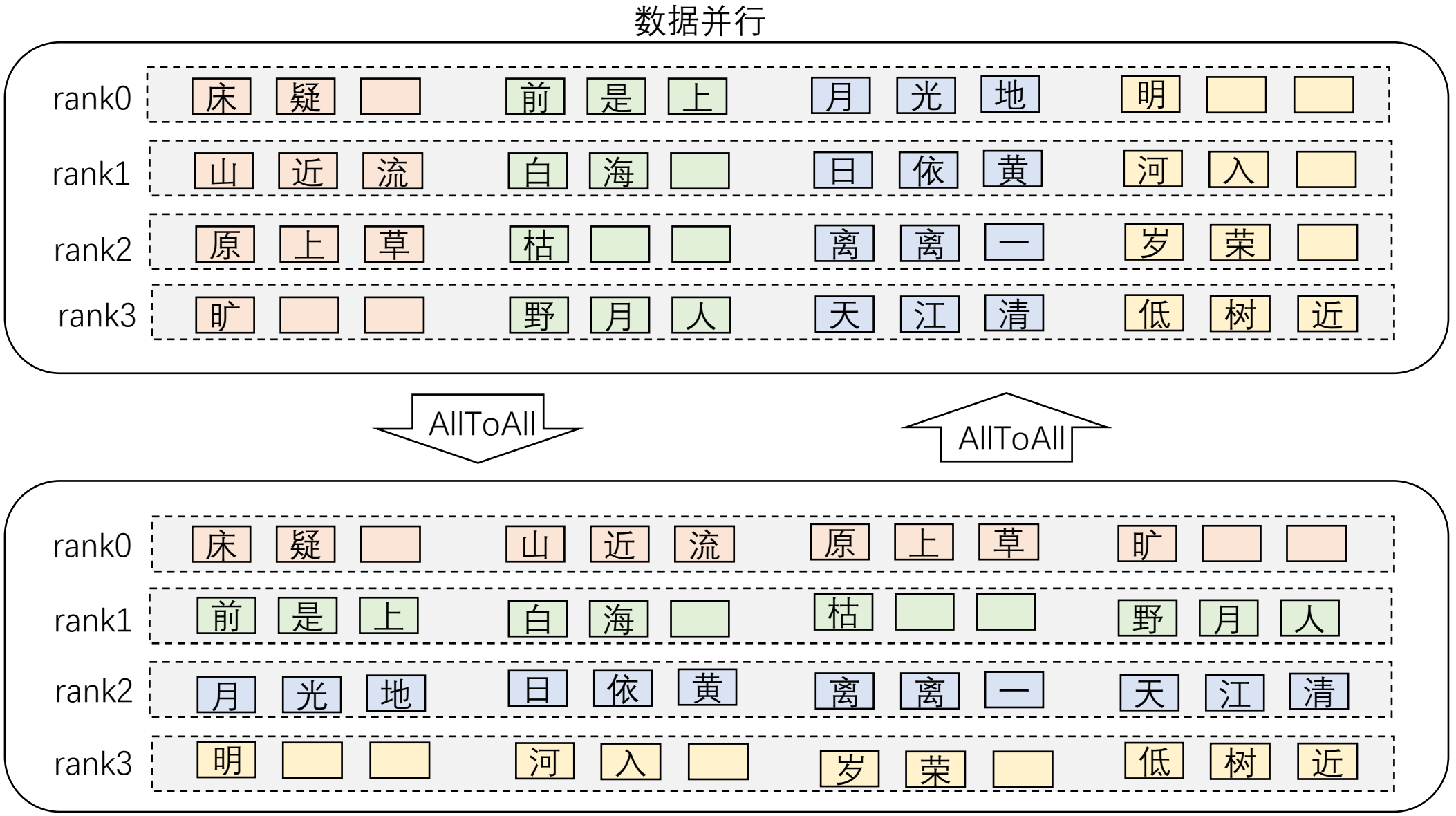
4. combine 专家序列→时间序列

# Optimize the implementation of TopKRouter

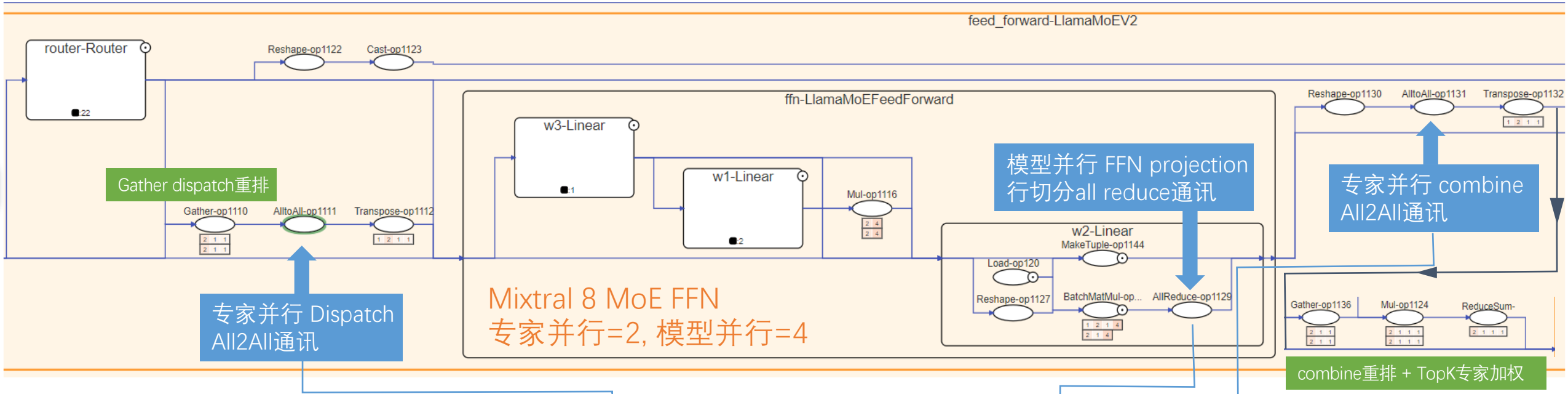
| | TopK | dispatch/combine | router order computing |
|---|---|---|---|
| TopKRouterV1 | K * argmax | BatchMatMul | ordering matrix = K* (onehot-cumsum-onehot*2) |
| TopKRouterV2 -dev | 1 * TopK | Gather | ordering index = 1* (onehot-cumsum-bmm) |
| TopKRouterV2 | 1 * TopK | Gather | ordering index = 1* (onehot-cumsum-sortFP32) |

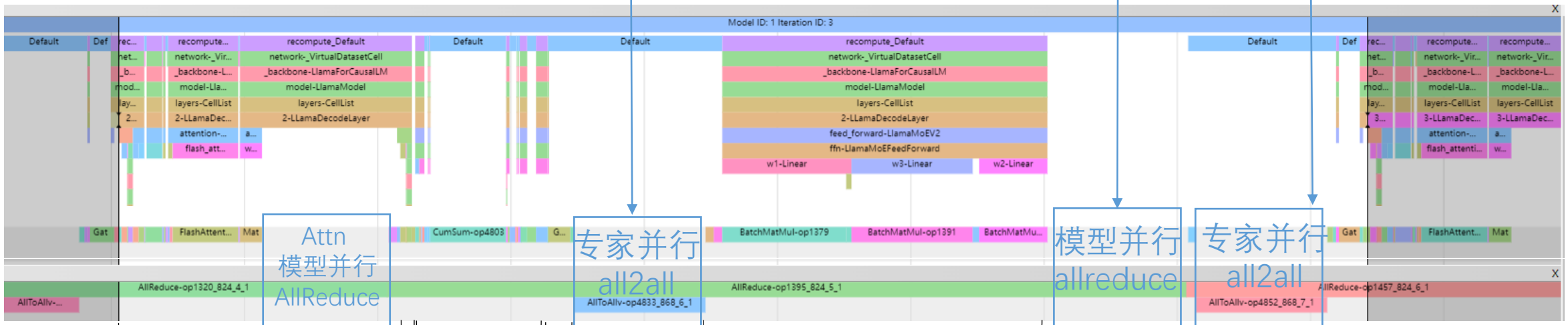


# 专家并行的示意图


数据并行
rank0
床 疑 前 是 上 月 光 地 明
rank1
山 近 流 白 海 日 依 黄 河 入
rank2
原 上 草 枯 离 离 一 岁 荣
rank3
旷 野 月 人 天 江 清 低 树 近
AllToAll
AllToAll
rank0
床 疑 山 近 流 原 上 草 旷
rank1
前 是 上 白 海 枯 野 月 人
rank2
月 光 地 日 依 黄 离 离 一 天 江 清
rank3
明 河 入 岁 荣 低 树 近
专家并行的FFN

# Mixtral 实现EP-MP等多维混合并行训练

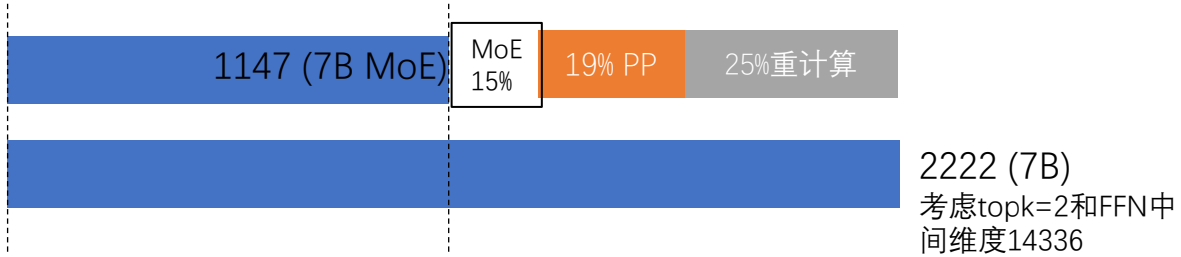


# Mixtral 实现EP-MP等多维混合并行训练

| | 参数量 | 测试集群 | 网络结构 | | | | | 并行策略 | | | | | | Batch Size | | | 重计算 | 性能 | 内存 (静态GB,动态GB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Seq | layer | hidden | 专家容量 | TopK | DP | EP | MP | PP | OP | SP | BS | num micro | GBS | | token/s/p | S0 | S1 | S2 | S3 |
| Mixtral 8x7B | 46 B | 4机32卡 | 4096 | 32 | 4096 | 1.1 | 2 | 1 | 1 | 8 | 4 | x | x | 4 | 16 | 64 | √ | 327.68 | 21,11 | 20,11 | 19,11 | 20,11 |
| | | | 4096 | 32 | 4096 | 1.1 | 2 | 4 | 4 | 2 | 4 | x | x | 1 | 16 | 64 | √ | 737.28 | 20,07 | 19,07 | 19,07 | 20,07 |
| | | | 4096 | 32 | 4096 | 1.1 | 2 | 8 | 8 | 1 | 4 | x | x | 1 | 16 | 128 | √ | 1146.88 | 22,09 | 20,07 | 20,07 | 23,08 |
| | | | 4096 | 32 | 4096 | 1.1 | 2 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | √ | **1146.88** | 21,08 | 19,06 | 19,06 | 21,06 |
| | | | 4096 | 32 | 4096 | 1.1 | 2 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | x | oom | | | | |
| Mixtral 8x13B | 89 B | 4机32卡 | 4096 | 40 | 5120 | 1.1 | 2 | 1 | 1 | 8 | 4 | x | x | 1 | 16 | 16 | √ | 245.76 | 34,12 | 34,12 | 34,12 | 33,12 |
| | | | 4096 | 40 | 5120 | 1.1 | 2 | 4 | 4 | 2 | 4 | x | x | 1 | 16 | 64 | √ | 450.56 | 38,13 | 36,13 | 36,13 | 37,13 |
| | | | 4096 | 40 | 5120 | 1.1 | 2 | 8 | 8 | 1 | 4 | x | x | 1 | 12 | 96 | √ | **573.44** | 42,16 | 40,14 | 40,14 | 42,15 |

- EP=8,MP=1时性能最佳,约1147 tokens/s/p。其中
  - MoE相关开销约15%(路由/容量系数/重排/通讯)
- 优化器并行在EP的基础上切分QKV优化器参数，静态与动态内存降低约1GB。
- 相比8x7B， 8X13B参数量约为2倍，最佳性能下的开销也约为2倍。



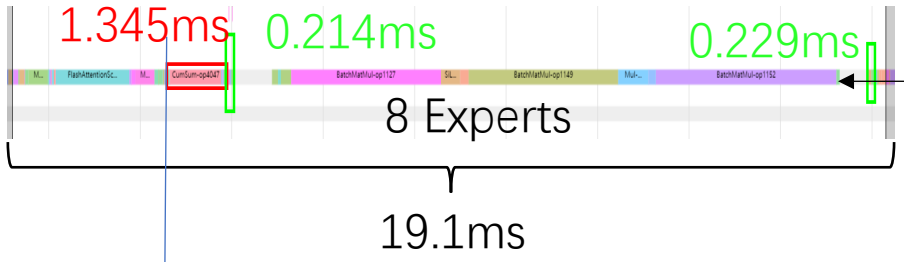

- 优化点:
  - MP开启时，可以通过Group-wise all2all提高a2a性能，通过SP降低allreduce 开销。
  - 内存尚有余量，可以考虑选择重计算。

# 扩展MoE 专家数对性能的影响

| 测试集群 | 网络结构 | | | | | | | | 并行策略 | | | | | | Batch Size | | | 重计算 | 性能 | 内存 (静态GB,动态GB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Seq | vocab | layer | hidden | 专家容量 | 专家dim | 专家总数 | 专家激活数 | DP | EP | MP | PP | OP | SP | BS | num micro | GBS | | token/s/p | S0 | S1 | S2 | S3 |
| 4机32卡 | 4096 | 32000 | 32 | 4096 | 1.1 | 11008 | 8 | 2 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | √ | 1310.72 | 18,06 | 15,05 | 15,05 | 17,05 |
| | 4096 | 32000 | 32 | 4096 | 1.1 | 5504 | 16 | 4 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | √ | 1105.92 | 18,06 | 15,05 | 15,05 | 17,05 |
| | 4096 | 32000 | 32 | 4096 | 1.1 | 2752 | 32 | 8 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | √ | 819.2 | 18,07 | 15,05 | 15,05 | 17,05 |
| | 4096 | 32000 | 32 | 4096 | 1.1 | 1376 | 64 | 16 | 8 | 8 | 1 | 4 | √ | x | 1 | 16 | 128 | √ | 573.44 | 18,07 | 15,06 | 15,06 | 17,06 |

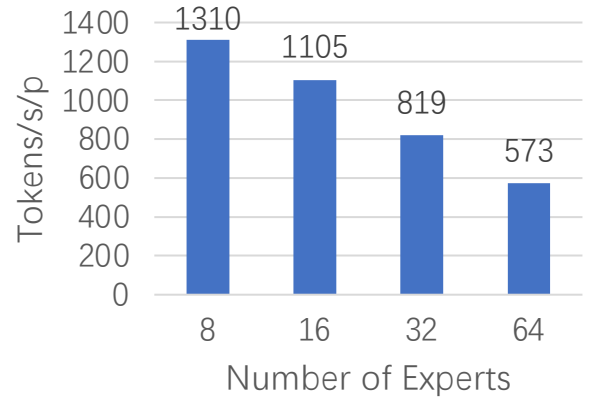
每卡1个专家，8路专家并行

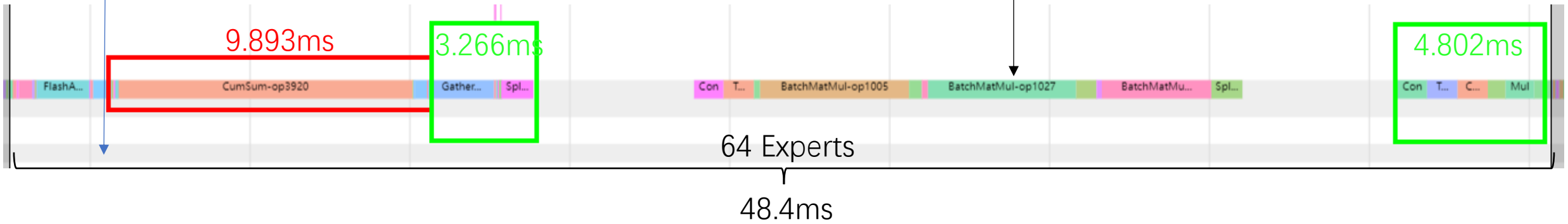
每卡8个专家，8路专家并行



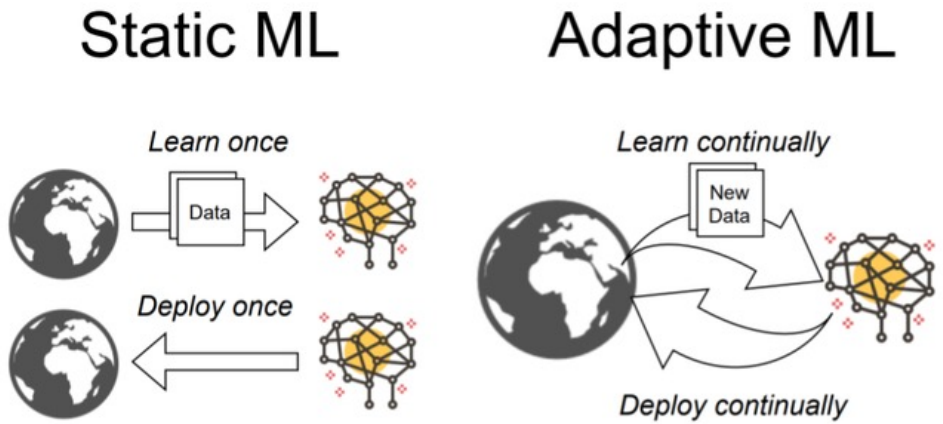

```
Inputs:
  name: Reshape-op1247
  shape: [ "1", "1", "9016", "4096" ]
  strategy: [ 1, 8, 1, 1 ]
  name: TensorMove-op2283
  shape: [ "1", "14336", "4096" ]
  strategy: [ 8, 1, 1 ]
Outputs:
  name: Reshape-op1271
  name: UpdateState-op2168
Output_Shape:
[ "1", "1", "9016", "14336" ]
```

```
Inputs:
  name: Reshape-op1247
  shape: [ "1", "8", "9016", "4096" ]
  strategy: [ 1, 8, 1, 1 ]
  name: TensorMove-op2283
  shape: [ "8", "1792", "4096" ]
  strategy: [ 8, 1, 1 ]
Outputs:
  name: Reshape-op1271
  name: UpdateState-op2168
Output_Shape:
[ "1", "8", "9016", "1792" ]
```



# Continual Learning




Continual learning is a machine learning paradigm, whose objective is to **adaptively learn across time** by leveraging previously learned tasks to improve generalization for future tasks. Hence, CL studies the problem of **sequential learning from a continuous stream of data**, drawn from a potentially non-stationary distribution, and reusing gained knowledge throughout the lifetime while **avoiding catastrophic forgetting**.

Biesialska, Magdalena, Katarzyna Biesialska, and Marta R. Costa-Jussa. "Continual lifelong learning in natural language processing: A survey." arXiv preprint arXiv:2012.09823 (2020).

# Desiderata of Continual Learning

| *Property* | *Definition* |
|---|---|
| **Knowledge retention** | The model is not prone to catastrophic forgetting. |
| **Forward transfer** | The model learns a new task while reusing knowledge acquired from previous tasks. |
| **Backward transfer** | The model achieves improved performance on previous tasks after learning a new task. |
| **On-line learning** | The model learns from a continuous data stream. |
| **No task boundaries** | The model learns without requiring neither clear task nor data boundaries. |
| **Fixed model capacity** | Memory size is constant regardless of the number of tasks and the length of a data stream. |

Table 1: Desiderata of continual learning.

| 性质 | 定义 |
|---|---|
| 知识记忆(knowledge retention) | 模型不易产生遗忘灾难 |
| 前向迁移(forward transfer) | 利用旧知识学习新任务 |
| 后向迁移(backward transfer) | 新任务学习后提升旧任务 |
| 在线学习(online learning) | 连续数据流学习 |
| 无任务边界(No task boudaries) | 不需要明确的任务或数据定义 |
| 固定模型容量(Fixed model capacity) | 模型大小不随任务和数据变化 |

Biesialska, Magdalena, Katarzyna Biesialska, and Marta R. Costa-Jussa. "Continual lifelong learning in natural language processing: A survey." arXiv preprint arXiv:2012.09823 (2020).

# Comparison of related ML paradigms

| *Paradigm* | *Definition* | *Properties** | *Related works* |
|---|---|---|---|
| **Transfer learning** | Transferring knowledge from a source task/domain to a target task/domain to improve the performance of the target task. | + forward transfer<br>– no backward transfer<br>– no knowledge retention<br>– task boundaries<br>– off-line learning | (Pan and Yang, 2010) (Howard and Ruder, 2018) (Peters et al., 2018) (Radford et al., 2019) (Devlin et al., 2019) (Houlsby et al., 2019) (Raffel et al., 2020) |
| **Multi-task learning** | Learning multiple related tasks jointly, using parameter sharing, to improve the generalization of all the tasks. | + positive transfer<br>– negative transfer<br>– task boundaries<br>– off-line learning | (Caruana, 1997) (Zhang and Yang, 2017) (Ruder, 2017) (McCann et al., 2018) (Stickland and Murray, 2019) (Phang et al., 2020) |
| **Meta-learning** | *Learning to learn.* Learning generic knowledge, given a small set of training examples and numerous tasks, and quickly adapting to a new task. | + forward transfer<br>– no backward transfer<br>– no knowledge retention<br>– off-line learning | (Thrun and Pratt, 1998) (Finn et al., 2017) (Xu et al., 2018) (Obamuyide and Vlachos, 2019) (Hospedales et al., 2020) (Beaulieu et al., 2020) |
| **Curriculum learning** | Learning from training examples arranged in a meaningful order – task or data difficulty gradually increases. | + forward transfer<br>+ backward transfer<br>+ knowledge retention<br>– task boundaries<br>– off-line learning | (Elman, 1993) (Bengio et al., 2009) (van der Wees et al., 2017) (Zhang et al., 2018) (Zhang et al., 2019) (Platanios et al., 2019) (Ruiter et al., 2020) |
| **On-line learning** | Learning over a continuous stream of training examples provided in a sequential order. Experiences *concept drift* due to non-i.i.d. data. | + on-line learning<br>+ forward transfer<br>– no backward transfer<br>– no knowledge retention<br>– single task/domain | (Bottou, 1998) (Bottou and LeCun, 2004) (Cesa-Bianchi and Lugosi, 2006) (Shalev-Shwartz, 2012) (C. de Souza et al., 2015) (Hoi et al., 2018) |
| **On-the-job learning** | Discovering new tasks, learning and adapting on-the-fly. On-the-job learning operates in an *open-world* environment, and it involves interaction with humans and the environment. It belongs to the CL family of methods. | + on-line learning<br>+ forward transfer<br>+ backward transfer<br>+ knowledge retention<br>+ no task boundaries<br>+ open-world learning<br>– interactive learning | (Xu et al., 2019) (Mazumder et al., 2019) (Liu, 2020) |

Biesialska, Magdalena, Katarzyna Biesialska, and Marta R. Costa-Jussa. "Continual lifelong learning in natural language processing: A survey." arXiv preprint arXiv:2012.09823 (2020).

# Forgetting

- Distribution of original dataset: A
- Distribution of new samples: B
- Distribution shift A ⇨ B
- Performance on both A & B matter!
  - Build general models is the trend.
- **“Forgetting issue”**: if we only train on B, performance on A will drop

# LLM paradigm(Pretrain+SFT)

| 性质 | | 说明 |
| --- | --- | --- |
| 知识记忆(knowledge retention) | √ | LLM预训练后，具备世界知识，小规模finetune不易对LLM造成遗忘灾难。**但大规模数据续训会造成。** |
| 前向迁移(forward transfer) | √ | 基于世界知识的**Zero shot**、few shot、finetune。 |
| 后向迁移(backward transfer) | - | Finetune后会可能会造成部分任务的性能下降。二次finetune会损失首次finetune性能。 |
| 在线学习(online learning) | × | 离线预训练、微调。 |
| 无任务边界(No task boudaries) | √ | **Unsupervised**预训练、微调，不区分任务。 |
| 固定模型容量(Fixed model capacity) | √ | LLM预训练后**大小不变**。 |

# MoE和Lifelong learning

MoE的特点：

- 多个Expert分别处理不同分布（domain/topic）的数据
- 推理仅需要部分Expert

LLM的终身学习：

- 世界知识底座持续学习。
- Expert可插拔
- Gating Network可增删。

| 性质 | |
|---|---|
| 知识记忆(knowledge retention) | √ |
| 前向迁移(forward transfer) | √ |
| 后向迁移(backward transfer) | - |
| 在线学习(online learning) | × |
| 无任务边界(No task boudaries) | √ |
| 固定模型容量(Fixed model capacity) | √ |

# Lifelong Language Pretraining with Distribution-Specialized Experts

- Distribution based MoE
  - Progressively add more experts for new data distribution
  - Add regularization to mitigate forgetting.

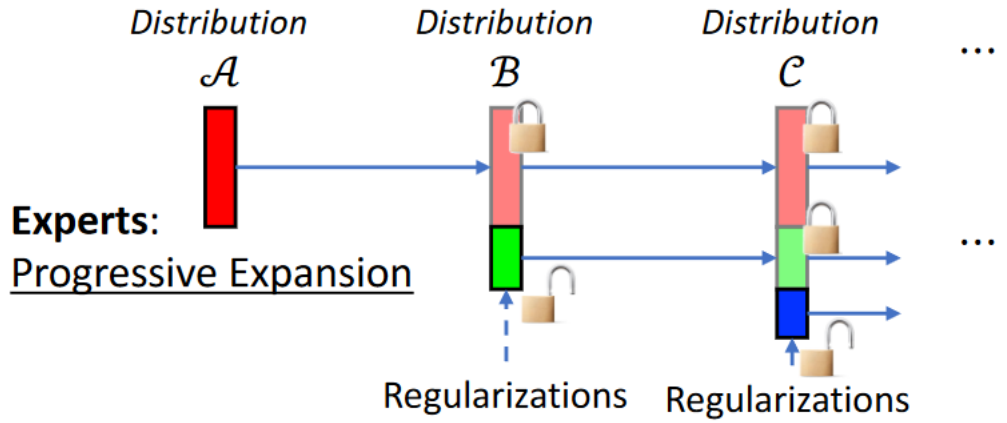



Figure 1: Overview of our Lifelong-MoE method: 1) During pretraining, the expanded experts (and gatings) are specialized for each data distribution; 2) We freeze the pretrained old experts and gatings; 3) We further introduce regularizations to the MoE to avoid the catastrophic forgetting.

Chen, Wuyang, et al. "Lifelong language pretraining with distribution-specialized experts." ICML, 2023.

# Lifelong Pretraining on MoE

- Distribution A ⇨ B ⇨ C
  - Simulation on Tarzan: A=wiki/web, B=non-English, C=dialog

Table 1: Data distributions in our lifelong pretraining set.

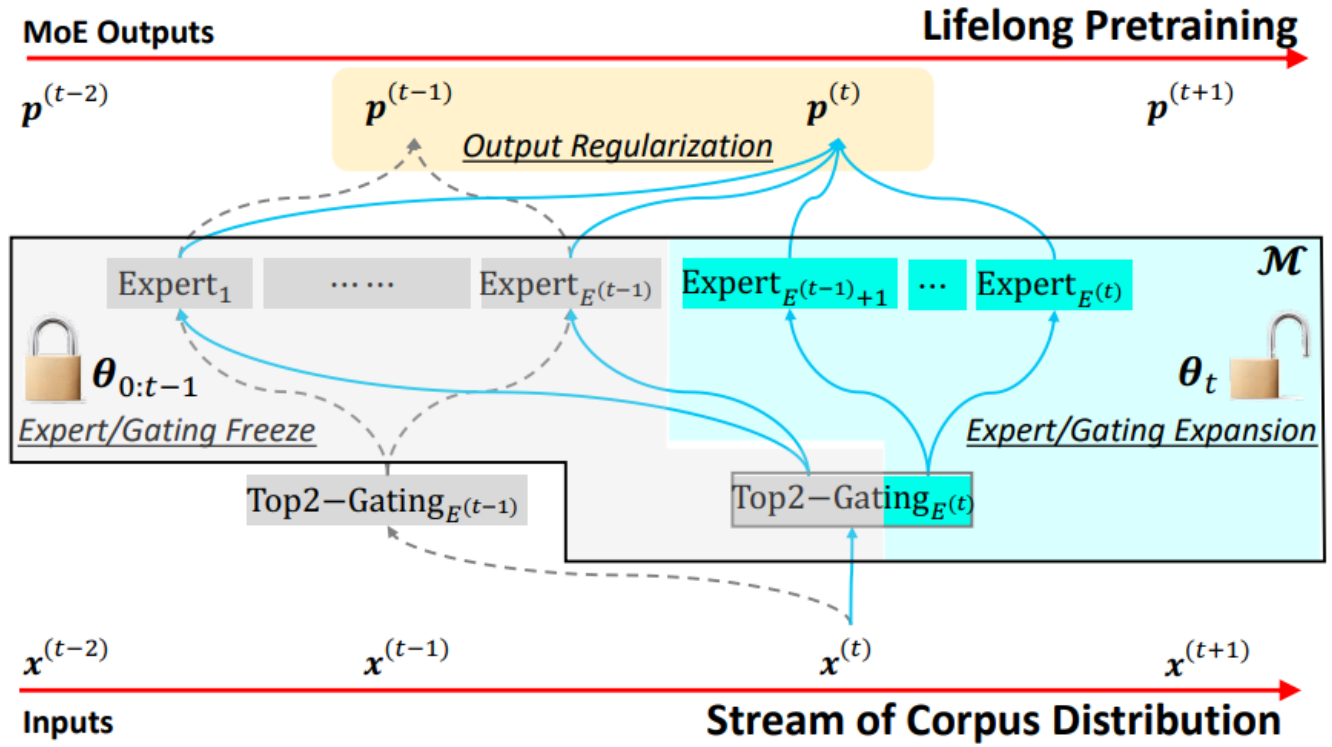
| Distribution | Corpus | Tokens (B) |
|---|---|---|
| $\mathcal{A}$ | Wikipedia (19%)<br>Filtered Webpages (81%) | 3<br>143 |
| $\mathcal{B}$ | i18n | 366 |
| $\mathcal{C}$ | Conversations | 174 |

- Regularization
  - We don't want models to overfit B
  - We don't want model weights to be updated too far from A
  - Fit B, while regularize model from A
- Expansion
  - Allow models to expand layers(experts) when fitting new distributions

Chen, Wuyang, et al. "Lifelong language pretraining with distribution-specialized experts." ICML, 2023.

# Expansion + Regularization

- Expand experts for new distributions
- Partially freeze old experts/gatings
- Train with “Learning without Forgetting” loss (Regularization)




Chen, Wuyang, et al. "Lifelong language pretraining with distribution-specialized experts." ICML, 2023.

# Results

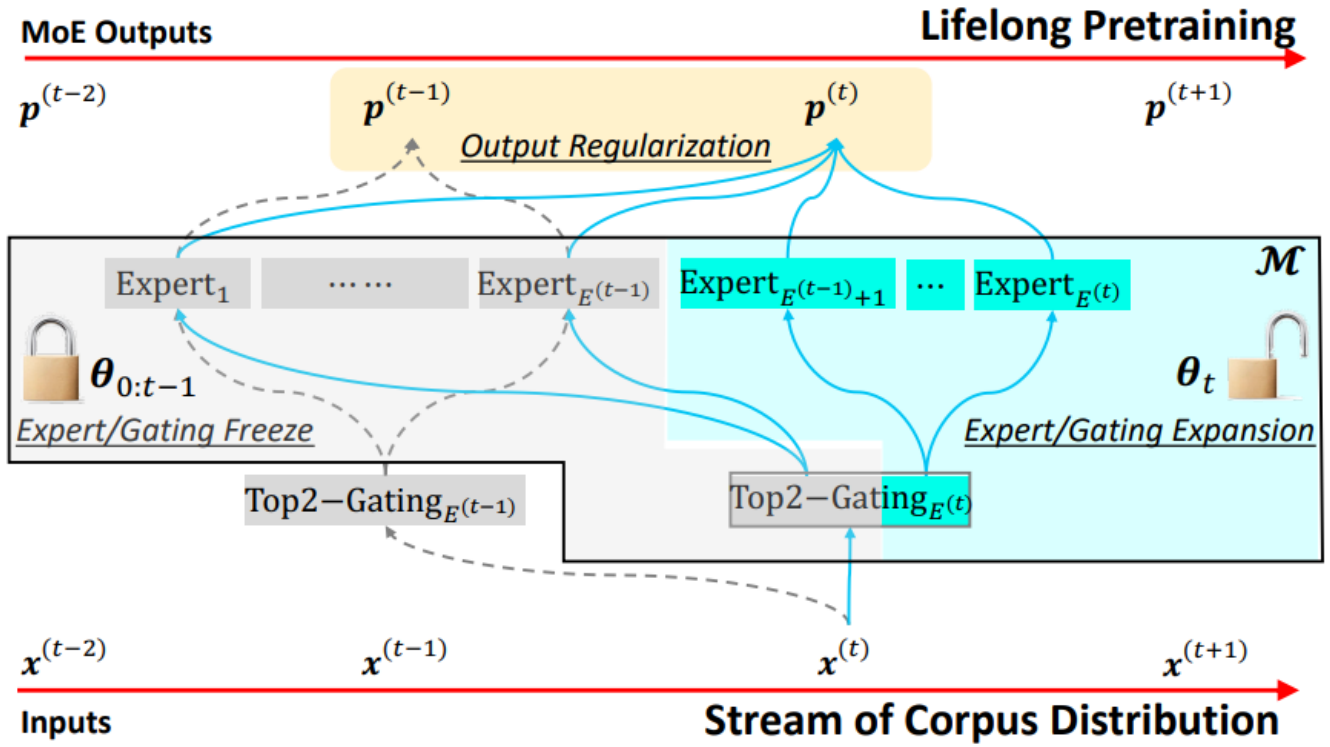
Table 4: Comparison between our Lifelong-MoE with dense GShard (Lepikhin et al., 2021), GLaM (Du et al., 2022), and classic lifelong learning methods. F1 score is evaluated on TriviaQA. Bleu is evaluated on WMT16.

| Experts | F1 Score | Bleu | Ubuntu | Avg. of 19 NLU Tasks |
|---|---|---|---|---|
| Dense + Online L2 Reg. | 12.99 | 5.66 | 27 | 48.65 |
| Dense + Memory Replay | 14.18 | 7.54 | 26 | 48.65 |
| Dense Oracle | 21.25 | 11.14 | 26 | 49.03 |
| GLaM | 21.76 | 6.97 | 26 | 50.9 |
| Lifelong-MoE (ours) | 20.22 | 19.16 | 27 | 50.26 |

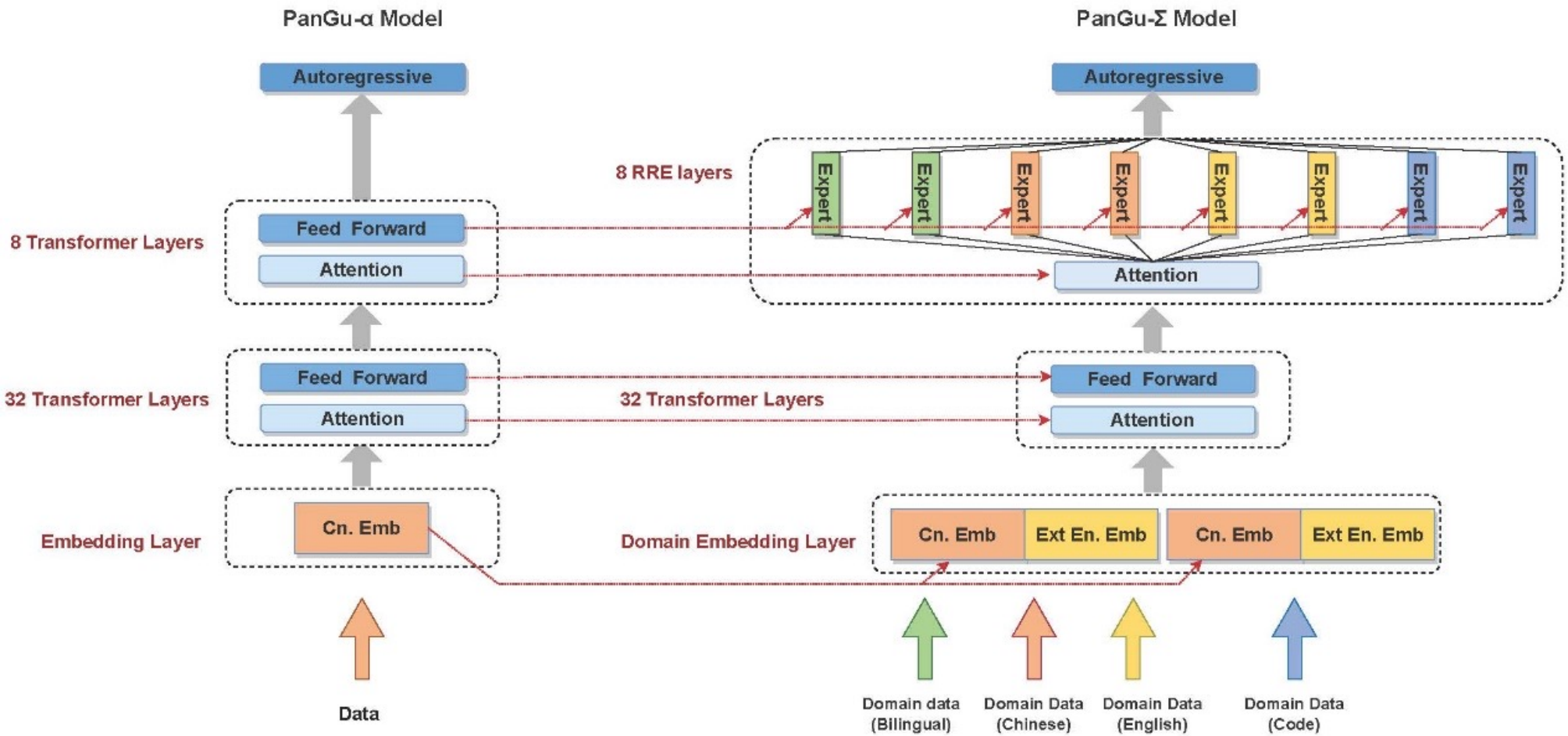
Chen, Wuyang, et al. "Lifelong language pretraining with distribution-specialized experts." ICML, 2023.

# Drawback

- Expand dimensions of gating layer with random initialization.
- Need extra regularization
- **Not pluggable**



# PanGu-Sigma

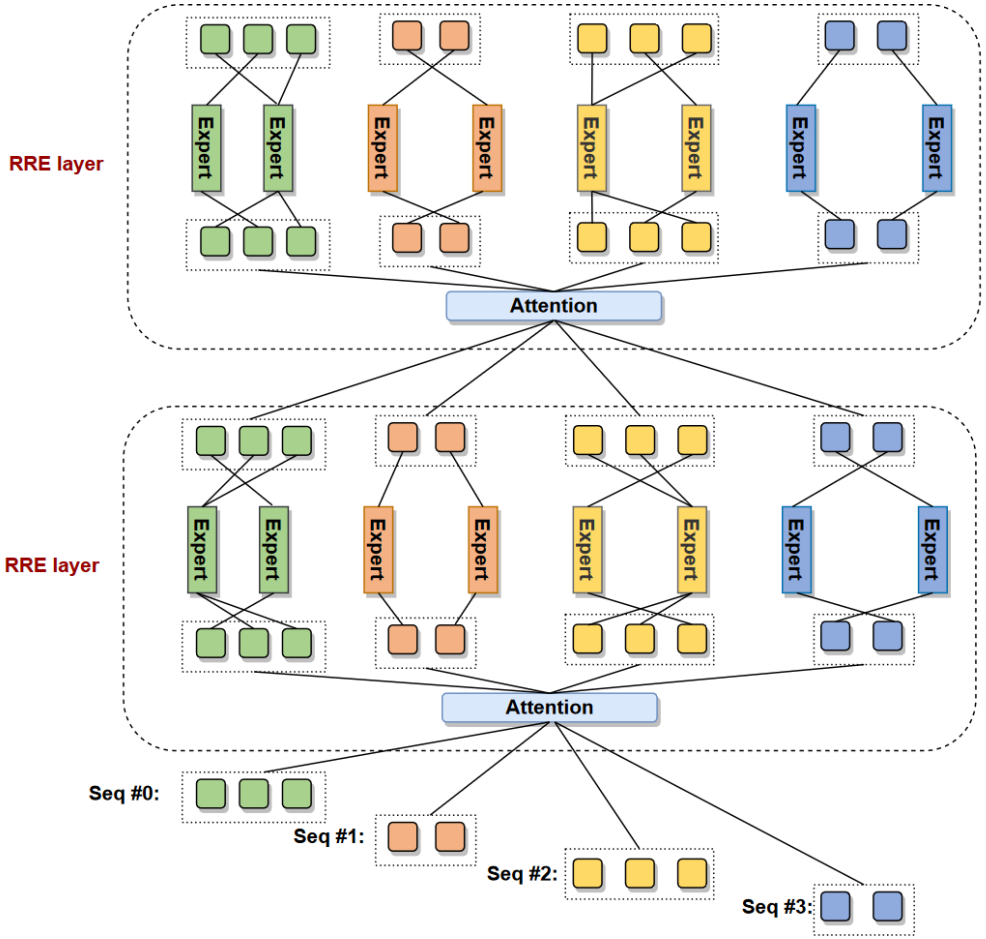



Ren, Xiaozhe, et al. "PanGu-{\Sigma}: Towards Trillion Parameter Language Model with Sparse Heterogeneous Computing." arXiv preprint arXiv:2303.10845 (2023).

# Random Routed Experts

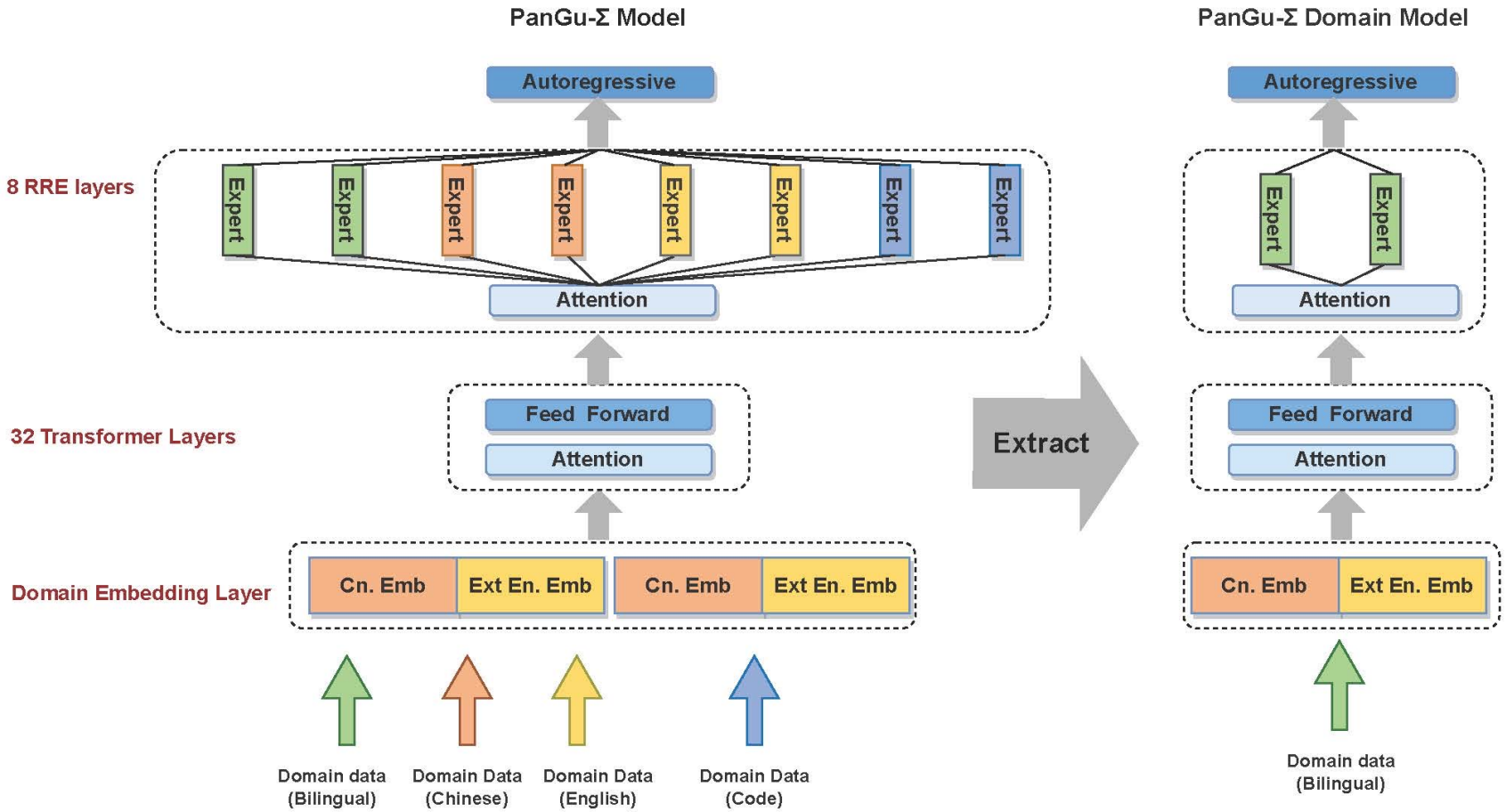
RRE做了双层的设计：

- 第一层，根据任务分配给不同的专家组（多个expert构成一个专家组，供一个task/domain使用）
- 第二层，使用组内随机Gating，让专家组的expert可以负载均衡。




Ren, Xiaozhe, et al. "PanGu-{\Sigma}: Towards Trillion Parameter Language Model with Sparse Heterogeneous Computing." arXiv preprint arXiv:2303.10845 (2023).

# Domain Extract




Ren, Xiaozhe, et al. "PanGu-{\Sigma}: Towards Trillion Parameter Language Model with Sparse Heterogeneous Computing." arXiv preprint arXiv:2303.10845 (2023).

# Deepseek

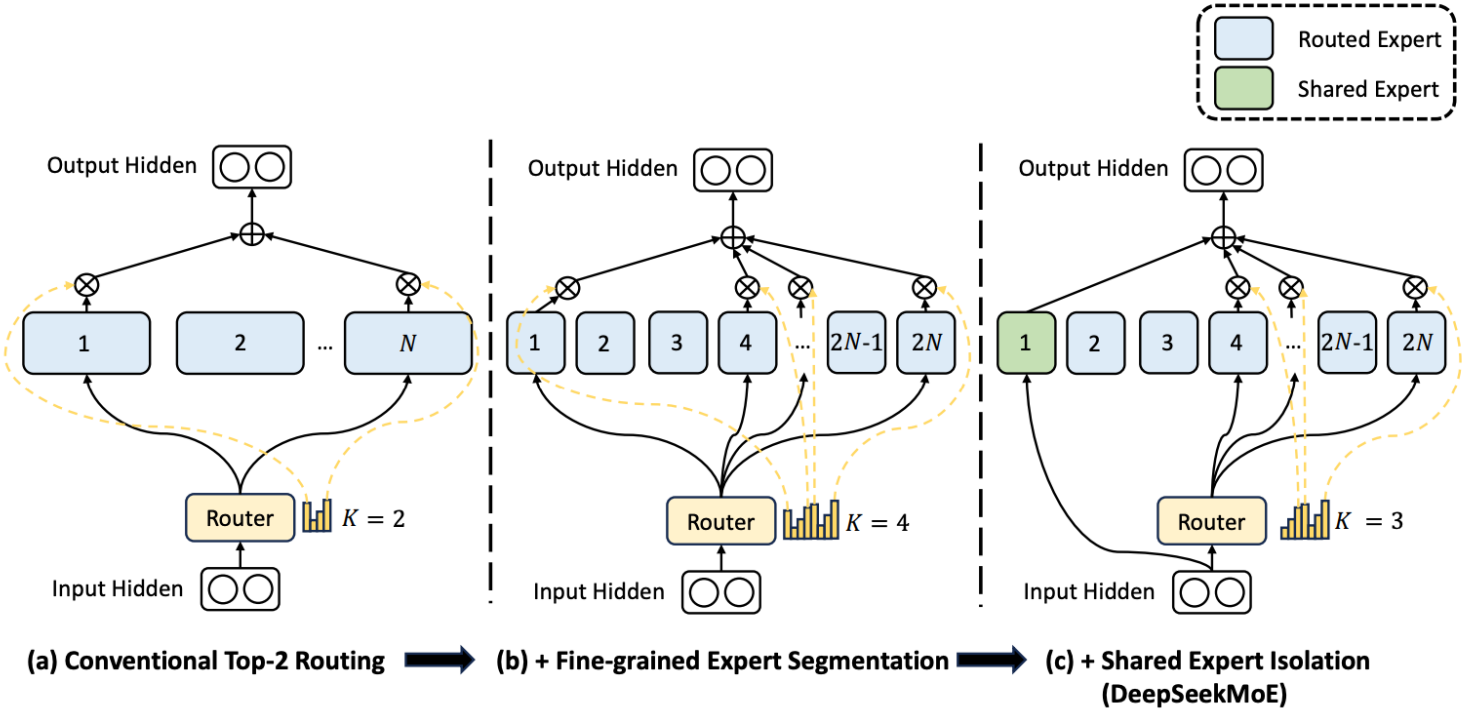



Figure 2 | Illustration of DeepSeekMoE. Subfigure (a) showcases an MoE layer with the conventional top-2 routing strategy. Subfigure (b) illustrates the fine-grained expert segmentation strategy. Subsequently, subfigure (c) demonstrates the integration of the shared expert isolation strategy, constituting the complete DeepSeekMoE architecture. It is noteworthy that across these three architectures, the number of expert parameters and computational costs remain constant.

# Deepseek

- (1)**知识杂交**：现有的MoE实践通常采用有限数量的专家(例如8或16)，因此分配给特定专家的标记可能会涵盖不同的知识。因此，指定的专家将倾向于在其参数中汇集不同类型的知识，这些知识很难同时利用。
- (2)**知识冗余**：分配给不同专家的token可能需要共同知识。因此，多个专家可能会在各自的参数中收敛到共享知识的获取，从而导致专家参数中的冗余。这些问题共同阻碍了现有MoE实践中的专家专业化，使其无法达到MoE模型的理论上限性能。

# Mixtral 8x7b演示

## Running large models

When running models that are too large to fit a single GPU's memory, use pipeline parallelism (PP) and `mpirun`. This is needed to run `Mixtral-7B-8x`. The code below does 2-way PP.

```
wget https://files.mixtral-8x7b-v0-1.mistral.ai/Mixtral-8x7B-v0.1-Instruct.tar
tar -xf Mixtral-8x7B-v0.1-Instruct.tar
```

Prepare `OpenMPI` lib:

```
conda install openmpi
```

```
mpirun -n 2 python -m main demo /path/to/Mixtral-8x7B-v0.1-Instruct/ --num_pipeline_ranks=2
```

> **Note**
>
> PP is not supported when running in interactive mode.