1. 论文名称：MoBA: Mixture of Block Attention for Long-Context LLMs
2. 第一作者：Moonshot-AI(月之暗面)
3. 论文链接：https://arxiv.org/abs/2502.13189
4. 最新日期：2025年2月18日
5. github：https://github.com/MoonshotAI/MoBA.git

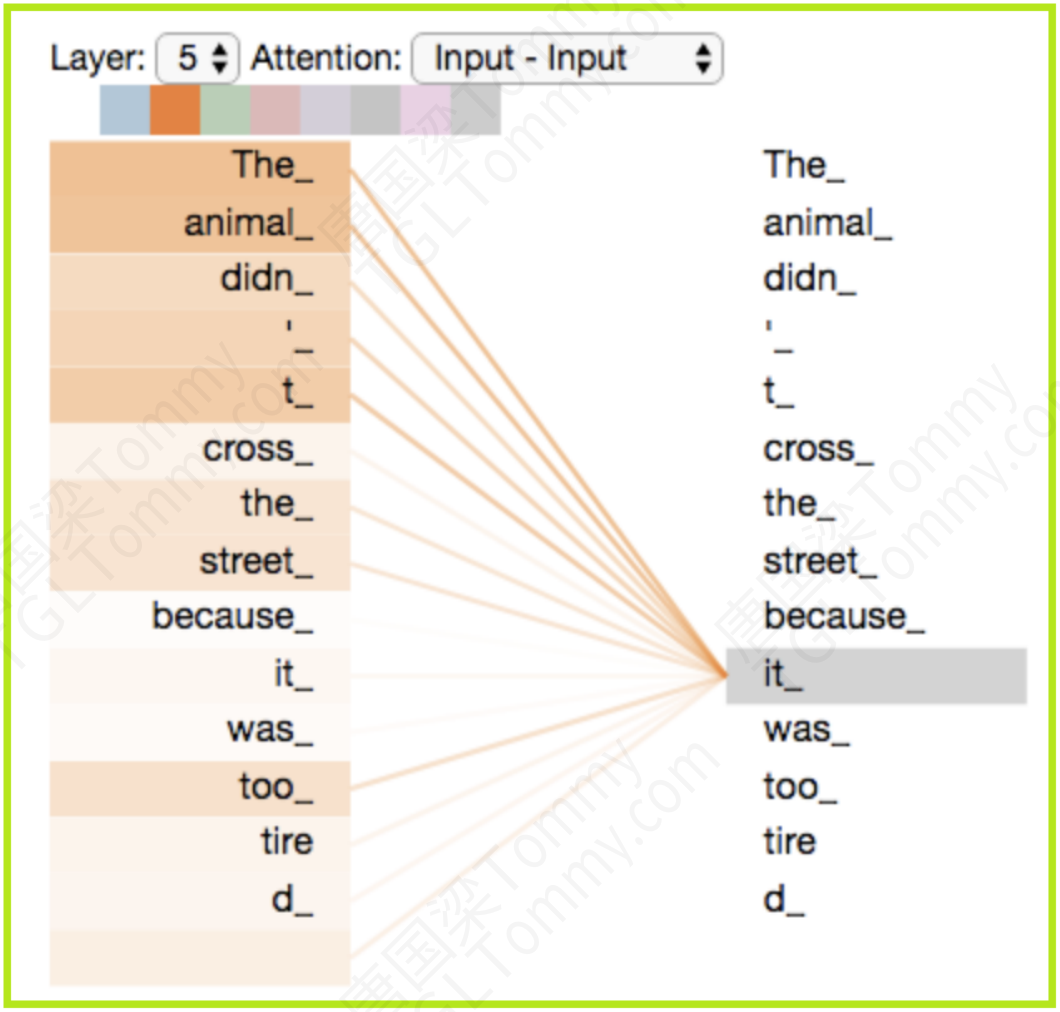
MoBA (Mixture of Block Attention) 混合块注意力

月之暗面（MoonshotAI）

1. 背景

- 问题概述

  大模型（LLMs）朝着通用人工智能（AGI）方向发展，扩展上下文长度至关重要。传统注意力机制的计算复杂度呈二次增长（计算复杂度O(N^2)）成为主要瓶颈，限制了长上下文（Long-Context）的处理能力。



- **现有方法的局限性**
  - **静态稀疏注意力**（如滑动窗口、汇聚注意力）：任务特定性强，通用性差。
  - **动态稀疏注意力**（如Reformer、Routing Transformer）：训练成本高，难以扩展。

- **解决方案：MoBA**
  - **核心思想**：

    提出 **块混合注意力**（MoBA, Mixture of Block Attention）架构，通过结合专家混合（MoE）和稀疏注意力机制，实现高效长序列处理。MoBA 允许模型自主决定注意力关注区域，避免了预定义偏差。

  - **关键特性**：
    - 支持全注意力(Full Attention)与稀疏(Sparse Attention)模式无缝切换。
    - 保持因果性，避免信息泄漏。
    - 计算复杂度降低至 $O(NlogN)$ 。
  - **优势**：
    - **效率**：相比全注意力，计算时间减少6.5倍（1M上下文）。
    - **性能**：在长上下文任务（如32K-1M序列）中损失接近全注意力。

2. 方法：MoBA 架构

一种新的架构，称为**块混合注意力（MoBA）**，通过动态选择历史片段（块）进行注意力计算，从而增强了Transformer模型的能力。

MoBA的关键创新在于将MoE的思想应用于注意力机制本身，进而更高效地处理长序列。

- **专家混合（MoE）**，该方法主要应用于Transformer架构中的前馈网络层。
- **稀疏注意力**，其核心思想是只选择最相关的部分进行计算。

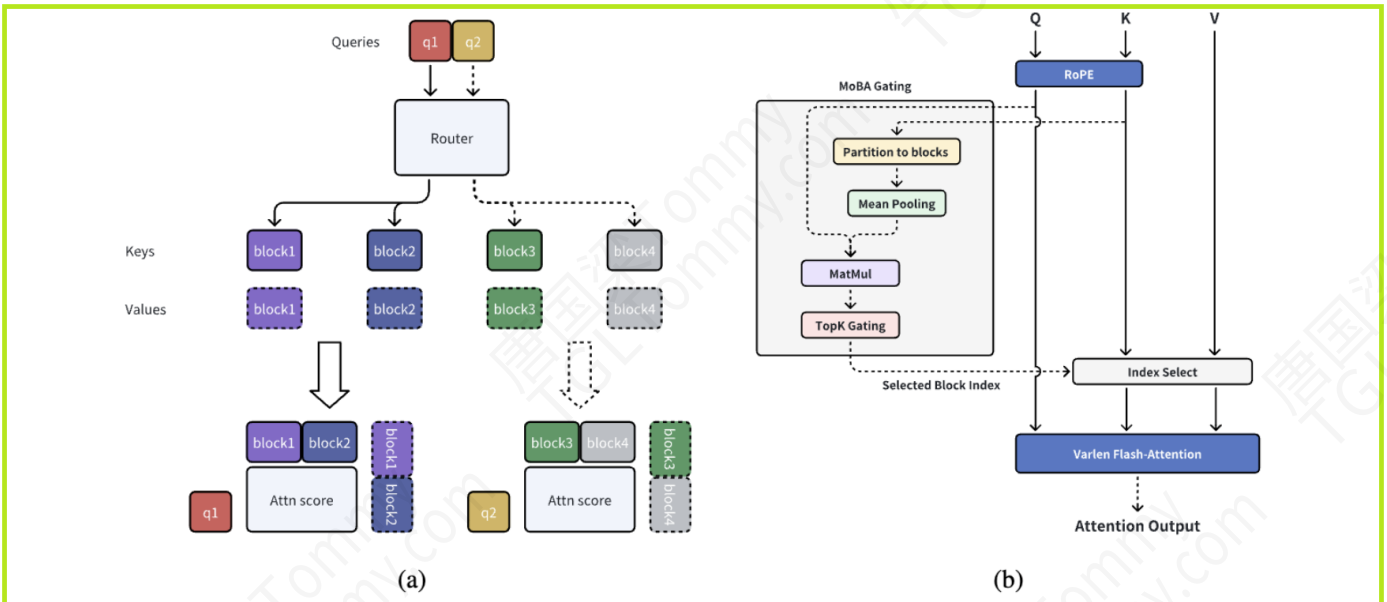


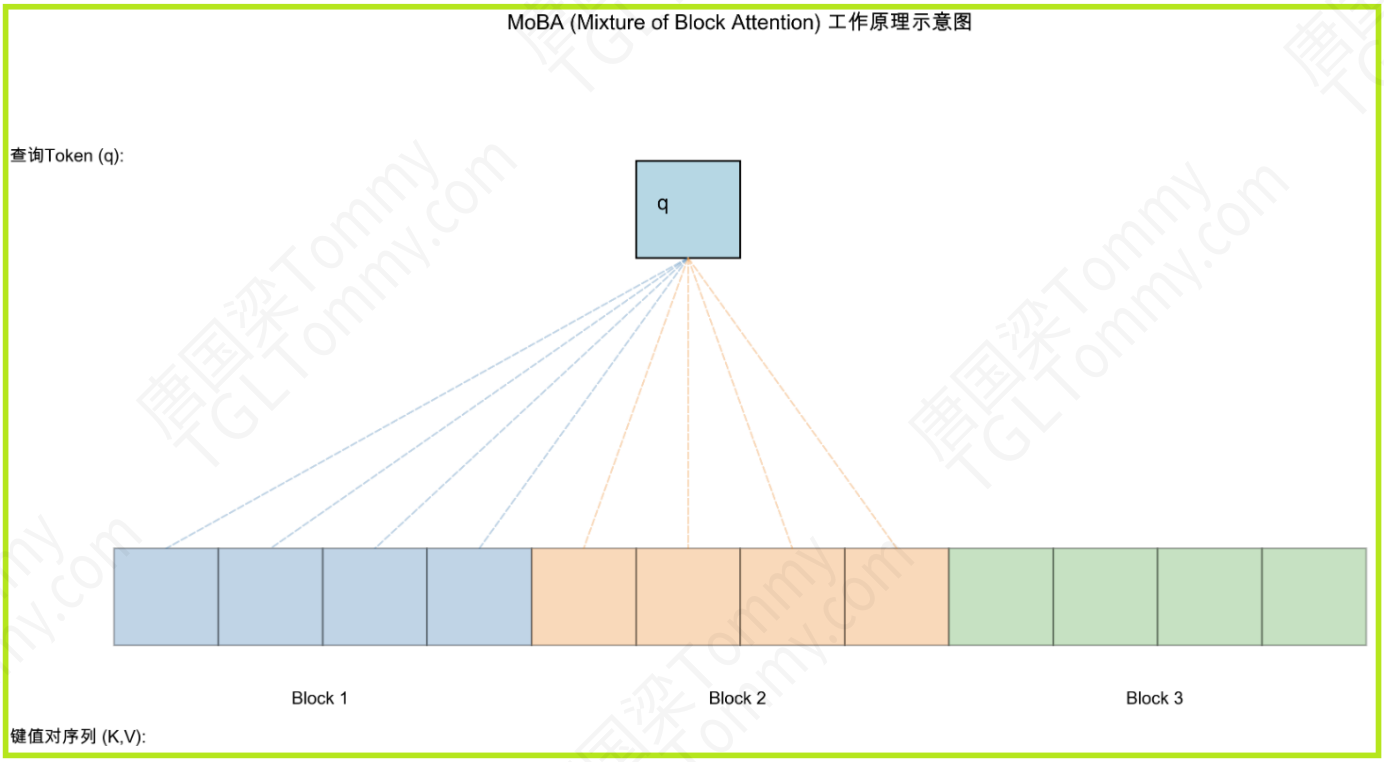
Figure 1: Illustration of mixture of block attention (MoBA). **(a)** A running example of MoBA; **(b)** Integration of MoBA into Flash Attention.

2.1 块划分

MoBA方法对标准注意力机制进行了修改，使得每个查询令牌不再关注整个上下文，而是只关注其中一个子集（称为“块”）。这些子集是动态选择的，每个查询令牌会只关注一部分键和值，如下公式所示：

$$\mathrm{MoBA}(q, K, V) = \mathrm{Softmax}(qK[I]^T)V[I]$$

其中，$I \subseteq [N]$ 表示选择的键值对子集。



MoBA的关键创新在于如何将整个上下文划分为**块**。具体来说，将长度为 N 的上下文划分为 n 个块，每个块代表一部分连续的tokens。假设 N 可以被 n 整除，则每个块的大小为 $B = \frac{N}{n}$ 。对于第 i 个块，其索引范围为：

$[(i-1)\times B+1, i\times B]$

即，第一个块包含从第 1 到第 B 的tokens，第二个块包含从第 $B+1$ 到第 $2B$ 的tokens，以此类推。

2.2 动态路由

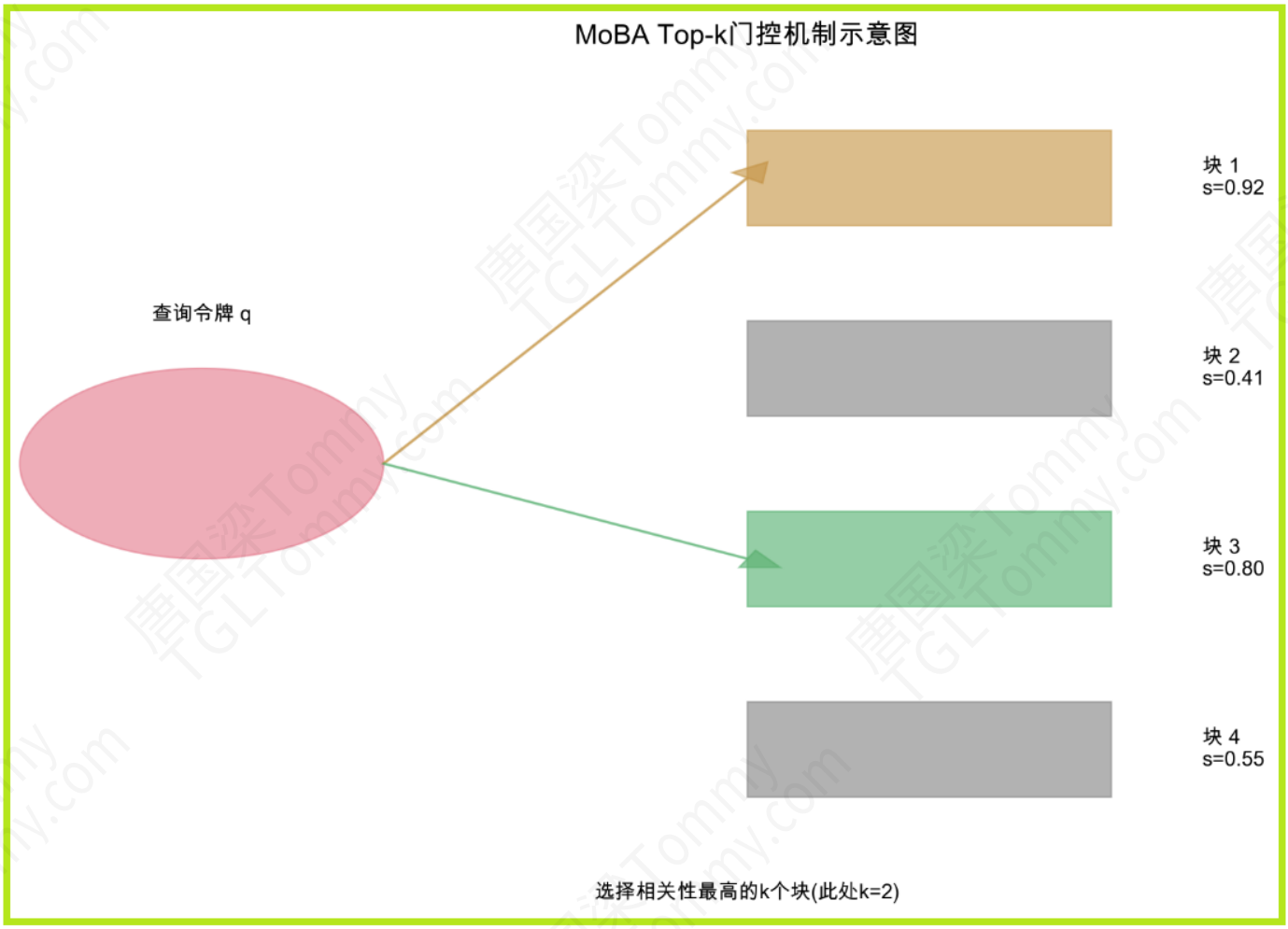
MoBA进一步应用了 **top-k门控机制**，这一机制借鉴了专家混合（MoE）的思想，能够选择最相关的块。第 i 个块的门控值 g_i 是根据查询令牌 q 与第 i 个块的相关性来计算的，公式如下：

$$g_i = \begin{cases} 1 & \text{如果 } s_i \in \text{Topk}(\{s_j | j \in [n]\}, k) \\ 0 & \text{否则} \end{cases}$$

其中，s_i 表示查询令牌 q 与第 i 块之间的 **亲和度分数** 。这个分数通过查询令牌 q 与该块中所有键的**均值池化**的内积来计算：

$$s_i = \langle q, \text{mean.pool}(K[I_i]) \rangle$$

这个分数表示每个块与查询的相关性，选择相关性最强的 k 个块来进行计算。通过这种机制，MoBA能够将注意力集中在最相关的块，而不是整个上下文。



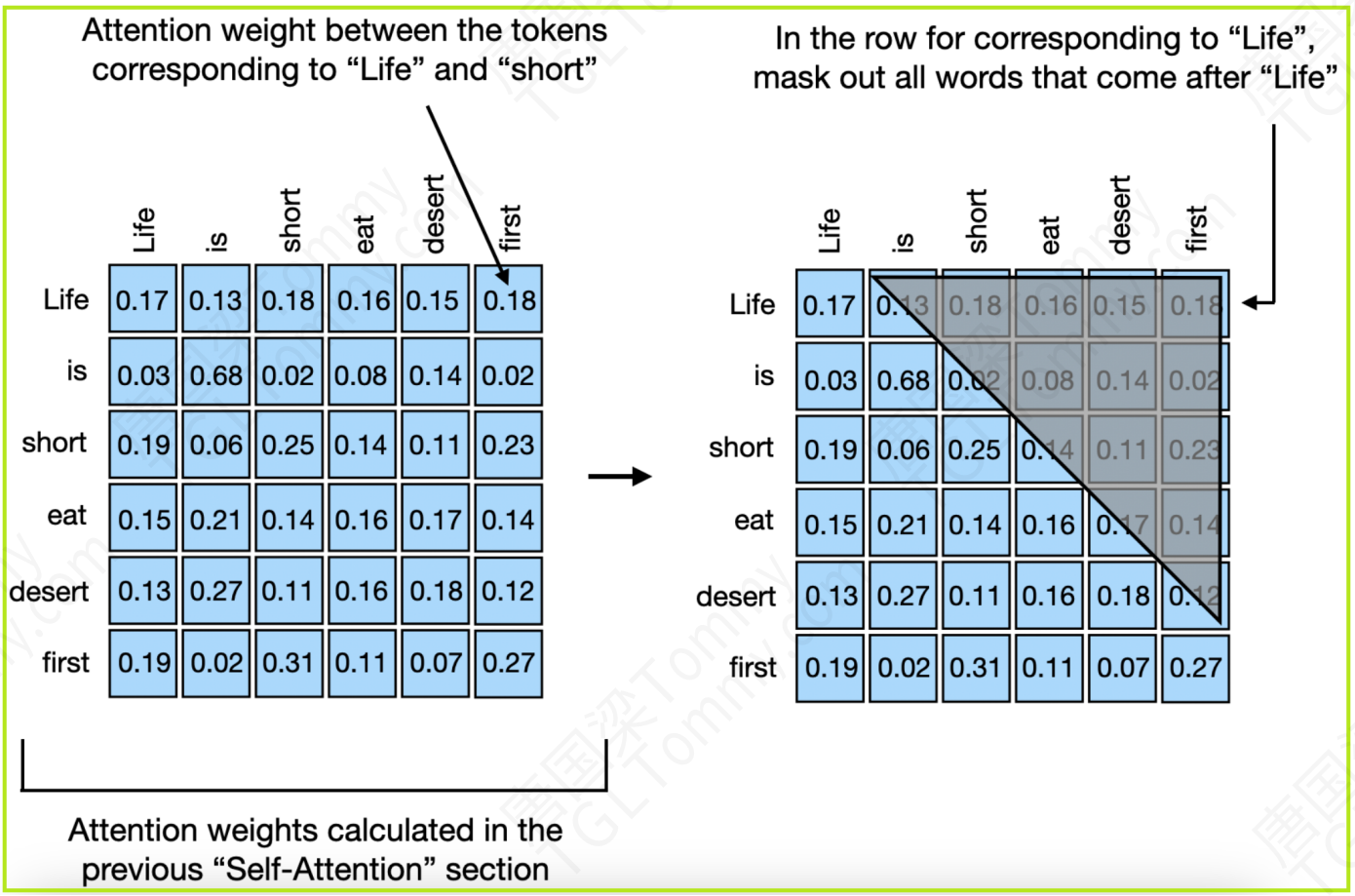
2.3 因果性约束

MoBA保证每个查询令牌只能关注其当前块，并在当前块的注意力计算中应用因果掩码（causal mask），以避免包含来自未来tokens的信息。

更形式化地表示，当查询令牌的位置 $pos(q)$ 属于当前块 I_i 时，MoBA确保查询令牌仅关注其当前块的信息：

$$g_i = 1 \quad (\text{if } pos(q) \in I_i)$$

同时，这种方法鼓励查询令牌对本地上下文进行注意，而不是泄漏未来信息。



3. MoBA 实现

MoBA的高效实现，结合了来自FlashAttention和MoE的优化技术。

3.1 主要步骤

1️⃣ **确定查询令牌与KV块的映射关系**：根据门控网络和因果掩码，确定每个查询令牌对应的KV块。简单来说，就是决定每个查询令牌应该关注哪些KV块。

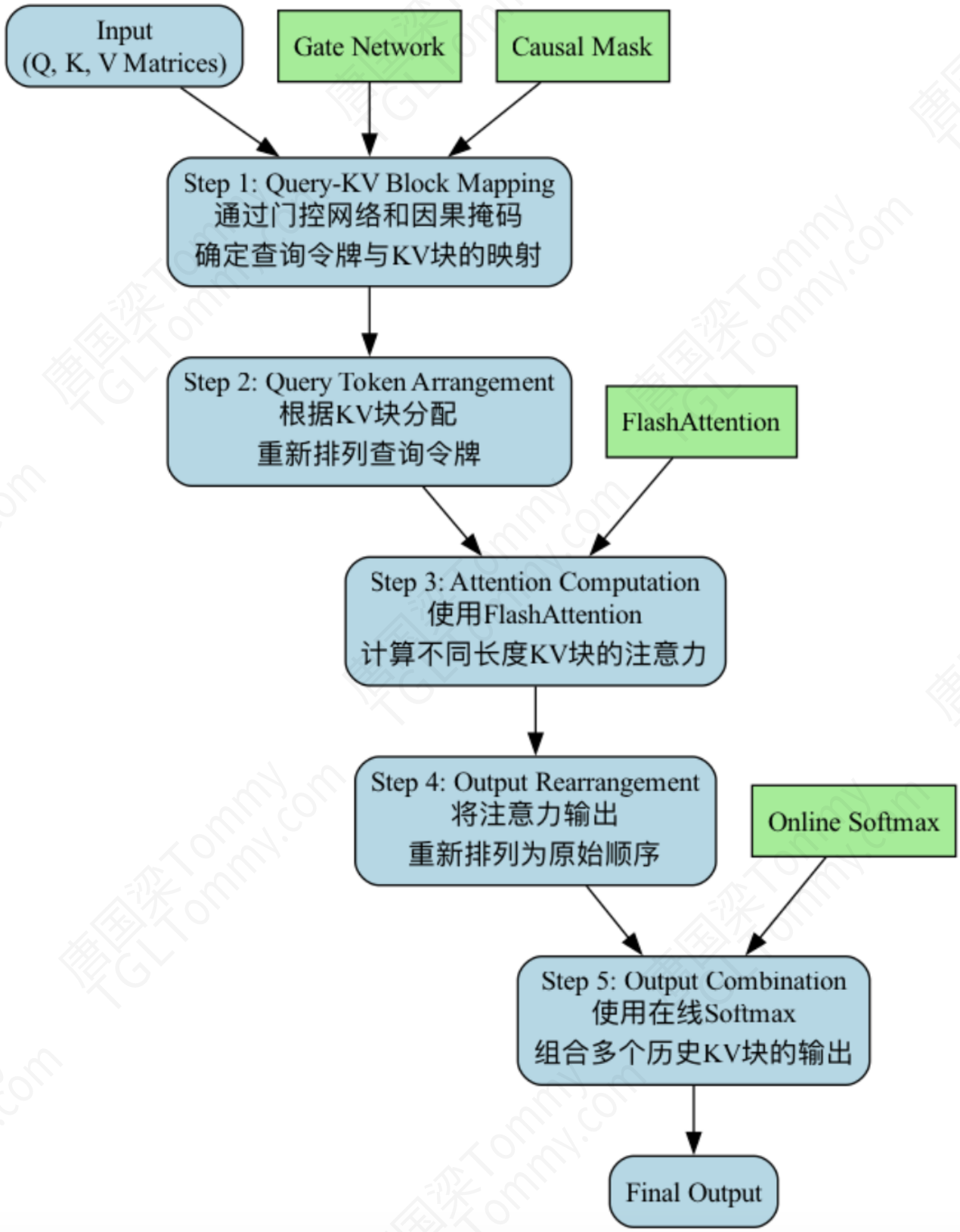
2️⃣ **安排查询令牌的顺序**：根据查询令牌分配到的KV块，重新排列查询令牌的顺序。这样做是为了便于后续的并行计算。

3️⃣ **计算每个KV块及其对应的查询令牌的注意力输出**：这一过程可以通过FlashAttention来优化，该方法能够处理具有不同长度的KV块。FlashAttention 是一种高效的注意力计算方法，能够加速计算过程。

4️⃣ **重新排列注意力输出**：将计算出的注意力输出重新排列为原始顺序。因为之前为了计算效率，调整了查询令牌的顺序，所以现在需要恢复原来的顺序。

5️⃣ **结合对应的注意力输出**：使用在线Softmax将查询令牌的注意力输出与多个历史KV块结合，允许查询令牌同时关注当前块和多个历史块。

> - Softmax函数通常用于将多个数值转换为概率分布，确保所有数值的总和为1。
> - "在线"：在计算过程中动态地进行Softmax归一化，而不是预先计算好的。



3.2 算法示例

实现MoBA（Mixture of Block Attention）模型，特别是对查询（Q）、键（K）和值（V）矩阵的操作，结合了门控网络和因果掩码。

- 输入参数：查询、键、值矩阵
  - $Q, K, V \in \mathbb{R}^{N \times h \times d}$，其中：
    - N 表示上下文的长度。
    - h 表示注意力头的数量。
    - d 是每个注意力头的维度。
  - MoB. A的超参数包括块的大小 B 和选择的 top-k。
  - $n = N/B$ 表示块的数量。

第1步：将KV拆分成多个块：

```
{K̃i, Ṽi} = split.blocks(K, V, B), where K̃i, Ṽi ∈ R^(B×h×d), i ∈ [n]
```

- 这一步将键（K）和值（V）矩阵按照块大小 B 进行拆分，得到多个子块 K_i 和 V_i，每个子块的大小为 $B \times h \times d$，i 表示第 i 个块。

第2步：计算门控分数：

```
K̃ = mean_pool(K, B) ∈ R^(n×h×d)
S = QK̃^T ∈ R^(N×h×n)
```

- 通过对键矩阵进行均值池化（mean pooling）操作，得到每个块的聚合表示 $\tilde{K}$，其维度为 $n \times h \times d$。
- 然后，计算查询矩阵 Q 和加权的键矩阵 $\tilde{K}$ 之间的点积，得到一个相关性矩阵 S，它的维度为 $N \times h \times n$，表示每个查询令牌和各个块之间的相关性。

第3步：选择满足因果约束的块：

```
M = create_causal_mask(N, n)
G = topk(S + M, k)
```

- 生成一个因果掩码矩阵 M，该掩码确保查询令牌不会“看到”未来的块，从而保持因果性。
- 然后，计算 S 和因果掩码 M 的和，使用 top-k 操作从中选择出最相关的 k 个块，以得到最终的块选择 G。

第4步：为计算效率组织注意力模式：

```
Qs, K̃s, Ṽs = get_self_attn_block(Q, K̃, Ṽ) # 提取查询令牌（Q）所在的当前块
Qm, K̃m, Ṽm = index_select_moba_attn_block(Q, K̃, Ṽ, G) # 根据门控G的结果，用于提取历史块
```

- 使用函数 `get_self_attn_block` 获取自注意力的块表示 Q_s, K_s, V_s 。
- 通过函数 `index_select_moba_attn_block`，根据门控 G 从 $\tilde{K}, \tilde{V}$ 中选择出最相关的块，得到查询和键值的子集。

第5步：分别计算注意力：

```
Os = flash_attention_varlen(Qs, K̃s, Ṽs, causal=True)
Om = flash_attention_varlen(Qm, K̃m, Ṽm, causal=False)
```

- 使用 **FlashAttention** 方法计算注意力输出，其中 `flash_attention_varlen` 是一个处理变长序列的高效注意力计算方法。
- 对于当前块的注意力（O_s），我们应用因果掩码（`causal=True`），**确保每个查询令牌只能关注到当前块中它前面的令牌**。
- 对于多个历史块的注意力（O_m），不应用因果掩码（`causal=False`），因为**历史块中的所有令牌都可以被当前查询令牌关注**。

第6步：结合结果并应用在线Softmax：

```
O = combine_with_online_softmax(Os, Om)
```

- 最后，将当前块和多个历史块的注意力输出 O_s 和 O_m 结合起来，并使用在线 Softmax进行归一化。

第7步：返回最终的输出：

```
return O
```

- 返回最终的注意力输出 O，它包含了查询令牌对于不同块的注意力值。

4. 模型评估

- **实验目标**：评估MoBA在真实世界下游任务中的表现，并与全注意力模型进行比较。
- **结果分析**：
  - **表2** 中展示了MoBA与全注意力在多个基准任务中的性能比较。MoBA的表现与全注意力模型非常接近，尤其在处理长上下文的任务中，如 **RULER**。

Benchmark	Llama-8B-1M-MoBA	Llama-8B-1M-Full
AGIEval [0-shot]	0.5144	**0.5146**
BBH [3-shot]	0.6573	**0.6589**
CEval [5-shot]	**0.6273**	0.6165
GSM8K [5-shot]	**0.7278**	0.7142
HellaSWAG [0-shot]	0.8262	**0.8279**
Loogle [0-shot]	**0.4209**	0.4016
Competition Math [0-shot]	0.4254	**0.4324**
MBPP [3-shot]	**0.5380**	0.5320
MBPP Sanitized [0-shot]	**0.6926**	0.6615
MMLU [0-shot]	0.4903	**0.4904**
MMLU Pro [5-shot][CoT]	0.4295	**0.4328**
OpenAI HumanEval [0-shot][pass@1]	0.6951	**0.7012**
SimpleQA [0-shot]	0.0465	**0.0492**
TriviaQA [0-shot]	**0.5673**	0.5667
LongBench @32K [0-shot]	**0.4828**	0.4821
RULER @128K [0-shot]	0.7818	**0.7849**

Table 2: Performance comparison between MoBA and full Attention across different evaluation benchmarks.

- 图7 中显示了MoBA在 **长上下文（1M tokens）** 上的表现，MoBA在“Needle in the Haystack”基准测试中表现良好，即使上下文长度达到1百万个令牌。

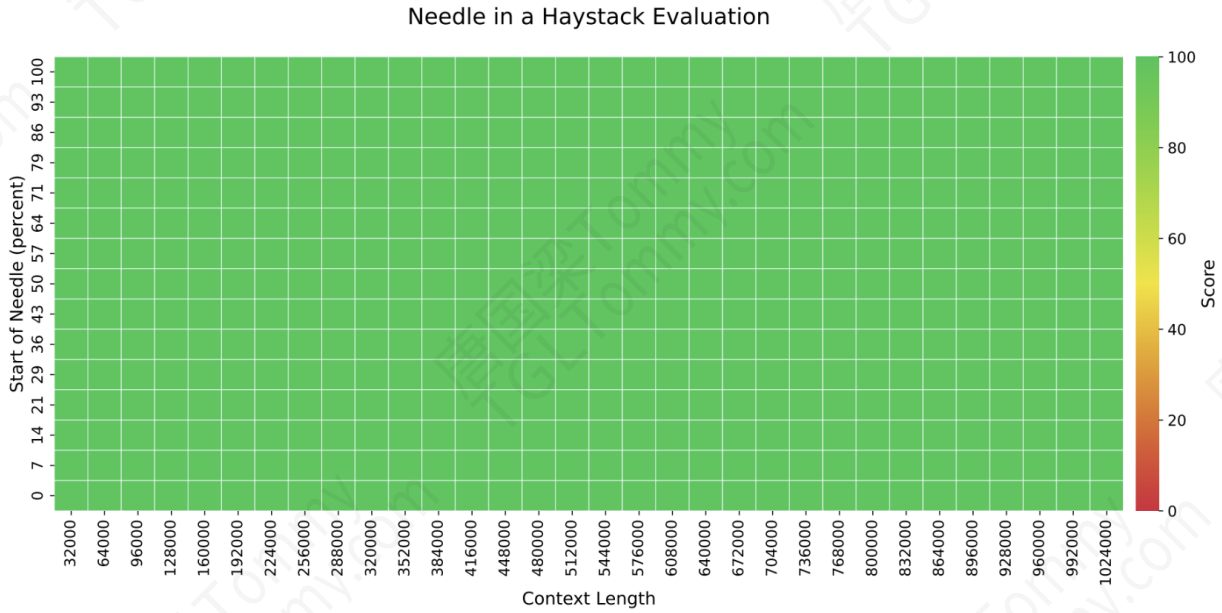


Figure 7: Performance of LLama-8B-1M-MoBA on the Needle in the Haystack benchmark (upto 1M context length).

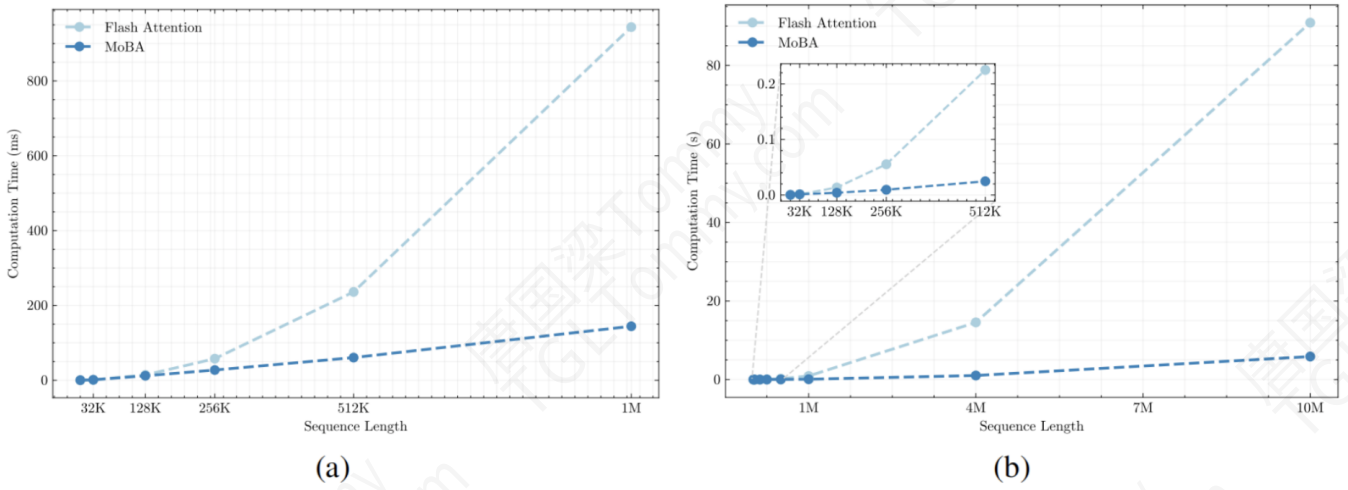


图2：MoBA与全注意力（使用Flash Attention实现）效率对比。 (a) 1M模型加速评估：在序列长度（8K-1M）增加的情况下，MoBA与Flash Attention的计算时间对比。 (b) 固定稀疏比缩放：在序列长度（8K-10M）增加的情况下，保持95.31%的固定稀疏比（固定64个MoBA块，top-k=3）。

5. 结论

- **MoBA的贡献**
  - 提出了 **Mixture of Block Attention (MoBA)**，一种新型的注意力架构，灵感来源于 **专家混合（MoE）** 原理，旨在提高大模型（LLMs）处理长上下文任务的效率和可扩展性。
  - MoBA通过将上下文划分为块，并通过动态路由机制选择最相关的键值块，显著降低了传统注意力机制的计算复杂度。

- **MoBA的优势**
  - MoBA不仅减少了计算复杂度，还在性能上与全注意力机制相当，尤其在长上下文任务中表现优越。
  - MoBA能够 **无缝切换** 传统全注意力与稀疏注意力模式，在训练和推理过程中实现更高的计算效率。
  - 通过 **实验验证**，MoBA在不同上下文长度下都表现出优越的性能，尤其在处理长序列时，能够显著提高效率，并保持较低的语言模型损失。

- **未来的研究方向**
  - 未来的研究可以进一步优化MoBA的 **块选择策略**，探索如何在不同任务中更高效地选择相关块。
  - 还可以考虑将MoBA应用到 **多模态任务**，并研究其对 **复杂推理任务** 的影响，以提升其在广泛应用场景中的表现和泛化能力。