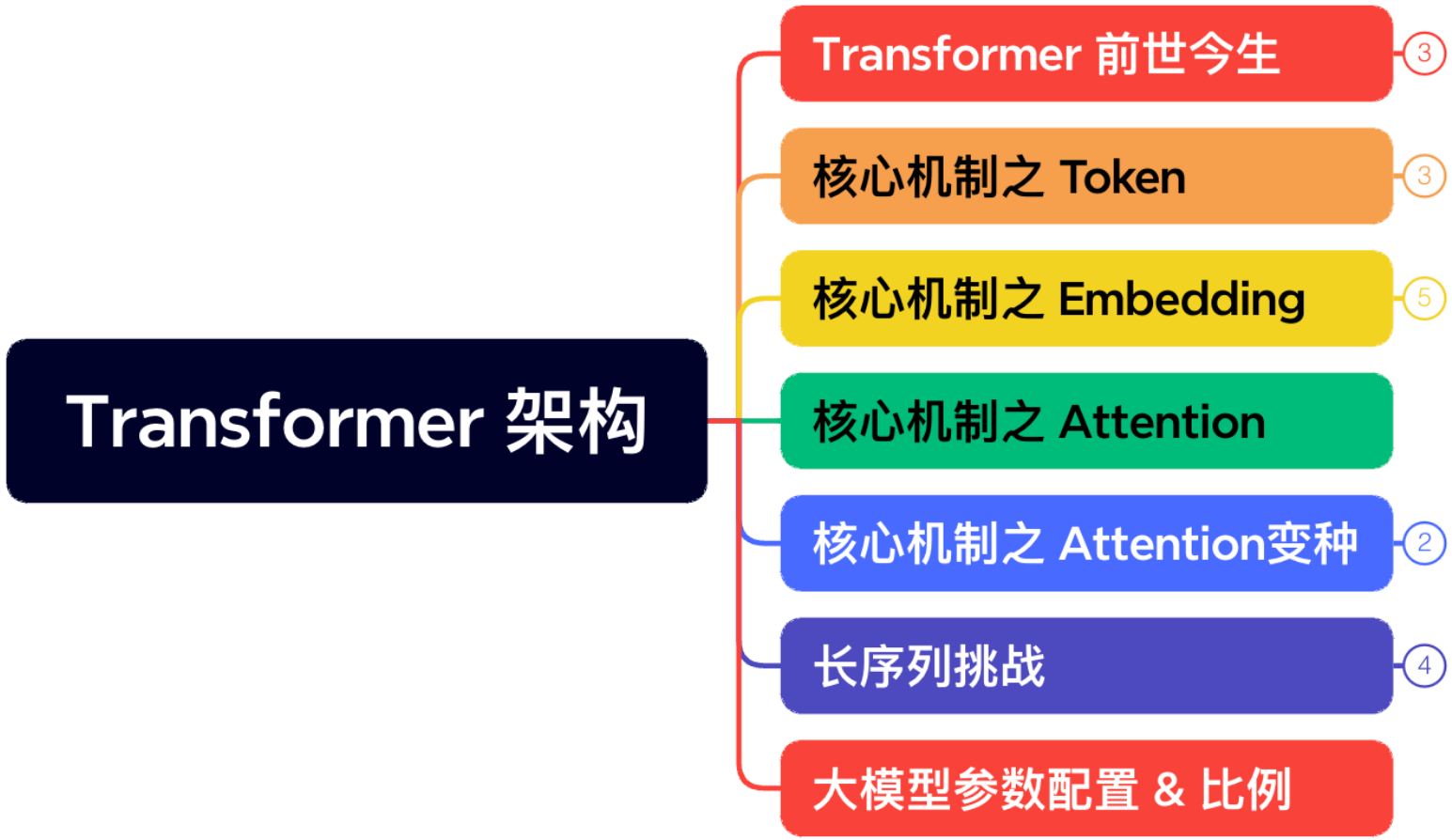



Attention
长序列支持


Contenet


Transformer 架构
Transformer 前世今生 3
核心机制之 Token 3
核心机制之 Embedding 5
核心机制之 Attention
核心机制之 Attention变种 2
长序列挑战 4
大模型参数配置 & 比例


好好想想

视频目录大纲

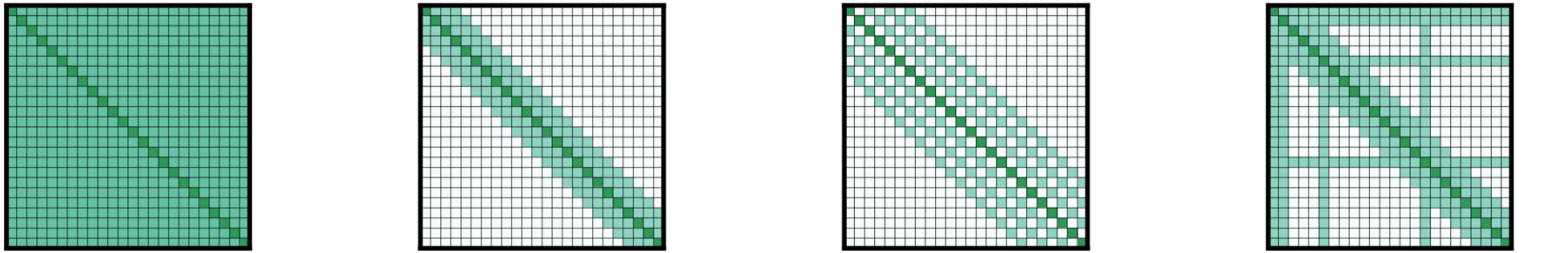
- 长序列 Transformer 架构（Longformer、 Transformer-XL）
- 长序列 Attention 架构（NSA、 MoBA）

01

Longformer

Sliding Window Attention

- Sliding window attention 设定窗口 w，规定序列中每个token只能看到 w 个token，其左右两侧能看到 $1/2\ w$ 个token，时间复杂度为 $O(n \times w)$。
- 不需要担心 Longformer 无法建立整个序列语义信息，transformer 模型结构本身是层层叠加结构，模型高层相比底层具有更宽广的感受野，因此能去建模融合全部序列信息的全局表示。

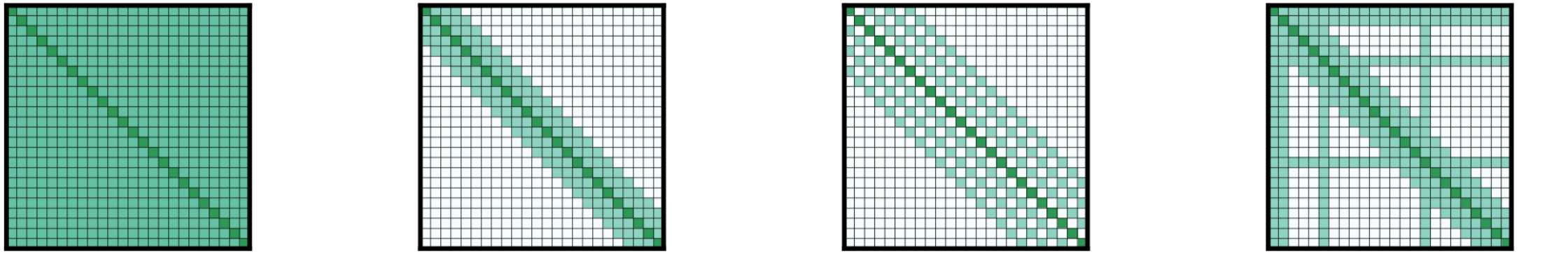


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Dilated Sliding Window

- Sliding Window Attention 只能考虑长度为 w 上下文，不增加计算量情况下，在进行 Self-Attention 两个相邻 token 间会存在大小为 d 间隙，这样序列中每个token感受野范围可扩展到 $d \times w$。第 m 层，感受野的范围是 $m \times d \times w$ 。

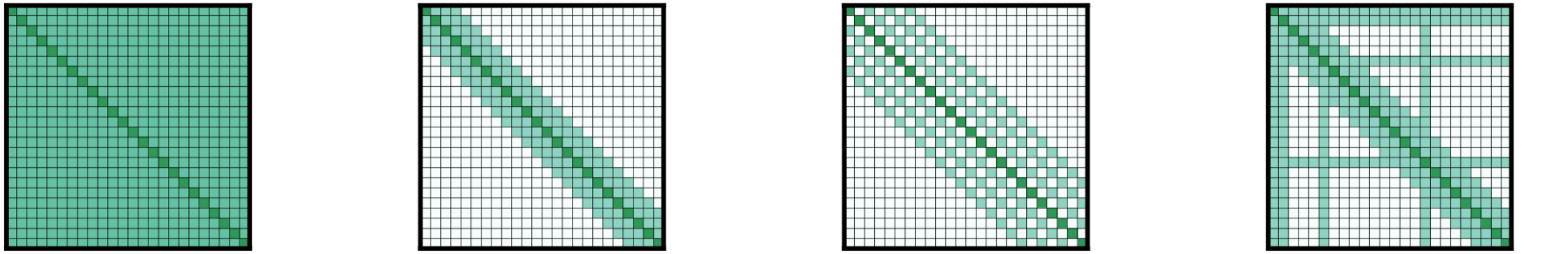


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Global+Sliding Window

- 将原始输入分别映射到 Q, K, V 三个空间后进行 Attention 计算，Global+Sliding Window 这里涉及到两种 Attention；
- Longformer 中分别将这两种 Attention 映射到两个独立空间，即使用 Qs, Ks, Vs 来计算 Sliding Window Attention，使用 Qg, Kg, Vg 来计算Global Attention。



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

02

Transformer-XL

Transformer-XL介绍

- 为了解决上下文碎片和推理速度慢的问题，作者推出了片段递归机制，为了解决长期依赖，作者对绝对位置编码进行了改进，并推出了相对位置编码机制。

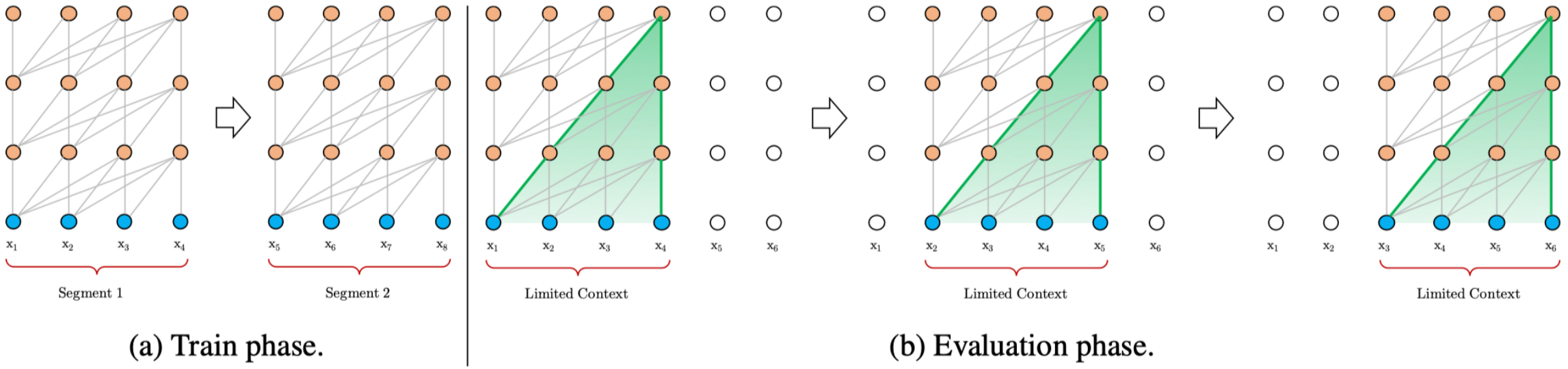
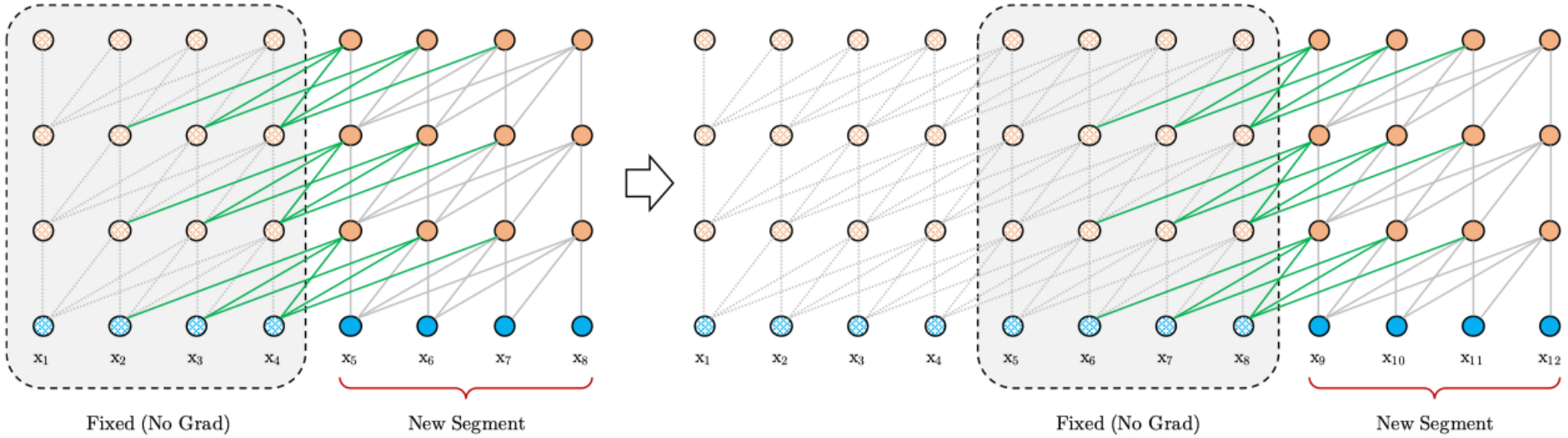


Figure 1: Illustration of the vanilla model with a segment length 4.

片段递归

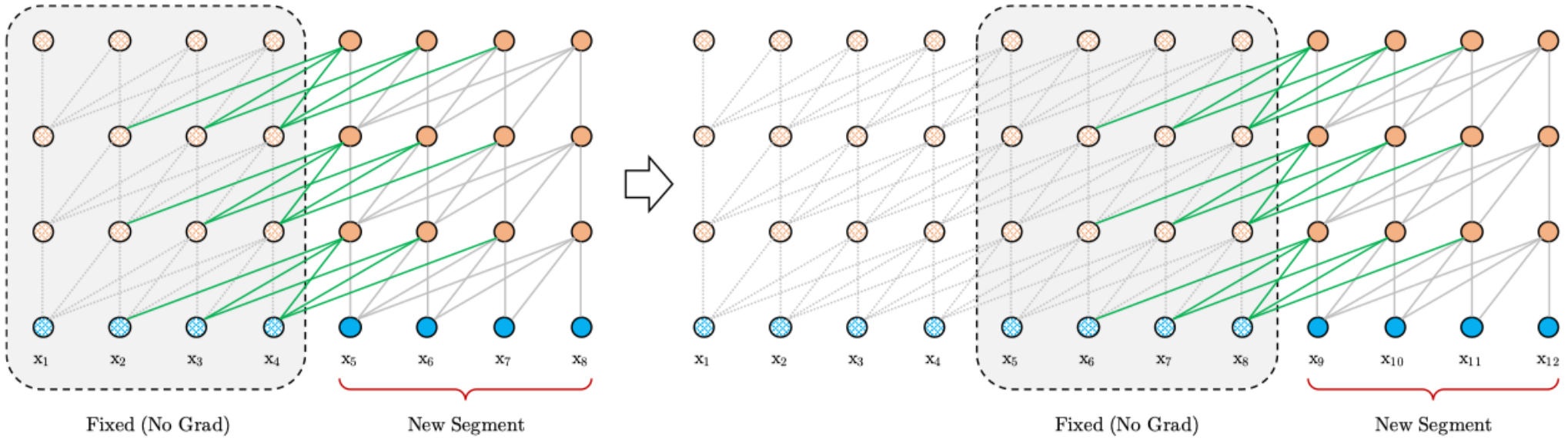
- Transformer-XL的上一个片段的状态会被缓存下来然后在计算当前段的时候再重复使用上个时间片的隐层状态。因为上个片段的特征在当前片段进行了重复使用，这也就赋予了Transformer-XL建模更长期的依赖的能力。



(a) Training phase.

片段递归

$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$
$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$
$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$



(a) Training phase.

相对位置编码

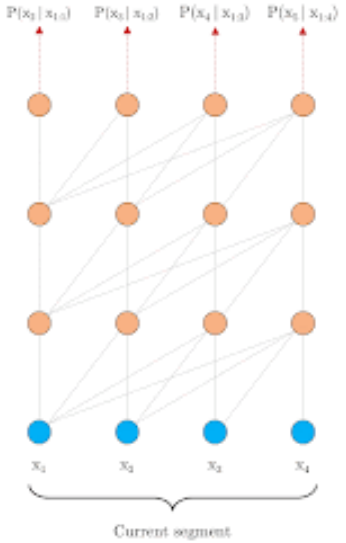
- 参考了RPR中把相对位置编码加入到self-attention中的思想：

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{(d)}.$$

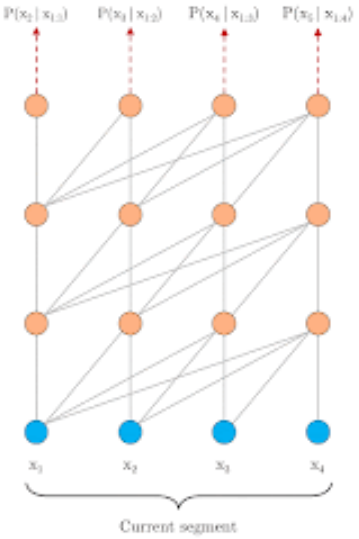
- **三个变化：**

1. W_k 被拆分成立 $W_{k,E}$ 和 $W_{k,R}$，输入序列和位置编码不再共享权值。
2. b 和 d 将绝对位置编码 U_j 换成了相对位置编码 R_{i-j}，其中 R 采用 sinsoid 编码矩阵。
3. c 和 d 引入两个可学习参数 $u \in \mathbb{R}^d$ 和 $v \in \mathbb{R}^d$ 替换 Transformer 中 query 向量 $U_i^T W_q^T$。表明对所有 query 位置对应 query 向量是相同，无论 query 位置如何，对不同词注意偏差都保持一致。

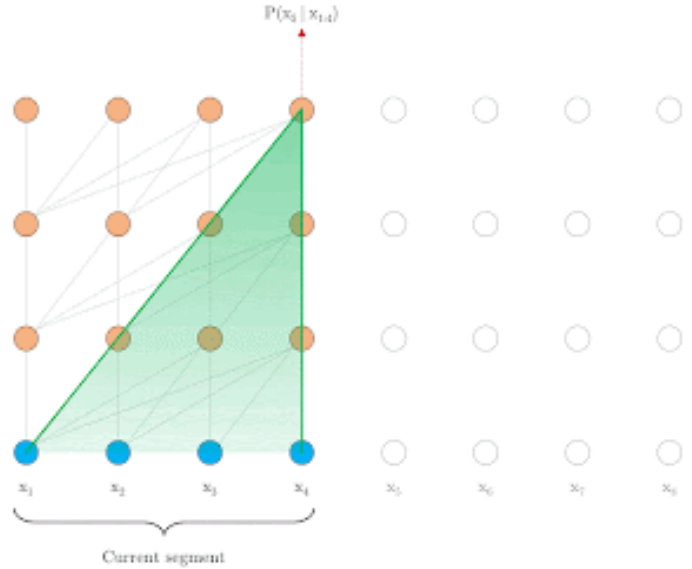
Vanilla Transformer



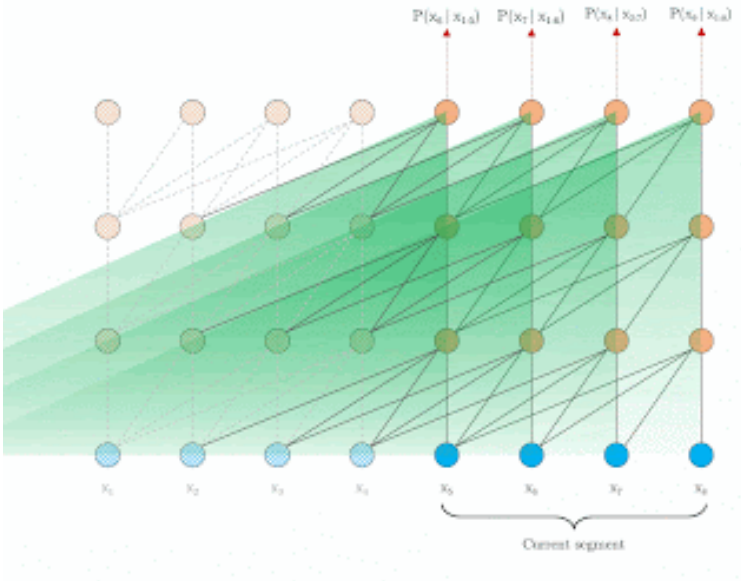
Transformer-XL



Vanilla Transformer



Transformer-XL



03

DS NSA

稀疏注意力（Sparse Attention）

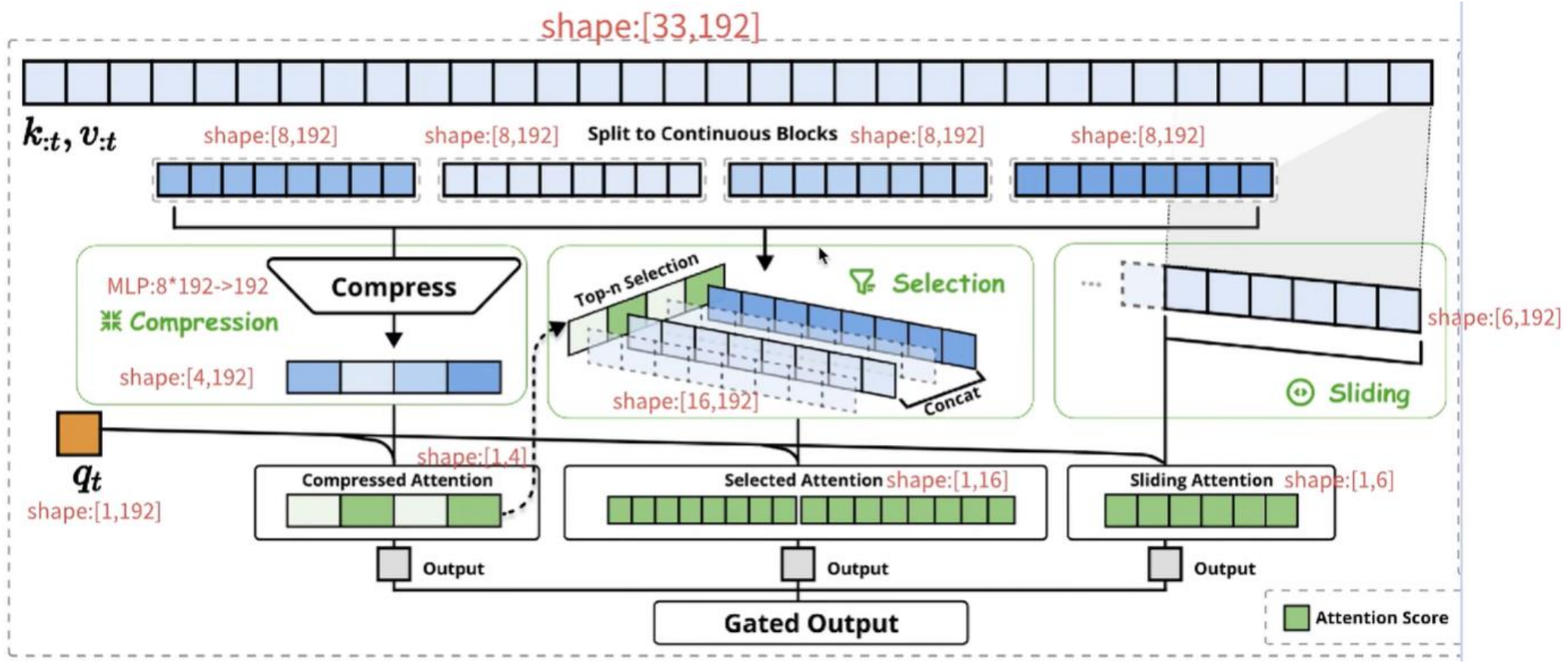
- **长上下文压缩：**长上下文带来平方级的运算或存储复杂度，给推理优化带来非常大影响，因此是当前大模型推理优化重要研究。

- **self-attention：**理论时间和空间占用为 $O(n^2)$ 其中 n 为序列长度，因为对于长度为 n 序列，任意向量间都需要计算相关度，得到 n^2 相关度矩阵，因此为 $O(n^2)$。

- **稀疏 Attention：**对于 self-attention 每个元素都跟序列内所有的元素都有关联，基本思路就是减少关联性计算，即每个元素只跟序列内部分元素相关。

Native Sparse Attention

- **核心思想：**设计一种原生可训练的稀疏注意力架构 (NSA)，通过层级化的 token 建模和硬件对齐的优化，实现高效的长文本建模。



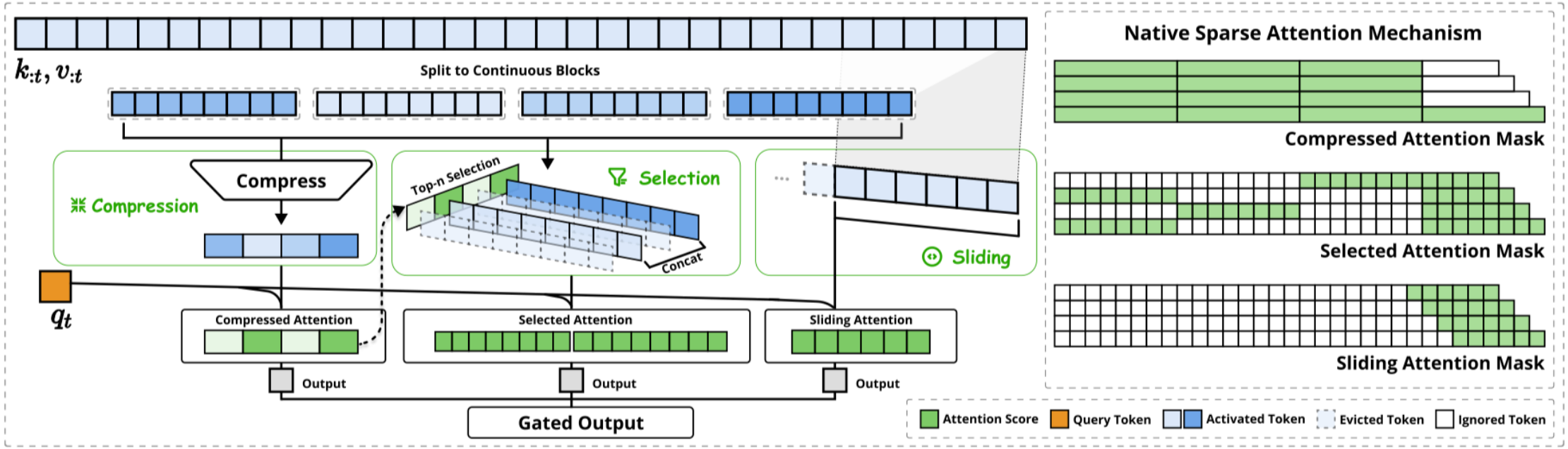
Native Sparse Attention

- **灵感来源：**
  - Attention 稀疏性：Attention score 存在内在的稀疏性，即只有少数 query-key 对是重要。
  - 硬件效率：充分利用现代 GPU 的特性和 FlashAttention 设计原则，实现高效的计算和内存访问。

- **主要创新：**
  - Hardware-aligned system：优化 block wise，利用 Tensor Core 和内存访问，确保算术强度平衡。
  - Training-aware design：通过高效的算法和反向传播算子，实现稳定的端到端训练。

三路注意力融合

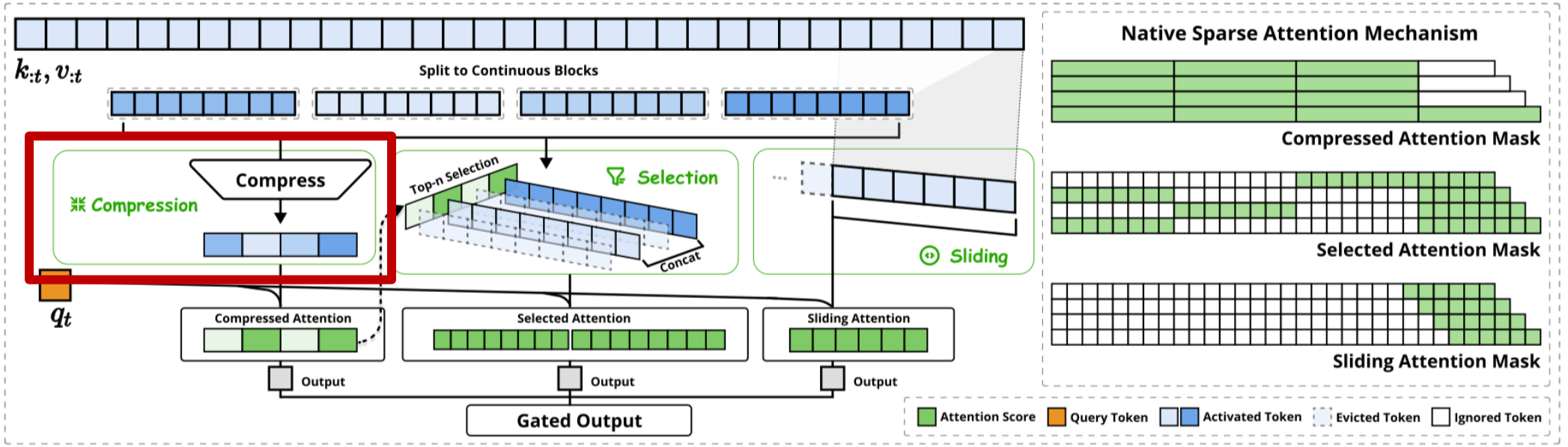
- 三条路径的结果通过门控机制动态加权融合，避免模型因局部信息优势而忽略全局：
  1. Compression：将长序列划为块（512 token/Block），通过MLP压缩为粗粒度捕获全局语义。
  2. Selection：基于压缩块的注意力分数筛选出Top-N关键块，还原细粒度信息。
  3. Sliding Window：固定窗口覆盖最近的局部token，确保上下文连贯性。



三路注意力融合 Compression

- 将全文划分为多个语义块（如每块512字），每个块通过可学习MLP压缩为一个"摘要向量"（如维度从512→64）。所有摘要向量组成精简版文档（长度缩减为原来的1/8）。

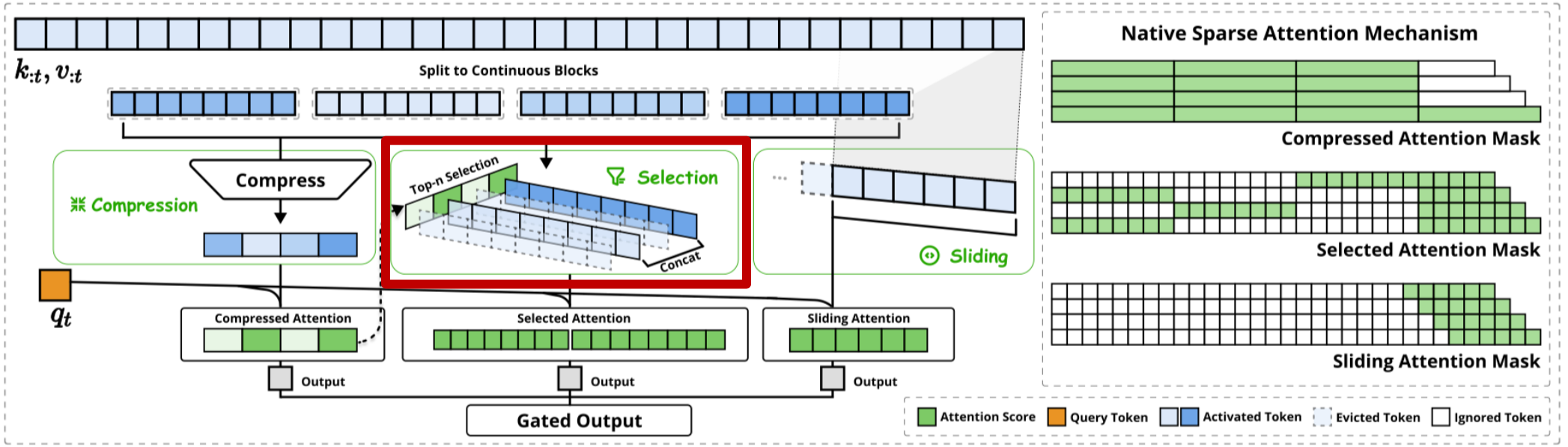
$$\tilde{K}_t^{\text{cmp}} = f_K^{\text{cmp}}(\mathbf{k}_{:t}) = \left\{ \varphi(\mathbf{k}_{id+1:id+l}) \middle| 0 \leqslant i \leqslant \left\lfloor \frac{t-l}{d} \right\rfloor \right\}$$



三路注意力融合 Selection

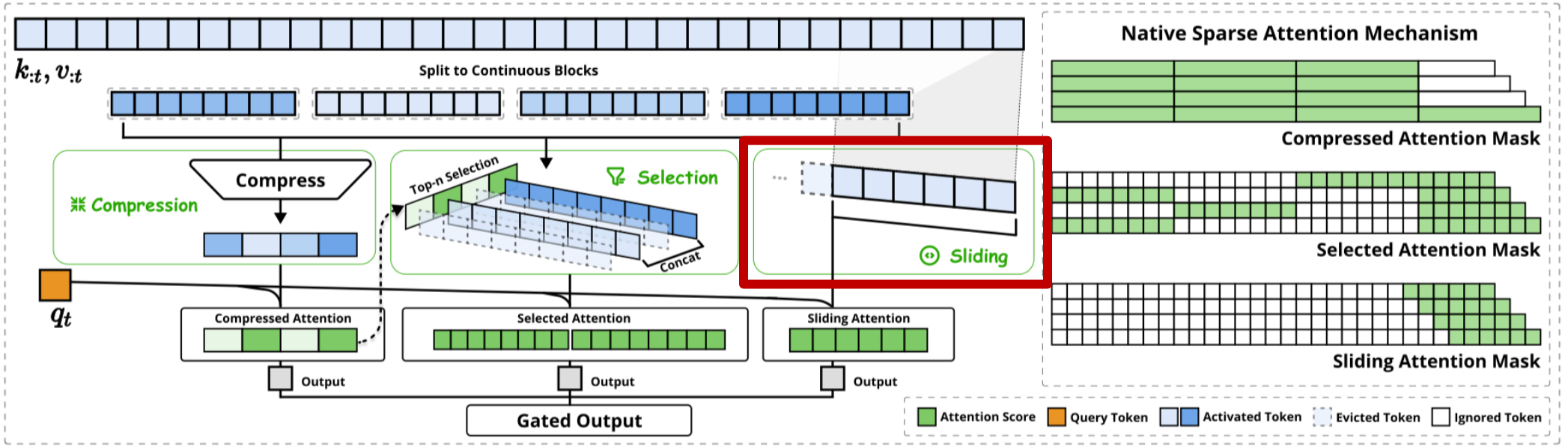
1. 关键块筛选：基于压缩路径注意力分数，选出Top-10%的关键块。
2. 细粒度还原：将选中块的摘要向量解码为原始token序列，进行细粒度注意力计算。

$$\mathbf{p}_t^{\text{cmp}} = \text{Softmax}\left(\mathbf{q}_t^T \tilde{K}_t^{\text{cmp}}\right), \qquad \mathbf{p}_t^{\text{slc}}[j] = \sum_{m=0}^{\frac{l'}{d}-1} \sum_{n=0}^{\frac{l}{d}-1} \mathbf{p}_t^{\text{cmp}} \left[\frac{l'}{d} j - m - n\right],$$



三路注意力融合 Sliding Window

- 对当前token的前后局部窗口（如1024 tokens）进行全注意力计算，确保上下文连贯性。处理到第N个token时，窗口自动滑动覆盖[N-512, N+512]的范围。



04

KIMI MoBA

https://www.bilibili.com/video/BV1J1PGevEPb

小总结

- 从思路上MoBA和NSA都有一个相通的点，对于 稀疏注意力实现，都是通过“筛选”操作；
- 但是注意力筛选势必要用到QKT计算，两者都会用一个小替换大（先分块再去压缩处理）思路。

总结与思考

总结

- **未来趋势：高效、可扩展、适合长上下文**
  - 减少复杂度：随着大模型发展，通过优化 Attention 计算复杂度提出 Linear Attention 等。
  - 长序列建模：结合稀疏注意力与动态路由，进一步压缩KV Cache。
  - 多模态扩展：探索跨模态注意力交互，如视觉-语言联合表征。

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

GitHub https://github.com/chenzomi12/AIInfra

引用与参考

- https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers
- https://shangzhih.github.io/jian-shu-attentionji-zhi.html
- https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html
- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- https://www.gnn.club/?p=2729
- http://www.myhz0606.com/article/kv-cache
- https://spaces.ac.cn/archives/10091

- PPT 开源在：https://github.com/chenzomi12/AIInfra