

# New Kernel with Multi-Scale KV Pooling

Xiaolong Li

July 2025

# 1 Forward Pass

---

**Algorithm 1** Forward Pass with Multi-Scale KV Pooling and Masking

---

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$  in HBM, block sizes  $B_r, B_c$ ,

- 1: sparse-block mask  $mask \in \{0, 1, 0.5, 0.25, 0.125\}^{T_r \times T_c}$
- 2: **Generate multi-scale KV blocks via pooling:**
- 3: pooling(x,zoom\_ratio) applies average pooling to merge each group of zoom\_ratio adjacent tokens into a single token.
- 4:  $\mathbf{K}_1, \mathbf{V}_1 \leftarrow \mathbf{K}, \mathbf{V}$
- 5: Blocks:  $T_c$  blocks of size  $B_c \times d$
- 6:  $\mathbf{K}_2, \mathbf{V}_2 \leftarrow \text{pooling}(\mathbf{K}_1, 2), \text{pooling}(\mathbf{V}_1, 2)$
- 7: Blocks:  $T_c$  blocks of size  $(B_c/2) \times d$
- 8:  $\mathbf{K}_4, \mathbf{V}_4 \leftarrow \text{pooling}(\mathbf{K}_2, 2), \text{pooling}(\mathbf{V}_2, 2)$
- 9: Blocks:  $T_c$  blocks of size  $(B_c/4) \times d$
- 10:  $\mathbf{K}_8, \mathbf{V}_8 \leftarrow \text{pooling}(\mathbf{K}_4, 2), \text{pooling}(\mathbf{V}_4, 2)$
- 11: Blocks:  $T_c$  blocks of size  $(B_c/8) \times d$
- 12:  $T_r \leftarrow \lceil N/B_r \rceil, T_c \leftarrow \lceil N/B_c \rceil$
- 13: Split  $\mathbf{Q}, \mathbf{O}$  into  $\{\mathbf{Q}_i\}, \{\mathbf{O}_i\}$ , each  $B_r \times d$
- 14: **for**  $i = 1$  to  $T_r$  **do**
- 15:   Load  $\mathbf{Q}_i$  to SRAM; initialize  $\mathbf{O}_i^{(0)} \leftarrow 0, \ell_i^{(0)} \leftarrow 0, m_i^{(0)} \leftarrow -\infty$
- 16:   **for**  $j = 1$  to  $T_c$  **do**
- 17:      $\alpha \leftarrow mask_{i,j}$
- 18:     **if**  $\alpha = 0$  **then**
- 19:       **continue**
- 20:     **else**
- 21:       **Select KV:** based on  $\alpha$
- 22:       **if**  $\alpha = 1$  **then**
- 23:          $\mathbf{K}_{\text{sel}}, \mathbf{V}_{\text{sel}} \leftarrow \mathbf{K}_1[j], \mathbf{V}_1[j]$
- 24:       **else if**  $\alpha = 0.5$  **then**
- 25:          $\mathbf{K}_{\text{sel}}, \mathbf{V}_{\text{sel}} \leftarrow \mathbf{K}_2[j], \mathbf{V}_2[j]$
- 26:       **else if**  $\alpha = 0.25$  **then**
- 27:          $\mathbf{K}_{\text{sel}}, \mathbf{V}_{\text{sel}} \leftarrow \mathbf{K}_4[j], \mathbf{V}_4[j]$
- 28:       **else if**  $\alpha = 0.125$  **then**
- 29:          $\mathbf{K}_{\text{sel}}, \mathbf{V}_{\text{sel}} \leftarrow \mathbf{K}_8[j], \mathbf{V}_8[j]$
- 30:       **end if**
- 31:       Compute scores:  $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_{\text{sel}}^\top$
- 32:       Apply compensation:  $\mathbf{S}_i^{(j)} += \log(1/\alpha)$
- 33:       Update max:  $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(\mathbf{S}_i^{(j)}))$
- 34:       Compute exp:  $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)})$
- 35:       Accumulate sum:  $\ell_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} \ell_i^{(j-1)} + \text{rowsum}(\tilde{\mathbf{P}}_i^{(j)})$
- 36:       Update output:  $\mathbf{O}_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_{\text{sel}}$
- 37:     **end if**
- 38:   **end for**
- 39:   Finalize:  $\mathbf{O}_i = \mathbf{O}_i^{(T_c)} / \ell_i^{(T_c)}, L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$
- 40:   Write  $\mathbf{O}_i, L_i$  to HBM
- 41: **end for**
- 42: **return**  $\mathbf{O}, L$

---

## 2 New Backward Pass

---

**Algorithm 2** FlashAttention-MSNew Backward Pass with Multi-Scale KV Pooling and Masking

---

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$  in HBM, vector  $\mathbf{L} \in \mathbb{R}^N$  in HBM, block sizes  $B_r, B_c$ , sparse-block mask  $mask \in \{0, 1, 0.5, 0.25, 0.125\}^{T_r \times T_c}$ .

- 1: INITIALIZATION FOR BACKWARD PASS
- 2:  $T_r \leftarrow \lceil N/B_r \rceil, T_c \leftarrow \lceil N/B_c \rceil$
- 3: Divide  $\mathbf{Q}, \mathbf{O}, \mathbf{dO}$  into  $\{\mathbf{Q}_i\}, \{\mathbf{O}_i\}, \{\mathbf{dO}_i\}$ , each  $B_r \times d$ ; divide  $\mathbf{L}$  into  $\{\mathbf{L}_i\}$ , each  $B_r$
- 4: Initialize  $\mathbf{dQ} = (0)_{N \times d}$  in HBM, divided into  $\{\mathbf{dQ}_i\}$ ; initialize  $\mathbf{dK}_1 = (0)_{N \times d}, \mathbf{dV}_1 = (0)_{N \times d}$  in HBM
- 5: Compute  $D = \text{rowsum}(\mathbf{dO} \circ \mathbf{O}) \in \mathbb{R}^N$ , write to HBM, divide into  $\{D_i\}$ , each  $B_r$
- 6: **for**  $\alpha \in \{1, 0.5, 0.25, 0.125\}$  **do**
- 7:     PROCESSMASK( $\alpha$ )
- 8: **end for**
- 9: BACKWARD FOR POOLING( $\mathbf{dK}_1, \mathbf{dV}_1, \mathbf{dK}_2, \mathbf{dV}_2, \mathbf{dK}_4, \mathbf{dV}_4, \mathbf{dK}_8, \mathbf{dV}_8$ )
- 10: **return**  $\mathbf{dQ}, \mathbf{dK}_1, \mathbf{dV}_1$

---



---

**Algorithm 3** INITIALIZATION FOR NEW BACKWARD PASS WITH MULTI-SCALE KV POOLING

---

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$  in HBM, vector  $\mathbf{L} \in \mathbb{R}^N$  in HBM, block sizes  $B_r, B_c$

- 1: **procedure** INITIALIZE( $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO}, \mathbf{L}, B_r, B_c$ )
- 2:     **Generate multi-scale KV blocks via pooling:**
- 3:          $pooling(x, zoom\_ratio)$  applies average pooling to merge each group of  $zoom\_ratio$  adjacent tokens into a single token.
- 4:          $\mathbf{K}_1, \mathbf{V}_1 \leftarrow \mathbf{K}, \mathbf{V}$  ▷ zoom\_ratio=1
- 5:         Divide  $\mathbf{K}_1, \mathbf{V}_1$  into  $T_c$  blocks  $\mathbf{K}_1[1], \dots, \mathbf{K}_1[T_c], \mathbf{V}_1[1], \dots, \mathbf{V}_1[T_c]$  of size  $B_c \times d$  each in HBM.
- 6:          $\mathbf{K}_2, \mathbf{V}_2 \leftarrow pooling(\mathbf{K}_1, 2), pooling(\mathbf{V}_1, 2)$  ▷ zoom\_ratio=2
- 7:         Divide  $\mathbf{K}_2, \mathbf{V}_2$  into  $T_c$  blocks  $\mathbf{K}_2[1], \dots, \mathbf{K}_2[T_c], \mathbf{V}_2[1], \dots, \mathbf{V}_2[T_c]$  of size  $\lceil B_c/2 \rceil \times d$  each in HBM.
- 8:          $\mathbf{K}_4, \mathbf{V}_4 \leftarrow pooling(\mathbf{K}_2, 2), pooling(\mathbf{V}_2, 2)$  ▷ zoom\_ratio=4
- 9:         Divide  $\mathbf{K}_4, \mathbf{V}_4$  into  $T_c$  blocks  $\mathbf{K}_4[1], \dots, \mathbf{K}_4[T_c], \mathbf{V}_4[1], \dots, \mathbf{V}_4[T_c]$  of size  $\lceil B_c/4 \rceil \times d$  each in HBM.
- 10:          $\mathbf{K}_8, \mathbf{V}_8 \leftarrow pooling(\mathbf{K}_4, 2), pooling(\mathbf{V}_4, 2)$  ▷ zoom\_ratio=8
- 11:         Divide  $\mathbf{K}_8, \mathbf{V}_8$  into  $T_c$  blocks  $\mathbf{K}_8[1], \dots, \mathbf{K}_8[T_c], \mathbf{V}_8[1], \dots, \mathbf{V}_8[T_c]$  of size  $\lceil B_c/8 \rceil \times d$  each in HBM.
- 12:          $T_r \leftarrow \lceil N/B_r \rceil, T_c \leftarrow \lceil N/B_c \rceil$
- 13:         Divide  $\mathbf{Q}$  into  $\{\mathbf{Q}_i\}_{i=1}^{T_r}, \mathbf{O}$  into  $\{\mathbf{O}_i\}_{i=1}^{T_r}, \mathbf{dO}$  into  $\{\mathbf{dO}_i\}_{i=1}^{T_r}$ , each  $B_r \times d$ , and  $\mathbf{L}$  into  $\{\mathbf{L}_i\}_{i=1}^{T_r}$ , each  $B_r$ , all in HBM.
- 14:         Initialize  $\mathbf{dQ} = (0)_{N \times d}$  in HBM, divide into  $T_r$  blocks  $\mathbf{dQ}_1, \dots, \mathbf{dQ}_{T_r}$ , each  $B_r \times d$ .
- 15:         Initialize and Divide  $\mathbf{dK}_1, \mathbf{dV}_1 \in (0)_{N \times d}$  into  $T_c$  blocks  $\mathbf{dK}_1[j], \mathbf{dV}_1[j]$ , each  $B_c \times d$  in HBM.
- 16:         Initialize and Divide  $\mathbf{dK}_2, \mathbf{dV}_2 \in (0)_{\lceil N/2 \rceil \times d}$  into  $T_c$  blocks  $\mathbf{dK}_2[j], \mathbf{dV}_2[j]$ , each  $\lceil B_c/2 \rceil \times d$  in HBM.
- 17:         Initialize and Divide  $\mathbf{dK}_4, \mathbf{dV}_4 \in (0)_{\lceil N/4 \rceil \times d}$  into  $T_c$  blocks  $\mathbf{dK}_4[j], \mathbf{dV}_4[j]$ , each  $\lceil B_c/4 \rceil \times d$  in HBM.
- 18:         Initialize and Divide  $\mathbf{dK}_8, \mathbf{dV}_8 \in (0)_{\lceil N/8 \rceil \times d}$  into  $T_c$  blocks  $\mathbf{dK}_8[j], \mathbf{dV}_8[j]$ , each  $\lceil B_c/8 \rceil \times d$  in HBM.
- 19:         Compute  $D = \text{rowsum}(\mathbf{dO} \circ \mathbf{O}) \in \mathbb{R}^N$ , write to HBM, divide into  $\{D_i\}_{i=1}^{T_r}$ , each  $B_r$ .
- 20: **end procedure**

---

---

```

procedure PROCESSMASK( $\alpha$ )
     $zoom\_ratio \leftarrow 1/\alpha$ 
    for  $j = 1$  to  $T_c$  do
        Select KV and gradient blocks based on  $\alpha$ :
        if  $\alpha = 1$  then
             $K_{sel} \leftarrow K_1[j]$ ,  $V_{sel} \leftarrow V_1[j]$ ,  $dK_{sel} \leftarrow dK_1[j]$ ,  $dV_{sel} \leftarrow dV_1[j]$ 
        else if  $\alpha = 0.5$  then
             $K_{sel} \leftarrow K_2[j]$ ,  $V_{sel} \leftarrow V_2[j]$ ,  $dK_{sel} \leftarrow dK_2[j]$ ,  $dV_{sel} \leftarrow dV_2[j]$ 
        else if  $\alpha = 0.25$  then
             $K_{sel} \leftarrow K_4[j]$ ,  $V_{sel} \leftarrow V_4[j]$ ,  $dK_{sel} \leftarrow dK_4[j]$ ,  $dV_{sel} \leftarrow dV_4[j]$ 
        else if  $\alpha = 0.125$  then
             $K_{sel} \leftarrow K_8[j]$ ,  $V_{sel} \leftarrow V_8[j]$ ,  $dK_{sel} \leftarrow dK_8[j]$ ,  $dV_{sel} \leftarrow dV_8[j]$ 
        end if
        Load  $K_{sel}$ ,  $V_{sel}$ ,  $dK_{sel}$ ,  $dV_{sel}$  from HBM to on-chip SRAM
        for  $i = 1$  to  $T_r$  do
            if  $mask[i, j] \neq \alpha$  then
                continue
            end if
            Load  $Q_i$ ,  $dO_i$ ,  $L_i$ ,  $D_i$  from HBM to on-chip SRAM
            On chip, compute  $S_i^{(j)} = Q_i K_{sel}^\top$ 
            On chip, apply compensation:  $S_i^{(j)} += \ln(zoom\_ratio)$ 
            On chip, compute  $P_i^{(j)} = \exp(S_i^{(j)} - L_i)$ 
            On chip, compute  $dP_i^{(j)} = dO_i V_{sel}^\top$ 
            On chip, compute  $dS_i^{(j)} = P_i^{(j)} \circ (dP_i^{(j)} - D_i)$ 
            Load  $dQ_i$  from HBM to on-chip SRAM
            On chip, update  $dQ_i \leftarrow dQ_i + dS_i^{(j)} K_{sel}$ 
            Write  $dQ_i$  back to HBM
            On chip, accumulate  $dK_{sel} \leftarrow dK_{sel} + (dS_i^{(j)})^\top Q_i$ 
            On chip, accumulate  $dV_{sel} \leftarrow dV_{sel} + (P_i^{(j)})^\top dO_i$ 
        end for
        Write  $dK_{sel}$ ,  $dV_{sel}$  back to HBM
    end for
end procedure

```

---