

# Flash-HOG

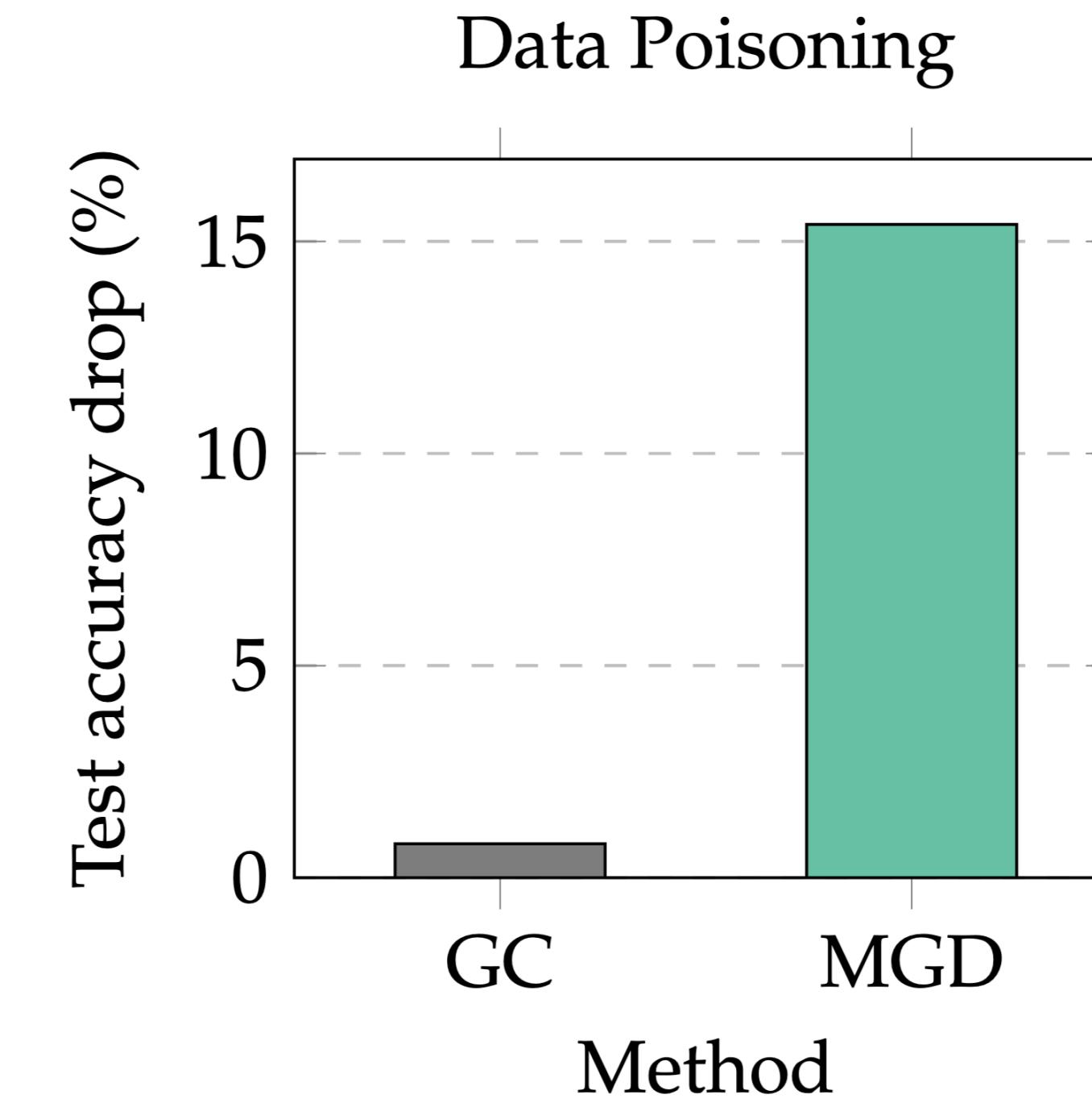
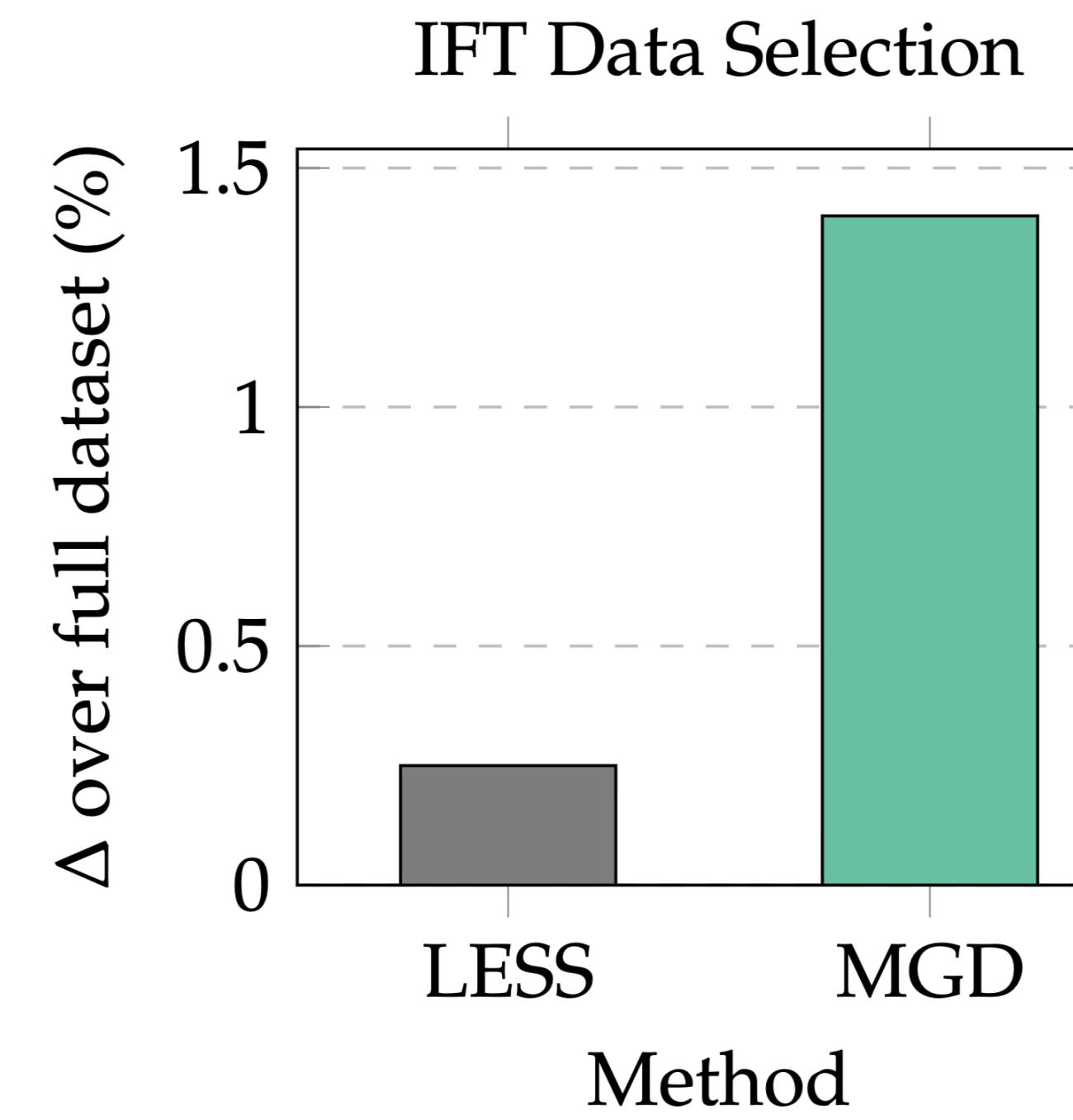
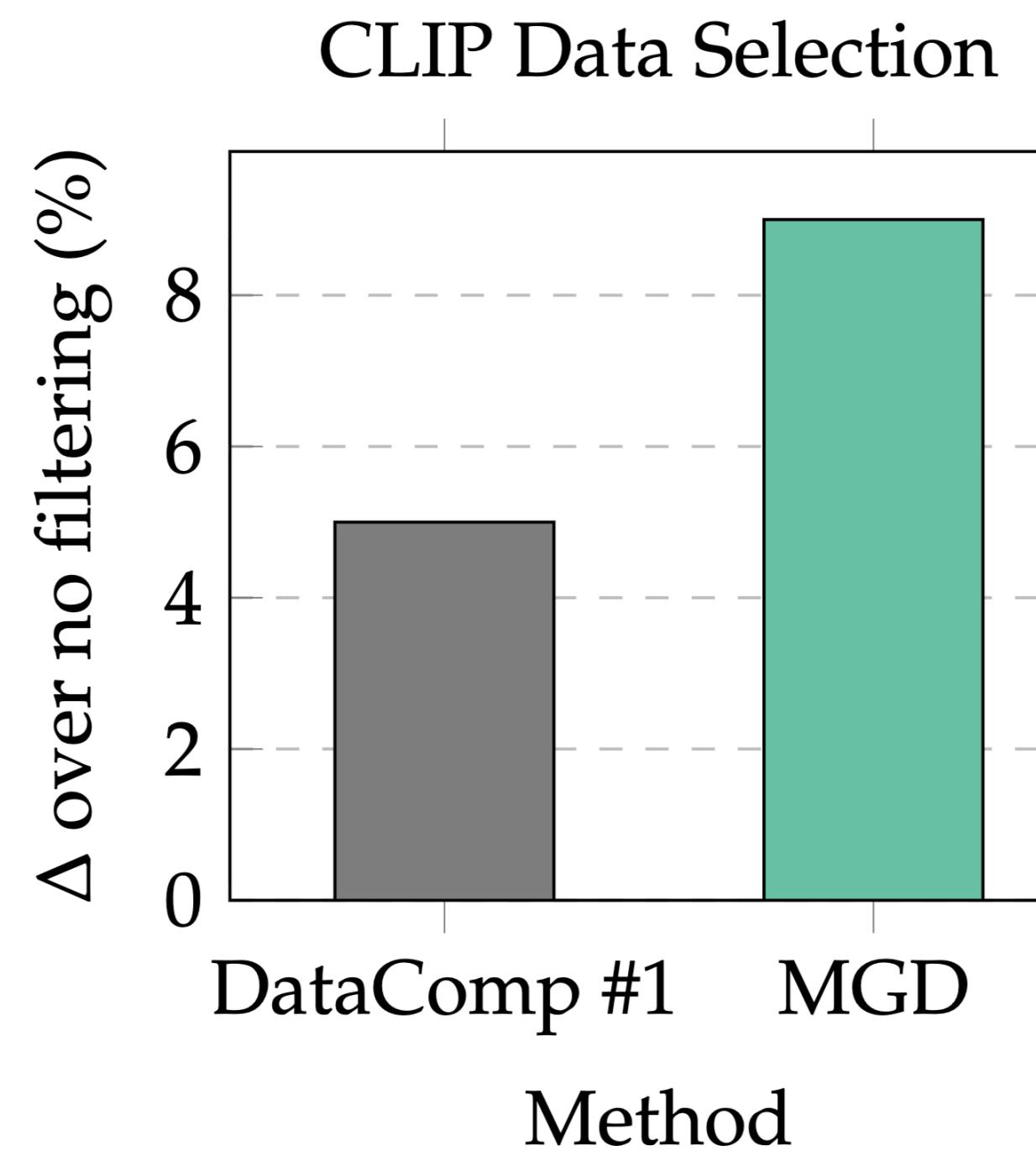
Higher-order gradients at lightspeed!



Marcel Roed, Neil Band, Herman Brunborg, Stephen Ge, Shiv Sundram

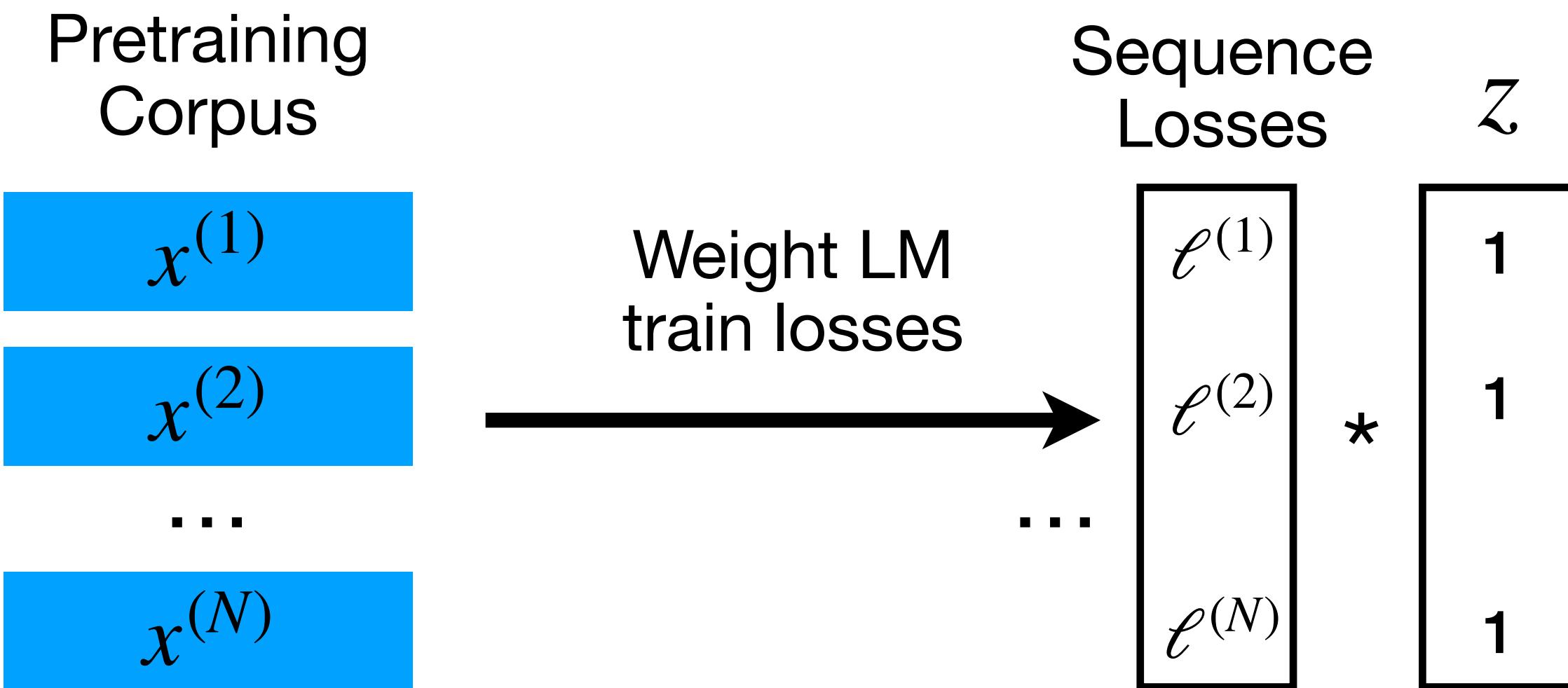
# Why higher-order gradients?

- State-of-the-art approach for pretraining/SFT data selection



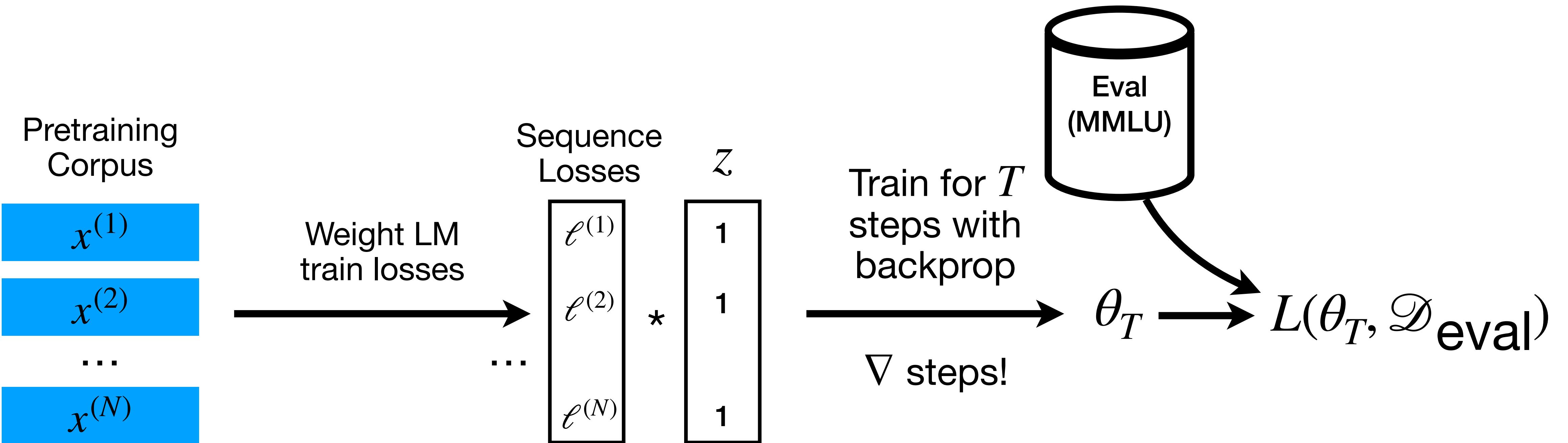
# Why higher-order gradients: data selection

1. Weight per-sequence train losses with scalar *data weights*  $z$



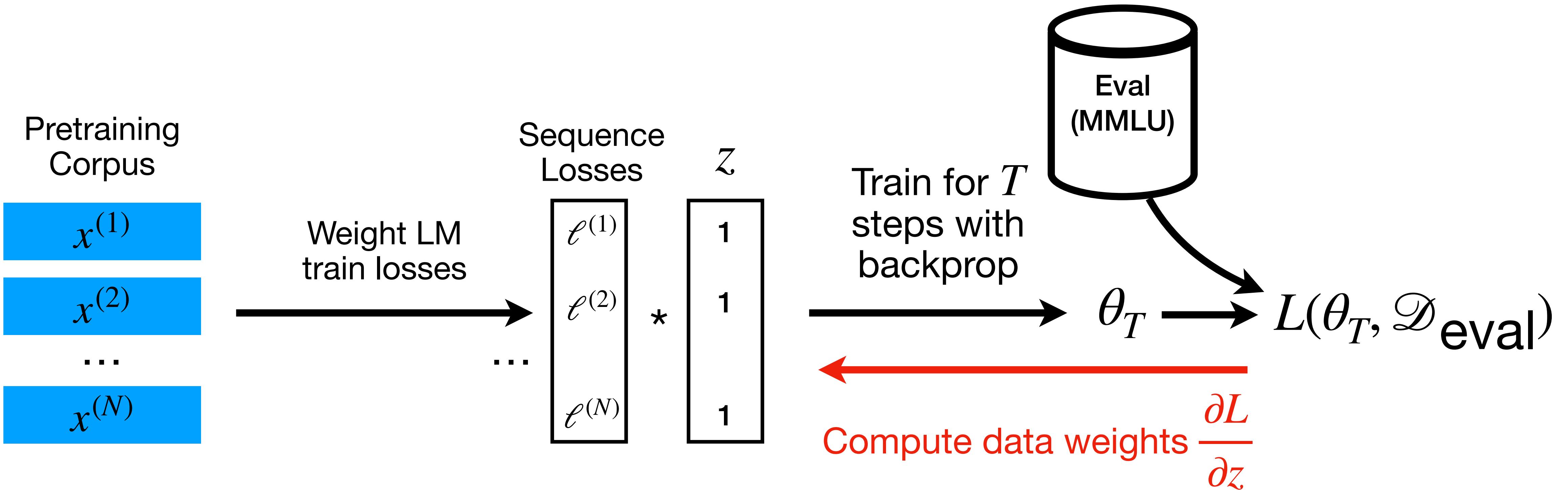
# Why higher-order gradients: data selection

1. Weight per-sequence train losses with scalar *data weights*  $z$



# Why higher-order gradients: data selection

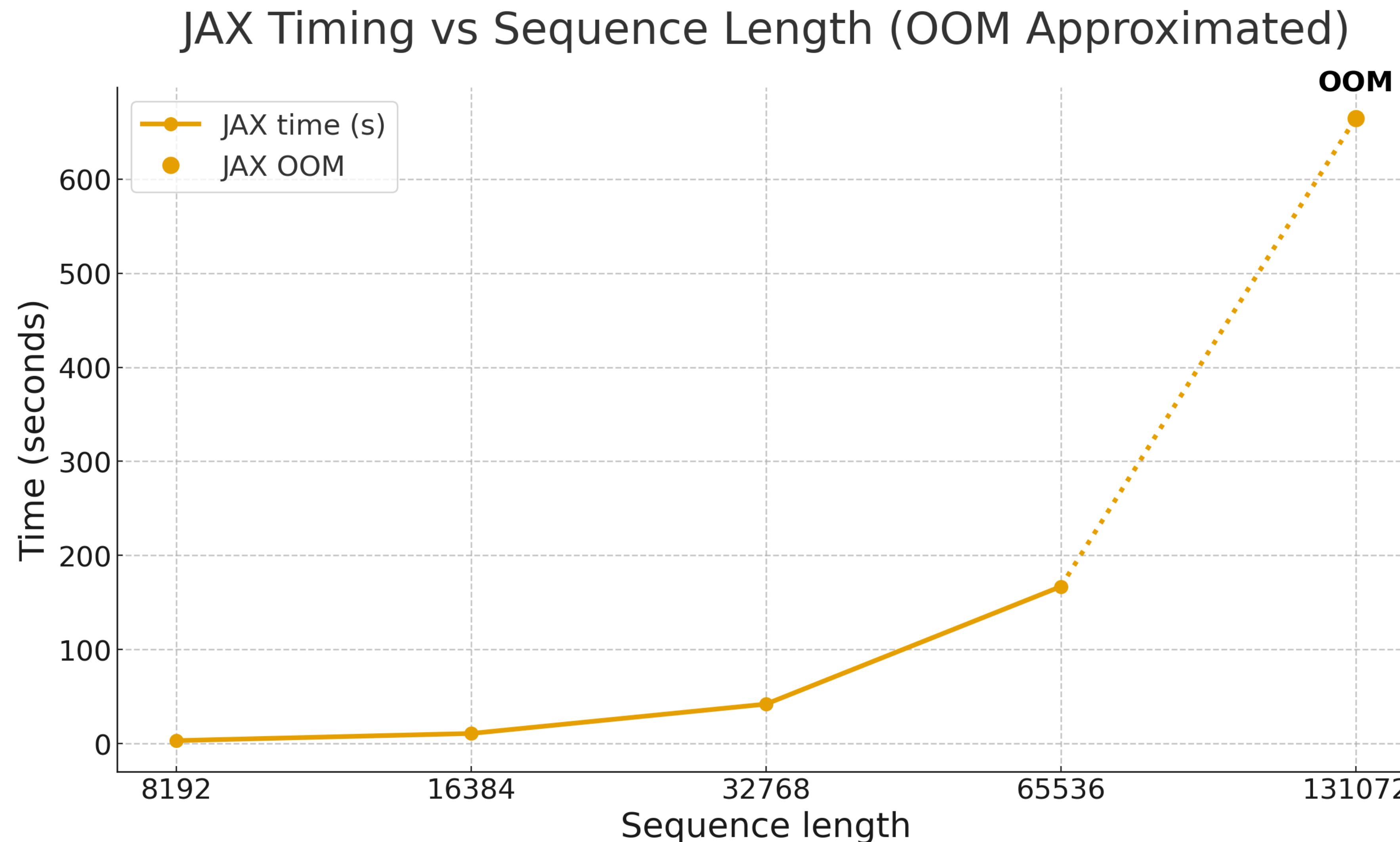
1. Weight per-sequence train losses with scalar *data weights*  $z$
2. Differentiate a validation loss w.r.t.  $z$  to get “**“data importance weights”**”



Challenge:  
Requires grad-on-grad,  
which is naively super slow and memory-intensive!



# Naive impl: OOM@131K seqlen (on B200)

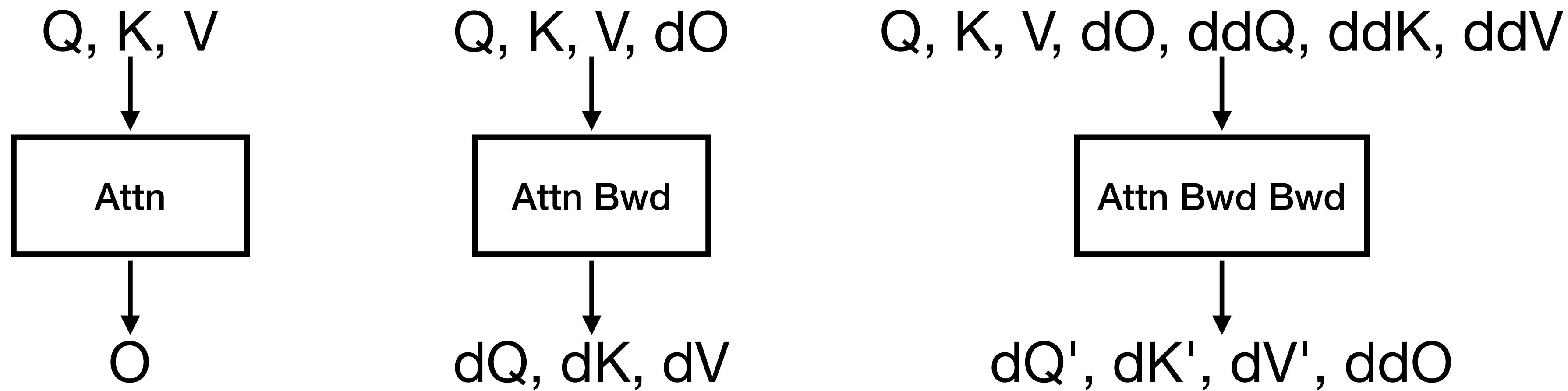


# Naive impl: @1M seqlen

```
Allocator (GPU_0_bfc) ran out of memory trying to allocate 4.00TiB (rounded to 4398046511104)requested by op
```

4.00TiB !!!

# Higher Order Backward



## Attn Bwd Bwd

### Attn

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d}$$

$$\mathbf{P}_{ij} = \text{softmax}_j(\mathbf{S})_{ij}$$

$$\mathbf{O} = \mathbf{PV}$$

### Attn Bwd

$$d\mathbf{V} = \mathbf{P}^\top d\mathbf{O}$$

$$d\mathbf{P} = d\mathbf{O}\mathbf{V}^\top$$

$$d\mathbf{S}_i = d\text{softmax}(\mathbf{dP}_i) = (\text{diag}(\mathbf{P}_i) - \mathbf{P}_i\mathbf{P}_i^\top) d\mathbf{P}_i$$

$$d\mathbf{Q} = d\mathbf{SK} / \sqrt{d}$$

$$d\mathbf{K} = d\mathbf{S}^\top \mathbf{Q} / \sqrt{d},$$

$$S = \mathbf{Q} \mathbf{K}^\top \cdot \text{scale}$$

$(N_a, N_k)$

$$P_{ij} = \text{softmax}_j(S_{ij}) = \exp(S_{ij} - L_i)$$

$(N_Q, N_k)$

$$dP_{ij} = dO_{id} V_{jd}$$

$(N_Q, N_k)$

$$ddS = \text{scale} \cdot (ddQ K^\top + Q ddK^\top)$$

$(N_a, N_k)$

$$d_{ii} = \underline{O_{id} dO_{id}}$$

$(N_Q, 1)$

$$dd_{ii} = \underline{ddS_{ik} P_{ik}}$$

$(N_Q, 1)$

$$dP' = dO ddV^\top - dP odd - ddS \cdot d + dP odd ds$$

$(N_Q, N_k)$

$$ddP = P \odot (ddS - dd)$$

$(N_Q, N_k)$

$$dS'_{ij} = \text{scale} \cdot P_i \left( \underline{dP'_{ij}} - \sum_k \underline{dP'_{ik} P_{ik}} \right) \xrightarrow{b_i}$$

$(N_Q, N_k)$

$$dS = \text{scale} \cdot P \odot (dP - d)$$

$(N_a, N_k)$

$$\underline{dQ'} = dS ddK + dS' K$$

$(N_Q, N_D)$

$$\underline{dK'} = dS^\top ddQ + dS'^\top Q$$

$(N_K, N_D)$

$$\underline{dV'} = ddP^\top dO$$

$(N_K, N_D)$

$$\underline{ddO} = P ddV + ddPV$$

$(N_Q, N_D)$

returns  $dQ', dK', dV', ddO$

inputs  $Q, K, V, O, dO, ddQ, ddK, ddV, L$

# Multi-Pass Reductions

- 4 total passes in one kernel
- Reduction over Q vs K
- Must avoid saving  $N \times N!$
- Linear memory costs

$$S = QK^T \cdot \text{scale}$$

$(N_Q, N_K)$

$$P_{ij} = \text{softmax}_j(S_{ij}) = \exp(S_{ij} - L_i)$$

$(N_Q, N_K)$

$$dP_{ij} = dO_{id} V_{jd}$$

$(N_Q, N_K)$

$$ddS = \text{scale} \cdot (ddQK^T + QddK^T)$$

$(N_Q, N_K)$

$$d_{ii} = \underline{O_{id} dO_{id}}$$

$(N_Q, 1)$

$$dd_{ii} = \underline{ddS_{ik} P_{ik}}$$

$(N_Q, 1)$

$$dP' = dO ddV^T - dP odd - ddS \cdot d + dP oddS$$

$(N_Q, N_K)$

$$ddP = P \circ (ddS - dd)$$

$(N_Q, N_K)$

$$dS'_{ij} = \text{scale} \cdot P_i \left( dP'_{ij} - \underbrace{\sum_k dP'_{ik} P_{ik}}_b \right) \xrightarrow{b_i}$$

$(N_Q, N_K)$

$$dS = \text{scale} \cdot P \circ (dP - d)$$

$(N_Q, N_K)$

$$dQ' = dS ddK + dS' K$$

$(N_Q, N_D)$

$$dK' = dS^T ddQ + dS'^T Q$$

$(N_K, N_D)$

$$dV' = ddP^T dO$$

$(N_K, N_D)$

$$ddO = P ddV + ddPV$$

$(N_Q, N_D)$

returns  $dQ'$ ,  $dK'$ ,  $dV'$ ,  $ddO$

inputs  $Q$ ,  $K$ ,  $V$ ,  $O$ ,  $dO$ ,  $ddQ$ ,  $ddK$ ,  $ddV$ ,  $L$

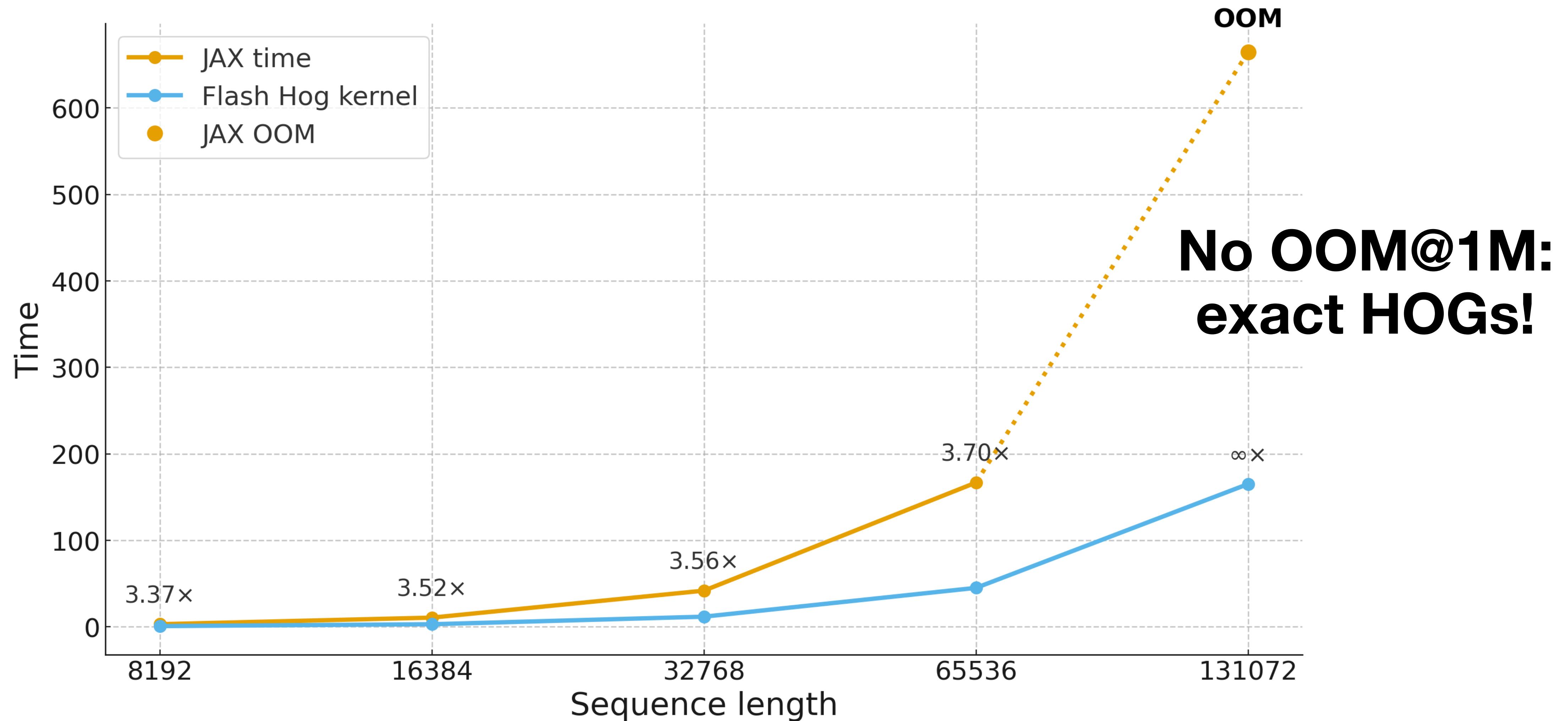
# Implementation: numerics look good

$\text{diff}(\text{Jax ref}, \text{Flash-HOG}) \approx \text{diff}(\text{Jax ref}, \text{Torch ref})$

Greatest absolute difference: 0.0126953125 at index (0, 49, 55)

# Results: >3x speedup, no OOMs!

JAX vs Flash Hog kernel: Timing vs Sequence Length with Speedups





# Flash-HOG

Higher-order gradients at lightspeed!