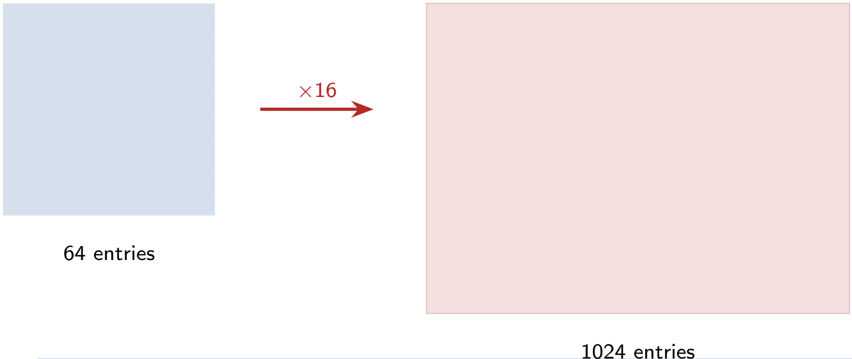
Long Context & Efficient Attention

Flash Attention · Sparse Patterns · KV Cache · State Space Models

The quadratic wall



Standard self-attention: $O(n^2)$ time and memory.

512 tokens $\rightarrow$ 262K ops · 128K tokens $\rightarrow$ 16 **billion** entries per head per layer

Doubling sequence length $\rightarrow$ 4$\times$ the cost. This is THE bottleneck.

Why long context matters

Documents

Legal contracts, research papers, novels, financial reports

Often 50–200 pages

Codebases

Entire repositories in context
Multi-file reasoning

30,000+ lines of code

Conversations

Multi-turn dialogue history
Agent memory and planning

Extended interactions

Multimodal

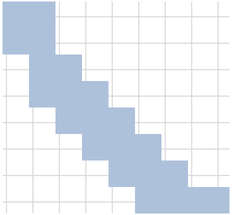
1 hour video $\approx$ 1M tokens
11 hours audio $\approx$ 1M tokens

Images, charts, diagrams

GPT-4 Turbo: **128K** Claude 3: **200K**
Gemini 1.5 Pro: **1M+** Gemini 2: **2M**

Sparse attention patterns

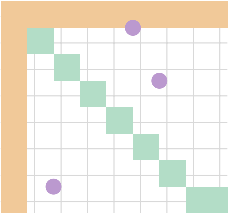
Sliding window



$O(n \cdot w)$

Longformer (2020)

Global + local + random



$O(n)$

BigBird (2020)

Full attention

$O(n^2)$

Standard

Key idea: most tokens don't need to attend to *all* other tokens.

Local context + a few global connections captures most of the useful information.

Linear attention approximations

Standard attention

$$Q \times K^T = n \times n \times V$$

$O(n^2 d)$ — bottleneck!

Linear attention

$$K^T \times V = d \times d \times Q$$

$O(nd^2)$ — linear in n!

Linformer (Wang et al., 2020)

Project K, V from $n \times d$ to $k \times d$
($k \ll n$, low-rank assumption)

Complexity: $O(nk)$

Performer / FAVOR+
(Choromanski et al., ICLR 2021)

Random feature maps $\varphi(Q), \varphi(K)$
Compute $\varphi(K)^T V$ first ($d \times d$)

No sparsity assumptions needed

Historically important but largely **superseded by FlashAttention** in practice — approximate methods degrade quality; exact attention can be made fast enough.

FlashAttention: the IO-aware revolution

HBM

Large (80 GB)
Slow (2 TB/s)

↕ IO bottleneck

SRAM

Small (20 MB)
Fast (19 TB/s)

Key insight

The bottleneck on modern GPUs is **data movement** (IO), not FLOPs.

Tile Q, K, V into blocks that fit in SRAM, compute attention **blockwise** with online softmax. Never materialize $n \times n$ matrix.

Dao et al., NeurIPS 2022

7.6$\times$ attention speedup
2–4$\times$ end-to-end training speedup
Memory: $O(n)$ instead of $O(n^2)$

Exact attention — no approximation!

FlashAttention changed the game: **don't approximate attention, just compute it smarter.**

Now the default in virtually every modern LLM training pipeline.

FlashAttention evolution

FlashAttention

Dao et al., 2022

IO-aware tiling
Online softmax
No $n \times n$ matrix

7.6× speedup

A100: 124 TFLOPs/s

FlashAttention-2

Dao, 2023

Better parallelism
Reduced non-matmul FLOPs
Warp-level optimization

2× over FA1
A100: 225 TFLOPs/s

(50–73% peak)

FlashAttention-3

Shah & Dao, NeurIPS 2024

Hopper GPU features
Async GEMM + softmax
FP8 quantization

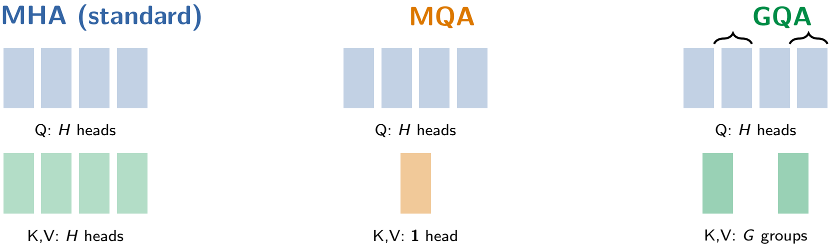
FP16: **740** TFLOPs/s

FP8: **1.2 PFLOPs/s**

The practical winner: **don't approximate attention, just optimize IO.**

FlashAttention made approximate methods (Linformer, Performer) largely obsolete.

Multi-Query & Grouped-Query Attention



MQA (Shazeer, 2019): **12**× faster decoding, slight quality drop

GQA (Ainslie et al., EMNLP 2023): near-MHA quality, near-MQA speed

GQA is the **de facto standard**: LLaMA 2/3, Mistral, Gemma, PaLM

DeepSeek MLA (2024): compress K,V into low-rank latent before caching. Each head gets unique K,V (unlike GQA), but cache stores only compressed latent. **93% cache reduction.**

KV cache: the inference bottleneck

Autoregressive generation

t_1 t_2 t_3 $\cdots$ t_n

new token

K cache V cache → grows with n

Memory per token:

$2 \times n_{\text{layers}} \times n_{\text{heads}} \times d_{\text{head}} \times$ precision

For 70B model at 128K:

KV cache alone $>$ **40 GB**!

At long context, the **KV cache** — not the model weights — becomes the memory bottleneck.

Every new token requires reading the entire cache. This dominates inference latency.

This is why efficient KV management is critical for production LLM serving.

KV cache optimization

PagedAttention

vLLM, SOSP 2023

OS-style paging: non-contiguous blocks

Fragmentation: **70%** → <**4%**

2–4× throughput

Prefix sharing

Quantized KV

Store cache in lower precision:

FP16 → FP8: **2**× smaller
FP16 → INT4: **4**× smaller

Minimal quality loss

Native on Hopper GPUs

Eviction

H2O: keep "heavy hitter" tokens (high cumulative attention), drop the rest

KV-Compress: up to **8**× compression with

negligible accuracy loss

DeepSeek MLA: compress K, V into a **low-rank latent** representation before caching.

93.3% KV cache reduction · 5.76× throughput improvement · outperforms GQA in quality

Positional encoding for length

Sinusoidal

Vaswani et al., 2017

Absolute, fixed
No extrapolation

Historical only

RoPE

Su et al., 2021

Rotation encodes relative position

Dominant in

modern LLMs

ALiBi

Press et al., ICLR 2022

No embeddings at all
Linear bias: $-m \cdot |i - j|$

Used in MPT, BLOOM

RoPE: rotate Q and K vectors by position-dependent angles.

$$\text{Attention}(q_m, k_n) = f(q, m)^T f(k, n) = g(q, k, m - n)$$

Dot product naturally captures **relative** position. Each dimension pair is rotated by a frequency that depends on position.

The extrapolation problem: all methods degrade when input length exceeds training length. Extending context requires additional techniques.

Context window extension

2K–4K $\xrightarrow{\text{extend}}$ **128K–2M**

training window → effective window

Position Interpolation (Meta, 2023)

Scale positions down to fit within original window. 2K $\rightarrow$ 32K in $\sim$1,000 fine-tuning steps.

NTK-aware scaling (2023)

Differential scaling: high-freq (local) scaled less, low-freq (global) scaled more.

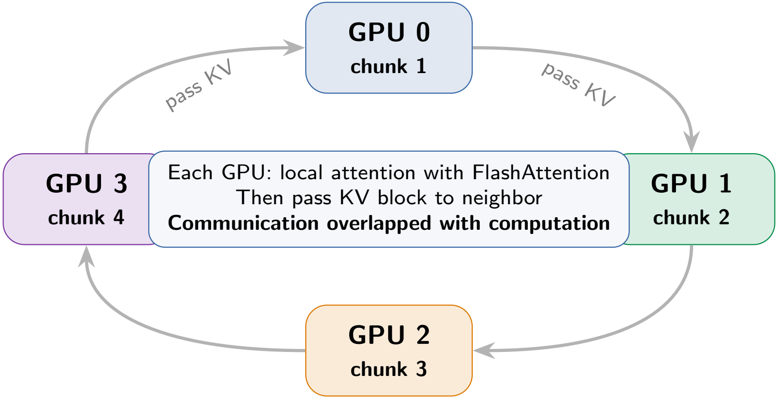
YaRN (Peng et al., ICLR 2024)

NTK-by-parts + temperature. 128K+ with only $\sim$400 fine-tuning steps.

LongRoPE (Microsoft, ICML 2024)

Non-uniform interpolation via efficient search. Up to **2M tokens** context.

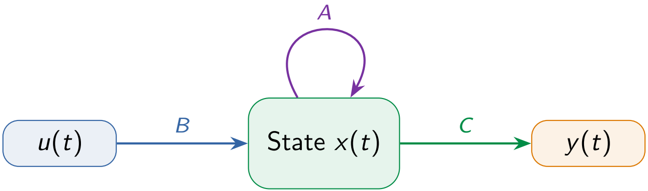
Ring Attention



Liu et al., ICLR 2024: max context scales **linearly** with number of GPUs.

Mathematically exact (no approximation). Builds on top of FlashAttention.

State Space Models: S4



$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

HiPPO initialization for A: mathematically optimal compression of continuous signals into finite state

Gu et al., ICLR 2022

First to solve Path-X (length 16,384)

Recurrence mode:
$O(n)$ for inference
(process one token at a time)

Convolution mode:
$O(n \log n)$ for training
(parallelizable via FFT)

Mamba: selective state spaces

S4 (fixed dynamics)

Parameters A, B, C are **fixed** (same transformation for all inputs)

Like a fixed filter:

same processing regardless of content

vs.

Mamba (selective)

Parameters Δ, B, C are **input-dependent** (functions of x_t)

Selection mechanism:

decide what to remember/forget

Gu & Dao, 2023: hardware-aware parallel scan on GPU

5$\times$ **throughput** over Transformers at inference · **Linear** scaling with n

Mamba-3B matches Transformer quality at **twice the size** (6B)

The selection mechanism is what makes Mamba work for language:

Content-dependent gating (like "what to remember") is essential for in-context learning and selective information propagation.

Mamba-2 & hybrid architectures

SSM $\xleftrightarrow{\text{mathematically dual}}$ **Linear Attention**

Dao & Gu, ICML 2024: same structured matrix

Mamba-2

Uses matmul instead of scan
2–8× faster than Mamba-1

Bridges SSM and
Transformer worlds

Hybrid architectures

Jamba (AI21, 2024):
Transformer + Mamba + MoE

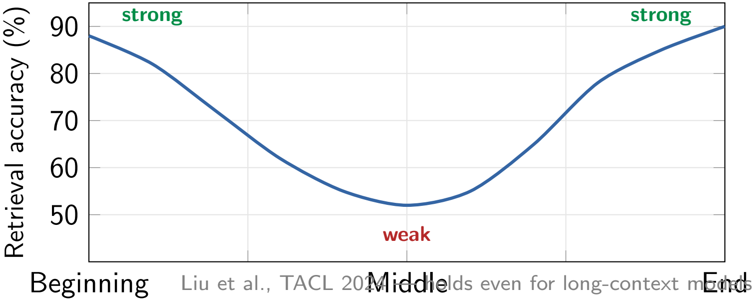
Gemma 3 (Google, 2025):
Interleaved local + global attention

The trend: mix attention types for best of both worlds.
Local sliding window (cheap) for most layers + full attention (expensive) every few layers

+ SSM layers for ultra-fast sequential processing.

Open question: will pure SSMs eventually
match Transformers, or will hybrids dominate?
Current evidence favors hybrids for language, but the field is evolving rapidly.

The “Lost in the Middle” problem

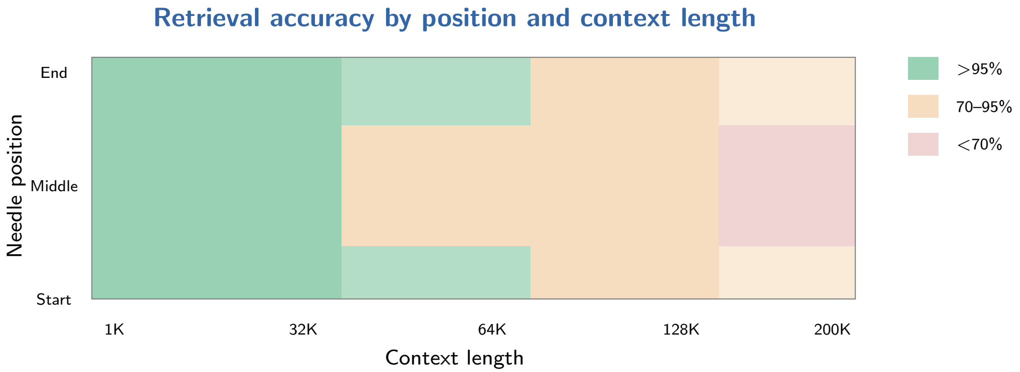


Liu et al., TACL 2024 — holds even for long-context models

Causes: RoPE long-term decay bias · training distribution (important info at start/end) · attention sink

Practical tip: put critical information at the **beginning** or **end** of your context.

Needle in a Haystack evaluation



Gemini 1.5 Pro: >**99.7%** recall up to **1M** tokens · GPT-4 Turbo: degrades beyond ~64K

Failure modes: middle positions, multi-needle retrieval, absent needle → hallucination

The full landscape

Architecture

Sparse attention (Longformer, BigBird)
Linear attention (Performer)
State Space Models (Mamba)
Hybrid (Jamba, Gemma 3)

Training

FlashAttention (IO-aware)
Ring Attention (distributed)
Position Interpolation
YaRN / LongRoPE

Inference

GQA / MQA / MLA
PagedAttention (vLLM)
KV cache quantization
KV eviction (H2O)

Evaluation

Needle in a Haystack
Lost in the Middle
Perplexity vs. length
Multi-needle retrieval

The practical stack: **FlashAttention** (training) + **GQA** (architecture) + **RoPE/YaRN** (positional) + **PagedAttention** (serving) + **sliding window** (efficiency).

Questions?

All DL4NLP topics complete!