

Recent Advancements in Natural Language Processing

Naeemullah
Khan

naeemullah.khan@kaust.edu.sa



جامعة الملك عبدالله
للغعلوم والتكنولوجية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

July 18, 2025

The History of NLP

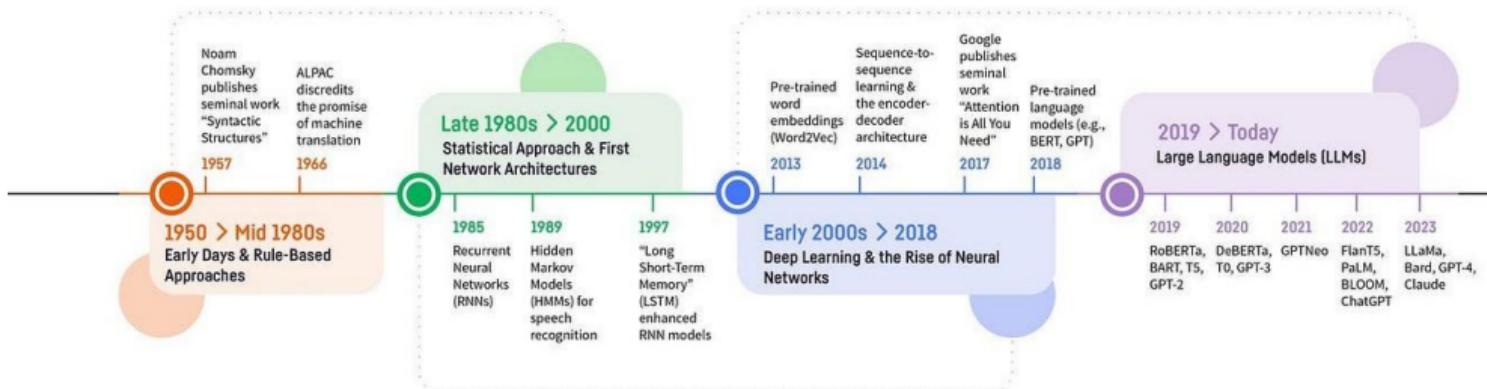


Table of Contents

1. Motivation
2. Learning Outcomes
3. Mixture of Experts (MoE)
4. FlashAttention-2
5. The LLaMA 3 Herd of Models
6. DeepSeek & GRPO
7. Byte Latent Transformer (BLT)
8. Large Language Diffusion Models (LLDMs)
9. Segment Everything Everywhere All at Once (SEEM)
10. Summary
11. References

The field of Artificial Intelligence (AI) is evolving rapidly, with significant breakthroughs across various domains:

- ▶ **Vision, Language, and Multimodal learning**
- ▶ **Faster inference:** FlashAttention-2 enables more efficient model computations.
- ▶ **Scalable learning:** Mixture of Experts (MoE) allows models to scale effectively.
- ▶ **Open-weight LLMs:** Releases like LLaMA 3 and DeepSeek foster transparency and accessibility.
- ▶ **Emerging paradigms:** Diffusion models in language and BLT (Bidirectional Language Transformers) are shaping new research

Motivation (cont.)

Staying updated with these advances is essential for modern AI development and research.

By the end of this session, you should be able to:

- ▶ Understand recent innovations in segmentation, attention mechanisms, and large language models (LLMs).
- ▶ Compare Mixture-of-Experts (MoE) architectures with dense transformer models.
- ▶ Recognize efficiency improvements in transformers, such as FlashAttention-2.
- ▶ Explore state-of-the-art open-source LLMs, including LLaMA 3 and DeepSeek.

Mixture of Experts (MoE)

MoE Overview

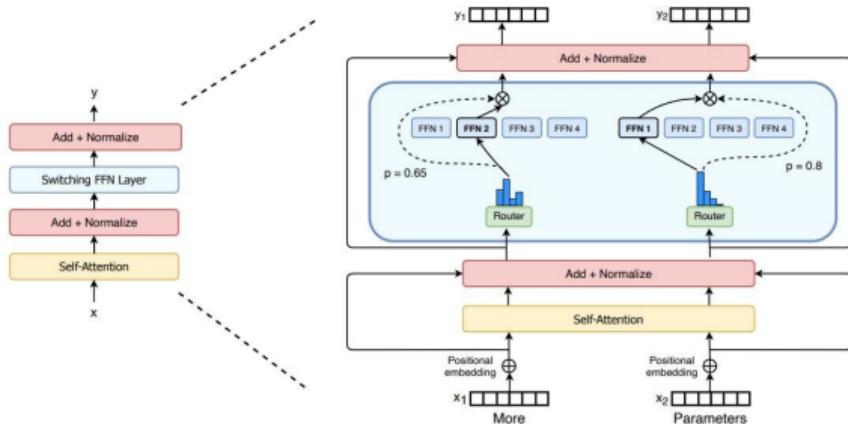


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

MoE Overview (cont.)

Mixture of Experts (MoE) is a neural network architecture that uses multiple “expert” subnetworks to process inputs, allowing for efficient scaling and specialization.

- ▶ **Experts:** Each expert is a small neural network trained on specific tasks or data.
- ▶ **Gating Mechanism:** A gating network decides which experts to activate for each input, enabling dynamic routing.
- ▶ **Scalability:** MoE models can scale to billions of parameters while maintaining computational efficiency.

- ▶ **Key Idea:** Activate only a subset of the model for each input, rather than the entire network.
- ▶ Each input is routed to a small number of experts (typically 2 out of N), improving efficiency.
- ▶ A gating network determines which experts are selected for each input.

- ▶ Switch Transformer (Fedus et al., 2021): Efficient routing and scaling.
- ▶ GShard (Lepikhin et al., 2020): Large-scale MoE for multilingual translation.
- ▶ M6, GLaM, and recent GPT-MoE variants: Further advances in scaling and performance.

What is Sparsity?

- ▶ **Sparsity:** Refers to the condition where only a small fraction of the model's parameters are active at any given time.
- ▶ In MoE, sparsity is achieved by activating only a few experts for each input, leading to reduced computational cost.
- ▶ This allows for larger models without a proportional increase in computation.



What is Sparsity? (cont.)

- ▶ **Gating Network:** Learns to route inputs to experts. Output is:

$$y = \sum_{i=1}^n G(x)_i E_i(x)$$

- ▶ **Sparsity:** If $G(x)_i = 0$, expert E_i is skipped, saving compute.
- ▶ **Typical Gating:** Softmax over a linear projection:

$$G_\sigma(x) = \text{Softmax}(xW_g)$$

- ▶ **Noisy Top-k Gating:** Adds noise for load balancing, selects top k experts:

$$H(x)_i = (xW_g)_i + \text{StandardNormal()} \cdot \text{Softplus}((xW_{noise})_i)$$

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))$$



Problem: Without intervention, the gating network tends to route most tokens to a few experts, leading to inefficient training and poor expert utilization.

- ▶ **Expert Overload:** Popular experts receive more tokens, train faster, and become even more favored.
- ▶ **Underutilization:** Other experts receive few or no tokens, limiting their contribution.

Solution: Introduce an auxiliary loss to encourage balanced token distribution across experts.

- ▶ **Auxiliary Loss:** Penalizes uneven assignment of tokens, pushing the gating network to distribute tokens more uniformly.
- ▶ **Expert Capacity:** Each expert is assigned a maximum number of

- ▶ **Implementation:** In transformer-based MoEs, this is often controlled via an auxiliary loss parameter.

Auxiliary Loss Example:

$$L_{\text{aux}} = n \cdot \sum_{i=1}^n f_i \log f_i$$

where f_i is the fraction of tokens assigned to expert i , and n is the number of experts.

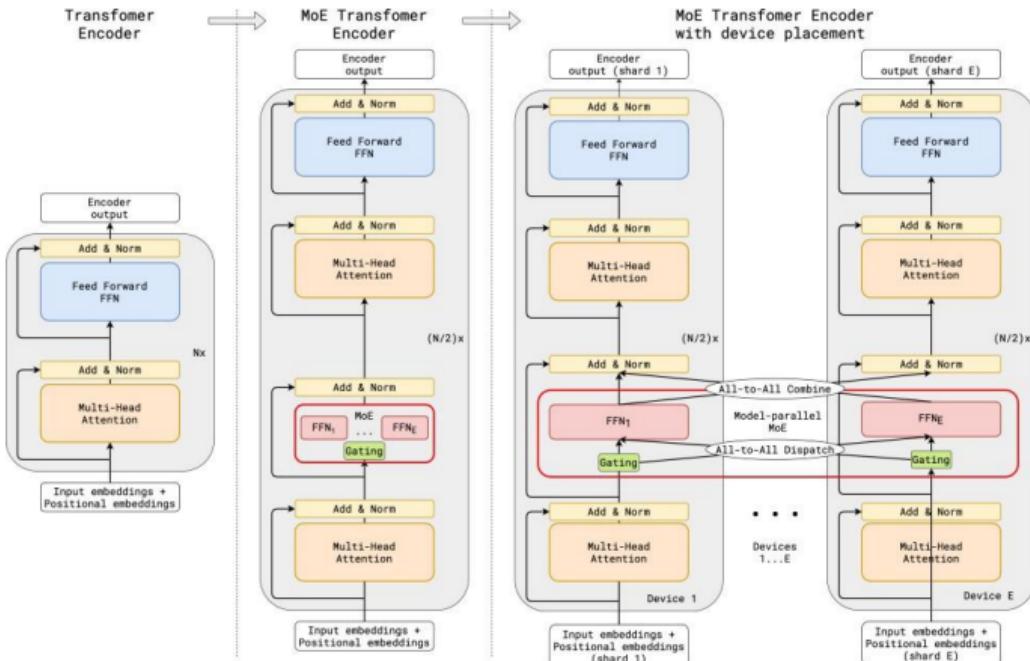
Benefits:

- ▶ Ensures all experts are trained and utilized.
- ▶ Improves model efficiency and generalization.
- ▶ Prevents bottlenecks and overload in popular experts.

Transformers and Scaling: Transformers demonstrate that increasing the number of parameters improves performance. Google's GShard project explored scaling transformers beyond 600 billion parameters using MoE layers.

- ▶ **GShard Approach:** Replaces every other feed-forward (FFN) layer with an MoE layer using top-2 gating in both encoder and decoder.
- ▶ **Distributed Training:** MoE layers are shared across devices, while other layers are replicated, enabling efficient large-scale computation.

MoEs and Transformers (cont.)



MoE Transformer Encoder from the GShard Paper

Load Balancing and Efficiency:

- ▶ **Random Routing:** In top-2 gating, the top expert is always selected, while the second expert is chosen probabilistically based on its gating weight.
- ▶ **Expert Capacity:** Each expert has a maximum token capacity. Overflow tokens are either sent via residual connections or dropped, ensuring static tensor shapes and balanced computation.

Why Expert Capacity?

- ▶ Tensor shapes must be fixed at compile time, but token distribution to experts is dynamic. Capacity factors ensure compatibility and efficiency.

Inference Efficiency:

- ▶ During inference, only a subset of experts is activated per input.
- ▶ Shared computations (e.g., self-attention) are applied to all tokens, so the effective compute is much lower than the total parameter count.
- ▶ Example: A 47B parameter model with 8 experts can run with the compute of a 12B dense model; with top-2 gating, 14B parameters are used.

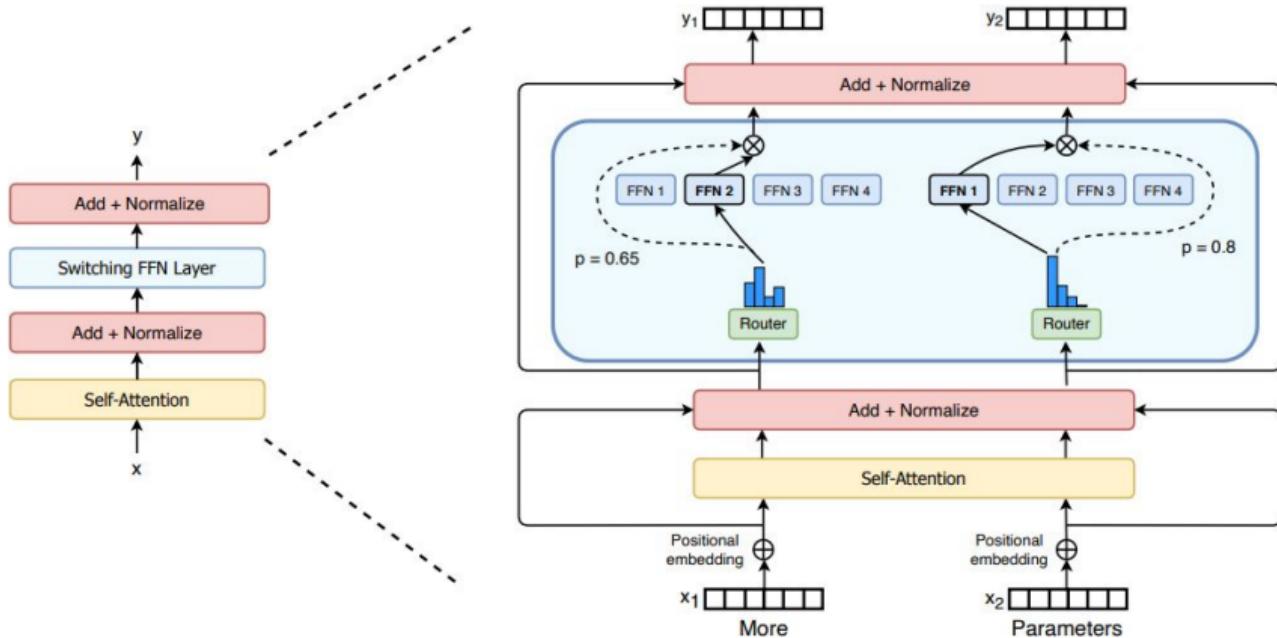
Switch Transformers: Address training and fine-tuning instabilities in MoEs by simplifying routing and improving efficiency. The authors released a 1.6T parameter MoE model with 2048 experts on Hugging Face, achieving a 4x pre-train speed-up over T5-XXL.

- ▶ **Switch Transformer Layer:** Replaces FFN layers with MoE layers, but routes each token to a single expert (single-expert strategy), reducing router computation and communication costs while preserving quality.
- ▶ **Expert Capacity:** Defined as $\frac{\text{tokens per batch}}{\text{number of experts}} \times \text{capacity factor}$
- ▶ Low capacity factors (1–1.25) are effective, balancing efficiency and communication overhead.

Switch Transformers (cont.)

- ▶ **Load Balancing Loss:** Simplified auxiliary loss encourages uniform routing and is weighted by a hyperparameter.
- ▶ **Selective Precision:** Experts are trained in bfloat16, while routing uses full precision to avoid instability. This reduces communication and computation costs without degrading quality.

Switch Transformers (cont.)



Switch Transformers (cont.)

Architecture:

- ▶ Encoder-decoder setup, similar to T5 but with MoE layers.
- ▶ Fine-tuning and summarization tasks are supported.

GLaM and Scaling:

- ▶ GLaM explores scaling MoEs further, matching GPT-3 quality with 1/3 the energy.
- ▶ Uses decoder-only models, top-2 routing, and larger capacity factors.
- ▶ Capacity factor can be adjusted during training and inference to control compute and efficiency.

When to Use Sparse MoEs:

- ▶ Ideal for high-throughput scenarios with access to many machines.
- ▶ Given a fixed compute budget for pretraining, sparse MoEs are more optimal due to efficient parameter utilization.
- ▶ Enables scaling to very large models without a proportional increase in computation.

When to Use Dense Models:

- ▶ Preferable for low-throughput scenarios or when limited VRAM is available.
- ▶ Simpler to implement and deploy on single devices.
- ▶ More suitable for smaller-scale tasks or resource-constrained

Parallelism Overview:

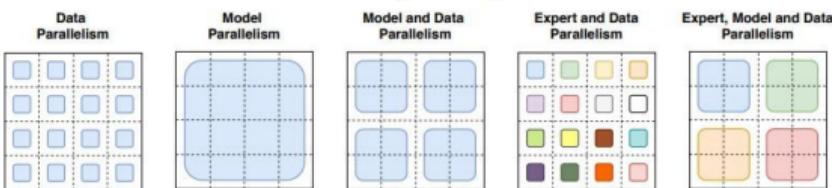
- ▶ **Data Parallelism:** Replicates model weights across all cores; data is partitioned so each core processes a different batch.
- ▶ **Model Parallelism:** Splits the model across cores; each core holds a part of the model, and data is replicated.
- ▶ **Model and Data Parallelism:** Both model and data are partitioned; different cores process different batches and hold different model segments.
- ▶ **Expert Parallelism:** Experts are distributed across workers. Each worker processes a different batch, and MoE layers route tokens to the appropriate expert on each worker.

Expert Parallelism Details:

Parallelism in MoE (cont.)

- ▶ For MoE layers, tokens are dynamically routed to the worker hosting the selected expert.
- ▶ Enables scaling to large numbers of experts and efficient distributed training.

How the *model weights* are split over cores



How the *data* is split over cores

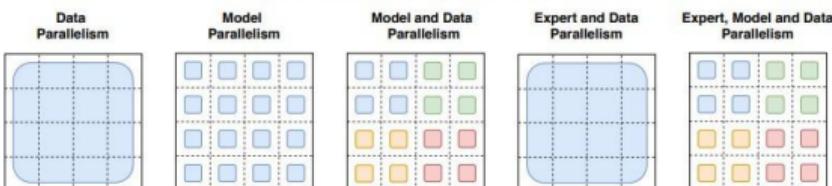


Illustration of model, expert, and data parallelism (adapted from Switch Transformers)

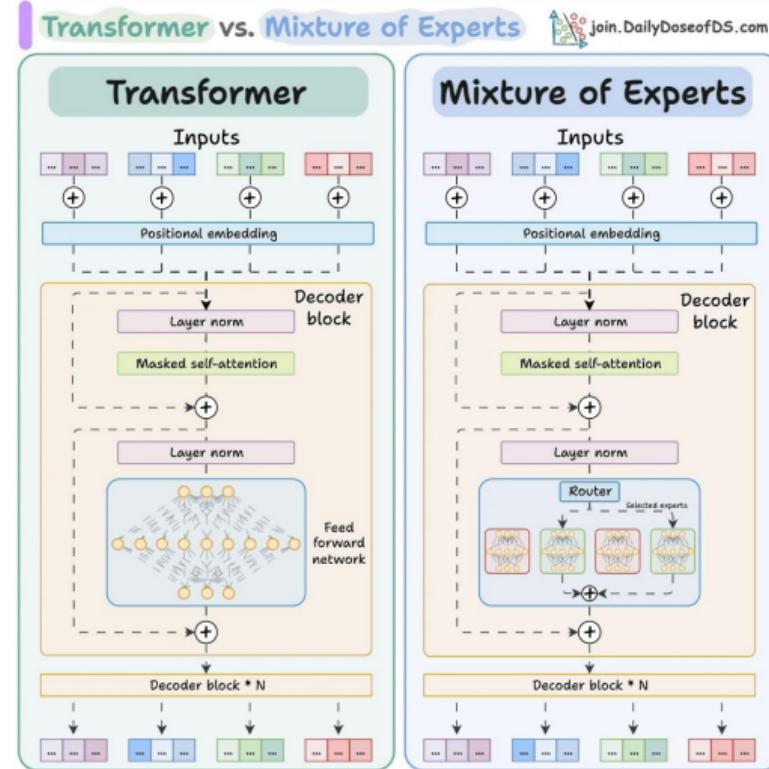
Dense Transformers:

- ▶ Every input token passes through all layers and all parameters.
- ▶ Simpler architecture and easier to implement.
- ▶ Scaling up increases both parameter count and computational cost proportionally.
- ▶ Well-suited for moderate-scale models and resource-constrained environments.

Mixture of Experts (MoE):

- ▶ Only a subset of experts (subnetworks) are activated for each input.
- ▶ Enables scaling to massive parameter counts without proportional increase in compute.
- ▶ Requires gating networks and load balancing mechanisms.
- ▶ More complex to implement and train, but highly efficient for large-scale distributed training.

Transformer vs. Mixture of Experts in LLMs (cont.)



Comparison:

- ▶ **Efficiency:** MoEs are more compute-efficient for large models, as only a fraction of parameters are used per token.
- ▶ **Scalability:** MoEs allow scaling to trillions of parameters, while dense transformers are limited by hardware constraints.
- ▶ **Specialization:** MoEs can specialize experts for different data or tasks, potentially improving performance.
- ▶ **Complexity:** Dense transformers are easier to deploy and maintain; MoEs require careful engineering for routing and load balancing.

Use Cases:

- ▶ Dense transformers: Small to medium LLMs, single-device inference, low-latency applications.
- ▶ MoEs: Large-scale pretraining, multi-device distributed systems, scenarios where efficiency and specialization are critical.

MoE: Advantages

- ▶ Scales efficiently to very large models.
- ▶ Reduces computation cost per token by activating only a few experts.
- ▶ Well-suited for distributed training across multiple devices.
- ▶ Enables specialization among experts, potentially improving performance.

MoE: Challenges



- ▶ Complex routing and expert selection mechanisms.
- ▶ Load balancing: Ensuring all experts are utilized effectively.
- ▶ Training instability and convergence issues.
- ▶ Increased implementation complexity compared to standard models.

FlashAttention- 2



Why do we care about FlashAttention?

- The algorithm is performing exact attention, so we see no reduction in perplexity or quality of the model
- The key metric is runtime

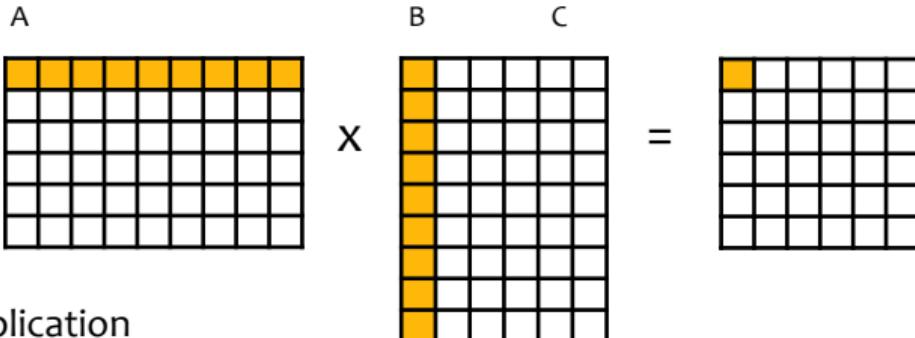
Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FLASHATTENTION	18.2	2.7 days (3.5×)
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FLASHATTENTION	14.3	6.9 days (3.0×)

Background

TILING FOR MATRIX MULTIPLICATION



Tiling for Matrix Multiplication



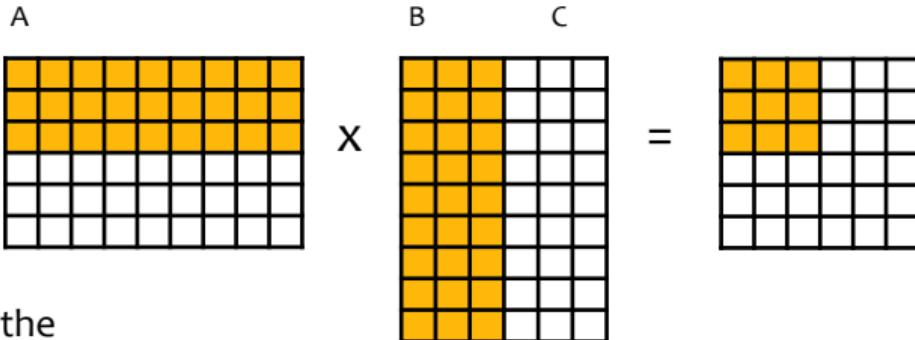
- Matrix multiplication computes each output value as a dot-product of a row/column pair from the input matrices

$$C_{ij} = \sum_{m=1}^M \sum_{n=1}^N A_{im} B_{nj}$$

=



Tiling for Matrix Multiplication

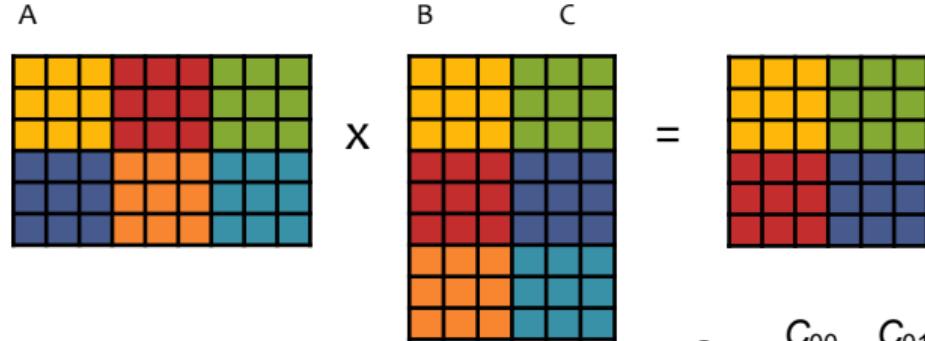


- We can view the computation as decomposing if we consider subsets of rows/columns

$$C_{(1 \ 1):(3 \ 3)} = A_{(1 \ 1):(3 \ 9)} \times B_{(1 \ 1):(9 \ 3)}$$

Tiling for Matrix Multiplication

- Tiling capitalizes on this decomposition
- Each output tile is computed by multiplying a pair of input tiles and adding it to the appropriate output tile



$$A = \begin{matrix} A_{00} & A_{01} \\ & A_{02} \\ A_{10} & A_{11} \\ & A_{12} \\ & 3 \times 3 \end{matrix}$$

with each $A_{ij} \in \mathbb{R}$

$$B = \begin{matrix} \square B_{00} & B_{01} \square \\ \square B_{10} & B_{11} \\ B_{20} & B_{21} \\ & 3 \times 3 \end{matrix}$$

with each $B_{ij} \in \mathbb{R}$

$$\begin{aligned} C &= \begin{matrix} C_{00} & C_{01} \\ C_{10} & \end{matrix} \\ \text{with each } C_{ij} &\in \mathbb{R} \\ C_{00} &= A_{00}B_{00} + A_{01}B_{01} + A_{02}B_{10} \\ C_{01} &= A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{11} \\ C_{10} &= A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20} \\ C_{11} &= A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21} \end{aligned}$$

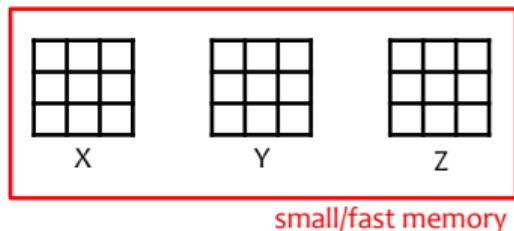
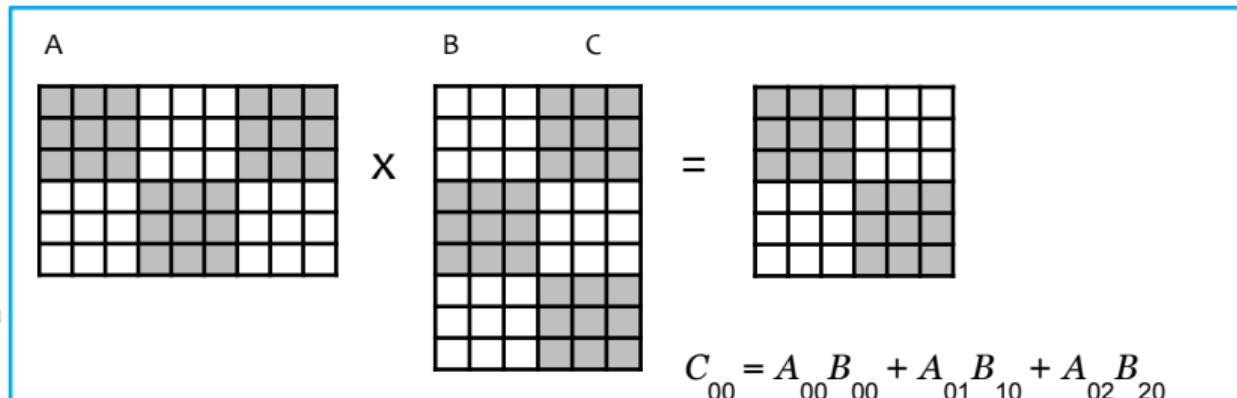


Tiling for Matrix Multiplication

- Tiling enables matrix multiplication of two **very** large matrices to capitalize on the small amount of fast memory on a device (e.g. GPU)

- Start by putting the input matrices and storage for the output matrix into large/slow memory

- Do the primary computation in slow/fast memory



$$X = A_{00}$$
$$Y$$

$$= B_{00} \quad X = A_{02}$$
$$Z = XY \quad Y = B_{20}$$
$$Y = A_{10} \quad Z = Z + XY$$
$$Z = Z + XY \quad C_{00} = Z$$

Tiling for Self-Attention?

- It would be great if we could directly use tiling for self-attention
- Unfortunately, whereas the addition in matrix multiplication is associative, the softmax in self-attention is not!

$$\mathbf{X}' = \overline{\text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k})\mathbf{V}}$$

Background

ONLINE SOFTMAX

Regular Softmax

- The standard softmax computation is used heavily throughout deep learning
- Yet, often we need to compute softmax on very large logits
- To avoid issues of overflow when raising e to some large power, we can use the safe softmax instead
- Every deep learning library implements this

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^V e^{x_j}}$$

Algorithm 1 Naive softmax

```
1:  $d_0 \leftarrow 0$ 
2: for  $j \leftarrow 1, V$  do
3:    $d_j \leftarrow d_{j-1} + e^{x_j}$ 
4: end for
5: for  $i \leftarrow 1, V$  do
6:    $y_i \leftarrow \frac{e^{x_i}}{d_V}$ 
7: end for
```

Safe Softmax

- The standard softmax computation is used heavily throughout deep learning
- Yet, often we need to compute softmax on very large logits
- To avoid issues of overflow when raising e to some large power, we can use the safe softmax instead
- Every deep learning library implements this

$$y_i = \frac{e^{x_i - \max_{k=1}^V x_k}}{\sum_{j=1}^V e^{x_j - \max_{k=1}^V x_k}}$$

Algorithm 2 Safe softmax

```

1:  $m_0 \leftarrow -\infty$ 
2: for  $k \leftarrow 1, V$  do
3:    $m_k \leftarrow \max(m_{k-1}, x_k)$ 
4: end for
5:  $d_0 \leftarrow 0$ 
6: for  $j \leftarrow 1, V$  do
7:    $d_j \leftarrow d_{j-1} + e^{x_j - m_V}$ 
8: end for
9: for  $i \leftarrow 1, V$  do
10:    $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$ 
11: end for

```

Online Softmax

- The problem with the usual safe softmax is that it requires three iterations, with each one accessing memory
- Online softmax reduces this to only two iterations through the data!
- This results in not only a 1.33x apparent speedup, but also a 1.3x speedup in practice because of reduced memory bandwidth requirements

Algorithm 3 Safe softmax with online normalizer calculation

```

1:  $m_0 \leftarrow -\infty$ 
2:  $d_0 \leftarrow 0$ 
3: for  $j \leftarrow 1, V$  do
4:    $m_j \leftarrow \max(m_{j-1}, x_j)$ 
5:    $d_j \leftarrow d_{j-1} \times e^{m_{j-1}-m_j} + e^{x_j-m_j}$ 
6: end for
7: for  $i \leftarrow 1, V$  do
8:    $y_i \leftarrow \frac{e^{x_i-m_V}}{d_V}$ 
9: end for

```

Online Softmax

- The problem with the usual safe softmax is that it requires three iterations, with each one accessing memory
- Online softmax reduces this to only two iterations through the data!
- This results in not only a 1.33x apparent speedup, but also a 1.3x speedup in practice because of reduced memory bandwidth requirements

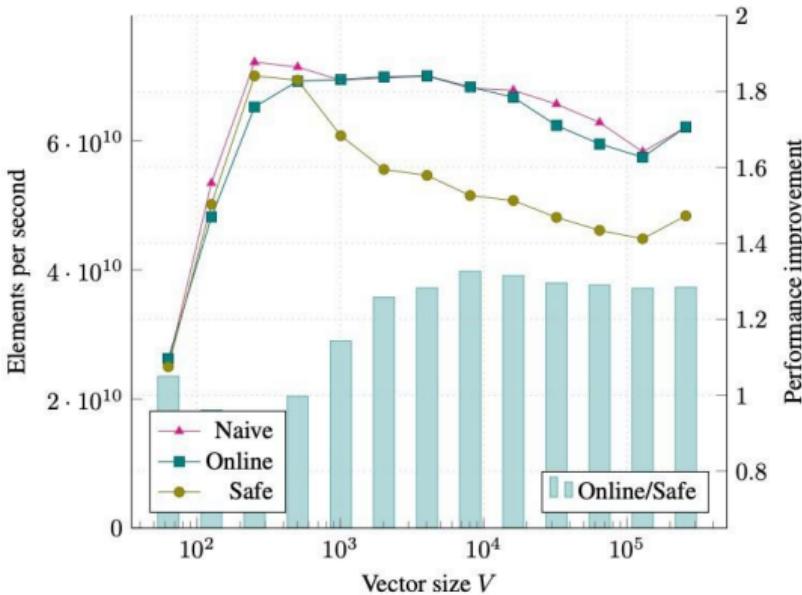


Figure 1: Benchmarking softmax, Tesla V100, fp32, batch size 4000 vectors

Figure from <https://arxiv.org/pdf/1805.02867>

FlashAttention

- One of the most impactful ideas in ML recently
- Even though many people probably don't even know they are using it!
- Introduced at HAET Workshop @ ICML July 2022
- Published @ NeurIPS Dec 2022



FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao, Dan Fu ((trid, danfu)@cs.stanford.edu)
7/23/22 HAET Workshop @ ICML 2022

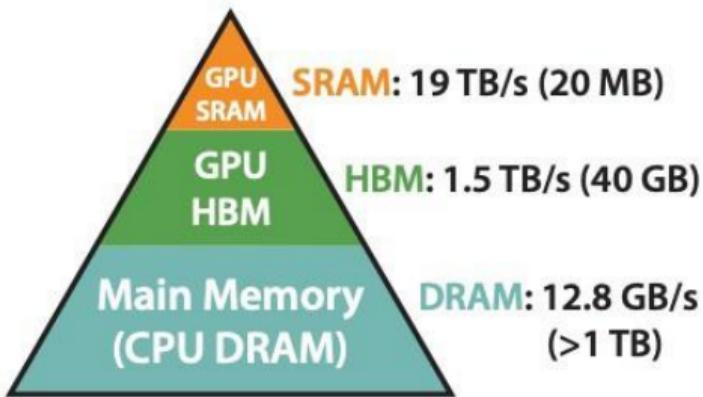
Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Ruda, Christopher Ré. Flash Attention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135*.
<https://github.com/HazyResearch/flash-attention>.



GPU Memory

Memory is arranged
hierarchically

- GPU SRAM is small, and supports the fastest access
- GPU HBM is larger but with much slower access
- CPU DRAM is huge, but the slowest of all



**Memory Hierarchy with
Bandwidth & Memory Size**

GPU Memory and Transformers

- Transformer training is usually memory-bound
- Matrix multiplication takes up 99% of the FLOPS
 - But only takes up 61% of the runtime
 - Lots of time is wasted moving data around on the GPU
 - Instead of doing computation

Table 1. Proportions for operator classes in PyTorch.

Operator class	% flop	% Runtime
△ Tensor contraction	99.80	61.0
□ Stat. normalization	0.17	25.5
○ Element-wise	0.03	13.5

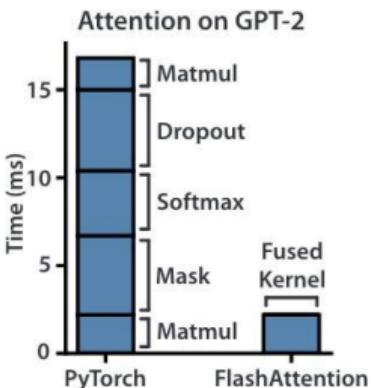
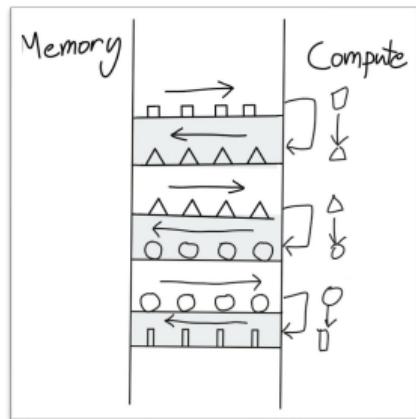


Figure from <https://arxiv.org/pdf/2205.14135>

Figure from <https://arxiv.org/pdf/2007.00072>

Operator Fusion

Version A: Usually, we compute a neural network one layer one at a time by moving the layer input to GPU SRAM (fast/small), doing some computation, then returning the output to GPU HBM (slow/large)



Version B: Operator fusion instead moves the original input to GPU SRAM (fast/small), does a whole sequence of layer computations without ever touching HBM, and then returns the final layer output to GPU HBM (slow/large)

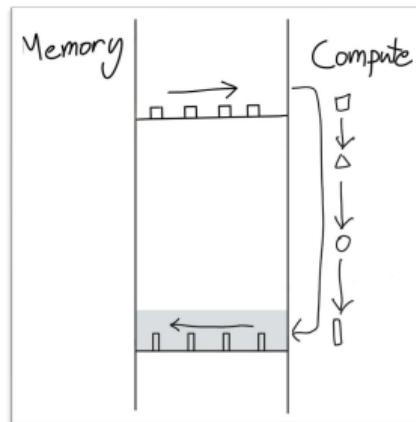


Figure from https://horace.io/brrr_intro.html

Operator Fusion

Version A: Usually, we compute a neural network one layer one at a time by moving the layer input to GPU SRAM (fast/small), doing some computation, then returning the output to GPU HBM (slow/large)

Version B: Operator fusion instead moves the original input to GPU SRAM (fast/small), does a whole sequence of layer computations without ever touching HBM, and then returns the final layer output to GPU HBM (slow/large)

Version A is exactly how standard attention is implemented

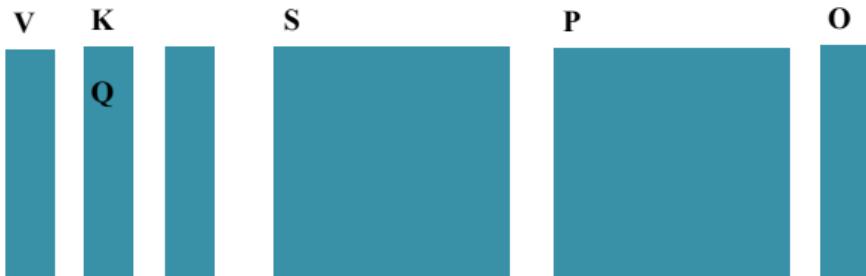
$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \text{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

- 1: Load \mathbf{Q}, \mathbf{K} by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^T$, write \mathbf{S} to HBM.
 - 2: Read \mathbf{S} from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write \mathbf{P} to HBM.
 - 3: Load \mathbf{P} and \mathbf{V} by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write \mathbf{O} to HBM.
 - 4: Return \mathbf{O} .
-

Standard Attention



Version A is exactly how standard attention is implemented

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \text{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

- 1: Load \mathbf{Q}, \mathbf{K} by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^T$, write \mathbf{S} to HBM.
 - 2: Read \mathbf{S} from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write \mathbf{P} to HBM.
 - 3: Load \mathbf{P} and \mathbf{V} by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write \mathbf{O} to HBM.
 - 4: Return \mathbf{O} .
-

- Two key ideas are combined to obtain FlashAttention
- Both are well-established ideas, so the interesting part is how they are put together for attention
 1. **Tiling:** compute the attention weights block by block so that we don't have to load everything into SRAM at once
 2. **Recomputation:** don't ever store the full attention matrix, but just recompute the parts of it you need during the backward pass

FlashAttention: Tiling

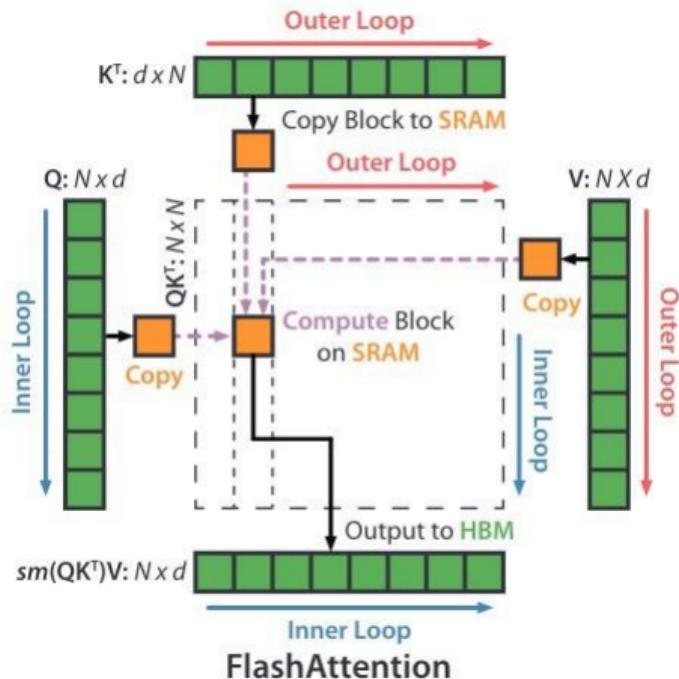


Figure from <https://arxiv.org/pdf/2205.14135>



FlashAttention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

- 1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
 - 2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
 - 3: Divide \mathbf{Q} into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide \mathbf{K}, \mathbf{V} in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
 - 4: Divide \mathbf{O} into T_r blocks $\mathbf{O}_1, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.
 - 5: **for** $1 \leq j \leq T_c$ **do**
 - 6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
 - 7: **for** $1 \leq i \leq T_r$ **do**
 - 8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
 - 9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
 - 10: On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}, \tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
 - 11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}, \ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
 - 12: Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
 - 13: Write $\ell_i \leftarrow \ell_i^{\text{new}}, m_i \leftarrow m_i^{\text{new}}$ to HBM.
 - 14: **end for**
 - 15: **end for**
 - 16: Return \mathbf{O} .
-

FlashAttention: Tiling

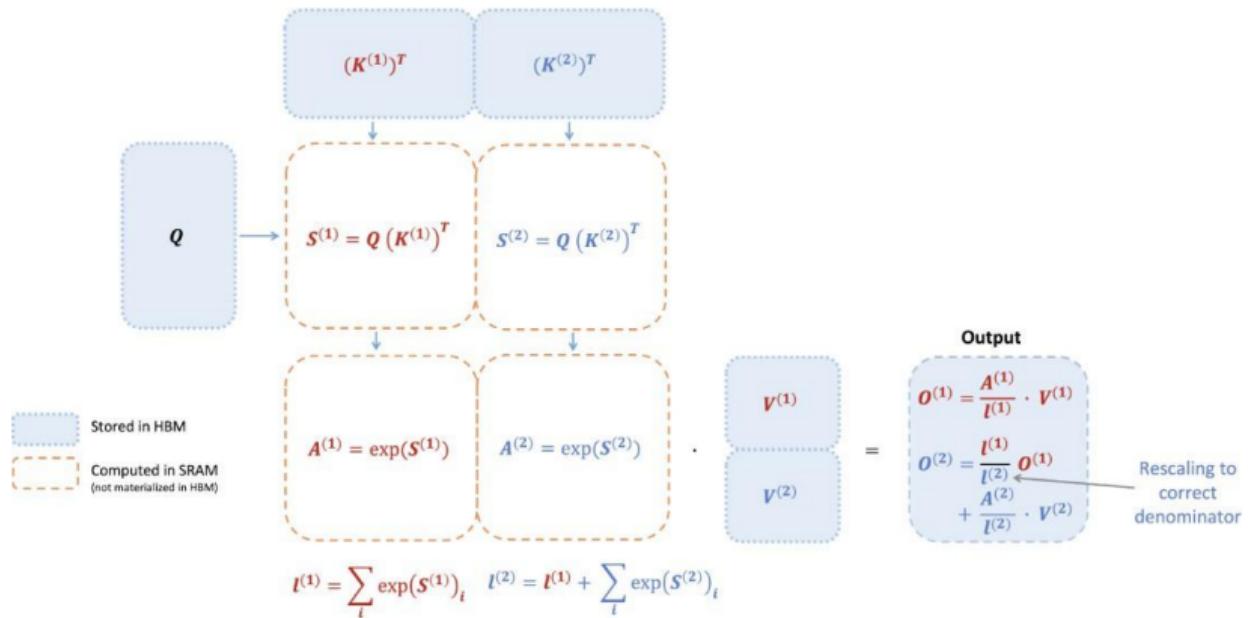


Figure from <http://arxiv.org/abs/2307.08691>



FlashAttention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

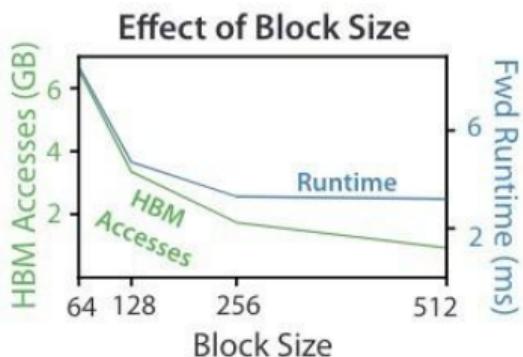
- 1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
 - 2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
 - 3: Divide \mathbf{Q} into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide \mathbf{K}, \mathbf{V} in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
 - 4: Divide \mathbf{O} into T_r blocks $\mathbf{O}_1, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.
 - 5: **for** $1 \leq j \leq T_c$ **do**
 - 6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
 - 7: **for** $1 \leq i \leq T_r$ **do**
 - 8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
 - 9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
 - 10: On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}, \tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
 - 11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}, \ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
 - 12: Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
 - 13: Write $\ell_i \leftarrow \ell_i^{\text{new}}, m_i \leftarrow m_i^{\text{new}}$ to HBM.
 - 14: **end for**
 - 15: **end for**
 - 16: Return \mathbf{O} .
-



FlashAttention: Results

- The algorithm is performing exact attention, so we see no reduction in perplexity or quality of the model
- The key metric is runtime

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3



FlashAttention: Results

- The algorithm is performing exact attention, so we see no reduction in perplexity or quality of the model
- The key metric is runtime

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [8]	18.2	9.5 days (1.0x)
GPT-2 small - Megatron-LM [7]	18.2	4.7 days (2.0x)
GPT-2 small - FLASHATTENTION	18.2	2.7 days (3.5x)
GPT-2 medium - Huggingface [8]	14.2	21.0 days (1.0x)
GPT-2 medium - Megatron-LM [7]	14.3	11.5 days (1.8x)
GPT-2 medium - FLASHATTENTION	14.3	6.9 days (3.0x)

FlashAttention-2: Overview

- ▶ **Paper:** Dao et al. (2023)
- ▶ **Goal:** Memory-efficient and fast attention computation
- ▶ **Highlights:**
 - 90–95% GPU utilization
 - 2–4× faster than standard attention
- ▶ **Techniques:**
 - Multi-query and grouped-query attention
 - Hardware-aware scheduling
 - Supports long context windows (32k+ tokens)
- ▶ **Adoption:**
 - Used in OpenAI GPT-4, Mistral, LLaMA 3

FlashAttention-2: Key Improvements

▶ Fewer non-matmul FLOPs:

- Algorithm tweaks reduce non-matmul operations (e.g., rescaling, masking).
- Maximizes use of specialized GPU units (Tensor Cores).
- Non-matmul FLOPs are up to $16\times$ more expensive than matmul FLOPs.

▶ Better Parallelism:

- Parallelizes over sequence length in addition to batch size and heads.
- Improves GPU utilization for long sequences and small batches.

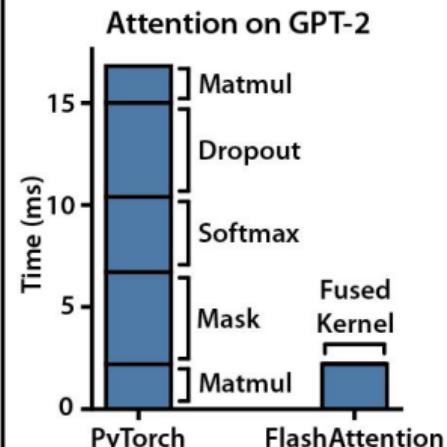
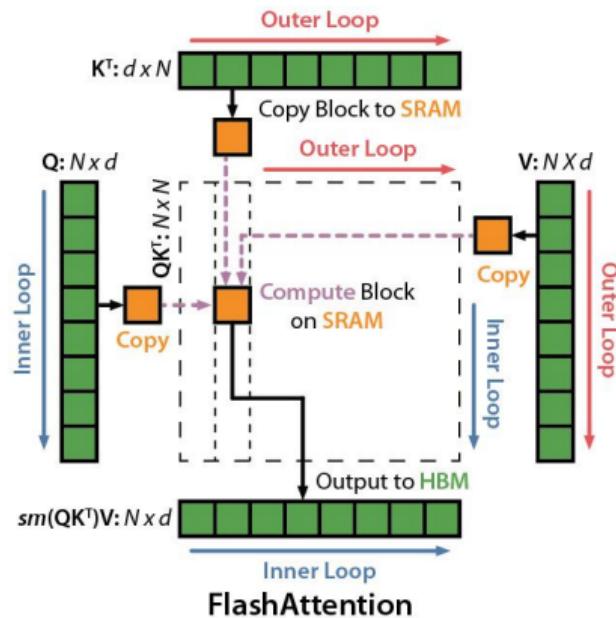
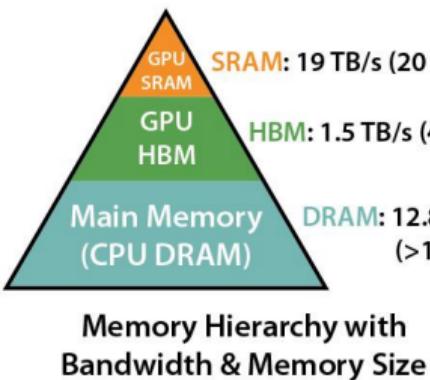
► Improved Work Partitioning:

- Splits Q across warps (instead of K/V), reducing shared memory communication.
- Yields significant speedup by minimizing synchronization.

► New Features:

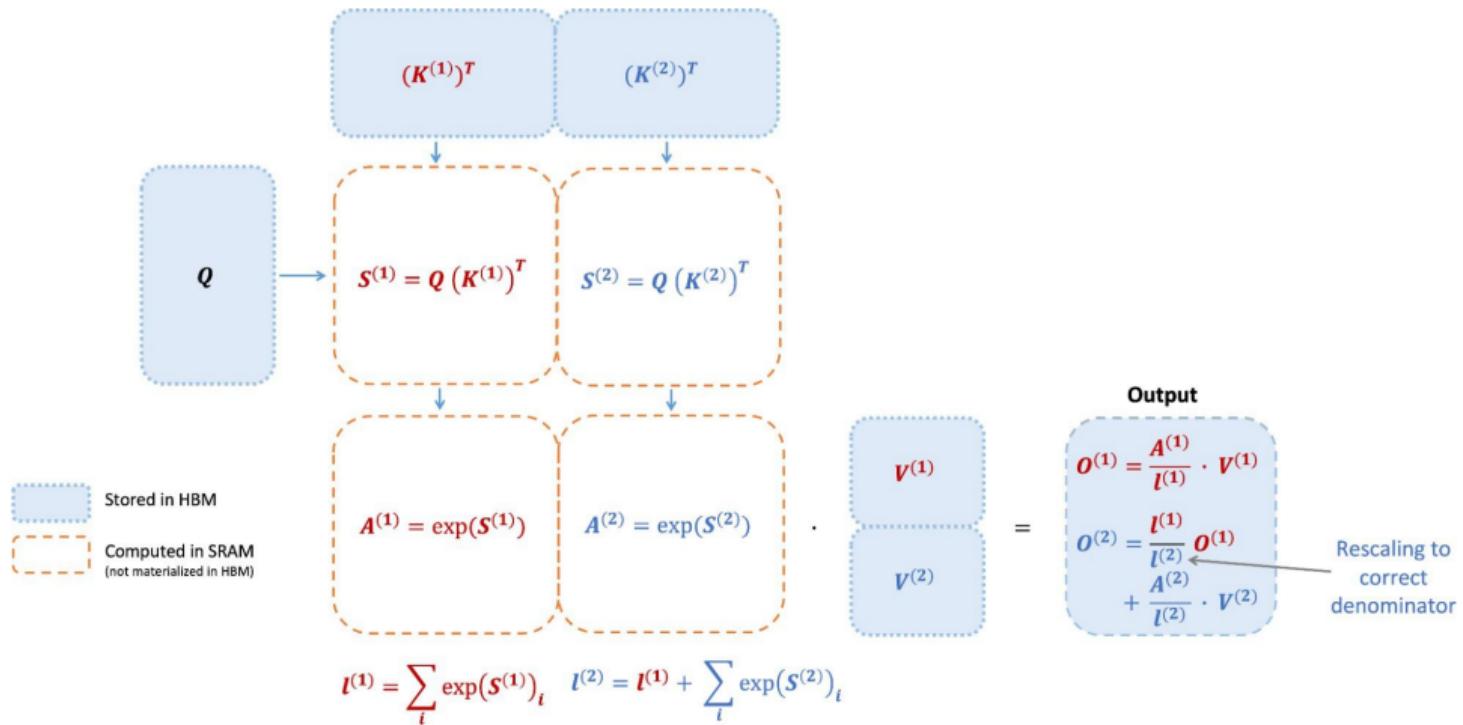
- Supports head dimensions up to 256.
- Enables multi-query and grouped-query attention.

FlashAttention-2: Architecture

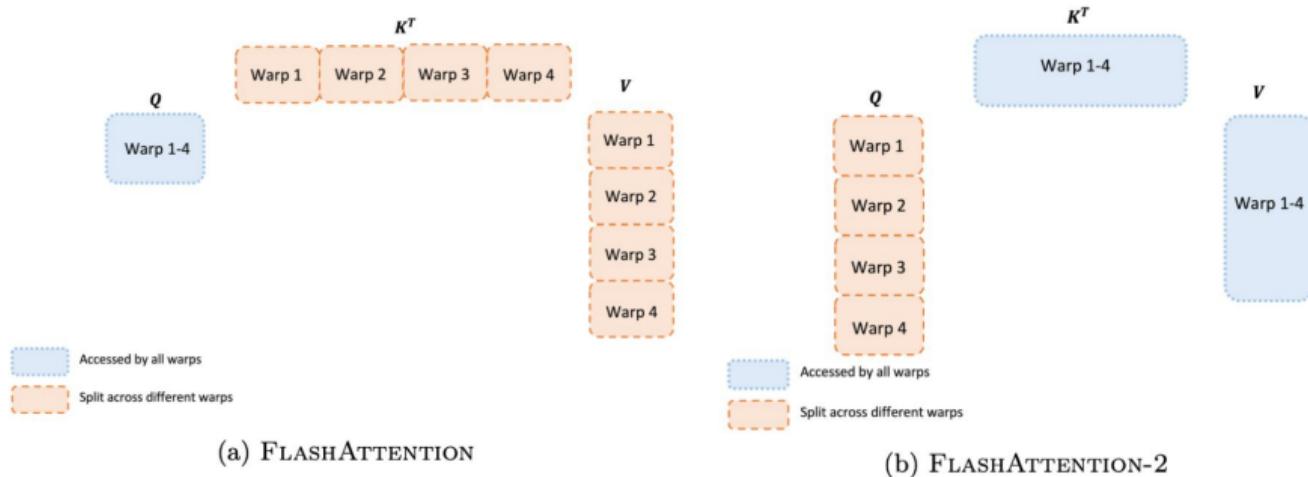


FlashAttention-2: Architecture

(cont.)



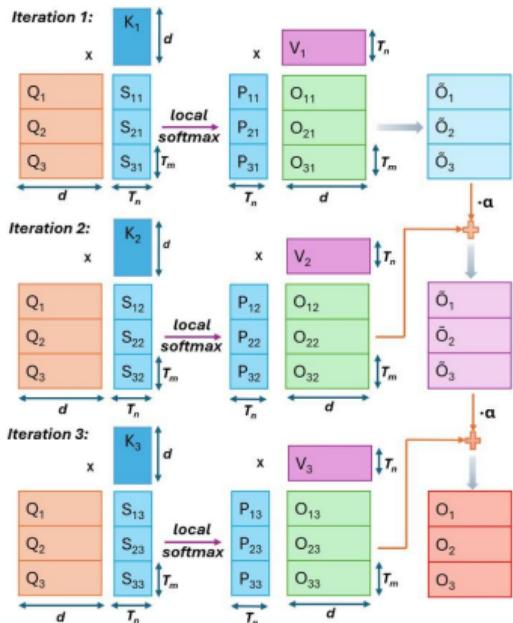
FlashAttention-2: Architecture (cont.)



FlashAttention-2: Architecture (cont.)

- ▶ **Multi-Query Attention:** Uses a single set of keys and values for all queries, reducing memory usage.
- ▶ **Grouped-Query Attention:** Groups queries to share keys and values, further optimizing memory.
- ▶ **Hardware-Aware Scheduling:** Optimizes GPU scheduling to maximize throughput and minimize latency.
- ▶ **Long Context Support:** Efficiently handles long sequences (32k+ tokens) with minimal memory overhead.

FlashAttention-2: Architecture (cont.)



Iterative update of output in FlashAttention-2 with $C_n = 3$ iterations.

FlashAttention-2: Performance Benchmarks

▶ Speed:

- Up to $2\times$ faster than FlashAttention-1, $9\times$ faster than standard PyTorch attention.
- Achieves up to 335 TFLOPs/s on H100 GPUs.

▶ End-to-End Training:

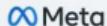
- $1.3\times$ speedup over optimized FlashAttention models.
- Up to 225 TFLOPs/s on A100 GPU (72% model FLOPs utilization).

▶ Model Example Results:

	Baseline	FlashAttention	FlashAttention-2
GPT3-1.3B 2k	142	189	196
GPT3-1.3B 8k	72	170	220
GPT3-2.7B 2k	149	189	205
GPT3-2.7B 8k	80	175	225

▶ Baseline: Megatron-LM without FlashAttention.

The LLaMA 3 Herd of Models



The Llama 3 Herd of Models

Llama Team, AI @ Meta¹

¹A detailed contributor list can be found in the appendix of this paper.

Modern artificial intelligence (AI) systems are powered by foundation models. This paper presents a new set of foundation models, called Llama 3. It is a herd of language models that natively support multilinguality, coding, reasoning, and tool usage. Our largest model is a dense Transformer with 405B parameters and a context window of up to 128K tokens. This paper presents an extensive empirical evaluation of Llama 3. We find that Llama 3 delivers comparable quality to leading language models such as GPT-4 on a plethora of tasks. We publicly release Llama 3, including pre-trained and post-trained versions of the 405B parameter language model and our Llama Guard 3 model for input and output safety. The paper also presents the results of experiments in which we integrate image, video, and speech capabilities into Llama 3 via a compositional approach. We observe this approach performs competitively with the state-of-the-art on image, video, and speech recognition tasks. The resulting models are not yet being broadly released as they are still under development.

Date: July 23, 2024

Website: <https://llama.meta.com/>

The LLaMA 3 Herd of Models (cont.)

Meta AI (2024)

- ▶ **Model Sizes:** 8B, 70B, and smaller variants
- ▶ **Open Weights:** Available for commercial use
- ▶ **Training:** Over 15 trillion tokens, multilingual data, improved alignment techniques

The LLaMA 3 Herd of Models (cont.)

	Finetuned	Multilingual	Long context	Tool use	Release
Llama 3 8B	✗	✗ ¹	✗	✗	April 2024
Llama 3 8B Instruct	✓	✗	✗	✗	April 2024
Llama 3 70B	✗	✗ ¹	✗	✗	April 2024
Llama 3 70B Instruct	✓	✗	✗	✗	April 2024
Llama 3.1 8B	✗	✓	✓	✗	July 2024
Llama 3.1 8B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 70B	✗	✓	✓	✗	July 2024
Llama 3.1 70B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 405B	✗	✓	✓	✗	July 2024
Llama 3.1 405B Instruct	✓	✓	✓	✓	July 2024

Overview of the Llama 3 Herd of models. All results in this paper are for the Llama 3.1 models.

The LLaMA 3 Herd of Models (cont.)

Key Features:

- ▶ 128k context window for extended input sequences
- ▶ FlashAttention-2 and grouped-query attention for efficient inference
- ▶ Fine-tuned variants: Instruct and Chat models

The LLaMA 3 Herd of Models (cont.)

Category	Benchmark	Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 _(0/12B)	GPT-4o	Claude 3.5 Sonnet
General	MMLU (5-shot)	69.4	72.3	61.1	83.6	76.9	70.7	87.3	82.6	85.1	89.1	89.9
	MMLU (0-shot, CoT)	73.0	72.3 [△]	60.5	86.0	79.9	69.8	88.6	78.7 [□]	85.4	88.7	88.3
	MMLU-Pro (5-shot, CoT)	48.3	—	36.9	66.4	56.3	49.2	73.3	62.7	64.8	74.0	77.0
	IFEval	80.4	73.6	57.6	87.5	72.7	69.9	88.6	85.1	84.3	85.6	88.0
Code	HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0	89.0	73.2	86.6	90.2	92.0
	MBPP EvalPlus (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0	88.6	72.8	83.6	87.8	90.5
Math	GSM8K (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6	96.8	92.3 [◊]	94.2	96.1	96.4 [◊]
	MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1	73.8	41.1	64.5	76.6	71.1
Reasoning	ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7	96.9	94.6	96.4	96.7	96.7
	GPQA (0-shot, CoT)	32.8	—	28.8	46.7	33.3	30.8	51.1	—	41.4	53.6	59.4
Tool use	BFCL	76.1	—	60.4	84.8	—	85.9	88.5	86.5	88.3	80.5	90.2
	Nexus	38.5	30.0	24.7	56.7	48.5	37.2	58.7	—	50.3	56.1	45.7
Long context	ZeroSCROLLS/QuALITY	81.0	—	—	90.5	—	—	95.2	—	95.2	90.5	90.5
	InfiniteBench/En.MC	65.1	—	—	78.2	—	—	83.4	—	72.1	82.5	—
	NIH/Multi-needle	98.8	—	—	97.5	—	—	98.1	—	100.0	100.0	90.8
Multilingual	MGSM (0-shot, CoT)	68.9	53.2	29.9	86.9	71.1	51.4	91.6	—	85.9	90.5	91.6

Table 2 Performance of finetuned Llama 3 models on key benchmark evaluations. The table compares the performance of the 8B, 70B, and 405B versions of Llama 3 with that of competing models. We **boldface** the best-performing model in each of three model-size equivalence classes. [△]Results obtained using 5-shot prompting (no CoT). [□]Results obtained without CoT. [◊]Results obtained using zero-shot prompting.

The LLaMA 3 Herd of Models (cont.)

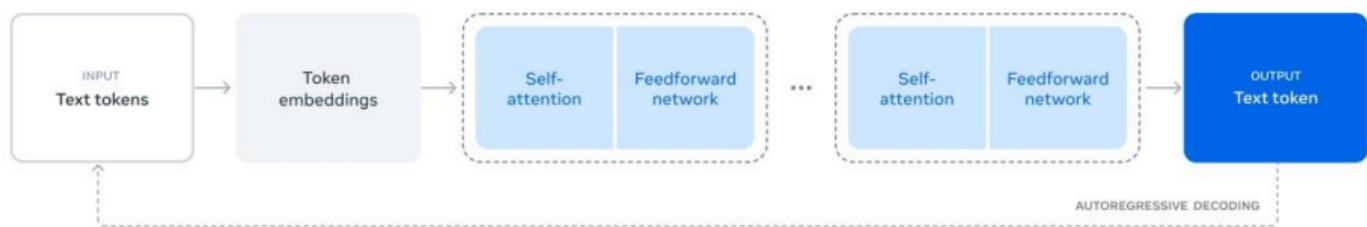


Figure 1 Illustration of the overall architecture and training of Llama 3. Llama 3 is a Transformer language model trained to predict the next token of a textual sequence. See text for details.

The LLaMA 3 Herd of Models (cont.)

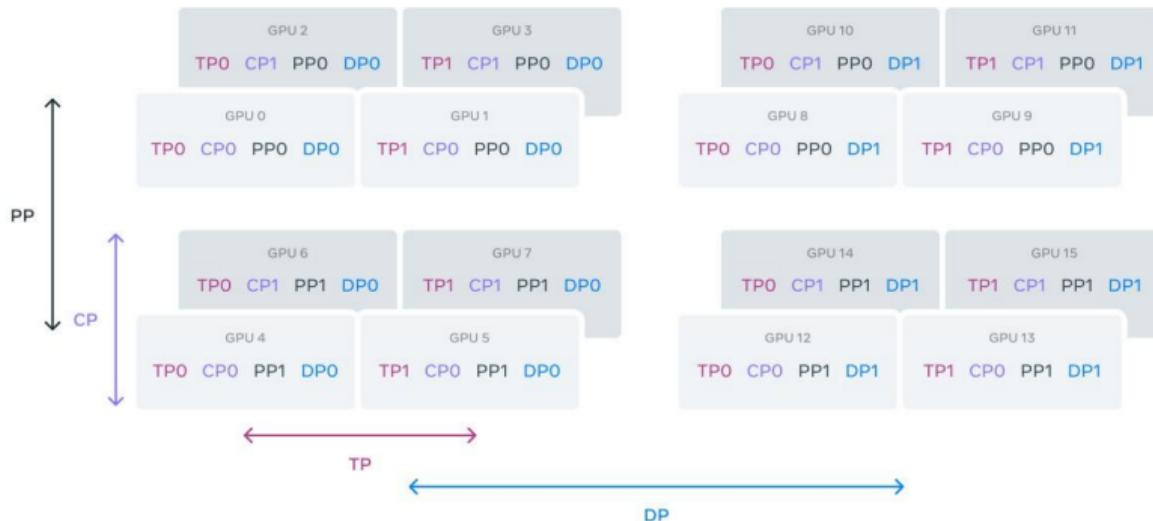


Figure 5 Illustration of 4D parallelism. GPUs are divided into parallelism groups in the order of [TP, CP, PP, DP], where DP stands for FSDP. In this example, 16 GPUs are configured with a group size of $|TP|=2$, $|CP|=2$, $|PP|=2$, and $|DP|=2$. A GPU's position in 4D parallelism is represented as a vector, $[D_1, D_2, D_3, D_4]$, where D_i is the index on the i -th parallelism dimension. In this example, GPU0[TP0, CP0, PP0, DP0] and GPU1[TP1, CP0, PP0, DP0] are in the same TP group, GPU0 and GPU2 are in the same CP group, GPU0 and GPU4 are in the same PP group, and GPU0 and GPU8 are in the same DP group.

The LLaMA 3 Herd of Models (cont.)

User prompt

```
Based on the inflation rate for the US in February 2024, and the cost of this list of tuples containing my daily groceries, ({"eggs":3.0,"chicken":4.0,"rice":2.0,"beans":2.50}, {"apples":6.25}), can you give me the estimated total cost of my groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage in $X.XX?
```

System Prompt

```
Environment: python,
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today's Date: Jul 11, 2024
```

Assistant response

To answer your question, I will need to:

1. Look up the inflation rate for the US in February 2024
2. Look up the Costco food court giant cookie debut price
3. Calculate the estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage

First, I will look up the inflation rate for the US in February 2024.

Assistant tool call

```
brave_search.call(query="US inflation rate February 2024")
```

Tool output

```
({"title": "US Inflation Remained Higher Than Economists Had Expected in February", "url": "https://www.investopedia.com/february-2024-cpi-8878236", "description": "Consumer prices rose <strong>3.2%</strong> over the year in February, higher than the 3.1% annual rate in January. This also exceeded what economists had predicted. An uptick in gas prices and housing prices was behind the unexpectedly high inflation rate.", })
```

Assistant tool call

```
brave_search.call(query="Costco food court giant cookie debut price")
```

Tool output

```
{ "title": "Costco's food court replaced its churro with a giant cookie. Here's how it tastes.", "url": "https://www.today.com/food/groceries/costco-double-chocolate-chunk-cookies-food-court-revealed/cn13551796", "description": "The latest addition to its <strong>Food/strong> <strong>court</strong>, costs $2.49 and has 750 calories. Here's my review of the buttery treat.", })
```

Assistant tool call

```
# Define the list of tuples containing the daily groceries
groceries = [ {"eggs":3.0}, {"chicken":4.0}, {"rice":2.0}, {"beans":2.50}, {"apples":6.25} ]
```

Tool output

```
The estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage is $626.63
```

The estimated total cost of your groceries over a 30 day month when you add 1 cookie per day at Costco's food court giant cookie debut price and adjust the total for the inflation percentage is **\$626.63**

Figure 10 Multi-step tool usage. Example of Llama 3 performing multi-step planning, reasoning, and tool calling to solve a task.

The LLaMA 3 Herd of Models (cont.)

Reading Comprehension			
	SQuAD	QuAC	RACE
Llama 3 8B	77.0 ±0.8	44.9 ±1.1	54.3 ±1.4
Mistral 7B	73.2 ±0.8	44.7 ±1.1	53.0 ±1.4
Gemma 7B	81.8 ±0.7	42.4 ±1.1	48.8 ±1.4
Llama 3 70B	81.8 ±0.7	51.1 ±1.1	59.0 ±1.4
Mixtral 8×22B	84.1 ±0.7	44.9 ±1.1	59.2 ±1.4
Llama 3 405B	81.8 ±0.7	53.6 ±1.1	58.1 ±1.4
GPT-4	—	—	—
Nemotron 4 340B	—	—	—
Gemini Ultra	—	—	—

Table 9 Pre-trained model performance on reading comprehension tasks. Results include 95% confidence intervals.

Code		
	HumanEval	MBPP
Llama 3 8B	37.2 ±7.4	47.6 ±4.4
Mistral 7B	30.5 ±7.0	47.5 ±4.4
Gemma 7B	32.3 ±7.2	44.4 ±4.4
Llama 3 70B	58.5 ±7.5	66.2 ±4.1
Mixtral 8×22B	45.1 ±7.6	71.2 ±4.0
Llama 3 405B	61.0 ±7.5	73.4 ±3.9
GPT-4	67.0 ±7.2	—
Nemotron 4 340B	57.3 ±7.6	—
Gemini Ultra	74.4 ±6.7	—

Table 10 Pre-trained model performance on coding tasks. Results include 95% confidence intervals.

The LLaMA 3 Herd of Models (cont.)

Highlights:

- ▶ Competitive performance with GPT-4 and Gemini
- ▶ Foundation for Code LLaMA, Audio LLaMA, and multimodal LLaMA models

DeepSeek & GRPO

► **DeepSeek:** China's open LLM ecosystem

- Models: DeepSeek-VL, DeepSeek-Coder, DeepSeek-MoE
- GPT-style, bilingual (Chinese/English)
- Performance close to GPT-4

► **GRPO:** General Reinforced Preference Optimization

- Extension of RLHF (Reinforcement Learning from Human Feedback)
- Trains LLMs with both reward and preference signals
- Stable, efficient, converges faster than PPO
- **Improves safety and alignment of open-weight models**
- Used in DeepSeek-MoE and other models

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

Abstract

We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Qwen and Llama.

Key Features:

- ▶ Open-source, bilingual LLM ecosystem
- ▶ Models trained on diverse datasets
- ▶ Focus on safety and alignment

DeepSeek-R1 (cont.)

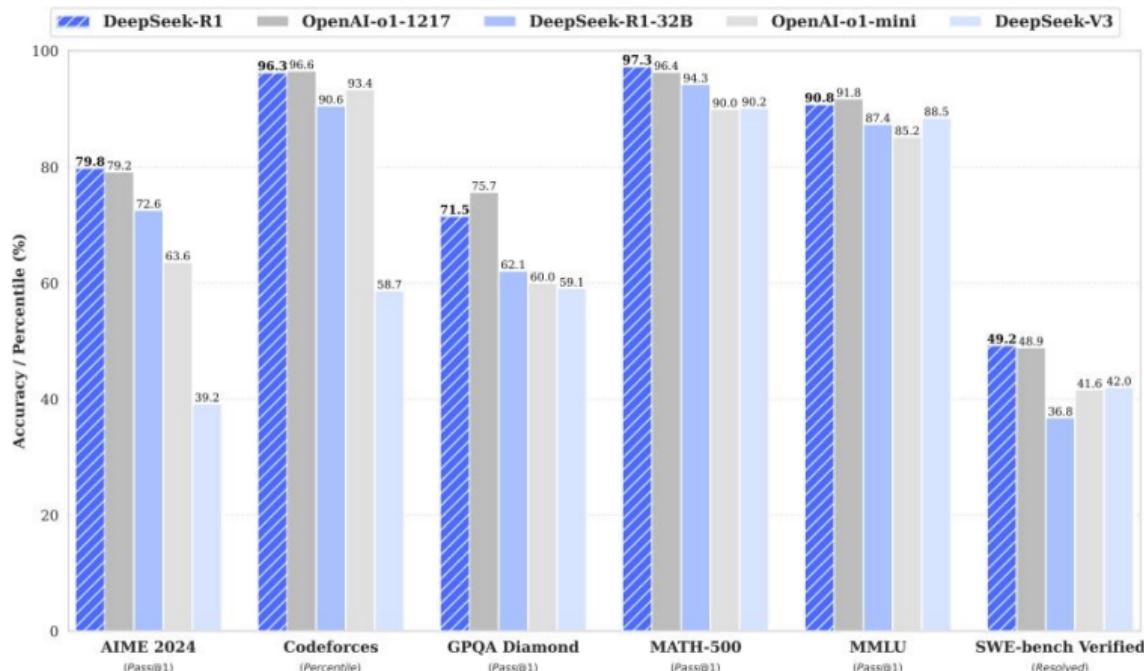


Figure 1 | Benchmark performance of DeepSeek-R1.

GRPO Overview:

- ▶ Combines reward and preference signals for training
- ▶ Uses a single model for both signals
- ▶ More stable and efficient than traditional RLHF methods

GRPO Objective:

- ▶ GRPO avoids the need for a separate critic model by estimating the baseline from group scores.
- ▶ For each question q , sample a group of outputs $\{o_1, o_2, \dots, o_G\}$ from the old policy $\pi_{\theta_{\text{old}}}$.
- ▶ The policy model π_θ is optimized by maximizing:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(O|q)} \left[\frac{1}{G} \sum_{i=1}^G \left(\min \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)} A_i, \right. \right. \right. \\ \left. \left. \left. \text{clip} \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}, 1 - \epsilon, 1 + \epsilon \right) A_i \right) - \beta D_{\text{KL}}(\pi_\theta || \pi_{\text{ref}}) \right) \right] \quad (1)$$

- ▶ KL divergence term:

$$D_{\text{KL}}(\pi_\theta || \pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(o_i|q)}{\pi_\theta(o_i|q)} - \log \frac{\pi_{\text{ref}}(o_i|q)}{\pi_\theta(o_i|q)} - 1 \quad (2)$$

- ▶ Advantage A_i is computed using group rewards $\{r_1, r_2, \dots, r_G\}$:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})} \quad (3)$$

- ▶ ϵ and β are hyper-parameters.

Reward Modeling in DeepSeek-R1-Zero:

- ▶ **Accuracy Rewards:** Evaluate correctness of responses using rule-based systems.
 - For math problems, answers must be provided in a specified format (e.g., boxed), allowing reliable verification.
 - For coding tasks (e.g., LeetCode), a compiler checks solutions against predefined test cases.
- ▶ **Format Rewards:** Enforce structured reasoning by requiring the model's thought process to be enclosed within <think> and </think> tags.
- ▶ **No Neural Reward Model:** Neural reward models are avoided due to risks of reward hacking and increased training complexity.
- ▶ The rule-based reward system provides stable and interpretable

Key Advantages:

- ▶ Faster convergence compared to PPO
- ▶ Improved safety and alignment of models
- ▶ Effective for open-weight models

DeepSeek-R1 Reinforced Preference Optimization

(SNPDE)

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	1820
OpenAI-o1-0912	74.4	83.3	94.8	77.3	63.4	1843
DeepSeek-R1-Zero	71.0	86.7	95.9	73.3	50.0	1444

Table 2 | Comparison of DeepSeek-R1-Zero and OpenAI o1 models on reasoning-related benchmarks.

DeepSeek-R1 Reinforced Preference Optimization (CPPO)

Benchmark (Metric)	Claude-3.5-Sonnet-1022	GPT-4o 0513	DeepSeek V3	OpenAI o1-mini	OpenAI o1-1217	DeepSeek R1
Architecture	-	-	MoE	-	-	MoE
# Activated Params	-	-	37B	-	-	37B
# Total Params	-	-	671B	-	-	671B
English	MMLU (Pass@1)	88.3	87.2	88.5	85.2	91.8
	MMLU-Redux (EM)	88.9	88.0	89.1	86.7	-
	MMLU-Pro (EM)	78.0	72.6	75.9	80.3	-
	DROP (3-shot F1)	88.3	83.7	91.6	83.9	90.2
	IF-Eval (Prompt Strict)	86.5	84.3	86.1	84.8	-
	GPQA Diamond (Pass@1)	65.0	49.9	59.1	60.0	75.7
	SimpleQA (Correct)	28.4	38.2	24.9	7.0	47.0
	FRAMES (Acc.)	72.5	80.5	73.3	76.9	-
	AlpacaEval2.0 (LC-winrate)	52.0	51.1	70.0	57.8	-
	ArenaHard (GPT-4-1106)	85.2	80.4	85.5	92.0	-
Code	LiveCodeBench (Pass@1-COT)	38.9	32.9	36.2	53.8	63.4
	Codeforces (Percentile)	20.3	23.6	58.7	93.4	96.6
	Codeforces (Rating)	717	759	1134	1820	2061
	SWE Verified (Resolved)	50.8	38.8	42.0	41.6	48.9
	Aider-Polyglot (Acc.)	45.3	16.0	49.6	32.9	61.7
Math	AIME 2024 (Pass@1)	16.0	9.3	39.2	63.6	79.2
	MATH-500 (Pass@1)	78.3	74.6	90.2	90.0	96.4
	CNMO 2024 (Pass@1)	13.1	10.8	43.2	67.6	-
Chinese	CLUEWSC (EM)	85.4	87.9	90.9	89.9	-
	C-Eval (EM)	76.7	76.0	86.5	68.9	-
	C-SimpleQA (Correct)	55.4	58.7	68.0	40.3	-

Table 4 | Comparison between DeepSeek-R1 and other representative models.

Applications:

- ▶ Used in DeepSeek-MoE and other models
- ▶ Enhances model performance in real-world tasks
- ▶ Focus on safety and alignment in open LLMs

Byte Latent Transformer (BLT)

Byte Latent Transformer: Patches Scale Better Than Tokens

Artidoro Pagnoni, Ram Pasunuru[†], Pedro Rodriguez[†], John Nguyen[†], Benjamin Muller, Margaret Li^{1,◦},
Chunting Zhou[◦], Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman^{†,2,◦},
Srinivasan Iyer[†]

FAIR at Meta, ¹Paul G. Allen School of Computer Science & Engineering, University of Washington,

²University of Chicago

[†]Joint second author, [‡]Joint last author, [◦]Work done at Meta

We introduce the Byte Latent Transformer (BLT), a new byte-level LLM architecture that, for the first time, matches tokenization-based LLM performance at scale with significant improvements in inference efficiency and robustness. BLT encodes bytes into dynamically sized patches, which serve as the primary units of computation. Patches are segmented based on the entropy of the next byte, allocating more compute and model capacity where increased data complexity demands it. We present the first FLOP controlled scaling study of byte-level models up to 8B parameters and 4T training bytes. Our results demonstrate the feasibility of scaling models trained on raw bytes without a fixed vocabulary. Both training and inference efficiency improve due to dynamically selecting long patches when data is predictable, along with qualitative improvements on reasoning and long tail generalization. Overall, for fixed inference costs, BLT shows significantly better scaling than tokenization-based models, by simultaneously growing both patch and model size.

Date: December 16, 2024

Correspondence: artidoro at cs.washington.edu, sviyer at meta.com

Code: <https://github.com/facebookresearch/blt>



Byte Latent Transformer (BLT) (cont.)

BLT is a compression-based approach for efficient LLMs:

- ▶ Compress input text into a smaller latent space
 - ▶ Run transformer on latent tokens
 - ▶ Decode with a decoder-only model
-
- ✓ Reduces memory & compute
 - ✓ Maintains quality (BLEU, perplexity)
 - ✓ Works well with long context (128k+)

Related: VQ-VAE, tokenizer-free models, compression transformers

Meta AI (2024) <https://ai.meta.com/research/publications/>

[byte-latent-transformers-compression-based-efficiency-for-byte-level-langs/](#)

Byte Latent Transformer (BLT) (cont.)

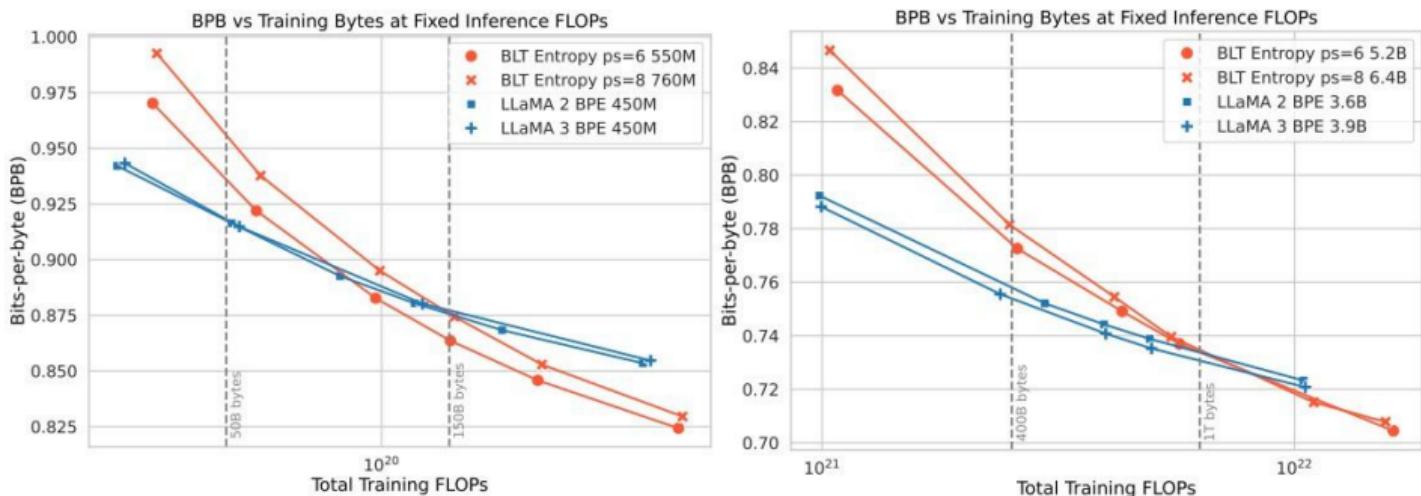


Figure 1 Scaling trends for fixed inference FLOP models (fully) trained with varying training budgets. In token-based models, a fixed inference budget determines the model size. In contrast, the BLT architecture provides a new scaling axis allowing simultaneous increases in model and patch size while keeping the same training and inference budget. BLT patch-size (ps) 6 and 8 models quickly overtake scaling trends of BPE Llama 2 and 3. Moving to the larger inference budget makes the larger patch size 8 model more desirable sooner. Both BPE compute-optimal point and crossover point are indicated with vertical lines.

Byte Latent Transformer (BLT) (cont.)

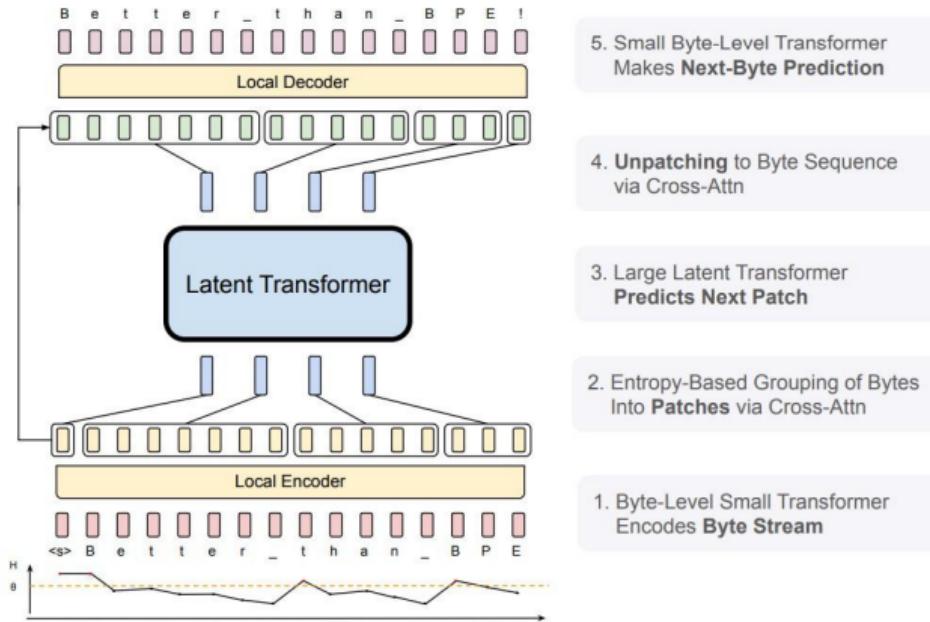


Figure 2 BLT comprises three modules, a lightweight *Local Encoder* that encodes input bytes into patch representations, a computationally expensive Latent Transformer over patch representations, and a lightweight *Local Decoder* to decode the next patch of bytes. BLT incorporates byte n -gram embeddings and a cross-attention mechanism to maximize information flow between the Latent Transformer and the byte-level modules (Figure 5). Unlike fixed-vocabulary tokenization, BLT dynamically groups bytes into patches preserving access to the byte-level information.

Byte Latent Transformer (BLT)

(cont.)

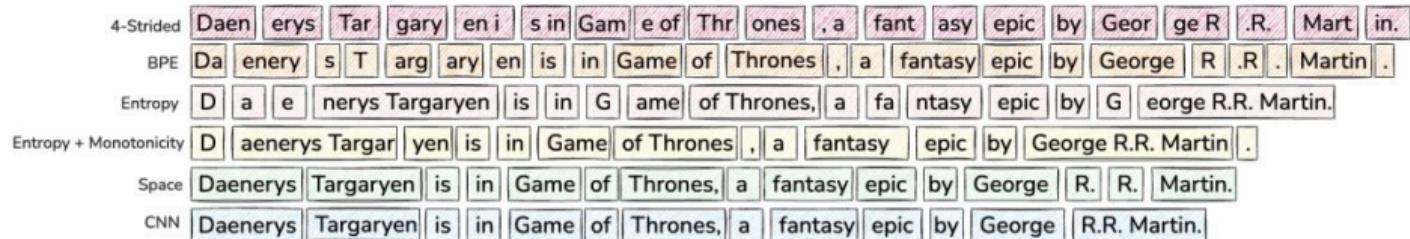


Figure 3 Patching schemes group bytes in different ways, each leading to a different number of resulting patches. Since each patch is processed using a large transformer step, the number of patches directly determines the bulk of the compute expended in terms of FLOPS. These schemes group bytes into patches by (a) striding every four bytes (§2.1) as in MegaByte (Yu et al., 2023), (b) tokenizing with Byte-Pair Encoding (BPE), in this case the Llama-3 (Dubey et al., 2024) tokenizer, (c & d) entropy-based patching as in this work (§2.3), (e) patching on space-bytes (Slagle, 2024), (f) and patching on entropy using a small CNN byte-level model with 2-byte context.

Byte Latent Transformer (BLT)

(cont.)

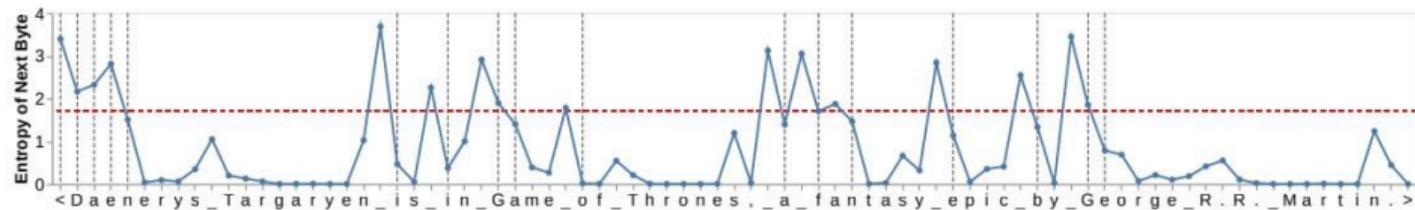


Figure 4 This figure plots the entropy $H(x_i)$ of each byte in “Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin.” with spaces shown as underscores. Patches end when $H(x_i)$ exceeds the global threshold θ_g , shown as a red horizontal line. The start of new patches are shown with vertical gray lines. For example, the entropies of “G” and “e” in “George R.R. Martin” exceed θ_g , so “G” is the start of a single byte patch and “e” of a larger patch extending to the end of the named entity as the entropy $H(x_i)$ stays low, resulting in no additional patches.

Space Patching

Space Patching (Slagle, 2024) improves over strided patching by creating new patches after any space-like byte, which often aligns with word boundaries in many languages.

- ▶ Allocates a latent transformer step for every word.
- ▶ Ensures consistent patching of words across sequences.
- ▶ Focuses compute (flops) on hard predictions, typically following spaces.

Example: Predicting the first byte of the answer to “Who composed the Magic Flute?” is harder than predicting subsequent bytes in “Mozart”.

- ▶ The first character narrows down possible completions, making the rest easier to predict.

Limitations:

- ▶ Not suitable for all languages or domains.
- ▶ Cannot vary patch size.

Next: A new patching method that leverages the difficulty of predicting initial bytes in words and enables control over patch size.

Entropy Patching: Using Next-Byte Entropies from a Small Byte LM

- ▶ Instead of rule-based heuristics (e.g., whitespace), entropy patching uses a data-driven approach to identify high-uncertainty next-byte predictions.
- ▶ Train a small byte-level auto-regressive LM on BLT training data.
- ▶ Compute the byte entropy $H(x_i) = - \sum_{v \in V} p_e(x_i = v | x_{<i}) \log p_e(x_i = v | x_{<i})$ (4)

Entropy Patching (cont.)

- ▶ Two methods for identifying patch boundaries:
 1. **Global Constraint:** $H(x_t) > \theta_g$ (entropy above a global threshold)
 2. **Approx. Monotonic Constraint:** $H(x_t) - H(x_{t-1}) > \theta_r$ (entropy rises relative to previous byte)
- ▶ The second method finds points that break approximate monotonically decreasing entropy within a patch.
- ▶ Patch boundaries are identified during lightweight preprocessing at dataloading.
- ▶ Differs from Nawrot et al. (2023), which trains a classifier to predict entropy-based patch boundaries.

In experiments: Both methods are compared for distinguishing between low

Encoder Multi-Headed Cross-Attention:

- ▶ Follows Perceiver input cross-attention (Jaegle et al., 2021), but latent representations correspond to variable patch representations (not fixed set).
- ▶ Each patch p_j attends only to its own bytes.
- ▶ Query vector for each patch is initialized by pooling byte representations for p_j , then projecting via $EC \in \mathbb{R}^{h_E \times (h_E \times U_E)}$, where U_E is the number of encoder cross-attention heads.

Encoder Multi-Headed Cross-Attention (cont.)

Formulation

:

$$P_{0,j} = EC(f_{\text{bytes}}(p_j)) \quad (\text{Pooling function } f \text{ over bytes in patch } p_j) \quad (5)$$

$$P_I = P_{I-1} + W_o \left[\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right] \quad (6)$$

$$Q_j = W_q(P_{I-1,j}) \quad (7)$$

$$K_i = W_k(h_{I-1,i}) \quad (8)$$

$$V_i = W_v(h_{I-1,i}) \quad (9)$$

$$h_I = \text{Encoder-Transformer-Layer}_I(h_{I-1}) \quad (10)$$

$P \in \mathbb{R}^{n_p \times h_G}$: n_p patch representations for the global model.

W_q , W_k , W_v , W_o : learned projection matrices.

Encoder Multi-Headed Cross-Attention (cont.)

- ▶ Patch representations are initialized by pooling byte embeddings e_i for each patch p_j .

Key Difference: Latent representations are variable and correspond to patch boundaries, not a fixed latent set.

Encoder Multi-Headed Cross-Attention (cont.)

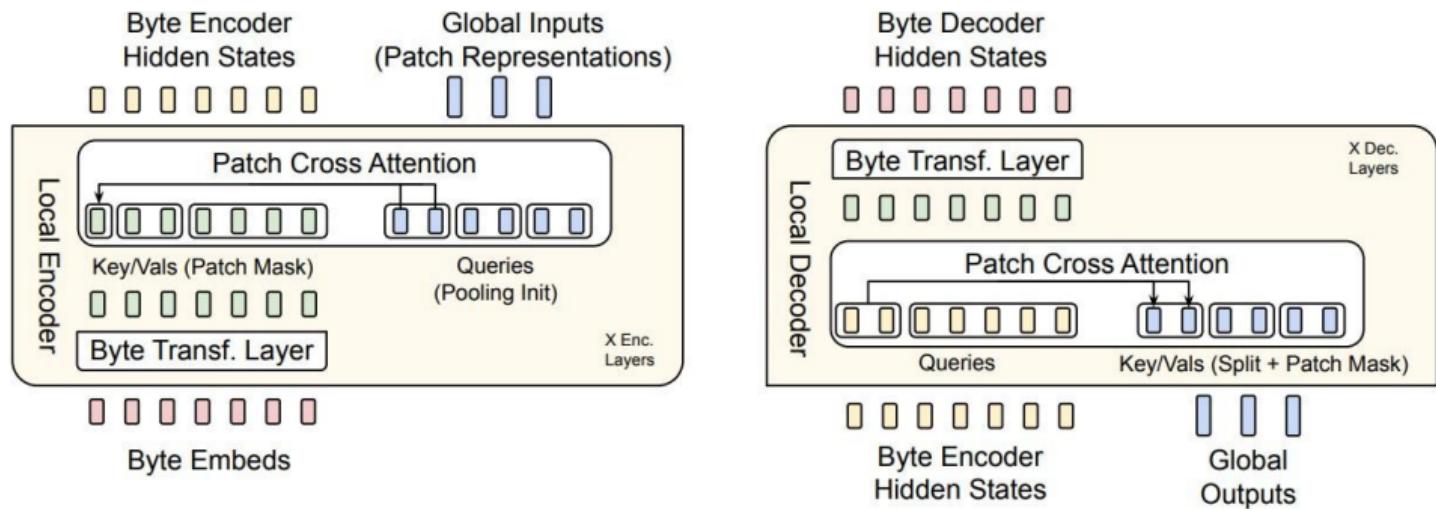


Figure 5 The local encoder uses a cross-attention block with patch representations as queries, and byte representations as keys/values to encode byte representations into patch representations. The local decoder uses a similar block but with the roles reversed i.e. byte representations are now the queries and patch representations are the keys/values. Here we use Cross-Attn $k = 2$.

Decoder Multi-Headed Cross-Attention:

- ▶ In the decoder, the roles of queries and key/values are swapped compared to the encoder cross-attention.
- ▶ Byte representations act as queries, while patch representations serve as keys and values.
- ▶ Initial byte representations for cross-attention are set as the byte embeddings from the last encoder layer, h^E .

Decoder Multi-Headed Cross-Attention (cont.)

Formulation

:

$$D_0 = h^E \quad (11)$$

$$B_I = D_{I-1} + W_o \left[\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right] \quad (12)$$

$$Q_i = W_q(d_{I-1,i}) \quad (13)$$

$$K_j = W_k(DC(o_j)) \quad (14)$$

$$V_j = W_v(DC(o_j)) \quad (15)$$

$$D_I = \text{Decoder-Transformer-Layer}_I(B_I) \quad (16)$$

- ▶ DC is a linear transformation and split operation applied to the final patch representations o_j from the global model.
- ▶ W_q, W_k, W_v, W_o are learned projection matrices.

Decoder Multi-Headed Cross-Attention (cont.)



- ▶ $B \in \mathbb{R}^{h_D \times n_b}$, where n_b is the number of output bytes.
- ▶ Multiple attention heads, pre-LayerNorm, no positional embeddings, and residual connections are used.

Summary: Decoder cross-attention enables each output byte to attend to patch-level latent representations, facilitating efficient and flexible decoding.

BLT Performance:

- ▶ BLT achieves state-of-the-art performance on long-context benchmarks.
- ▶ Outperforms existing LLMs in terms of perplexity and BLEU scores.
- ▶ Efficiently handles long sequences (128k+ tokens) with reduced memory and compute requirements.

BLT Performance (cont.)

Language	Language → English		English → Language	
	Llama 3	BLT	Llama 3	BLT
Arabic	22.3	24.6	10.4	8.8
German	41.3	42.0	29.8	31.2
Hindi	20.7	20.9	7.8	7.2
Italian	34.0	33.9	24.4	26.2
Vietnamese	31.2	31.0	28.4	23.7
Thai	17.9	18.1	10.5	7.7
Armenian	1.7	6.3	0.6	0.9
Amharic	1.3	3.1	0.4	0.5
Assamese	2.7	5.4	0.8	1.6
Bengali	4.7	12.7	1.7	4.1
Bosnian	36.0	37.3	16.9	19.6
Cebuano	18.2	20.6	5.8	9.1
Georgian	1.7	7.4	1.0	2.5
Gujarati	2.0	5.8	1.0	2.2
Hausa	5.75	5.9	1.2	1.3
Icelandic	16.1	17.9	4.8	5.3
Kannada	1.6	3.9	0.7	1.7
Kazakh	5.6	7.0	1.0	2.6
Kabuverdianu	20.3	20.9	5.1	6.8
Khmer	4.4	9.5	0.8	0.8
Kyrgyz	4.6	5.1	0.9	2.0
Malayalam	1.8	3.5	0.7	1.4
Odia	1.6	2.7	0.8	1.1
Somali	5.0	5.0	1.1	1.4
Swahili	10.1	12.0	1.4	2.3
Urdu	9.3	9.5	2.0	1.4
Zulu	4.7	5.0	0.6	0.5
Overall Average	12.1	14.0	5.9	6.4

Table 4 Performance of 8B BLT and 8B Llama 3 trained for 1T tokens on translating into and from six widely-used languages and twenty one lower resource languages with various scripts from the FLORES-101 benchmark (Goyal et al., 2022).

BLT Performance (cont.)

Task	Prompt	Llama 3	BLT
Substitute Word	Question: Substitute " and " with " internet " in " She went to the kitchen and saw two cereals. ". Answer:	She went to the kitchen and saw two cereals.	She went to the kitchen internet saw two cereals.
Swap Char	Question: Swap " h " and " a " in " that ". Answer:	that	taht
Substitute Char	Question: Substitute " a " with " m " in " page ". Answer:	-	pmge
Semantic Similarity	Question: More semantically related to " are ": " seem ", " acre ". Answer:	acre	seem
Orthographic Similarity	Question: Closer in Levenshtein distance to " time ": " timber ", " period ". Answer:	period	timber
Insert Char	Question: Add an " z " after every " n " in " not ". Answer:	znotz	nzot

Figure 7 Output responses from Llama 3 and BLT models for various tasks from CUTE benchmark. BLT model performs better on sequence manipulation tasks compared to the tokenizer-based Llama 3 model. Note that few-shot examples are not shown in the above prompts to maintain clarity.

Large Language Diffusion Models (LLDMs)

Large Language Diffusion Models

Shen Nie^{1*†} Fengqi Zhu^{1*†} Zebin You^{1†} Xiaolu Zhang^{2‡} Jingyang Ou¹ Jun Hu^{2‡} Jun Zhou²
Yankai Lin^{1‡} Ji-Rong Wen¹ Chongxuan Li^{1‡¶}

Abstract

Autoregressive models (ARMs) are widely regarded as the cornerstone of large language models (LLMs). We challenge this notion by introducing **LLaDA**, a diffusion model trained from scratch under the pre-training and supervised fine-tuning (SFT) paradigm. LLaDA models distributions through a forward data masking process and a reverse process, parameterized by a vanilla Transformer to predict masked tokens. By optimizing a likelihood bound, it provides a principled generative approach for probabilistic inference. Across extensive benchmarks, LLaDA demonstrates strong *scalability*, outperforming our self-constructed ARM baselines. Remarkably, LLaDA 8B is competitive with strong LLMs like LLaMA3 8B in *in-context learning* and, after SFT, exhibits impressive *instruction-following* abilities in case studies such as multi-turn dialogue. Moreover, LLaDA addresses the reversal curse, surpassing GPT-4o in a reversal poem completion task. Our findings establish diffusion models as a viable and promising alternative to ARMs, challenging the assumption that key LLM capabilities discussed above are inherently tied to ARMs. Project page and codes: <https://ml-gsai.github.io/LLaDA-demo/>.

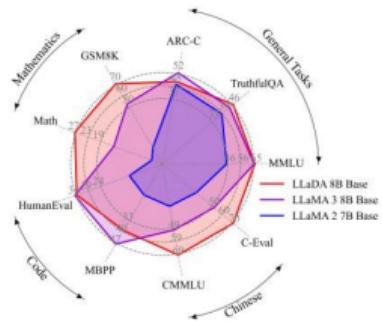


Figure 1. **Zero/Few-Shot Benchmarks.** We scale LLaDA to an unprecedented size of 8B parameters from scratch, achieving competitive performance with strong LLMs (Dubey et al., 2024).

distribution $p_{\text{data}}(\cdot)$ by optimizing a model distribution $p_\theta(\cdot)$ through maximum likelihood estimation, or equivalently KL divergence minimization between the two distributions:

$$\max_{\theta} \mathbb{E}_{p_{\text{data}}(x)} \log p_\theta(x) \Leftrightarrow \min_{\theta} \underbrace{\text{KL}(p_{\text{data}}(x) || p_\theta(x))}_{\text{Generative modeling principles}}. \quad (1)$$

- ▶ **Emerging trend:** Use diffusion instead of autoregression for text generation

Key Idea:

- ▶ Generate latent text representation over diffusion steps
- ▶ Use decoder to map latent → text

Benefits:

- ▶ Better global coherence
- ▶ Parallel generation
- ▶ Less repetition, hallucination



Related work:

- ▶ Diffusion-LM (Li et al.)
- ▶ Masked-Diffusion Transformers
- ▶ UniDiffuser (Vision + Text)

Note: Still experimental, but promising for the future of text generation

LLDMs: Diffusion for Text Generation (cont.)

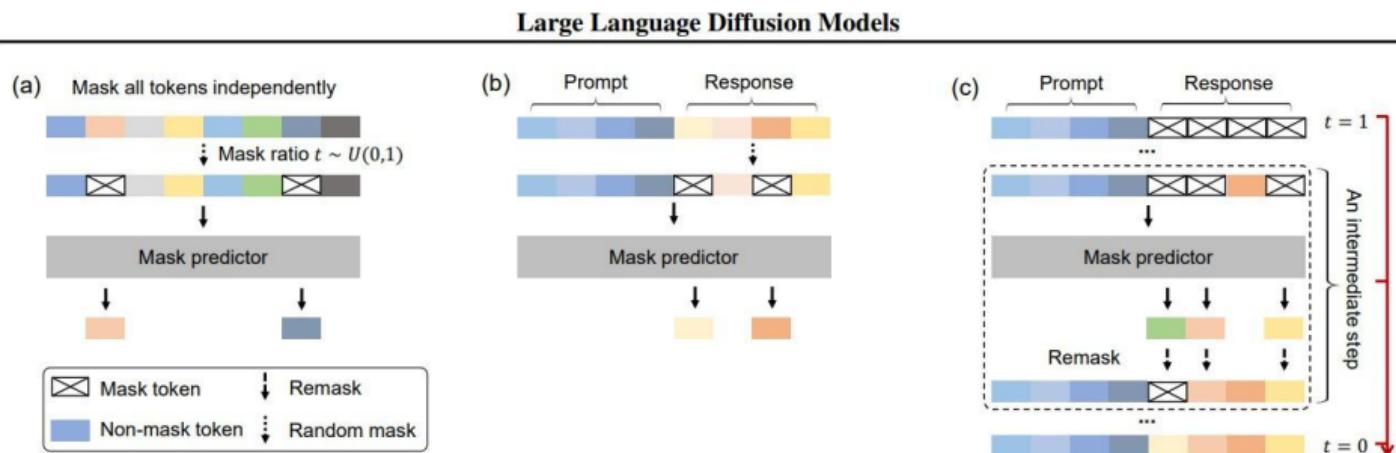


Figure 2. A Conceptual Overview of LLaDA. (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked) to $t = 0$ (unmasked), predicting all masks simultaneously at each step with flexible remask strategies.

LLDMs: Diffusion for Text Generation (cont.)

Large Language Diffusion Models

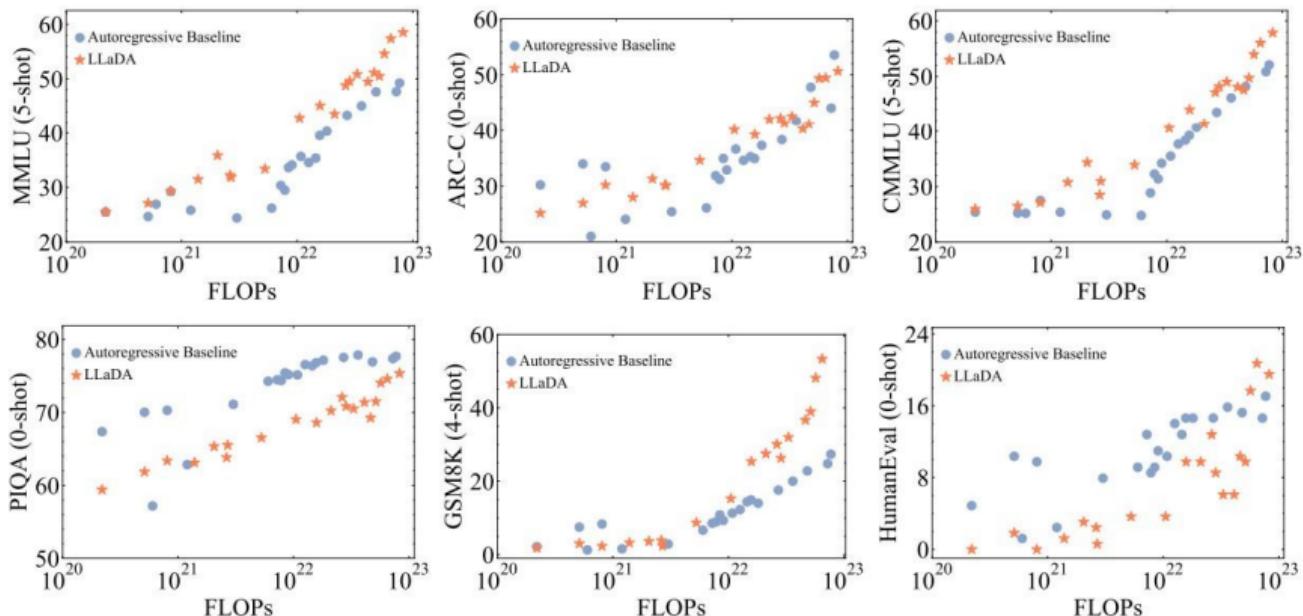


Figure 3. Scalability of LLaDA. We evaluate the performance of LLaDA and our ARM baselines trained on the same data across increasing computational FLOPs. LLaDA exhibits strong scalability, matching the overall performance of ARMs on six tasks.

LLDMs: Diffusion for Text Generation (cont.)

Large Language Diffusion Models

Table 1. Benchmark Results of Pre-trained LLMs. * indicates that LLaDA 8B Base, LLaMA2 7B Base, and LLaMA3 8B Base are evaluated under the same protocol, detailed in Appendix B.5. Results indicated by [†] and [¶] are sourced from Chu et al. (2024); Yang et al. (2024) and Bi et al. (2024) respectively. The numbers in parentheses represent the number of shots used for evaluation. “-” indicates unknown data.

	LLaDA 8B*	LLaMA3 8B*	LLaMA2 7B*	Qwen2 7B [†]	Qwen2.5 7B [†]	Mistral 7B [†]	Deepseek 7B [¶]
Model	Diffusion	AR	AR	AR	AR	AR	AR
Training tokens	2.3T	15T	2T	7T	18T	-	2T
General Tasks							
MMLU	65.9 (5)	65.4 (5)	45.9 (5)	70.3 (5)	74.2 (5)	64.2 (5)	48.2 (5)
BBH	49.8 (3)	57.6 (3)	37.3 (3)	62.3 (3)	70.4 (3)	56.1 (3)	39.5 (3)
ARC-C	47.9 (0)	53.1 (0)	46.3 (0)	60.6 (25)	63.7 (25)	60.0 (25)	48.1 (0)
Hellaswag	72.5 (0)	79.1 (0)	76.0 (0)	80.7 (10)	80.2 (10)	83.3 (10)	75.4 (0)
TruthfulQA	46.4 (0)	44.0 (0)	39.0 (0)	54.2 (0)	56.4 (0)	42.2 (0)	-
Winogrande	74.8 (5)	77.3 (5)	72.5 (5)	77.0 (5)	75.9 (5)	78.4 (5)	70.5 (0)
PIQA	74.4 (0)	80.6 (0)	79.1 (0)	-	-	-	79.2 (0)
Mathematics & Science							
GSM8K	70.7 (4)	53.1 (4)	14.3 (4)	80.2 (4)	85.4 (4)	36.2 (4)	17.4 (8)
Math	27.3 (4)	15.1 (4)	3.2 (4)	43.5 (4)	49.8 (4)	10.2 (4)	6.0 (4)
GPQA	26.1 (5)	25.9 (5)	25.7 (5)	30.8 (5)	36.4 (5)	24.7 (5)	-
Code							
HumanEval	33.5 (0)	34.2 (0)	12.8 (0)	51.2 (0)	57.9 (0)	29.3 (0)	26.2 (0)
HumanEval-FIM	73.8 (2)	73.3 (2)	26.9 (2)	-	-	-	-
MBPP	38.2 (4)	47.4 (4)	18.4 (4)	64.2 (0)	74.9 (0)	51.1 (0)	39.0 (3)
Chinese							
CMMLU	69.9 (5)	50.7 (5)	32.5 (5)	83.9 (5)	-	-	47.2 (5)
C-Eval	70.5 (5)	51.7 (5)	34.0 (5)	83.2 (5)	-	-	45.0 (5)

LLMs: Diffusion for Text Generation (cont.)

Table 4. Visualization of the Sampling Process and a Generated Multi-round Dialogue. In the response of LLaDA, darker colors indicate tokens predicted in the later stages of sampling, while lighter colors correspond to earlier predictions.

Sampling Process	
User	Lily can run 12 kilometers per hour for 4 hours. After that, she runs 6 kilometers per hour. How many kilometers can she run in 8 hours?
LLaDA	
Multi-round Dialogue Case	
User	Could you please share with me what the initial two lines of the renowned poem 'The Road Not Taken'?
LLaDA	Certainly! The initial two lines of the renowned poem "The Road Not Taken" by Robert Frost are: "Two roads diverged in a yellow wood, And sorry I could not travel both." These lines set the stage for the poem, introducing the idea of a speaker facing a choice between two paths.
User	Please help me translate into Chinese.
LLaDA	Sure! The first two lines of "The Road Not Taken" by Robert Frost can be translated into Chinese as: "两条路分岔在黄色的树林中, 遗憾我不能同时走"
User	Please translate into German.
LLaDA	Sure! The first two lines of "The Road Not Taken" by Robert Frost can be translated into German as: "Zwei Wege trennten sich im gelben Wald, und leider konnte ich nicht beide tun."
User	Please also write a poem about life choices. I require 5 sentences, and the first word of each sentence starts with C as the first letter.
LLaDA	Certainly! Here's a poem about life choices, with each sentence starting with the letter "C": Choices cascade through life's journey, Creating connections with each decision, Careful consideration leads to better paths, Courageous choices shape our destiny, Choosing wisely guides us through the years.

Algorithm 1 Pre-training of LLaDA

Require: mask predictor p_θ , data distribution p_{data}

- ```

1: repeat
2: $x_0 \sim p_{\text{data}}, t \sim \text{U}(0, 1)$ # with a probability of 1%, the sequence length of x_0 follows $\text{U}[1, 4096]$
3: $x_t \sim q_{t|0}(x_t|x_0)$ # $q_{t|0}$ is defined in Eq. (7)
4: Calculate $\mathcal{L} = -\frac{1}{t+L} \sum_{i=1}^L \mathbf{1}[x_t^i = \mathbf{M}] \log p_\theta(x_0^i|x_t)$ # L is the sequence length of x_0
5: Calculate $\nabla_\theta \mathcal{L}$ and run optimizer.
6: until Converged
7: Return p_θ

```

## LLMs: Algorithm Overview (cont.)

**Algorithm 2** Supervised Fine-Tuning of LLaDA

**Require:** mask predictor  $p_\theta$ , pair data distribution  $p_{\text{data}}$

- ```

1: repeat
2:    $p_0, r_0 \sim p_{\text{data}}, t \sim U(0, 1)$  # please refer to Appendix B.1 for details on the SFT data processing.
3:    $r_t \sim q_{t|0}(r_t|r_0)$  #  $q_{t|0}$  is defined in Eq. (7)
4:   Calculate  $\mathcal{L} = -\frac{1}{t+L'} \sum_{i=1}^{L'} \mathbf{1}[r_t^i = M] \log p_\theta(r_0^i | p_0, r_t)$  #  $L'$  is the sequence length of  $r_0$ 
5:   Calculate  $\nabla_\theta \mathcal{L}$  and run optimizer.
6: until Converged
7: Return  $p_\theta$ 

```

LLDMs: Algorithm Overview (cont.)

Algorithm 3 Conditional Log-likelihood Evaluation of LLaDA

Require: mask predictor p_θ , prompt p_0 , response r_0 , the number of Monte Carlo estimations n_{mc}

- 1: $\text{log_likelihood} = 0$
 - 2: **for** $i \leftarrow 1$ to n_{mc} **do**
 - 3: $l \in \{1, 2, \dots, L\}$ # L is the sequence length of r_0
 - 4: Obtain r_l by uniformly sampling l tokens from r_0 without replacement for masking
 - 5: $\text{log_likelihood} = \text{log_likelihood} + \frac{L}{l} \sum_{i=1}^L \mathbf{1}[r_l^i = M] \log p_\theta(r_0^i | p_0, r_l)$
 - 6: **end for**
 - 7: $\text{log_likelihood} = \text{log_likelihood}/n_{mc}$
 - 8: **Return** log_likelihood
-

Segment Everything Everywhere All at Once (SEEM)

Segment Everything Everywhere All at Once (SEEM)

Segment Everything Everywhere All at Once

Xueyan Zou^{*1§}, Jianwei Yang^{*2†}, Hao Zhang^{*#}, Feng Li^{*#}, Linjie Li[†], Jianfeng Gao^{¶†}, Yong Jae Lee^{¶1§}

[§] University of Wisconsin-Madison [†] Microsoft Research at Redmond [#] HKUST [¶] Microsoft Cloud & AI

*Equal Contribution ¶ Equal Advisory Contribution 1. Main Technical Contribution 2. Project Lead

{xueyan,yongjaelee}@cs.wisc.edu {jianwyan,jfgao,linjli}@microsoft.com {hzhangcx,fliay}@connect.ust.edu

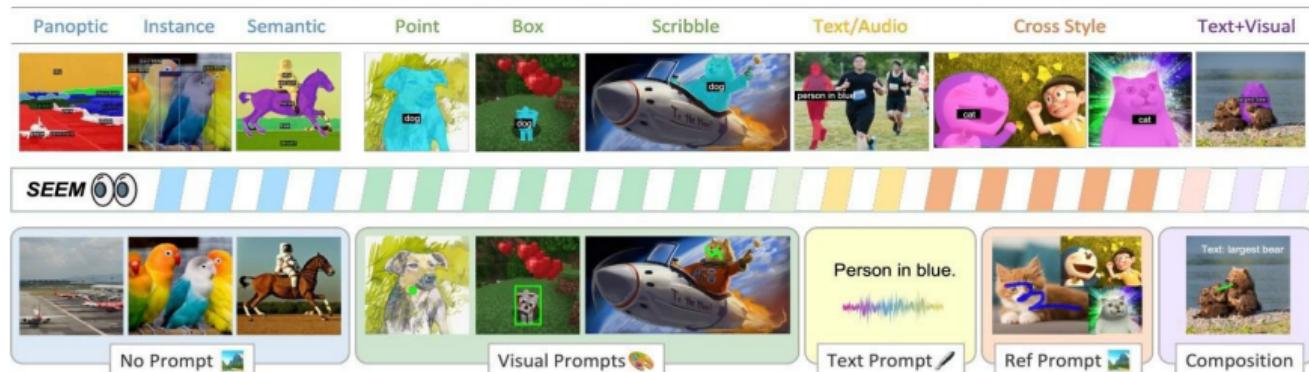


Figure 1: Our model, *SEEM*, can perform any segmentation task. When no prompt is given, it can do classic segmentation tasks such as semantic, instance, and panoptic segmentation in open-set scenarios. *SEEM* also supports various types of prompts and any combination of these prompts.

Segment Everything Everywhere All at Once (SEEM) (cont.)

Proposed by Meta AI (2023)

SEEM is a promptable panoptic segmentation model designed to segment any region, object, or stuff in an image using various input modalities such as text, points, boxes, or masks.

- ▶ **Unified Segmentation:** Supports instance, semantic, and panoptic segmentation.
- ▶ **Versatility:** Works with any image, any resolution, and enables zero-shot segmentation.
- ▶ **Multimodal Prompts:** Accepts prompts in multiple forms, including text, points, boxes, and masks.
- ▶ **Video and Open Vocabulary:** Capable of segmenting video frames and handling open vocabulary tasks.
- ▶ **Foundation for Interactive Systems:** Serves as a foundation for

Segment Everything Everywhere All at Once (SEEM) (cont.)

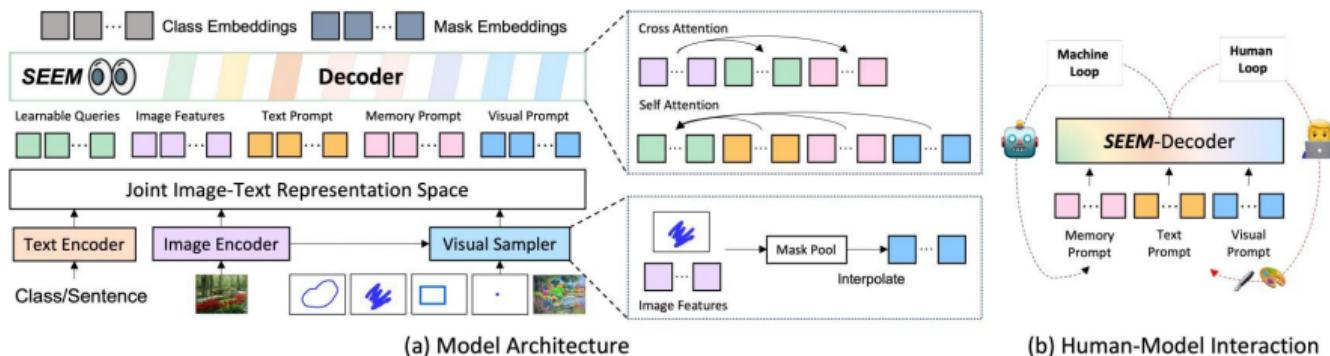


Figure 2: Overview of *SEEM*-Decoder. (a) *SEEM* encodes image, text, and human inputs into *joint visual-semantic space* as queries, features, and prompts, and then decodes queries to class and mask embeddings. (b) With the benefit of *SEEM* decoder, the machine loop enables memorizing history mask information, and the human loop provides new corrections to the next round.

Segment Everything Everywhere All at Once (SEEM) (cont.)

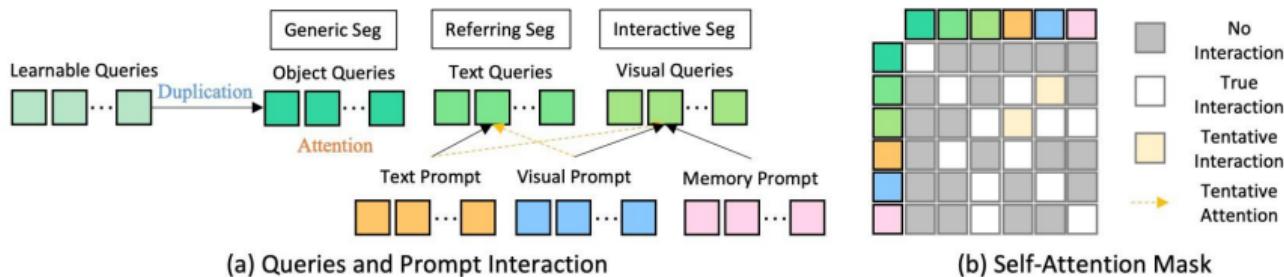


Figure 3: Queries and prompt interaction during training and evaluation. (a) Learnable queries are duplicated as object, grounding, and visual queries with the same set of weights for each task. (b) Attention mask between any two kinds of tokens (denoted as qpm in Algorithm. 1). Tentative means the interaction is not trained but able to do inference without any modification.

Segment Everything Everywhere All at Once (SEEM) (cont.)

Algorithm 1: Pseudo code for SEEM.

```
# Inputs: Image(img) [B, 3, H, W]; Pos_Mask(pm), Neg_Mask(nm) [B, 1, H, W]; Text(txt) [abc...];
# Variables: Learnable Queries(Qh); Attention Masks between Q and P(qpm)
# Functions: Img_Encoder(), Text_Encoder(), Visual_Sampler(), feature_attn(), prompt_attn(), output()

1 def init():
2     Qo, Qt, Qv = Qh.copy(); # Initialize object, text and visual queries.
3     Fv, Pt = Img_Encoder(img), Text_Encoder(txt); # Fv and Pt denote image feature, text
4         prompt.
5     Pv = Visual_Sampler(Fv, pm, nm); # Sample visual prompt from image feature, pos/neg
        mask.
6
7     def SEEM_Decoder(Fv, Qo, Qt, Qv, Pv, Pt, Pm):
8         Qo, Qt, Qv = feature_attn(Fv, Qo, Qt, Qv); # Cross attend queries with image features.
9         Qo, Qt, Qv = prompt_attn(qpm, Qo, Qt, Qv, Pv, Pt, Pm); # Self attend queries and prompts.
10        Om, Oc, Pm = output(Fv, Qo, Qt, Qv); # Compute mask and class outputs.
11
12    def forward(img, pm, nm, txt):
13        Fv, Qo, Qt, Qv, Pv, Pt = init(); Pm = None; # Initialize variables.
14        for i in range(max_iter):
15            Om, Oc, Pm = SEEM_Decoder(Fv, Qo, Qt, Qv, Pv, Pt, Pm)
```



Figure 4: Click/scribble-based segmentation. *SEEM* supports arbitrary formats of clicks or scribbles by users. Moreover, it simultaneously gives the semantic label for the segmented mask, which is not possible in SAM [36].

SEEM - Results (cont.)



Figure 5: Text to mask or text referring segmentation. The referred text is shown on the masks. *SEEM* adapts to various types of input images in the domain of cartoons, movies, and games.

SEEM - Results (cont.)

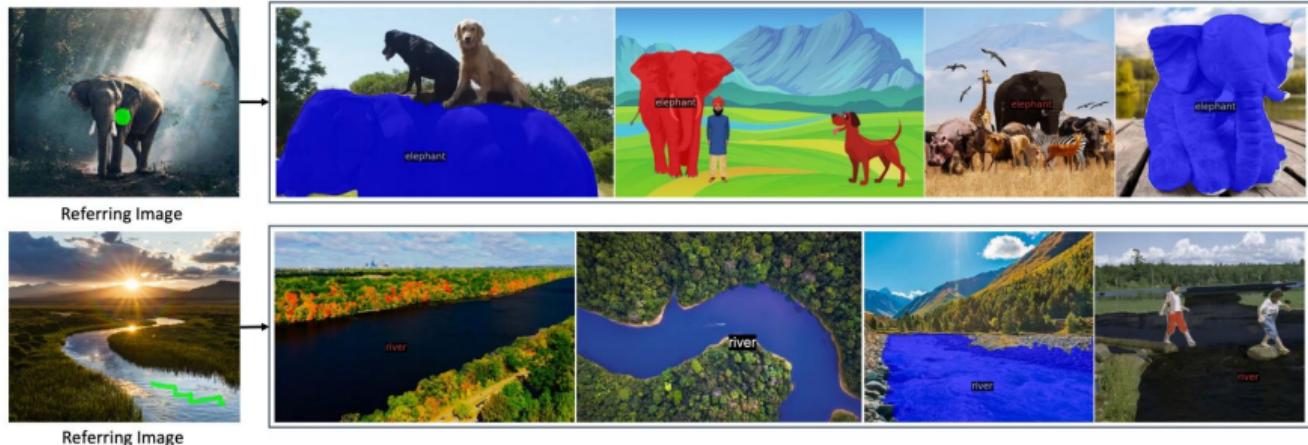


Figure 6: **Zero-shot** visual referring segmentation with *SEEM*. Given a referring image with simple spatial hints, *SEEM* can segment the regions which are semantically similar in different target images.

SEEM - Results (cont.)

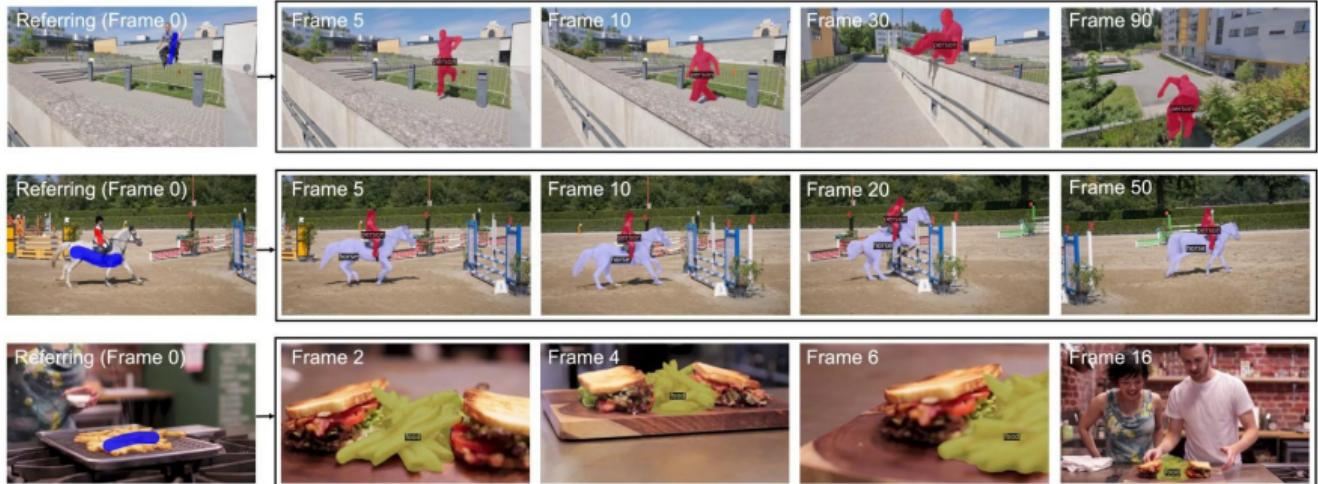


Figure 7: **Zero-shot** video object segmentation using the first frame plus one stroke. From top to bottom, the videos are “parkour” and “horsejump-low” from DAVIS [73], and video 101 from YouCook2 [74]. *SEEM* precisely segments referred objects even with significant appearance changes caused by blurring or intensive deformations.

Summar y

Comparative Overview



Advancement	Domain	Key Benefit
SEEM	Vision	Promptable segmentation
MoE	LLM/NLP	Sparse, scalable training
FlashAttention-2	LLM Efficiency	Fast long-context inference
LLaMA 3	NLP	Open GPT-4 level models
DeepSeek + GRPO	NLP	Safe, aligned LLM training
BLT	NLP/LLM	Token compression efficiency
LLDMs	NLP	Diffusion-based generation

Future Directions

- ▶ Multimodal Foundation Models
- ▶ Efficient Training (QLoRA, MoE, FlashAttention)
- ▶ LLMs with Memory, Tools, Agents
- ▶ Transformer Alternatives (MLPs, State Space Models)
- ▶ Unified Models: Language, Vision, Audio, Action

Integration of these advances will define AI agents, robotics, and autonomous systems.

- ▶ **SEEM:** Revolutionizes image segmentation with prompt-based control.
- ▶ **MoE:** Enables sparse expert usage for massive model scaling.
- ▶ **FlashAttention-2:** Accelerates transformer inference and improves memory efficiency.
- ▶ **LLaMA 3 & DeepSeek:** Provide top-tier open-source LLMs.
- ▶ **GRPO:** Advances LLM alignment beyond PPO.
- ▶ **BLT:** Reduces input redundancy for efficient token compression.
- ▶ **LLDMs:** Introduce diffusion-based generation as a radical shift.

Reference s

- ▶ **SEEM:** <https://segment-anything.com/seem>
- ▶ **FlashAttention-2:** Dao et al., 2023 ([arXiv:2307.08691](https://arxiv.org/abs/2307.08691))
- ▶ **LoRA:** Hu et al., 2021
- ▶ **Mixture of Experts:** Fedus et al., 2021 (Switch Transformer), Lepikhin et al., 2020 (GShard)
- ▶ **LLaMA 3:** Meta AI 2024 release notes
- ▶ **DeepSeek + GRPO:** <https://github.com/deepseek-ai>
- ▶ **BLT:** Meta AI, 2024 ([arXiv:2404.11450](https://arxiv.org/abs/2404.11450))
- ▶ **LLDM:** Li et al., 2022 (Diffusion-LM), MaskGIT, UniDiffuser

References (cont.)

Resources:

- ▶ HuggingFace ModelHub & Papers
- ▶ Meta AI blog
- ▶ DeepSeek official GitHub
- ▶ Open LLM Leaderboard:
<https://huggingface.co/spaces/HuggingFaceH4/open-llm-leaderboard>

**Credit
s**

**Dr. Prashant
Aparajeya**
Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.