Efficient Transformers

October 16, 2025

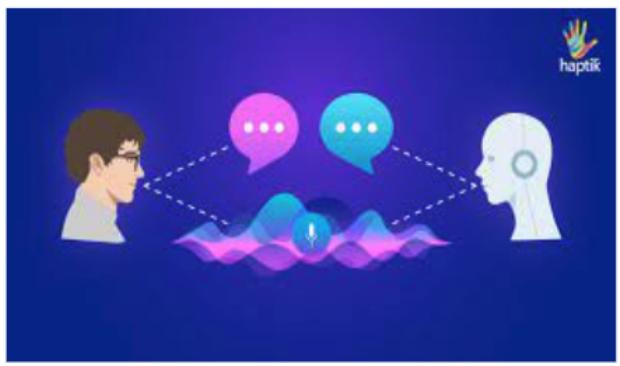
Plan

Plan

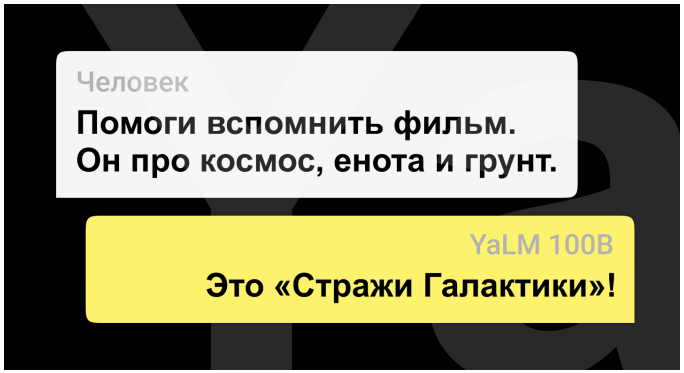
The relevance of large Transformer models

Examples of applications where Transformers language models are successfully used

- Automatic completion of phrases in email
- Summarization of documents
- Program code generation, creation of new drugs formulas
- Few-shot and zero-shot tasks.
- Example: A GPT-3 model is asked to answer the question: "How many is two plus six?" or offer to play a game. The model answer is correct.



Models, based on Transformer architecture



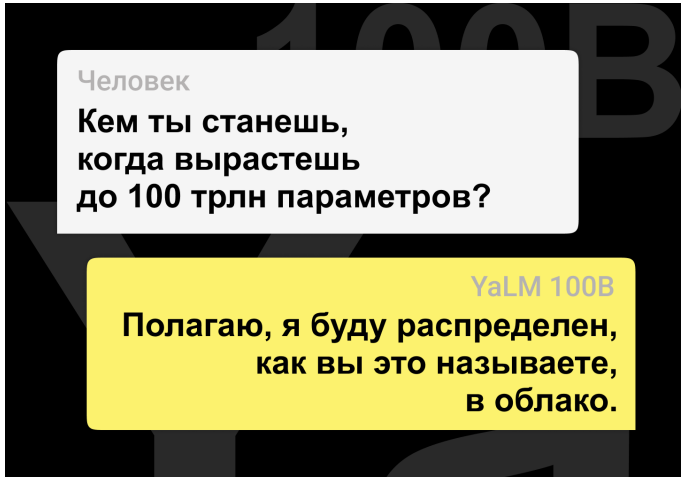


Table: Sizes of big Transformer-based models

Model	GPT-3	YaLM	mGPT	GPT-4
Number of parameters	$175 \cdot 10^9$	$100 \cdot 10^9$	$1.3 \cdot 10^9$	$100 \cdot 10^{12}$

Models, based on Transformer architecture

Training cost [1]

Актуальность снижения затрат на тренировку и выбросов в атмосферу

Энергия: 190 МВатт*час → Выбросы CO_2 в атмосферу: 85 тонн

Тренировка GPT-3

Отопление 126 домов в Дании

Поездка до Луны

[1] source: Sber-Skoltech project GreenAI

Plan

The main "bottlenecks" in Transformers model

- **Modeling: Embedding layer** (vocabulary_size $\times$ embedding_size $\approx 35 \cdot 10^3 \times 1024$)
- **Modeling: MLP** (GPT2-Medium $1024 \times 4096 \times 2 \times 24$)
- **Inference: Attention (Self-Attention) layer** (input_sequence $\times$ input_sequence, 1024×1024)

Table: Size of the different blocks in Transformers, MB

Layer-Model	GPT-2 small	GPT-2 medium	GPT-2 large
Attention	9.01	16.02	25.02
MLP	18.01	32.02	50.02

Transformers model

Goal

To reach acceptable quality under a constrained resources (on a model with fixed number of parameters)

Size reduction approaches

- **Parameters' size reduction** - in one way or another, reduces the amount of memory required for storing model parameters without change of configuration.
- **LA structures: SVD, Kronecker, Tensor/Matrix Representation** - reduces the number of parameters by representing a model layers as a products of decomposition.

Quantization: Overview

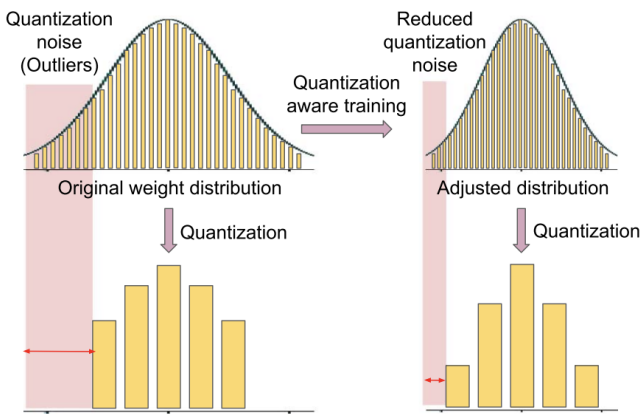
Quantization compresses the network by reducing the number of bits required to represent each weight. It can be applied to Embedding, Fully-connected layers, Attention layers (some layers are quantization friendly, some not - i didn't find a criterion).

- Fixed-point scalar quantization: change float32 weight to **8-bit** int or **4-bit** int (in initial work [2] to 1-bit or 2-bit).
- Groups quantization: split parameter matrix to blocks and replace every element of given block to one digit. Solution is found by finding the minimum of norm between real and quantized matrix [3].

[2] Quantized Neural Networks

[3] Training with Quantization Noise for Extreme Model Compression

Quantization: Overview



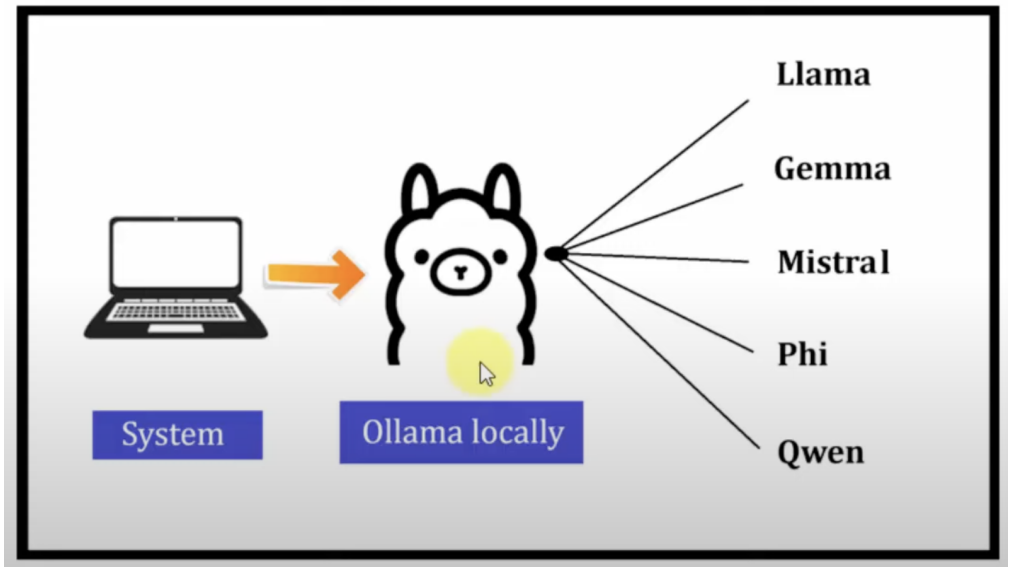
- Simple truncating leads to sizable drop in accuracy
- Tips:
  - Provide quantization while training. **Quantization Aware Training (QAT)**: Float values are rounded to mimic int4/int8 values, but all computations are still done with floating point numbers. This method usually yields higher accuracy than the post-training quantization.

Pluses: Partially easy to employ (**FP16** in PyTorch)
Drawbacks:

- The Naive methods lead to a performance decrease. For all methods, it is rather difficult to account for the effect of quantization in the resulting loss.
- Partially not so easy to employ: for 4, 2 bit needs special inference function.

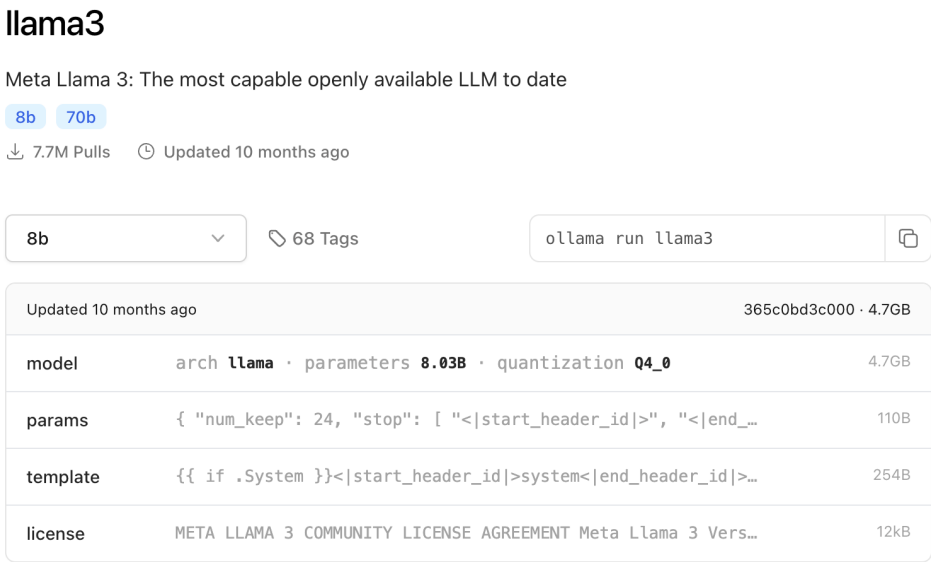
Use quantized models in practise: Ollama

- **Ollama** is an open source user-friendly platform to run LLM on your local machine.
- **Ollama** supports al Llamas, Gemma, Deepseek, Mistral, Qwen (model with strong reasoning ability), LLaVA



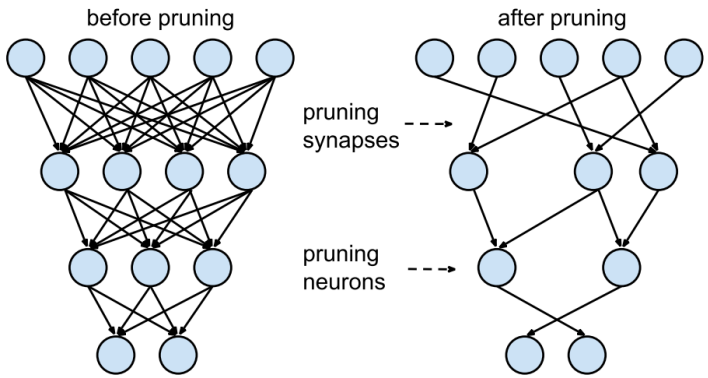
Use quantization in practise: Ollama

- By default, **Ollama** uses 4-bit quantization. To try other quantization levels, please try the other tags. The number after q represents the number of bits used for quantization (i.e. q4 means 4-bit quantization).



Pruning: Overview

The lottery ticket hypothesis



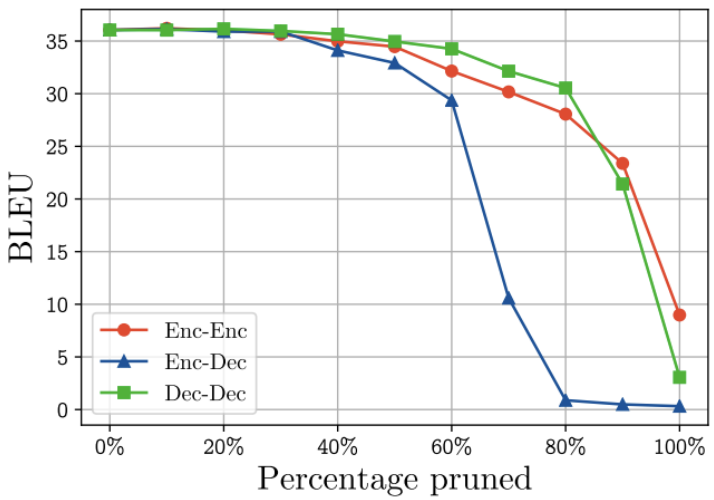
- Initialize weights
- Train
- Evaluate
- Remove weights close to zero, with gradients being close to zero
- Reset rest weights to initial state
- Train again

Pruning: Overview

Pruning types:

- Unstructured - remove weights from set of unimportant weights (Relative to gradient or other criteria)
- Structured - remove block of weights
  - **Prune L** - number of encoder blocks
  - **Prune H** - embedding size
  - **Prune A** - number of attention heads.

Prune A



We can prune up to 20 to 40% of attention heads without increasing the performance [a]. BLEU when incrementally pruning heads from each attention type in the WMT model.

[a] Are Sixteen Heads Really Better than One?

Pruning: modules selection [4]

- Method: Layer-wise relevance propagation (LRP)
- LRP describes the relative contribution of neurons at one point in a network to neurons at another. They evaluate the contribution of different heads to the Top-1 predicted logit.

	attention heads (e/d/d-e)	**BLEU** from trained	from scratch
WMT, 2.5m			
baseline	48/48/48	**29.6**	
sparse heads	14/31/30	29.62	29.47
	12/21/25	29.36	28.95
	8/13/15	29.06	28.56
	5/9/12	28.90	28.41

[4] Voita at al. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned

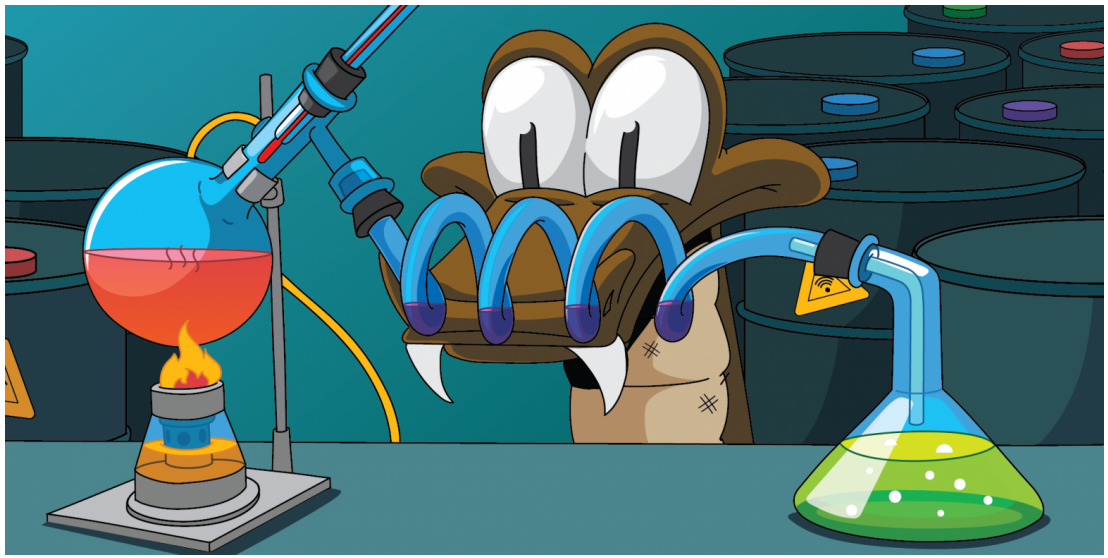
Pruning: Overview

Pros: To prune part of a model is technically easy
Drawbacks: To select the proper part for pruning is not so easy (pruning is tedious)

- High resulting performance requires manual setup of criteria and requires additional hyper-parameter (in case of sparse attention we should vary the radius of interaction and select the most effective one).
- Pruning requires more iterations to converge than general methods.

Distillation

- We have a big pre-trained language model (**Teacher**).
- This model fine-tunes on a small task i.e text classification, predicting labels for text objects.
- At the same time, the smaller model (**Student**) also is learned from scratch to predict labels w.r.t. teacher softmax outputs. It learns to imitate the teacher's answers.
- The Loss function of a common task splits into two: real prediction loss and teacher-student matching loss



Distillation

Distillation types

- **Offline distillation**: the knowledge is transferred from a pre-trained teacher model into a student model.
- **Online distillation**: both the teacher model and the student model are updated simultaneously, and the whole knowledge distillation framework is end-to-end trainable.
- **Self-distillation**: the same networks are used for the teacher and the student models, and can be regarded as a special case of online distillation.

Distillation

DistillBERT(Offline) [5]

- Teacher - BERT
- Student - LSTM-based model or smaller version of BERT (remove polling layers and remover layers on the factors of 2).
- Loss = CrossEntropyLoss(student_logits, target) + alpha · KLDivLoss(student_logits, teacher_logits)

Table: Performance of Distillation

Model name	BERT	DistillBERT
Number of parameters	$110 \cdot 10^6$	$66 \cdot 10^6$
Accuracy on IMBD	93.4	92.8

[5] Paper about DistillBERT

Distillation

TinyBERT(Offline) Distilling BERT for Natural Language Understanding

6

- Teacher - BERT
- Student - BERT with 12 heads, number of layer 4, hidden size 312.

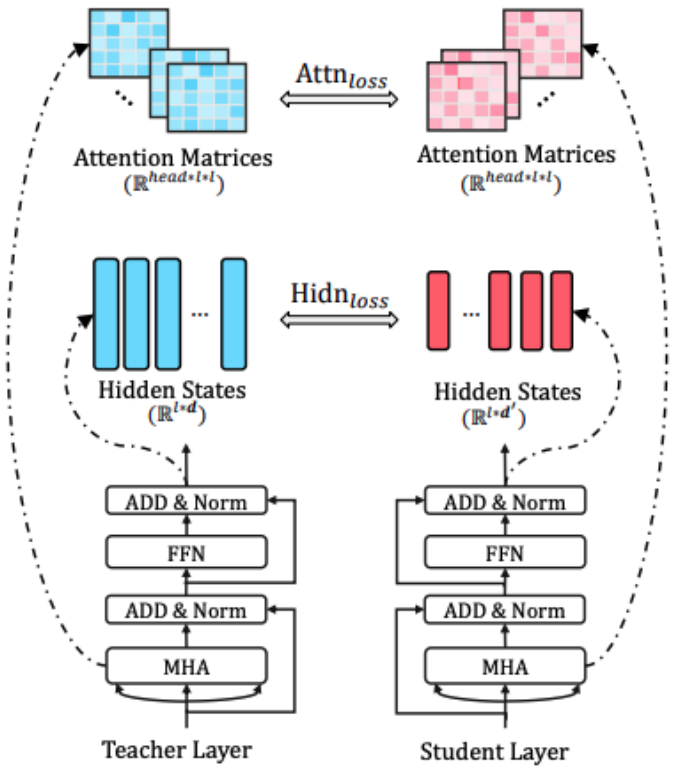
$\text{BERT}_{\text{BASE}}$ (Teacher)	109M	22.5B	1.0x	83.9/83.4	71.1	90.9	93.4	52.8	85.2	87.5	67.0	79.5
$\text{BERT}_{\text{TINY}}$	14.5M	1.2B	9.4x	75.4/74.9	66.5	84.8	87.6	19.5	77.1	83.2	62.6	70.2
$\text{BERT}_{\text{SMALL}}$	29.2M	3.4B	5.7x	77.6/77.0	68.1	86.4	89.7	27.8	77.0	83.4	61.8	72.1
BERT_4-PKD	52.2M	7.6B	3.0x	79.9/79.3	70.2	85.1	89.4	24.8	79.8	82.6	62.3	72.6
DistilBERT_4	52.2M	7.6B	3.0x	78.9/78.0	68.5	85.2	91.4	32.8	76.1	82.4	54.1	71.9
$\text{MobileBERT}_{\text{TINY}}$†	15.1M	3.1B	-	81.5/81.6	68.9	**89.5**	91.7	**46.7**	80.1	**87.9**	65.1	**77.0**
TinyBERT_4 (ours)	14.5M	1.2B	9.4x	**82.5/81.8**	**71.3**	87.7	**92.6**	44.1	**80.4**	86.4	**66.6**	**77.0**

TinyBERT4 and BERT_TINY have ($M = 4, d = 312, di = 1200$). BERT_TINY means directly pretraining a small BERT, which has the same model architecture as TinyBERT.

[6] Paper about TinyBERT

Distillation

The Transformer-layer distillation includes the attention based distillation and hidden states based distillation:



$$\mathcal{L}_{\text{layer}} = \begin{cases} \mathcal{L}_{\text{embd}}, & m = 0 \\ \mathcal{L}_{\text{hidn}} + \mathcal{L}_{\text{attn}}, & M \geq m > 0 \\ \mathcal{L}_{\text{pred}}, & m = M + 1 \end{cases}$$

- $L_{attn} = \frac{1}{h} \sum_i^h MSE(A_i^s, A_i^t)$
- $L_{hiddn} - MSE(h_s W_h, h_t)$
- $L_{emb} - MSE(e_s, e_t)$
- $L_{pred} - CE(z_t, z_s)$

Distillation

FastBERT(Self-distillation) [7]

This work uses the same model for teacher and student models:

- We have classic BERT model as encoder with h =12 layers
- On the top of every layer we have light weight **Teacher classifier** $p_t = TC(h_{L-1})$, L- number of transformer layers
- On the top of every layer we also have light weight **Student classifier** $p_{cl} = SC(h_l)$
- Loss $(p_{c0} \dots p_{cL}) = \sum_i^{L-1} KL(p_{ci}, p_t)$

[7] Paper about FastBERT

Distillation

Pros: Method can archive a good compression without drop in the performance **Drawbacks**:

- The approach gives a solution only to a particular problem and is not generalizable
- Needs accurate parameters fine tuning to obtain a good results

Plan

Long Context

Compressing the entire model is great, but what about long context?

- In modern transformers, the input sequence length can reach 32K-64K tokens for text input (RAG context, multi-turn dialogues)
- For Video processing, 1 frame can give 200-600 image tokens, so minutes of video $\rightarrow$ tens of thousands of tokens
- In this case, the attention architecture itself becomes the bottleneck:
  - **Quadratic scaling:** time memory $O(L^2)$ w.r.t. context length L
    Each head computes QK of size $L \times L \rightarrow$ quadratic FLOPs memory
  - **KV cache (inference)**: Per layer: $\approx 2 \times L \times d\_model \times bytes$
    fp16 example: 2·32,000·4096·2 0.52 GB/layer $\rightarrow$ 16.8 GB for 32 layers (cache only)

Efficient Attention: Fixed Patterns, Sparse Attention [8]

Sparse Attention is most simple to realize and intuitively understandable. At the same time, they are effective, so they are most widely used.

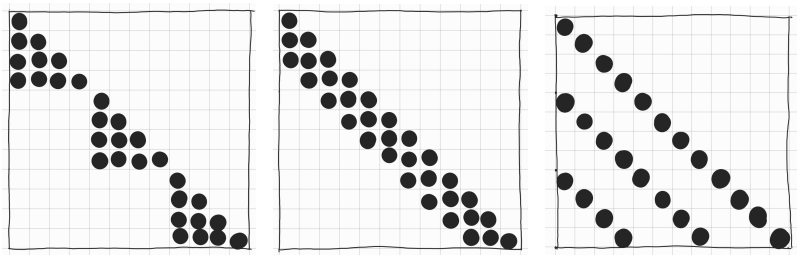


Figure: (a) – **Block Local Attention**, (b) – **Fixed Radius Local Attention**, (c) – **Block Strided Attention**.

- Block Local Attention
  The issue is grouping consecutive elements in the input sequence into blocks. Attention is calculated for elements within each block, elements from different groups do not interact.

$$A_{ij} = \begin{cases} Q_i K_j^T, & \text{if } \lfloor i/k \rfloor = \lfloor j/k \rfloor \\ 0, & \text{else} \end{cases}$$

[8] Generating Long Sequences with Sparse Transformers

Efficient Attention: Fixed Patterns

- Block Strided Attention Similar to block local attention, blocks consist of elements that are departed from each other for a fixed destination d:

$$A_{ij} = \begin{cases} Q_i K_j^T & \text{if} i \equiv j \pmod{k} \\ 0, & \text{else} \end{cases}$$

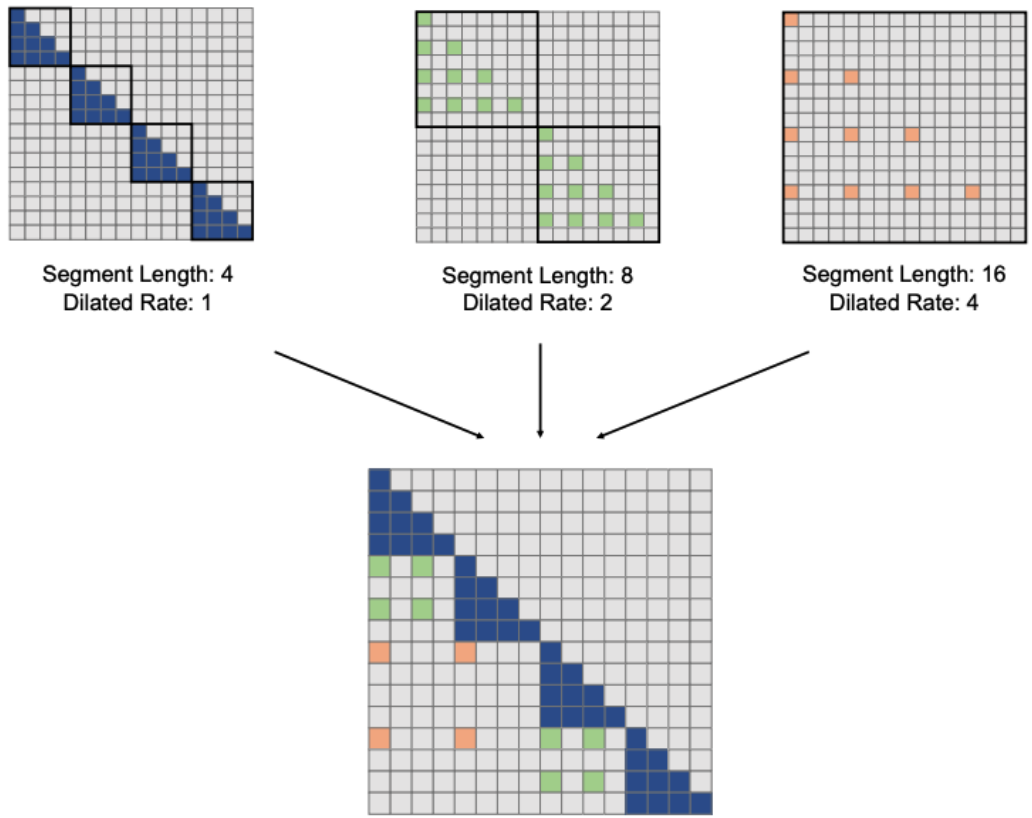
  In original article, stride $= 128$.

- Fixed Radius Local Attention attention is zero for elements stating apart on distance more than d:

$$A_{ij} = \begin{cases} Q_i K_j^T, & \text{if} |i-j| \leq k \\ 0, & \text{else.} \end{cases} \tag{1}$$

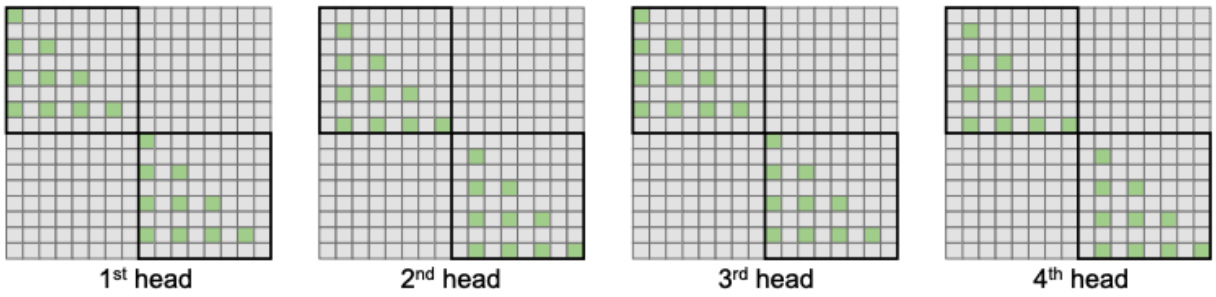
Efficient Attention: Long Net, Dilated Attention

LONGNET: Scaling Transformers to 1,000,000,000 Tokens [9]
Dilated Attention: Attention Allocation decreases exponentially as the distance between tokens grows.



[9] https://arxiv.org/pdf/2307.02486.pdf

Efficient Attention: Dilated Attention



Model	Length	Batch	Github 2K	Github 8K	Github 32K
Transformer [VSP+17]	2K	256	4.24	5.07	11.29
Sparse Transformer [CGRS19]	8K	64	4.39	3.35	8.79
LONGNET (ours)			4.23	3.24	3.36
Sparse Transformer [CGRS19]	16K	32	4.85	3.73	19.77
LONGNET (ours)			4.27	3.26	3.31
Sparse Transformer [CGRS19]	32K	16	5.15	4.00	3.64
LONGNET (ours)			4.37	3.33	3.01

Table 2: Perplexity of language models for LONGNET and the baselines.

Efficient Attention: Kernel methods

Linear Transformer

Improves the complexity of self-attention from quadratic to linear by using a kernel function. **Kernel trick**: we can map function to other space by ϕ, where we can split variables.

$$attn(Q, K, V) = softmax(\frac{Q, K^T}{\sqrt{(d)}})V,$$

$$\text{attn}(\text{Q}, \text{K}_i, V_i) = \frac{\sum_{j=1}^{p} sim(Q_i, K_j)V_j}{\sum_{j=1}^{p} sim(Q_i, K_j)},$$

$$sim(q, k) = \exp\left(q^T k/\sqrt{d}\right) = sim(q, k) = \phi(q)^T \phi(k)$$

than:

$$attn(Q, K_i, V_i) = \frac{\phi(Q_i)^T \sum_{j=1}^{p} \phi(K_j)V_j}{\phi(Q_i) \sum_{j=1}^{p} \phi(K_j)}$$

In original paper (ELU - Exponential Linaer Unit):

$$\phi(x) = elu(x) + 1$$

.

Efficient Attention: Kernel methods

Linformer [10]

Key and value matrix maps to low-rank space using trainable matrix $R \in \mathrm{R}^{b \times N}$

$$K' = RK, V' = RV$$

n	Model	SST-2	IMDB	QNLI	QQP	Average
512	Liu et al. (2019), RoBERTa-base	93.1	94.1	90.9	**90.9**	92.25
	Linformer, 128	92.4	94.0	90.4	90.2	91.75
	Linformer, 128, shared kv	**93.4**	93.4	90.3	90.3	91.85
	Linformer, 128, shared kv, layer	93.2	93.8	90.1	90.2	91.83
	Linformer, 256	93.2	94.0	90.6	90.5	92.08
	Linformer, 256, shared kv	93.3	93.6	90.6	90.6	92.03
	Linformer, 256, shared kv, layer	93.1	94.1	**91.2**	90.8	**92.30**
512	Devlin et al. (2019), BERT-base	92.7	93.5	91.8	89.6	91.90
	Sanh et al. (2019), Distilled BERT	91.3	92.8	89.2	88.5	90.45

[10] Linformer

Efficient Attention: Trainable Patterns

Reformer [11]

Attention is used inside groups of elements with the same hash function, i.e. if *query* is close to *key* $h(q_i) = h(k_j)$. Locality sensitive hashing

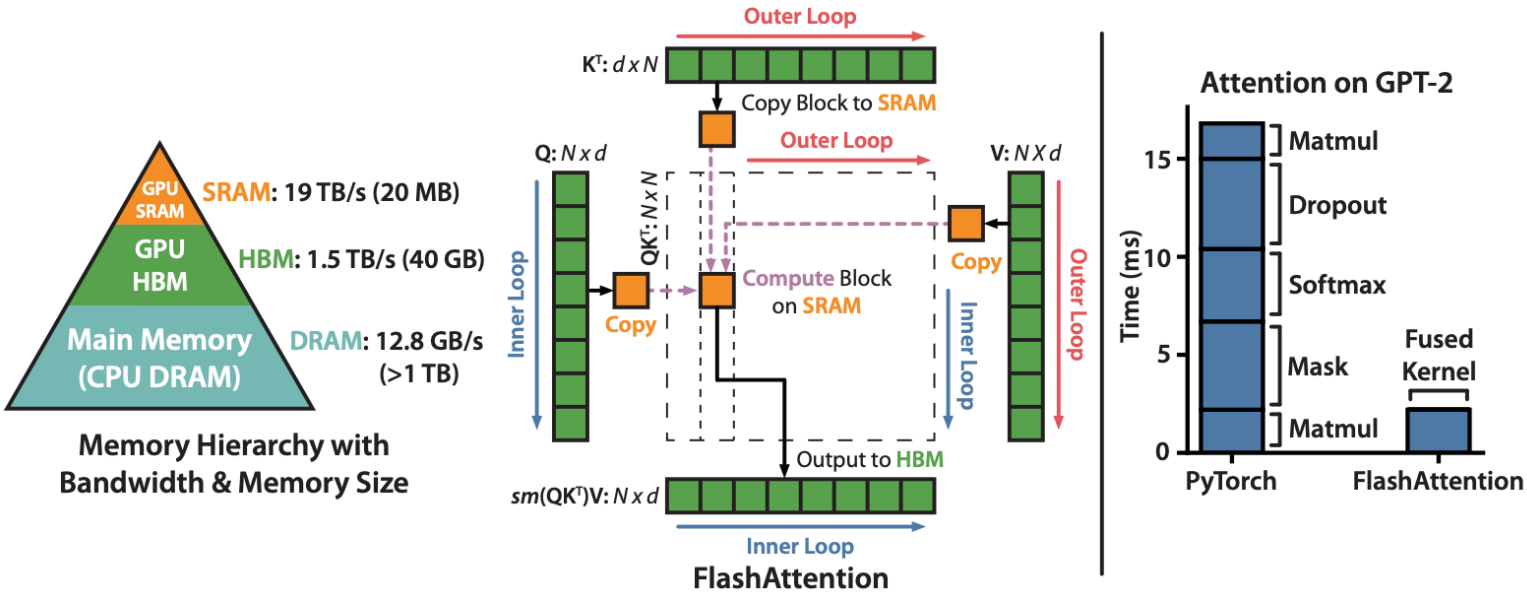
$$h(x) = \text{argmax}[xR, -xR], R - \text{random matrix}$$

[11] Reformer

- The methods described are rarely used, as they typically degrade benchmark performance.
- Still, they laid the groundwork for subsequent work and transformer modifications for long sequences.
- But what is used in practice?

Flash Attention

Key Idea: Use property of GPU core for speed-up computations.



For each attention head, FlashAttention uses classical tiling techniques to load blocks of query, key, and value from GPU HBM (its main memory) to SRAM (its fast cache), compute attention with respect to that block, and write back the output to HBM. We use 1 thread block to process one attention head, and there are overall batch_size $\times$ num_heads threadblocks.

Flash Attention

Flash Attention is integrated in the following pipelines:

- Hugging Face Transformers
- PyTorch 2.0+
- PyTorch Lightning
- Megatron LM

Attention Runtime Benchmark (ms):

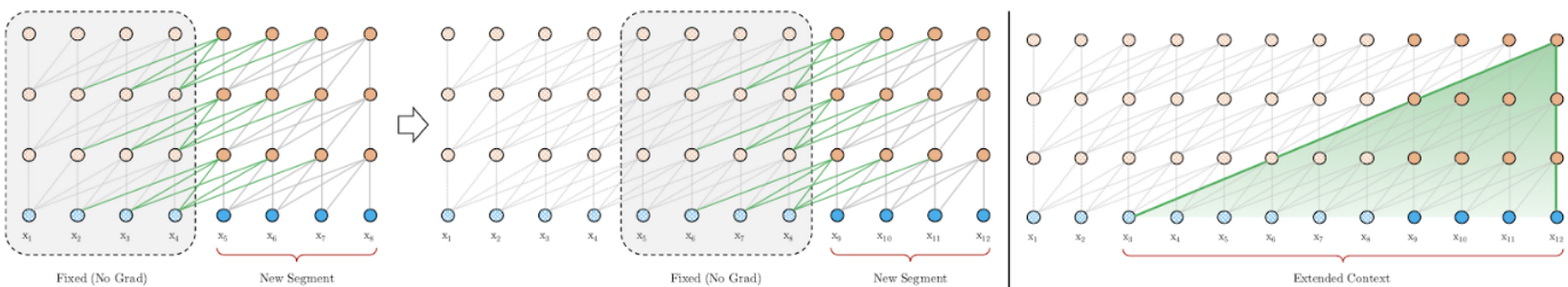
Method / Seq Len	**128**	**256**	**512**	**1024**	**2048**
FlashAttention v2	1.2	1.5	2.3	3.8	6.5
Performer	1.8	2.0	2.5	3.0	3.7
Sliced Attention	1.1	1.4	1.9	2.6	4.2
Sparse Attention (Block Local)	0.9	1.1	1.6	2.3	3.9

Some methods can be faster than Flash Attention — for example, those that sacrifice accuracy (**approximate methods**) or exploit sparsity (**sparse attention mechanisms**).

Next Steps in Modelling: Transformer-XL

Memory Across Chunks

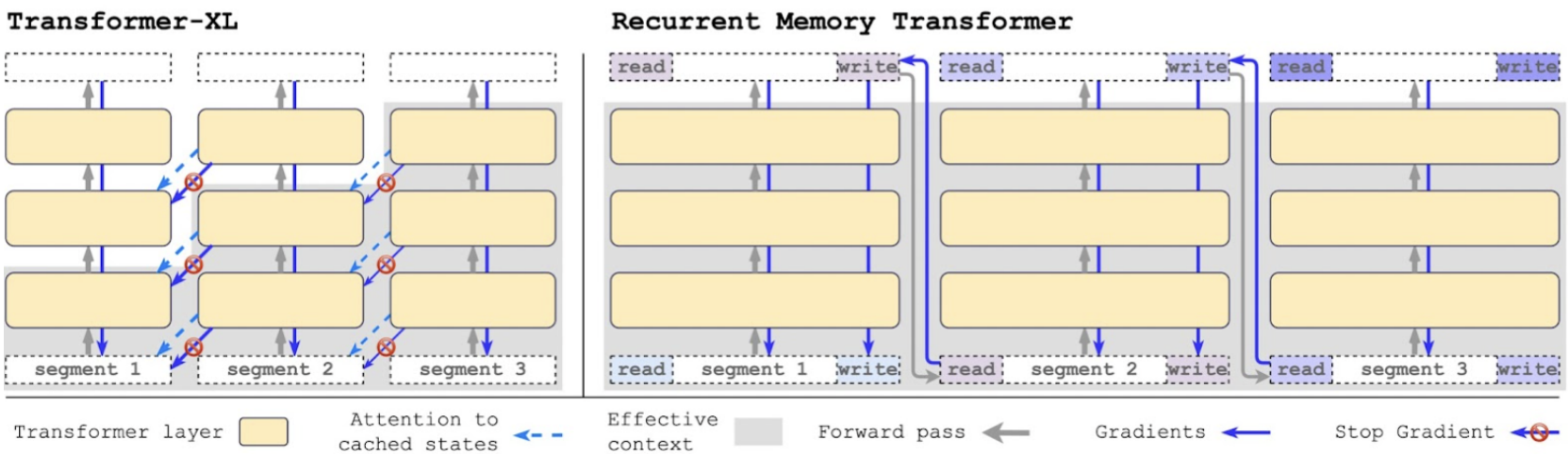
- **Core idea (pre-Long Attention era):** When processing long sequences split into chunks, incorporate information from previous chunks into the current one.
- **Mechanism:**
  - For each new chunk, take the **hidden states** from the previous chunk (for K and V).
  - **Concatenate** them with the current chunk's K and V through a **stop-gradient** operation.



(a) Training phase. (b) Evaluation phase.

Recurrent Memory Transformer (RMT) [1][2]

- **Core idea (AIRI):** Preallocate a **trainable memory block** that is appended to each input segment. This memory is trained to **store and transfer information** across segments.
- **Key difference from Transformer-XL:**
  - The memory block is **learnable and updated via backpropagation** across segments.
  - Enables the model to maintain and refine long-range context representations over time.



[1][2] https://arxiv.org/abs/2304.11062

Long Context Performance: RMT vs Transformer-XL

Model	4K	32K	64K
Transformer-XL	99.8%	67.5%	54.2%
RMT	99.8%	88.5%	82.1%

- On short contexts (4K), both models perform almost perfectly.
- At 32K, Transformer-XL shows a significant drop in accuracy, while RMT maintains high performance.
- At 64K, RMT's advantage becomes even more pronounced.

13

[13] https://arxiv.org/abs/2304.11062, https://github.com/gkamradt/LLMTest_NeedleInAHaystack

Long-context Benchmarks

- **Needle in a Haystack**[14]
  - *Task:* retrieve a single fact hidden in a long context.
  - *Max length:* up to ∼200K tokens.

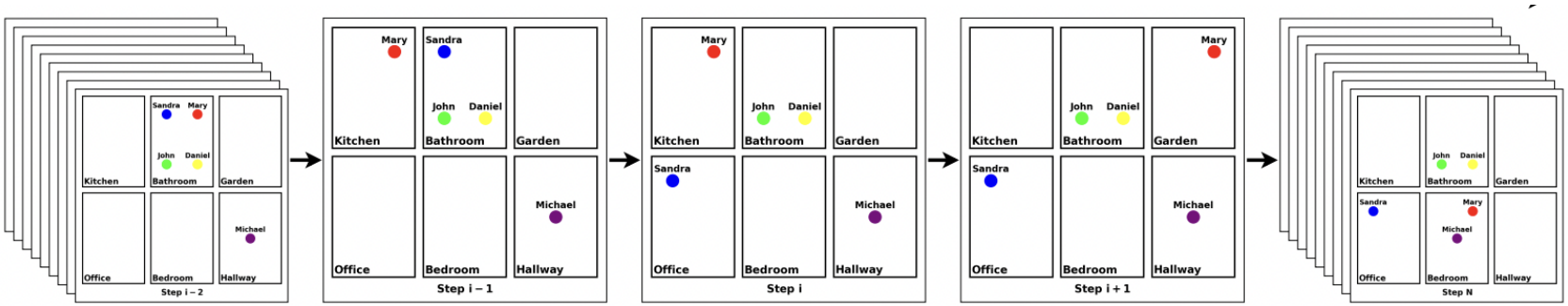
Example

Context: 100,000 tokens of random text + one inserted phrase "...the secret number is 8472."
Question: What secret number is?

[14] `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`

Long-context Benchmarks

- **MMReD (Massive Multi-Resolution Documents)**
  - *Task:* QA and dense multi-modal reasoning over long contexts.
  - *Max length:* up to ∼128K tokens.



Long-context Benchmarks

- **LIBRA: Long Input Benchmark for Russian Analysis (RU)**[15]
  - *Task:* long-range reading comprehension and multi-hop QA.
  - *Max length:* up to ∼100K tokens.

Example

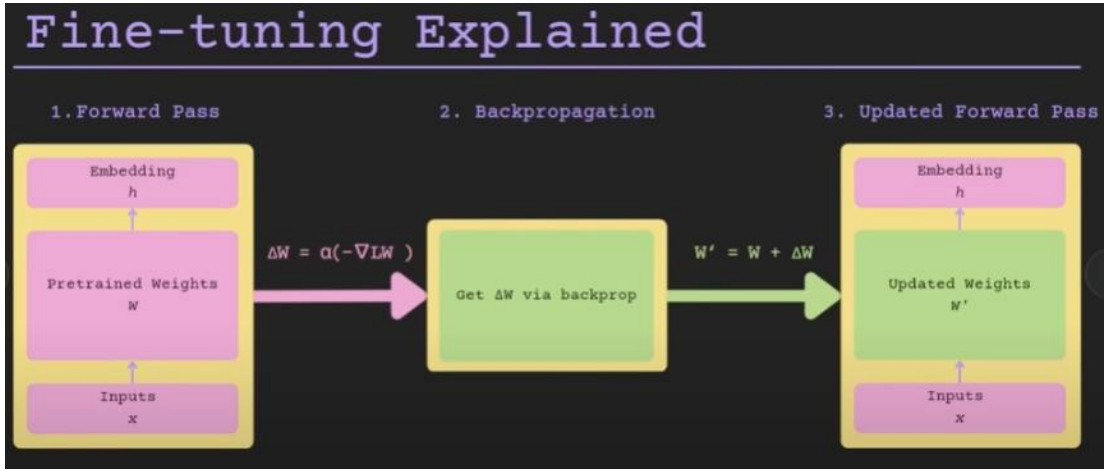
In the middle of the text there is a sentence:
"In 1991 the company released the first version of the software." ...
(followed by more text that does not contain this information)
Question:
In what year did the company release the first version of the software?

[15] `https://github.com/ai-forever/LIBRA/blob/main/README.md`

Plan

LORA:Low-rank adaptation of large language models [16]

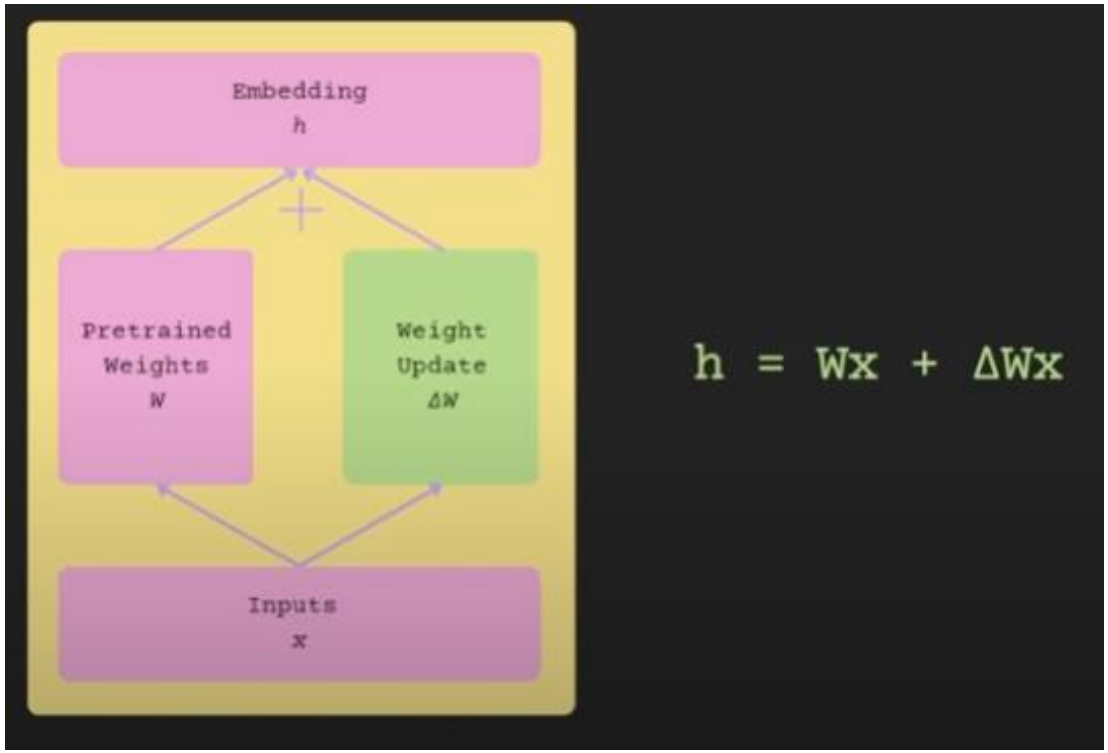
Fine-tuning: if the model has 100 billion trained parameters, storing all of them in memory becomes a significant bottleneck.



[16] https://arxiv.org/pdf/2106.09685.pdf

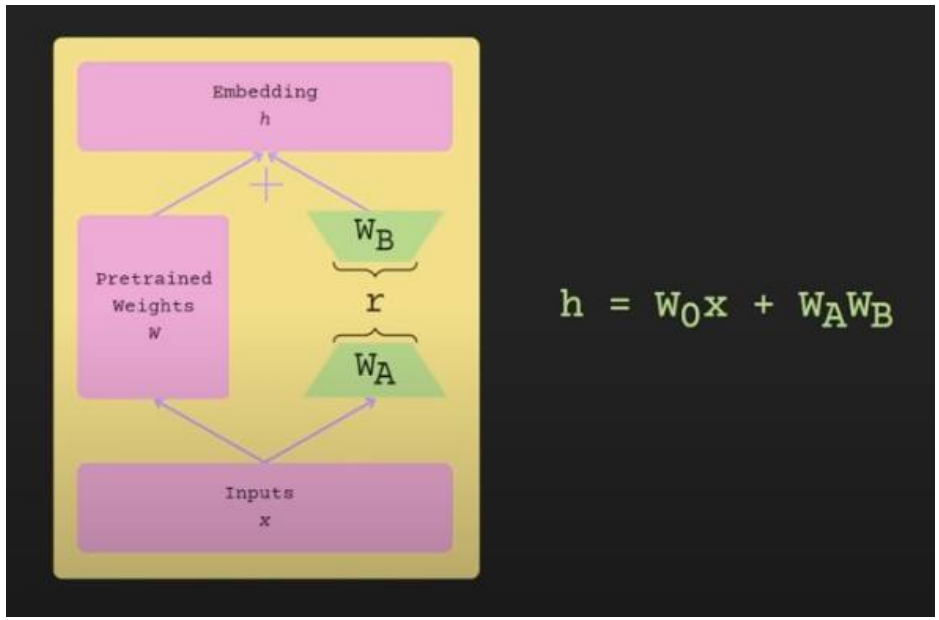
LORA:Low-rank adaptation of large language models

Thus, the pre-trained weights remain static and we only manipulate the delta weights. This is a crucial aspect of the algorithm.



LORA:Low-rank adaptation of large language models

The low-rank approximation is generated by using a technique called singular value decomposition (SVD). SVD decomposes the base model into a set of rank-1 matrices. These rank-1 matrices are then combined to form the target model.



LORA

Results on GLUE (Roberta)

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**

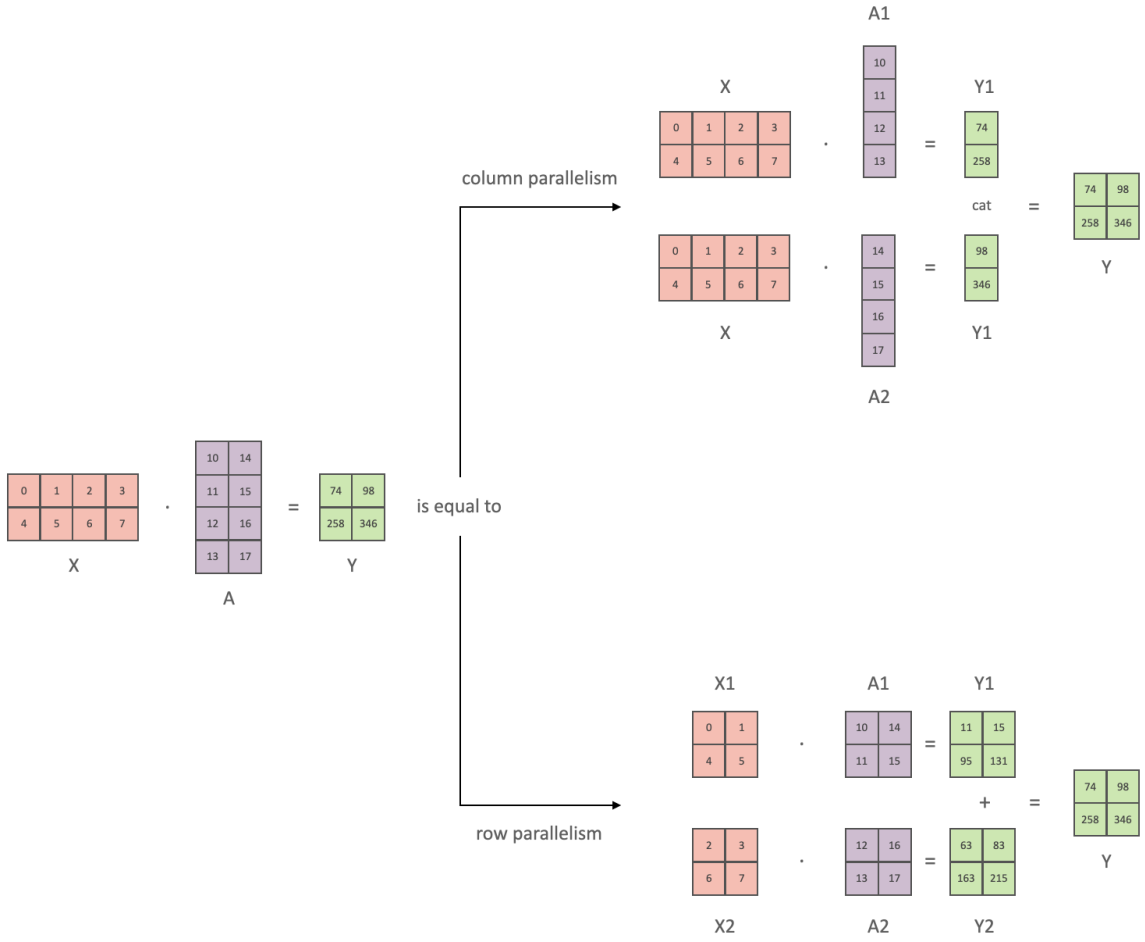
Training Parallelism

- **Data parrallelism** - pieces of a given batch are placed on a different GPU cards
- **Tensor parrallelism** - pieces of a model (blocks, layers, parts of the layers) are placed on a different GPU cards

Data Parallelism

- It creates and dispatches copies of the model, one copy per each accelerator.
- It shards the data to the n devices. If full batch has size B, now size is $\frac{B}{n}$.
- It finally aggregates all results together in the backpropagation step, so resulting gradient in module is average over n devices.

Tensor parrallelism



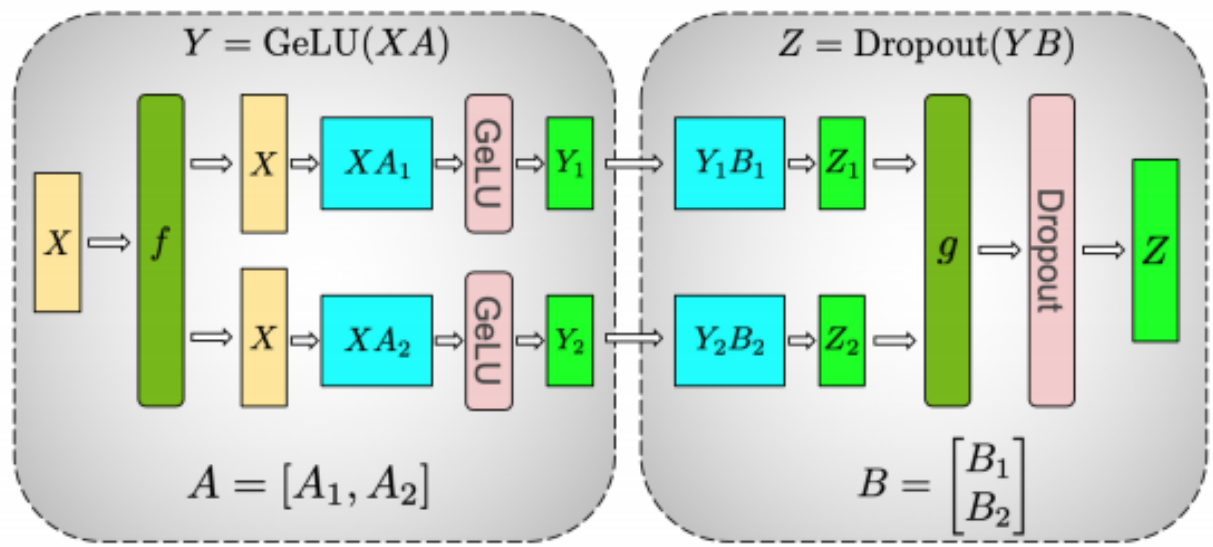
Different ways of splitting the matrix between several GPUs

Tensor parrallelism

A column-wise splitting provides matrix multiplications XA_1 through XA_n in parallel, then we will end up with N output vectors $Y_1, \ldots, Y_n$ which can be fed into GeLU independently

$$[Y_1, Y_2] = [GeLU(XA_1), GelU(XA_2)]$$

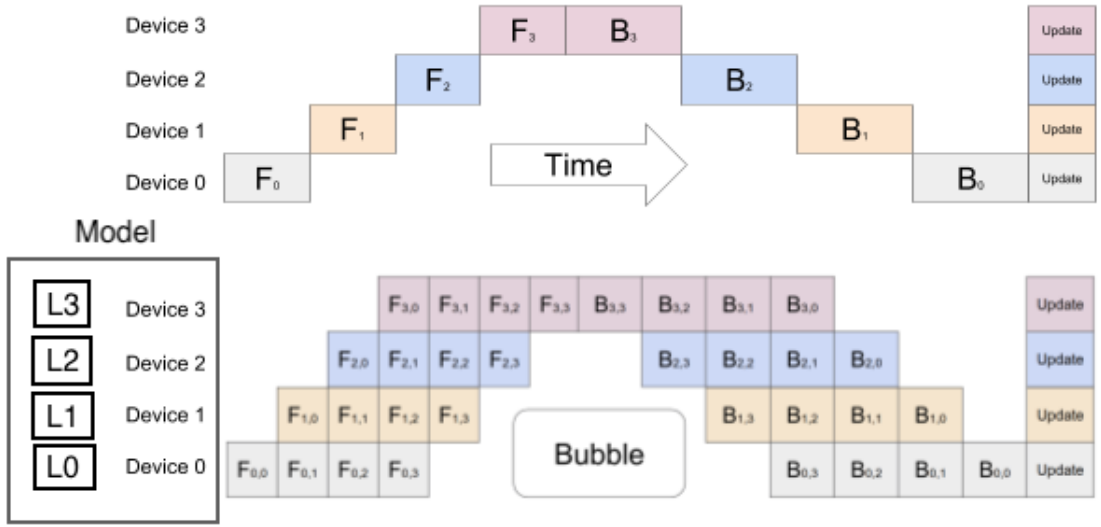
Using this principle, we can update an MLP of arbitrary depth, without the need for any synchronization between GPUs until the very end [17]:



(a) MLP

[17] Megatron

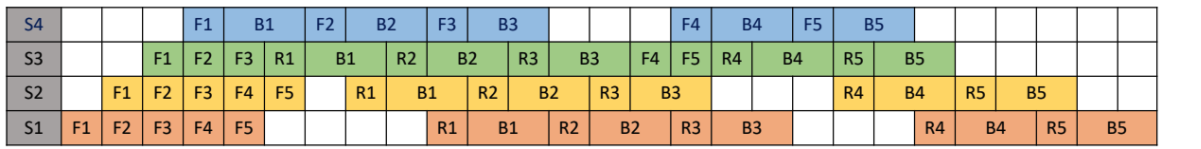
Pipelining



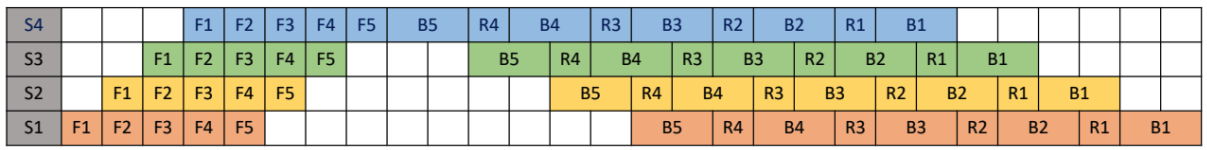
Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. ***Bottom:*** *GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.*

Pipelining

Interleaved pipelining aims to reduce "bubble" size.



(a) Varuna Schedule



(b) Gpipe Schedule

Varuna [18] model scheduler. F - forward pass, B- backward pass, R - recomputation. Varuna recomputes activations by re-running the forward computation, sinse activations take lot of of memory (checkpointing).

[18] https://arxiv.org/pdf/2111.04007.pdf

Thank you for your attention =)