# Transformer-based model compression

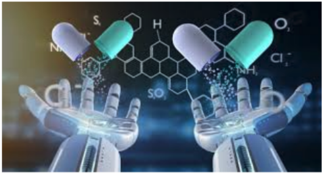
March 31, 2025

# Plan

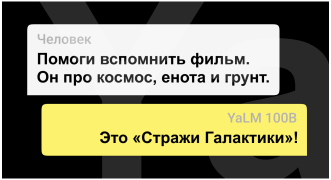
# The relevance of large Transformer models

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



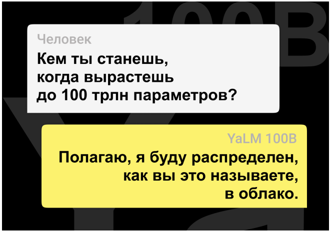




Table: Sizes of big Transformer-based models

| 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

# The main "bottlenecks" in Transformers model

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

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

- **Efficient attention** - provide methods to accelerate the computation of attention vector and make attention matrix more sparse.
- **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

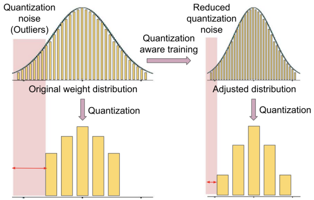
Quantization compresses the network by reducing the number of bits required to represent each weight. It can be applied to Embedding, Fully-connected layers, Attention layers (some layers are quantization friendly, some not - i didn't find a criterion).

- 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

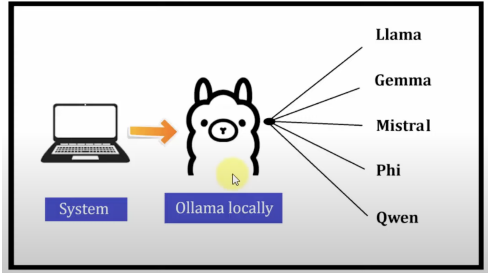



- Simple truncating leads to sizable drop in accuracy
- Tips:
  - Provide quantization while training. **Quantization Aware Training (QAT)**: Float values are rounded to mimic int4/int8 values, but all computations are still done with floating point numbers. This method usually yields higher accuracy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

**Pluses**: 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.

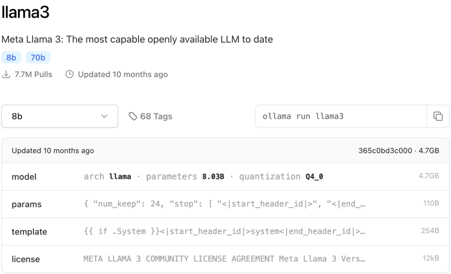
# Use quantization in practise: Ollama

- **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



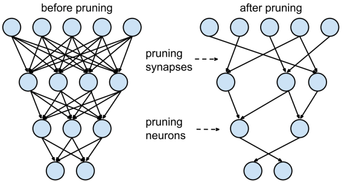
# Use quantization in practise: Ollama

- 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



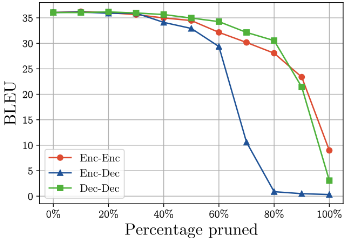

- Initialize weights
- Train
- Evaluate
- Remove weights close to zero, with gradients being close to zero
- Reset rest weights to initial state
- Train again

# 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

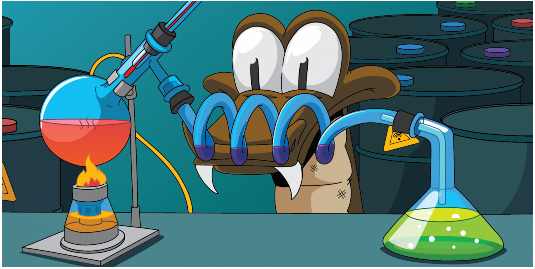
# Pruning: Overview

**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.

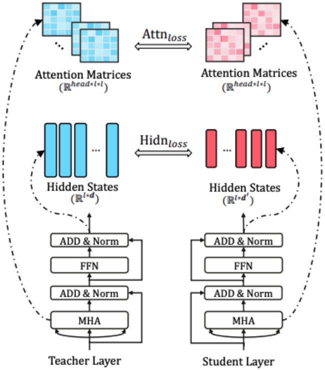
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ (Teacher) | 109M | 22.5B | 1.0x | 83.9/83.4 | 71.1 | 90.9 | 93.4 | 52.8 | 85.2 | 87.5 | 67.0 | 79.5 |
| $\text{BERT}_{\text{TINY}}$ | 14.5M | 1.2B | 9.4x | 75.4/74.9 | 66.5 | 84.8 | 87.6 | 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 |
| $\text{BERT}_4$-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 |
| $\text{DistilBERT}_4$ | 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 |
| $\text{MobileBERT}_{\text{TINY}}\dagger$ | 15.1M | 3.1B | - | 81.5/81.6 | 68.9 | **89.5** | 91.7 | **46.7** | 80.1 | **87.9** | 65.1 | **77.0** |
| $\text{TinyBERT}_4$ (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** $p_{c\,l} = SC(h_l)$
- Loss $(p_{c0} \dots p_{c\,L}) = \sum_i^{L-1} KL(p_{c\,i}, p_t)$

[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

# 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.

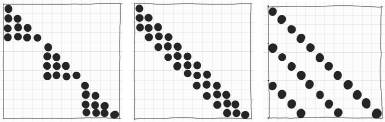


Figure: (a) – **Block Local Attention**, (b) – **Fixed Radius Local Attention**, (c) – **Block Strided Attention**.

- 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 $d$:

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

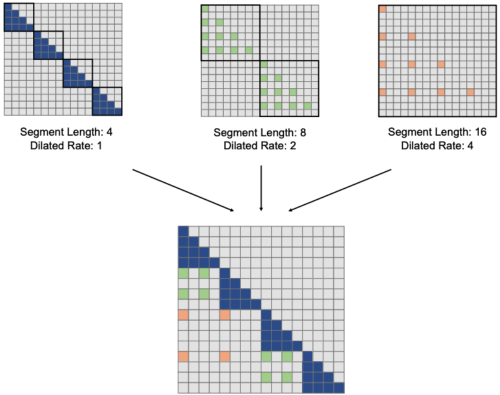
  In original article, stride $= 128$.

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

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

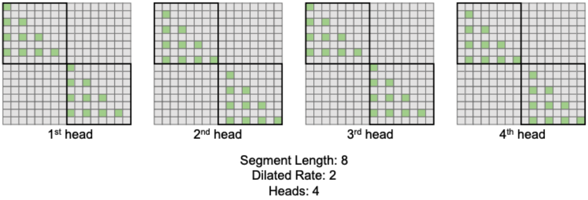
# Efficient Attention: Long Net, Dilated Attention

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)** | 8K | 64 | 4.23 | 3.24 | 3.36 |
| Sparse Transformer [CGRS19] | 16K | 32 | 4.85 | 3.73 | 19.77 |
| **LongNet (ours)** | 16K | 32 | 4.27 | 3.26 | 3.31 |
| Sparse Transformer [CGRS19] | 32K | 16 | 5.15 | 4.00 | 3.64 |
| **LongNet (ours)** | 32K | 16 | 4.37 | 3.33 | 3.01 |

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

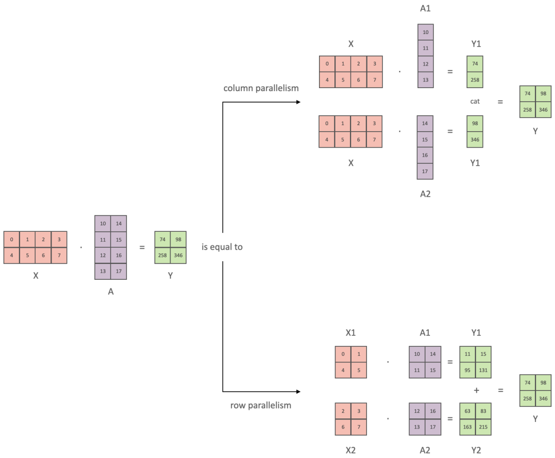
# Training Parallelism

- **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



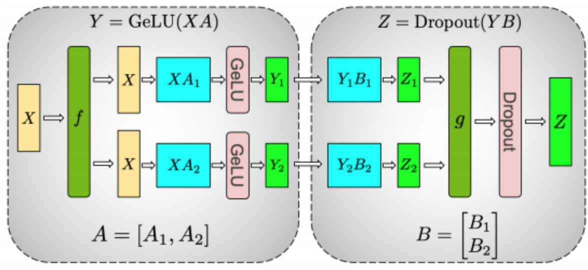

Different ways of splitting the matrix between several GPUs

# 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

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

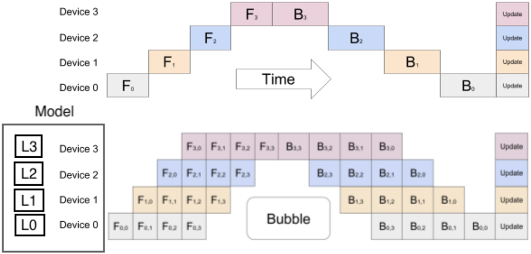
Using this principle, we can update an MLP of arbitrary depth, without the need for any synchronization between GPUs until the very end [10]:




(a) MLP

[10] Megatron

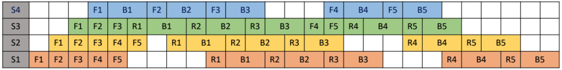
# Pipelining




***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 [11] 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).

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

Thank you for your attention =)