Efficient models: Introduction

Egor Shvetsov

>>> First things First

Всего баллов: 100 - Тесты и ДЗ: 60 - Обзор статей: 10 - Проект: 30	 - больше 80 баллов - отлично - 60 - 80 баллов хорошо - 40 - 60 баллов удв. - 40 и меньше не удв.

Обзор статей в виде презентации: Минимум 4 статьи за 2025 год по теме курса, из тех что не упоминались на курсе

- Число участников в командах ограничено, но не число команд на проект может быть любым

- Регламент выступления - не более 10 минут на команду

- Дедлайн по выбору проектов 27.09.2025. После этого записаться на другой проект будет нельзя.

Критерии сдачи проектов.

1. Наличие кода в репозитории.
2. Наличие презентации для отчета.

Презентация:
- должна содержать мотивация к работе, почему это важно
- план проведенных работ
- описание используемых фреймворков и библиотек
- подробное описание и обзор используемых статей
- описание результатов проекта
- выводы на основании результатов эксперимента, это может большая часть презентации на несколько слайдов, которая обсуждает полезность результатов проекта и намечает дальнейшие, возможные шаги для улучшения и развития
- Презентация должна быть подробной, покрывающей весь пласт проделанной работы. Часть слайдов на презентации можно пропустить на оффлайн изучение.

Код:
- должен содержать полноценный Readme с инструкцией, с помощью которой можно воспроизвести результаты (будет большим плюсом готовый docker-image на dockerhub или Dockerfile со статичными зависимостями)
- В Readme все то что в презентации, но в более краткой форме. Более технические аспекты запуска, воспроизводимости + результаты работы

HW №	Dates	Type	Theme / Goal	Format / Tasks
HW 1	Oct 04 → Oct 25 (3 weeks)	**Coding (Triton kernels)**	*Triton warm-ups*: implement a set of simple kernels (add, outer add, ReLU-mul, backward, sum, softmax, flash attention).	9 practical tasks with `torch.allclose` checks against PyTorch.
HW 2	Oct 06 → Oct 13 (1 week)	**Theory test**	*JIT & Compile in PyTorch 2.0*: JIT trace, JIT compile, `torch.compile` internals.	15 questions (multiple choice + short answer).
HW 3	Oct 13 → Oct 20 (1 week)	**Theory test**	*Pruning & Sparsification + NAS*: overview of pruning/sparsification methods, motivation, sparsity types, and intro to NAS.	15 questions (multiple choice + short answer).
HW 4	Oct 20 → Nov 10 (3 weeks)	**Coding (Triton kernels)**	*Quantized Linear*: symmetric int8 quantization, forward (INT8 matmul + fused dequant), backward in Triton.	Implement `CustomLinearFunction` + `QuantizedLinear` class with forward/backward.

>>> Motivation type one

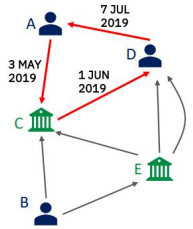
Dataset Example - How to model it?

Do we model it as:

- Sequences
- Tabular data
- Graph

Does it depend on a problem?

Trans. ID	Timestamp	Source bank ID	Source Account	Target bank ID	Target Account	Amount	Currency	Payment type
0	3 MAY 2019 12:45	1	A	1	C	1400	USD	Cheque
1	15 MAY 2019 07:34	2	B	1	C	710	EUR	ACH
2	18 MAY 2019 16:55	3	E	1	C	950	USD	Credit card
3	1 JUN 2019 10:06	1	C	3	D	1200	CHF	Wire
4	27 JUN 2019 13:18	3	E	3	D	2300	EUR	Credit card
5	7 JUL 2019 11:14	3	D	1	A	1100	USD	Credit card
6	14 JUL 2019 09:37	2	B	3	E	650	USD	ACH
7	20 JUL 2019 14:02	3	E	3	D	2500	USD	Wire



- f4 Amount
- f3 Product category
- f2 Geo location
- f1 Currency

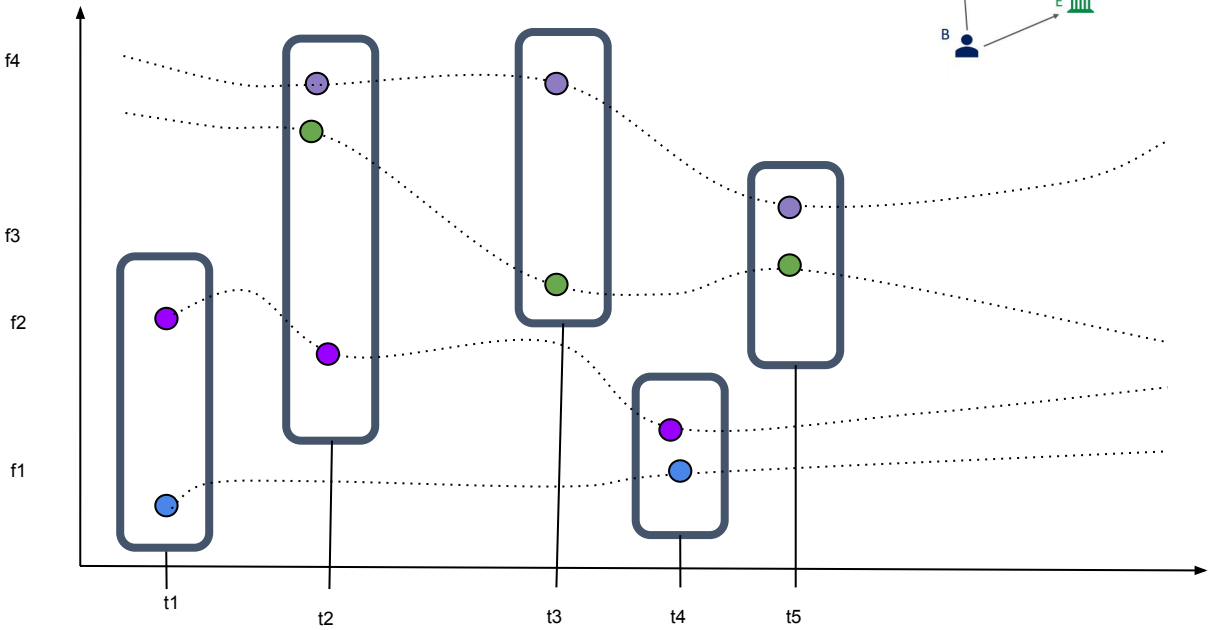


Image source: [Realistic Synthetic Financial Transactions for Anti-Money Laundering Models]

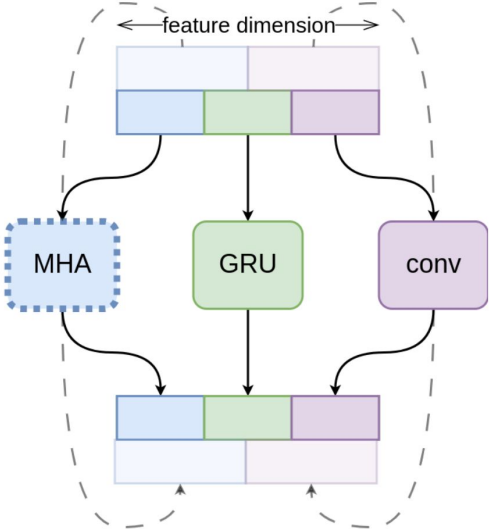
Real life example:

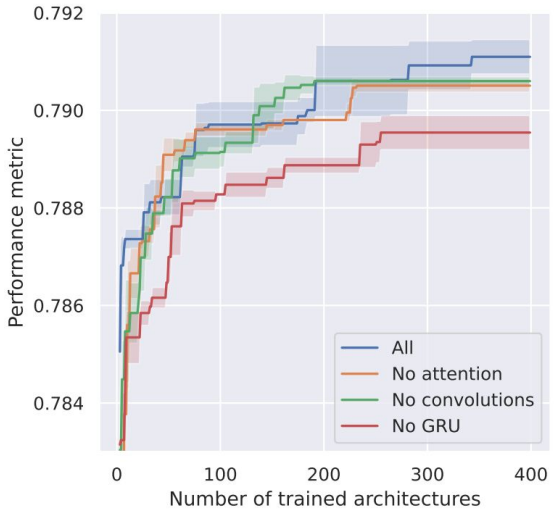
Encoder has both a searchable number of layers and the operations within each layer. Input of each Encoder layer

is divided into one to three blocks along the feature dimension, which can be processed using one of six potential

operations, such as MHA, GRU, or Convolution. The number of heads in MHA is searchable and is chosen from the set {1, 2, 4,8}. In total it provides up to 19 variations for a single Encoder layer. It is worth noting that each layer has a distinct set of operations. The outputs from each block are concatenated and sent to the next layer within the

Encoder. This structure is illustrated in Figure 5. The encoder may have three possible values for the number of layers (1, 2, or 4), resulting in approximately 130×10^3 possible Encoder variations. [1]





[1] SeqNAS: Neural Architecture Search for Event Sequence Classification

Example:

Top-performing banks can potentially save between $1-2$ million USD per year for every 0.5% increase of classification performance, as measured by AUC, in some applied problems, involving classification of users' transactions.

Developing a new model and implementing it into a bank's system could cost approximately $100,000 to $150,000 over a 6-month project period, with half of the effort usually dedicated to model development.

How to find such models?
What efficiency depends on?

Example:

Top-performing banks can potentially save between $1-2$ million USD per year for every 0.5% increase of classification performance, as measured by AUC, in some applied problems, involving classification of users' transactions.

Developing a new model and implementing it into a bank's system could cost approximately $100,000 to $150,000 over a 6-month project period, with half of the effort usually dedicated to model development.

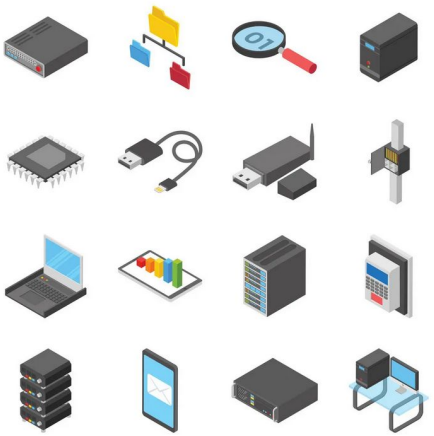
How to find such models?
What efficiency depends on?

- Data pre-processing and transformations
- Expert domain knowledge
- Hyper-parameter optimization
- Architecture design
- Else ... (Your ideas) ?

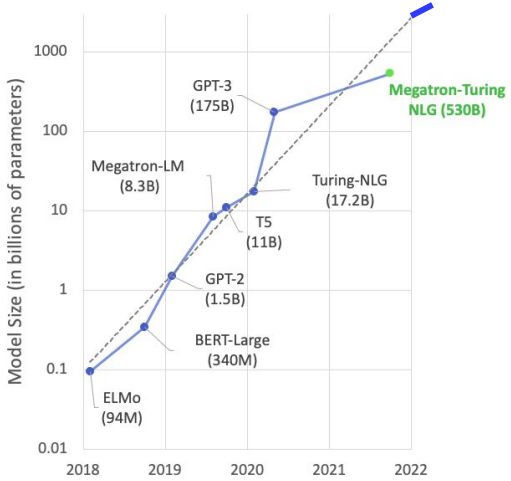
But search space might be too big?

>>> Motivation type two
Compute

Edge devices
Autonomous devices
Robotics



Modern LLMs



Trends

- LLM
- LLM based agents
- Multimodal models
- Scientific

applications

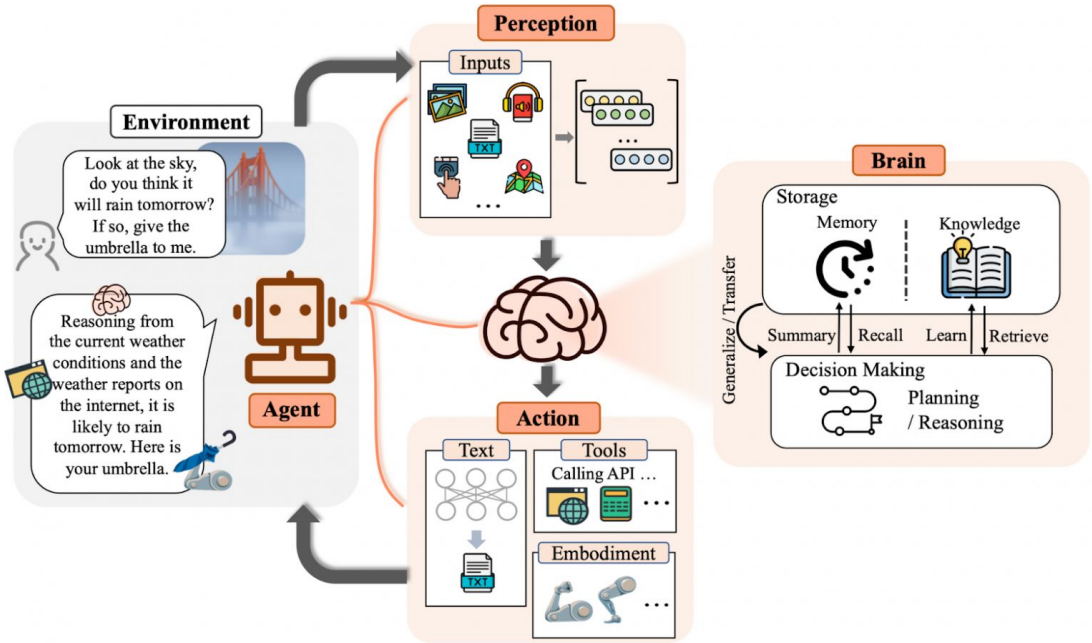
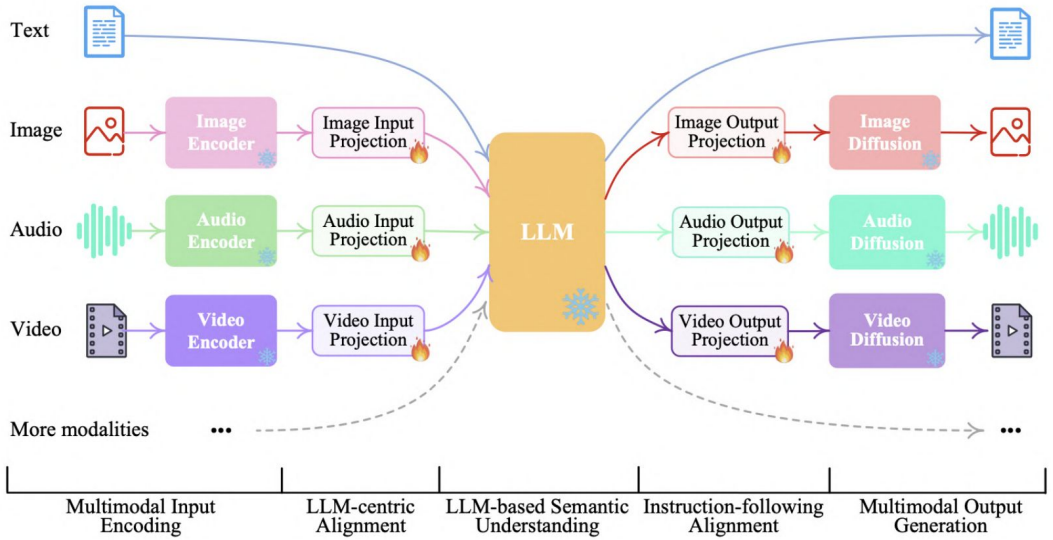


Image source: [Кто такие LLM-агенты и что они умеют?]

Trends

- LLM
- LLM based agents
- Multimodal models
- Scientific applications

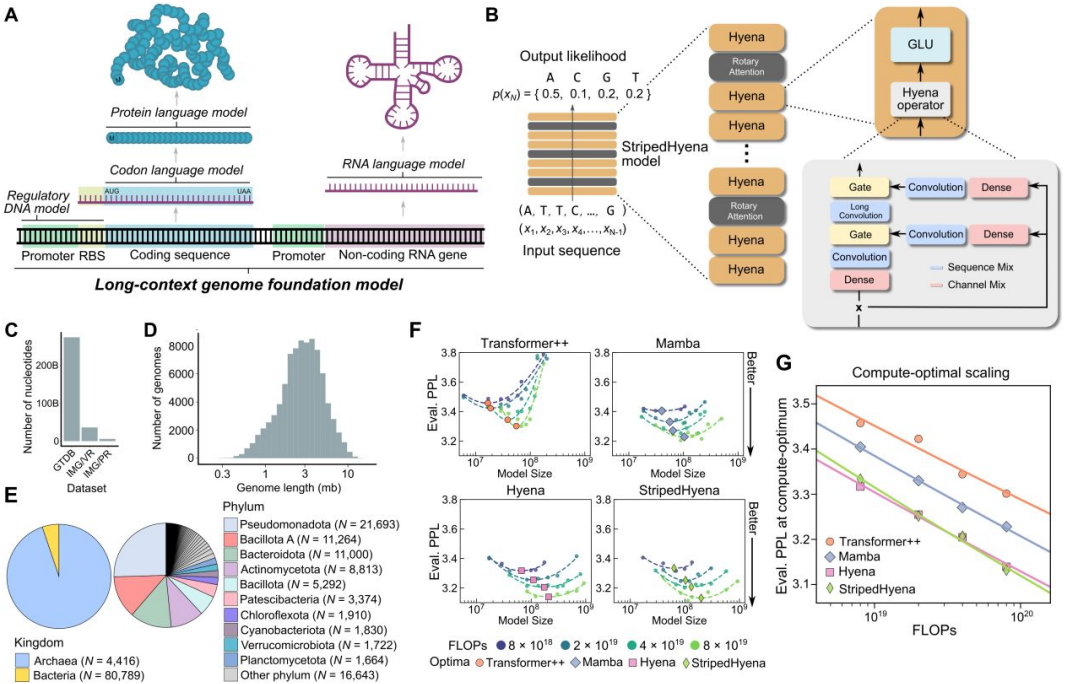


Trends

- LLM
- LLM based agents
- Multimodal models
- Scientific

applicat

Max sequence lengths: 131 000

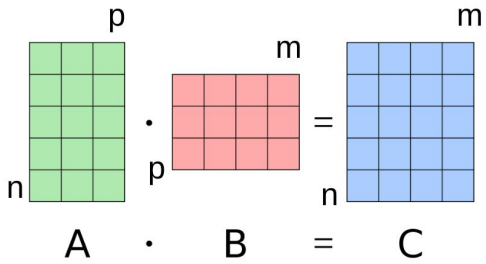


paper: [Sequence modeling and design from molecular to genome scale with Evo]

How do we measure computational efficiency?

Memory
FLOPs
Latency
Energy

$$F[(n \times p) \times (p \times m)] = ?$$

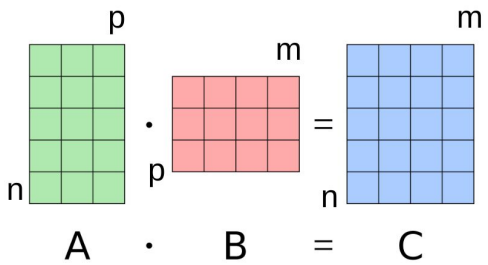


Memory
FLOPs
Latency
Energy

FLOPs - number of floating point operations?

How much is it for a matrix multiplication?

$$F[(n \times p) \times (p \times m)] = 2nmp$$

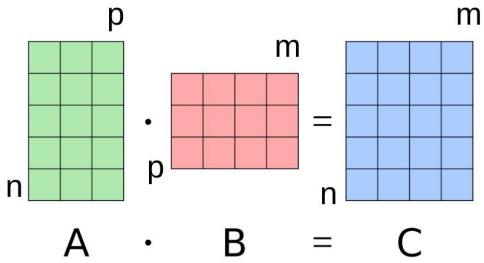


?

Multiply–accumulate (MAC) or multiply-add (MAD) operation is a common step that computes the product of two numbers and adds that product to an accumulator. The hardware unit that performs the operation is known as a multiplier–accumulator (MAC unit).

Сколько флопов уходит на умножение матриц?

- 2npm сложений и lmn умножений, всего 2npm флопов.
- Это 8 умножений для матриц размера 2×2.
- Алгоритм Штрассена делает это за 7 умножений.
- На практике не используется из-за архитектуры «железа».
- На самом деле ответ pmn, а не 2pmn потому что fma. (Fused multiply–add)



Memory
FLOPs
Latency
Energy

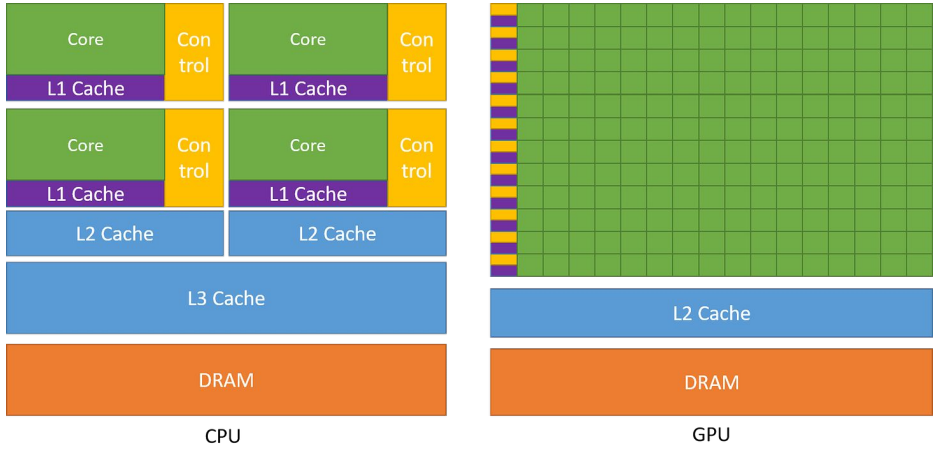
- Под скоростью работы часто понимают разные вещи.
- Latency — время отклика, то есть время от воздействия до реакции.
- Throughput — пропускная способность, то есть сколько данных можно обработать за единицу времени.

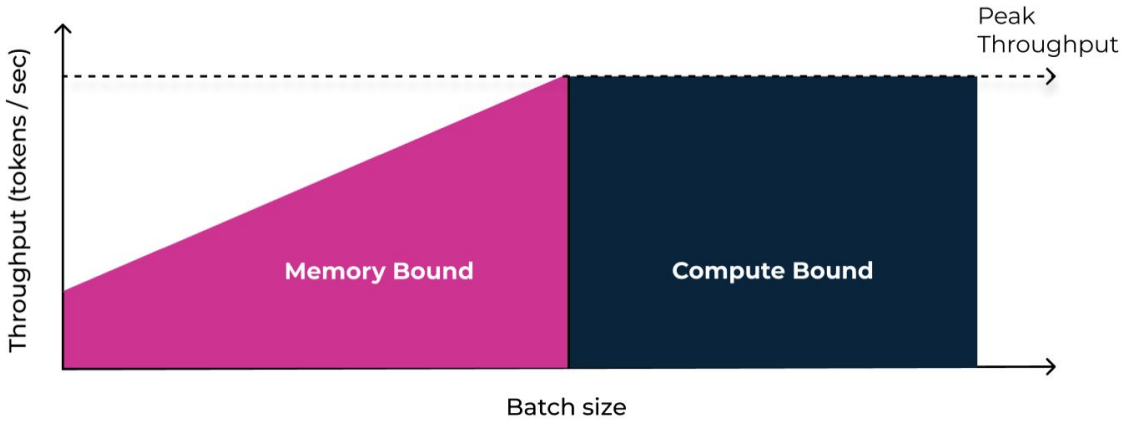
А в чем разница - примеры ?

Latency optimized

Throughput optimized

Memory
FLOPs
Latency
Energy





Memory
FLOPs
Latency
Energy

An example:

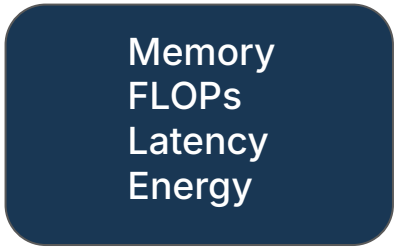
Linear layers:

a weight matrix of **M by M** is used to process
a vector of M values with b bits.

<u>Total data transfer</u> is: **b x (M+M²) or ~bM²**

If the linear layer is used only for one vector **M**, it will require to send the entire **~M²** matrix of weights as computation occurs.

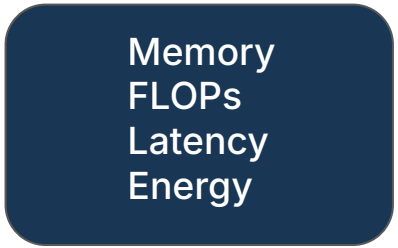
What happens if we have batch of vectors of size M?



Memory limited layers - слои которые почти не ограничены в compute bound:

- Ваши идеи.

source: memory-Limited Layers



Memory limited layers - слои которые почти не ограничены в compute bound:

- Normalization (scaling)
- Batch Normalization
- Activations
- Pooling
- Else ... (Your ideas) ?

source: memory-Limited Layers

How to understand if an operation is memory of arithmetic bound?

$$\text{Arithmetic Intensity} = \frac{\text{number of FLOPS}}{\text{number of byte accesses}} = \frac{2 \cdot (M \cdot N \cdot K)}{2 \cdot (M \cdot K + N \cdot K + M \cdot N)} = \frac{M \cdot N \cdot K}{M \cdot K + N \cdot K + M \cdot N}$$

Let's consider a M x N x K = 8192 x 128 x 8192 GEMM (General matrix multiply)

For this specific case, the arithmetic intensity is 124.1 FLOPS/B, lower than **V100's 138.9 FLOPS:B**, thus this operation **would be memory limited**.

source: Nvidia Matrix Multiplication Background User's Guide

Memory
FLOPs
Latency
Energy

Operation	Arithmetic Intensity	Usually limited by...
Linear layer (4096 outputs, 1024 inputs, batch size 512)	315 FLOPS/B	arithmetic
Linear layer (4096 outputs, 1024 inputs, batch size 1)	1 FLOPS/B	memory
Max pooling with 3x3 window and unit stride	2.25 FLOPS/B	memory
ReLU activation	0.25 FLOPS/B	memory
Layer normalization	< 10 FLOPS/B	memory

source: memory-Limited Layers

Memory
FLOPs
Latency
Energy

Input Type	Accumulation Type	Relative math throughput	Bandwidth savings
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

*Relative to FP32

Model	Params. (B)	Energy / Joule per response
Gemma 2 2B	2	40.42
Mistral 7B	7	43.76
Phi 3 Small	7	44.89
Llama 3.1 8B	8	51.12
Phi 3 Mini	4	54.59
Mistral Nemo	12	66.71
Gemma 2 9B	9	68.24
Phi 3 Medium	14	96.26
Mixtral 8x7B	47	121.49
Gemma 2 27B	**27**	**192.55**
Llama 3.1 70B	70	512.84
Mistral Large	123	869.17
Mixtral 8x22B	141	1161.61

source: ml.energy

How much energy consumes **Gemma 2 27B** in terms of 60W a light bulb:

- ~3 seconds of work
- ~5 seconds of work
- ~60 seconds on work

Model	Params. (B)	Energy / Joule per response
Gemma 2 2B	2	40.42
Mistral 7B	7	43.76
Phi 3 Small	7	44.89
Llama 3.1 8B	8	51.12
Phi 3 Mini	4	54.59
Mistral Nemo	12	66.71
Gemma 2 9B	9	68.24
Phi 3 Medium	14	96.26
Mixtral 8x7B	47	121.49
Gemma 2 27B	**27**	**192.55**
Llama 3.1 70B	70	512.84
Mistral Large	123	869.17
Mixtral 8x22B	141	1161.61

source: ml.energy

4000 mAh at 3.7V: 4000 × 3.7 / 1000 = 14.8 Wh ≈ 15 Watt - hours

14.8 Wh × 3600 J/Wh = 53,280 J

One fully charged Phone:
~ 50 Mistral 8x22B requests
~ 1000 LLama 3.1 8B requests

Model	Params. (B)	Energy / Joule per response
Gemma 2 2B	2	40.42
Mistral 7B	7	43.76
Phi 3 Small	7	44.89
Llama 3.1 8B	8	51.12
Phi 3 Mini	4	54.59
Mistral Nemo	12	66.71
Gemma 2 9B	9	68.24
Phi 3 Medium	14	96.26
Mixtral 8x7B	47	121.49
Gemma 2 27B	**27**	**192.55**
Llama 3.1 70B	70	512.84
Mistral Large	123	869.17
Mixtral 8x22B	141	1161.61

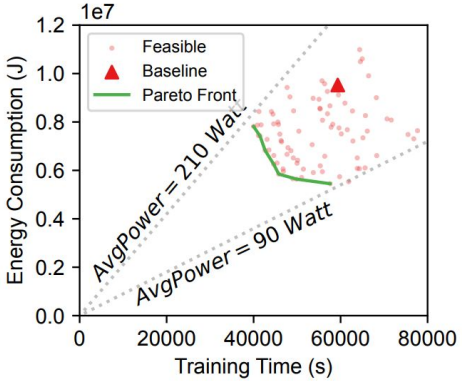
source: ml.energy

Brain consumes 20 joules energy per second.

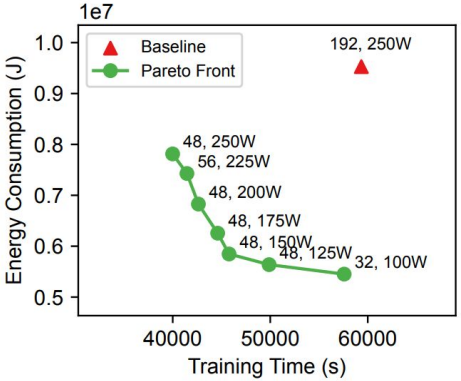
Memory
FLOPs
Latency
Energy

Some facts:

- AlphaGo: 1920 CPUs and 280 GPUs, $3000 electric bill per game
- At extreme scales, training the GPT-3 model just once consumes **1,287 MWh** which is enough to supply an average US household for **120 years**.



(a) *Energy-Time Tradeoff*



(b) *Pareto Front Zoom-in*

paper: [Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training]

Memory
FLOPs
Latency
Energy

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

Memory size	64-bit Memory access
8KB	10 pJ
32KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ

Accessing data from DRAM can consume 100–1000x more energy than performing an arithmetic operation [LINK]

paper: Energy-Efficient Model Compression and Splitting for Collaborative Inference Over Time-Varying Channels

Why does it work & why can’t we train a smaller model from the beginning?

Models are overparameterized
But why?

Initialization

Training (optimization) dynamic

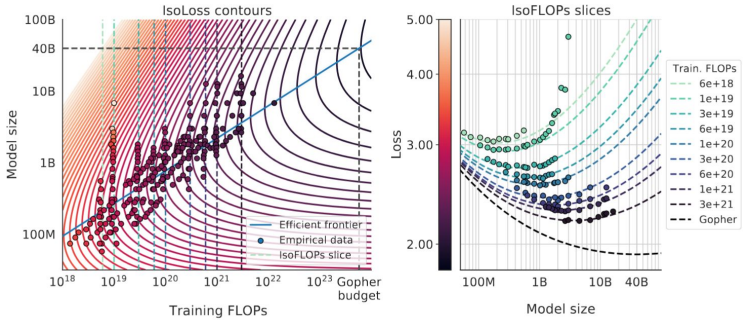
Data size and its quality

Scaling Laws

$$L(N, D) = \left(\frac{N_0}{N}\right)^{\alpha_N} + \left(\frac{D_0}{D}\right)^{\alpha_D}$$

Chinchilla's Golden Rule: Optimal tokens = 20 × model size

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion



Scaling Laws

$$L(N, D) = \left(\frac{N_0}{N}\right)^{\alpha_N} + \left(\frac{D_0}{D}\right)^{\alpha_D}$$

$$\alpha_N \approx 0.074, \quad \alpha_D \approx 0.11$$

Lever	Scaling Exponent	Real-World Impact
Model Size (N)	αN = 0.076	↑ 10× params → ↓ 7.6% loss "(e.g.7B → 70B params)"
Training Data (D)	αD = 0.11	↑ 10× tokens → ↓ 11% loss "(e.g.300B → 3T tokens)"
Compute (C)	γ = 0.052	↑ 10× FLOPs → ↓ 5.2% loss "(e.g.1e22 → 1e23 FLOPs)"

Scaling Laws

$$L(N, D) = \left(\frac{N_0}{N}\right)^{\alpha_N} + \left(\frac{D_0}{D}\right)^{\alpha_D}$$

$$N \sim C^{0.7}, \quad D \sim C^{0.3}$$

Model	Parameters	Tokens	Loss (C4)	Inference Cost*
Chinchilla	70.0B	1.40T	1.45	$0.0001/token
Gopher	280.0B	300.0B	1.58	$0.0007/token
GPT-3	175.0B	300.0B	1.62	$0.0005/token

Scaling Laws

$$L(N, D, C) = \frac{a}{(N \cdot \mathrm{eff}(C))^b} + \frac{c}{D^d} + e$$

Results:

eff(Q):

8-bit: 0.875
4-bit: 0.747
2-bit: 0.289
1-bit: 0.0067

eff(Sparse):

50%: 0.871
75%: 0.622

Compression Scaling Laws: Unifying Sparsity and Quantization

Scaling Laws

$$L(N, D, C) = \frac{a}{(N \cdot \mathrm{eff}(C))^b} + \frac{c}{D^d} + e$$

Results:

eff(Q):

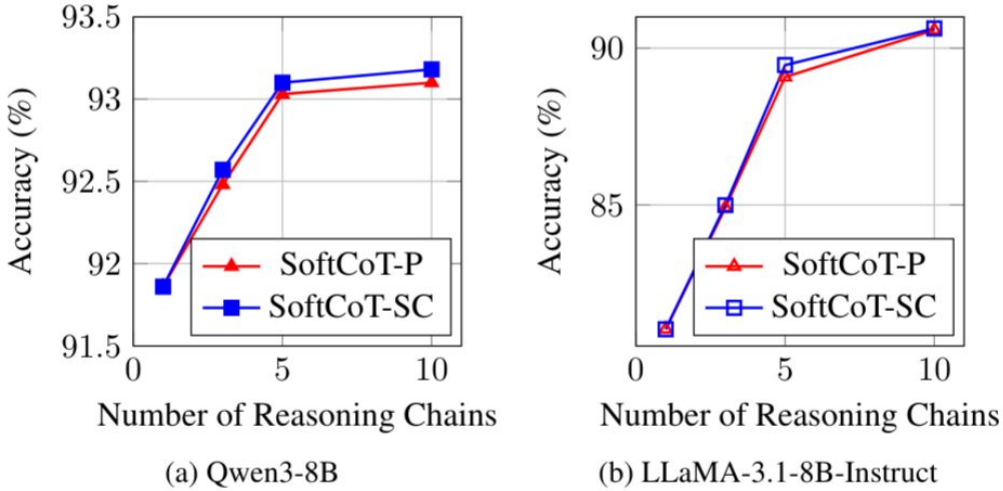
8-bit: 0.875
4-bit: 0.747
2-bit: 0.289
1-bit: 0.0067

eff(Sparse):

50%: 0.871
75%: 0.622

[Compression Scaling Laws: Unifying Sparsity and Quantization]

Scaling Laws



TEST - TIME COMPUTE SCALING

[SoftCoT++: Test-Time Scaling with Soft Chain-of-Thought Reasoning]
[Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters]

>>> Экономика LLM

Computational efficiency & Money:

Using NVIDIA A100 GPUs ChatGPT is capable of outputting around 15-20 words per second;

Given ChatGPT's version 3.5 has over 175 billion parameters, to get an output for a single query, it needs at least five A100 GPUs;

Running this on AWS with the above suggestion would cost $25/hr;

Q1: What the cost of one query should be to break even assuming uniform requests in time?

Q2: What if we improve latency 10%? How much would we save over one day?

Q3: What if we improve memory 10%? How much would we save over one day?

Computational efficiency & Money:

- Using NVIDIA A100 GPUs ChatGPT is capable of outputting around 15-20 words per second;
- Given ChatGPT's version 3.5 has over 175 billion parameters, to get an output for a single query, it needs at least five A100 GPUs;
- Running this on AWS with the above suggestion would cost $25/hr;

Q1: What the cost of one query should be to break even assuming uniform requests in time?

Q2: What if we improve latency 10%? How much would we save over one day?

Q3: What if we improve memory 10%? How much would we save over one day?

A1: ?
A2: We would serve about 8640 requests more!
A2: ?

Мы хотим более хорошие модели, для этого нам нужно:

- Больше данных
- Более большие модели

Более большие модели с большим числом данных будут обучаться дольше, но если обучать долго, модели могут устареть. Мало кто обучает модели дольше чем пол года.

Что делать?

Первый Трансформер 2017 обучался на 8 GPU и 12 часов.

GPT-3 (2018) **1024** GPU V100 и 34 дня

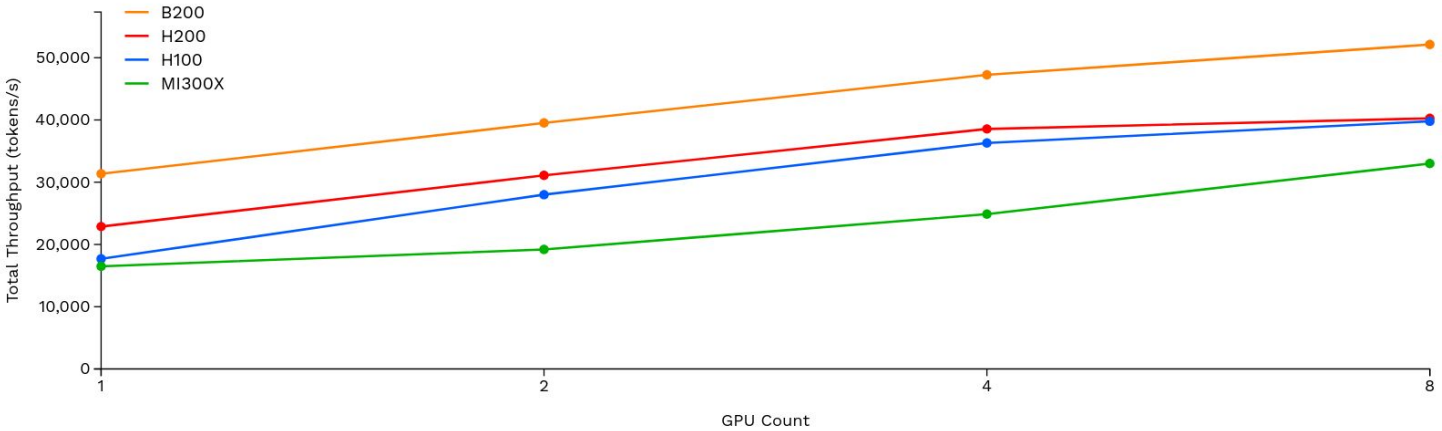
Llama 3.1 405B - **16 000** GPU и примерно 2 месяцев

Initial scaling: Performance scales nearly linearly for small numbers of GPUs (e.g., 2 to 8), especially within a single node connected by high-speed links like NVLink or a fast PCI Express bus.

Bottlenecks and saturation: Scaling efficiency decreases as the number of GPUs increases, particularly when scaling across multiple nodes over an InfiniBand or Ethernet network. The all-reduce communication operation becomes the main bottleneck, as the time spent exchanging gradients grows with the number of GPUs.

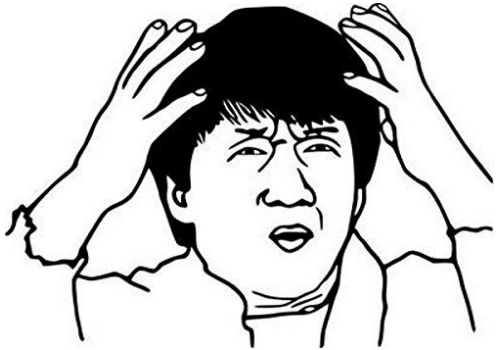
Multi-GPU benchmark results

Total throughput vs. GPU count



source

How to scale to 1000 of GPUs then?



How to scale to 1000 of GPUs then?

Hybrid strategies:

(model parallels) x (data parallel) x (and more)

*Not the scope of this class!

Стоимость обучения

Увеличение
в 3,5 раза/год с 2020 года
Доверительный интервал 90%: от 2,8x до 4,4x раз.

Вычислительные ресурсы для обучения

Увеличение
в 5 раз/год с 2020 года.
Доверительный интервал 90%: от 2,8x до 4,4x раз.

Стоимость приобретения оборудования

Увеличение в 2,5 раза/год с 2016
Доверительный интервал 90%: от 2,1x до 2,9x раз.

Пропускная способность памяти (DRAM)

Увеличение в 1,18 раза/год

Размер датасетов для обучения LLM

Увеличение в 3,6 раза/год с 2010

Когда закончатся данные?

В 2028
Согласно медианному-прогнозу, большая часть эффективного запаса общедоступного текста, созданного человеком, будет использована в 2028

Крупнейший dataset для обучения LLM

>30 триллионов токенов

Модели Llama 4 были обучены на наборе данных > 30 триллионах данных

Объем данных в интернете

510 триллионов токенов

Доверительный интервал 95%: от 130 до 2100 триллионов токенов.

Закон масштабирования (Chinchilla scaling laws)

20 токенов на параметр

Размеры моделей меняются в отношении 1 параметр на 20 токенов

Модель которая обучится на всех токенах

25.5 триллионов параметров

Размеры моделей меняются в отношении 1 параметр на 20 токенов

Самая дорогая модель ИИ на сегодня

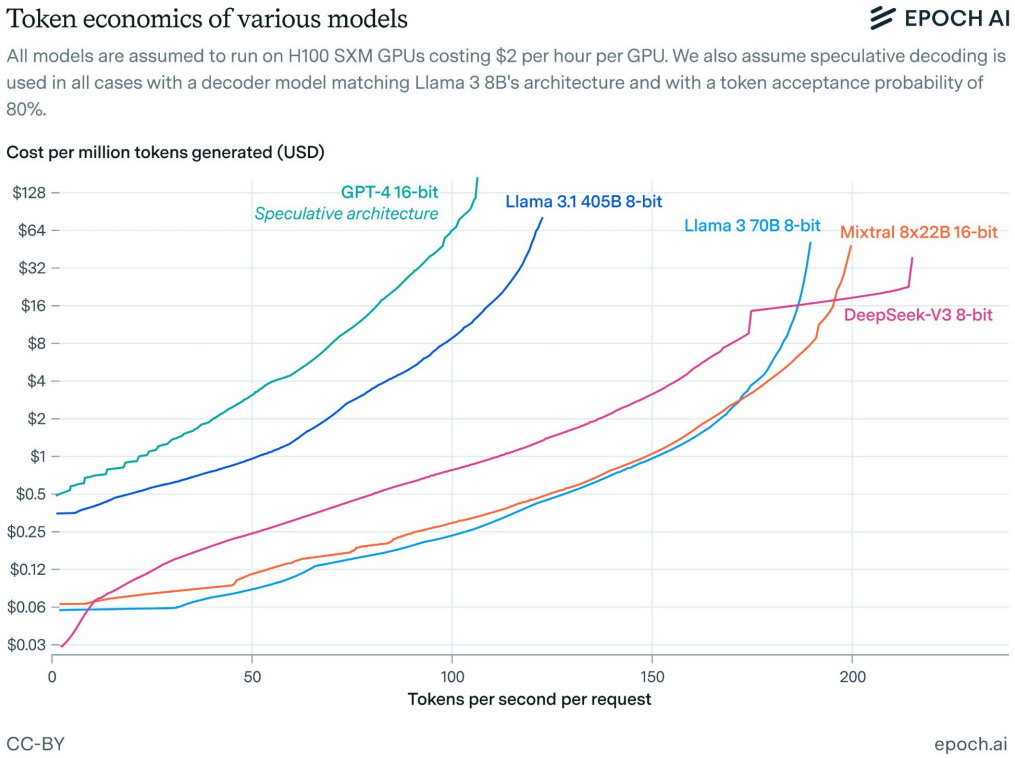
Grok-4 $480 миллионов

Общая амортизированная стоимость разработки Grok-4, включая электроэнергию, оценивается в 480 миллионов долларов США

Самая большая на сегодня модель

DeepSeek-R1 - 685 миллиардов параметров

*Среди моделей для которых количество параметров известно



Из чего формируется стоимость работы моделей.

Время вычислений — это время, которое занимают у ядер GPU выполнение основных операций: сложения и умножения.

Время чтения-записи в память — это время, необходимое для загрузки данных из высокоскоростной памяти (HBM) в ядра. В более сложных моделях учитывается вся иерархия памяти, но здесь мы это опускаем.

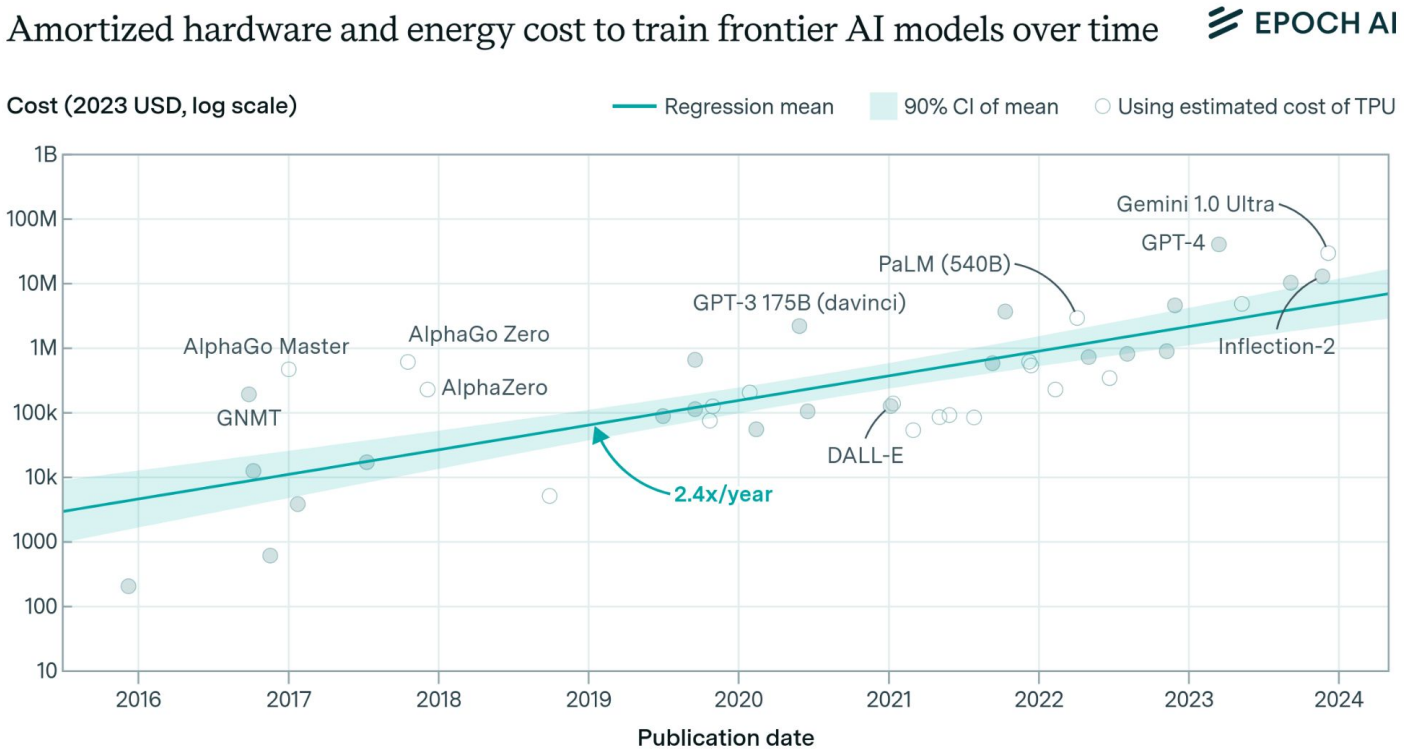
Время передачи по сети — вычисляется делением объёма данных, которые получает каждый GPU, на пропускную способность его сетевого канала.

Задержка — это фиксированное время, которое занимают такие операции, как запуск задач и операции GPU, независимо от их размера.

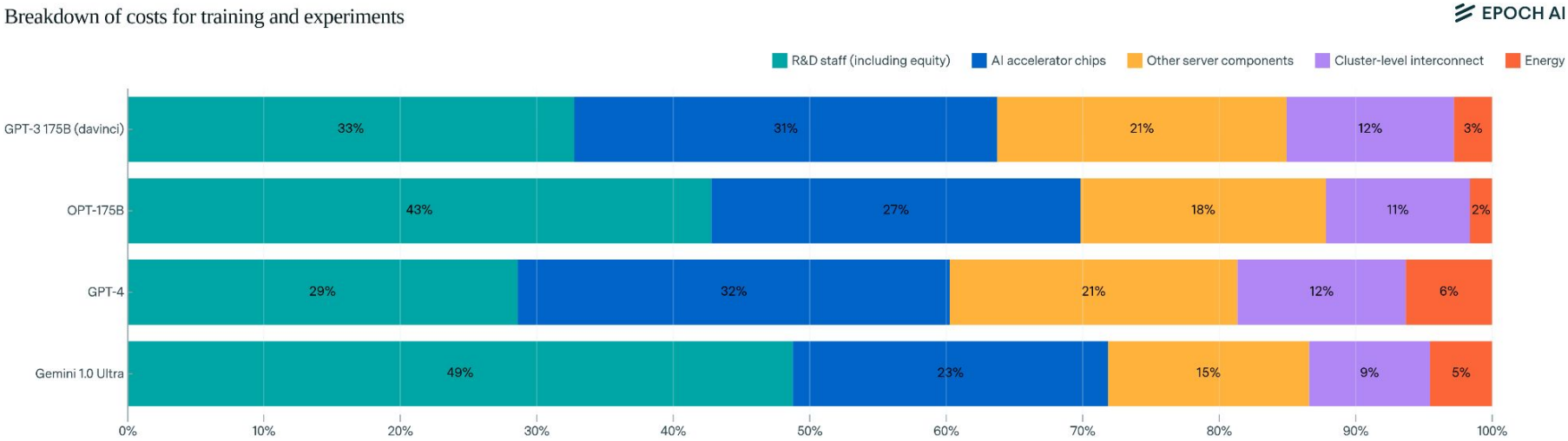
(Например, протокол LL128 в библиотеке NCCL для операции all-reduce имеет базовую задержку в 30 микросекунд на машине DGX H100 — даже если передается всего один байт данных.)

Внимание, в этом расчете нет людей!

Сколько стоит обучить модель?



Сколько стоит обучить модель?



How to make models faster?

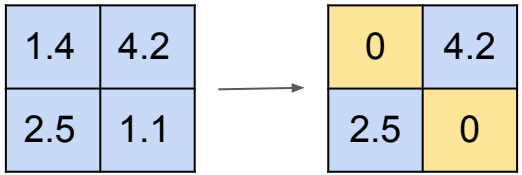
Reduce memory
Reduce FLOPS
Make operations faster

HOW?

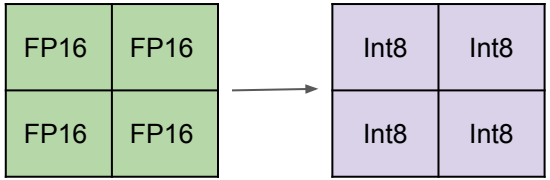
Here we focus on resource efficient methods for faster NN training which accelerate training without significant usage of additional resources.

	General methods		Task or Architecture specific
Methods	**Sparsification**	**Quantization**	**Heuristics**
How	Remove some weights to **perform fewer computations** and reduce memory.	Change data type into a lower precision (e.g. FP16 -> Int8) to **perform faster** computations and reduce memory.	Find some flaws in the current design and fix them. E.g. token pruning in LLMs may not apply for diffusion based LLMs. Another approach is to use a cleaner dataset but this would be task specific.
Cons	Requires some structure or very high sparsity more than 90%. Otherwise Index operations would overcome all of the gains.	Coarse approximation and limited representative range of values. Requires both matrices e.g. X.TW to be the same data type for faster arithmetic.	Will not work for other tasks or architectures.
Pros	Preserves Floating Point range	Retains better performance for weight quantization than weight sparsification. Wider hardware support.	Sometimes leads to really good results.
	Requires hardware support and low level kernel implementation.		Vary a lot based on a task

Perform FEWER computations
&
Reduce memory

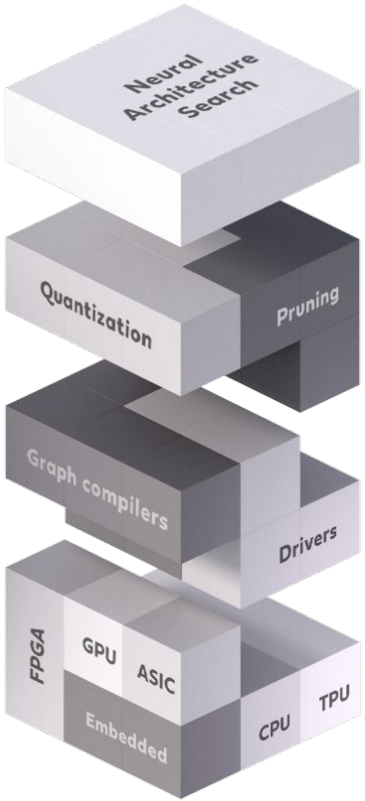


Perform FASTER computations
&
Reduce memory



TALK later

Software meets Hardware



HARDWARE	SOFTWARE	ALGORITHMIC
• GPU • TPU • CPU • FPGA • Ascend • etc.	• CUDA • Graph Compilers • MKL - DNN • etc.	• PRUNING • QUANTIZATION • NAS • etc.

A majority of machine learning architectures lack explicit consideration for software or hardware properties during development, resulting in computational inefficiency.

Furthermore, many models concentrate exclusively on achieving optimal results rather than accounting for practical implementation issues.

Image source: deci.ai

HARDWARE

- GPU
- TPU
- CPU
- FPGA
- Ascend

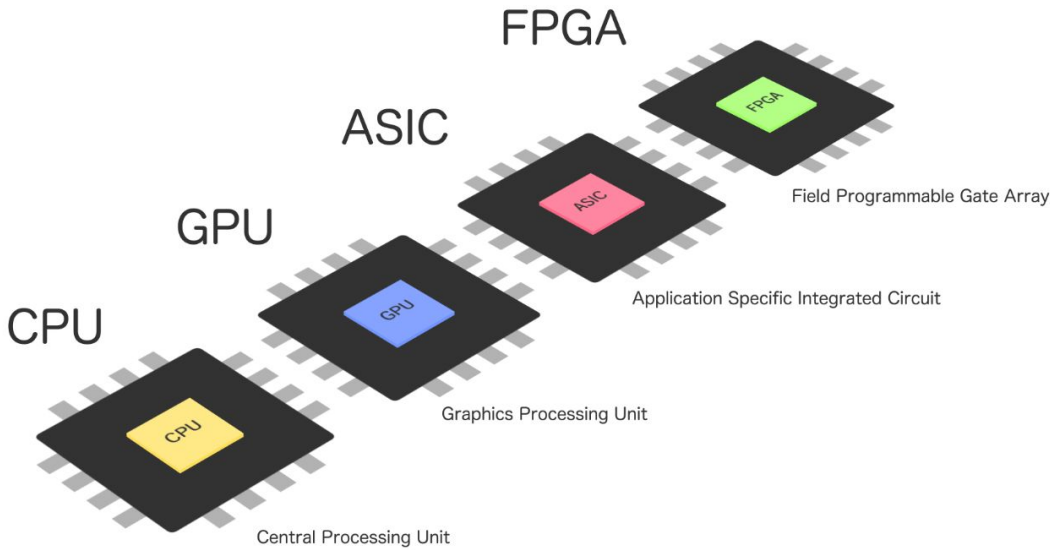


Image source: deci.ai

CPU	The central processing unit (CPU) is a general-purpose processor typically containing between 4 and 16 cores.
GPU	Graphics Processing Units (GPUs) are highly parallel cores (100 or 1000) for high-speed graphics rendering. They provide high-performance processing and typically take up more space and consume more power than CPUs. Because of their large number of small cores, GPUs are well suited for AI workloads, facilitating both neural network training and AI operation.
FPGA	Field Programmable Gate Array (FPGA), a technology that creates a chip with a set of logic elements, flip-flops, sometimes RAM and programmable electrical connections between them, consumes less power than CPUs and GPUs. Can be reprogrammed on site by engineers with programming experience.
ASICS	Application-specific integrated circuits (ASICs) are custom logic designed using manufacturer circuit libraries and offer the benefits of low power consumption, speed and small footprint. However, they are time-consuming to develop and more expensive than other options, so ASICs are recommended for products that will operate in very high volumes. ASICS Types: - Image processing units (VPUs), image and image processors, and coprocessors - Tensor processing units (TPUs), such as the first TPU developed by Google for its machine learning platform, TensorFlow. - Neural Compute Units (NCU), including from ARM
QPU	A quantum processing unit (QPU) is the central component of a quantum computer or quantum simulator. It is a physical or simulated processor that contains a series of interconnected qubits that can be manipulated to compute quantum algorithms. A QPU uses the behavior of particles such as electrons or photons to perform certain types of calculations much faster than the processors in modern computers. QPUs rely on behavior called superposition, the ability of a particle to be in many states at once, described in a relatively new branch of physics called quantum mechanics. In contrast, CPUs, GPUs, and DPUs apply the principles of classical physics to electrical currents. This is why modern systems are called classical computers.
Neuromorphic chips	Neuromorphic computing—brain-inspired computing—has emerged as a new technology that can process information at very low energy costs using electronic devices that mimic the electrical behavior of (biological) neural networks.
Photonic computing	Оптические вычисления или фотонные вычисления используют световые волны, создаваемые лазерами или некогерентными источниками, для обработки данных, хранения данных или передачи данных для вычислений. На протяжении десятилетий фотоны обещают обеспечить более высокую пропускную способность, чем электроны, используемые в обычных компьютерах (см. оптические волокна).

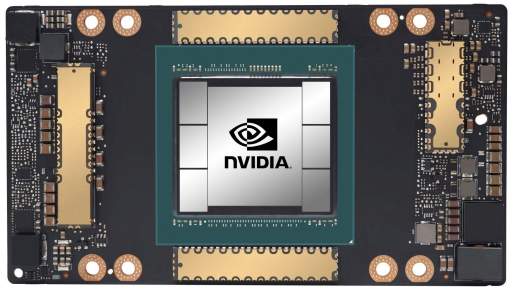
Support of low precision computations in GPU

- Float16
- Float32
- Bfloat16

Will be discussed in future series …

Supported by hardware and software for most modern GPU

Not supported by hardware or software



Supported by some hardware only

Problems with the current processors (CPU/GPU) are:

Energy efficiency:
AlphaGo, GPT and other examples

Architecture:
good for matrix multiplication (still the essence of DL) but other?

source: Hardware for Deep Learning

Cerebras Systems

California-based Cerebras Systems has unveiled the Wafer Scale Engine (WSE-3), its latest artificial intelligence (AI) chip with a whopping four **trillion transistors**. It delivers twice the performance of its predecessor, the Cerebras WSE-2, which previously held the record for the fastest chip.

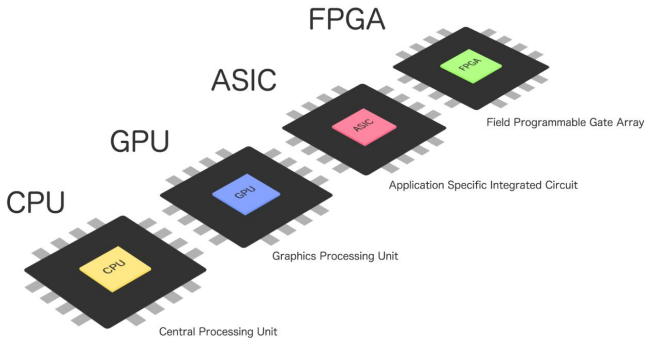
Systems built using WSE-3 will be able to accurately train models with 70 billion parameters in just one day.

The chip is almost size of iPad.

Fast but still has a huge energy consumption.

FPGA

- Amazon has FPGA F1 instances in the cloud
- Alibaba has FGPA F3 instances in the cloud
- Yandex uses FPGA for its own DL output
- Microsoft launched (in 2015) Project Catapult, which uses FPGA clusters
- Microsoft Project Brainwave: FPGA-Based AI Inference
- Microsoft Azure allows you to deploy pre-trained models on FPGA(!)
- Baidu has FPGA instances.



source: [Hardware for Deep Learning]

ASIC

An ASIC (Application Application Integrated Circuit) is an integrated circuit that is customized for a specific use rather than intended for general use.

- Google has Tensor Processing Units (TPU v2/v3/v4) in the cloud
- Intel has acquired Habana, Mobileye, Movidius, Nervana and has processors for training and inference
- Graphcore has a second generation IPU
-
- AWS has its own features for training and inference with ASICS
- Alibaba Hanguang 800
- Huawei Ascend 310, 910
- Bitmain Sophon, Cerebras, Groq and many, many others
- Many ASICs are designed for multi-chip and supercomputer configurations!

source: Hardware for Deep Learning

ASIC/FPGA

Problems with FPGA/ASIC and edge devices

Energy efficiency:
- Better than CPU/GPU, but still far from 20 watts used by the human brain

Architecture:
- Even more specialized for ML/DL computations

source: Hardware for Deep Learning

Neuromorphic computing

Neuromorphic computing - brain-inspired computing - has emerged as a new technology to enable information processing at very low energy cost using electronic devices that emulate the electrical behaviour of (biological) neural networks.

- Usually uses spiking neural networks (SNN)
- Neuromorphic chips attempt to model in silicon the massively parallel way the brain processes information as billions of neurons and trillions of synapses respond to sensory inputs such as visual and auditory stimuli.
- DARPA SyNAPSE program (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) Intel Loihi 2, IBM TrueNorth, Tianjic; Stanford Neurogrid; HRL neuromorphic chip; Human Brain Project BrainScaleS-2, SpiNNaker and HICANN.

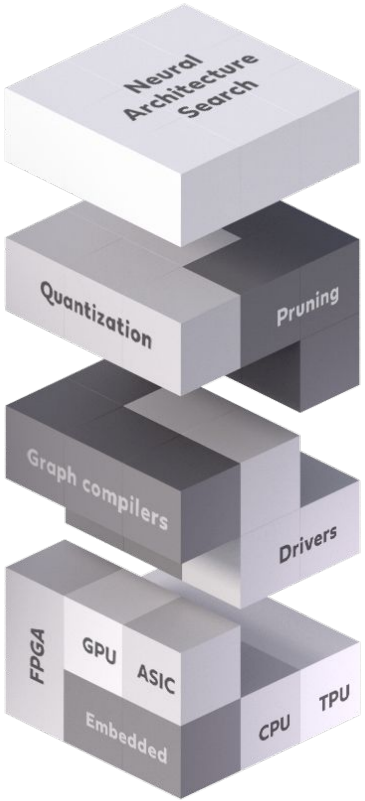
source: Hardware for Deep Learning

Other

- DNA computing
- Unconventional computing: cellular automata, reservoir computing, using biological cells/neurons, chemical computation, membrane computing, slime mold computing and much more

source: Hardware for Deep Learning

CPU	The central processing unit (CPU) is a general-purpose processor typically containing between 4 and 16 cores.
GPU	Graphics Processing Units (GPUs) are highly parallel cores (100 or 1000) for high-speed graphics rendering. They provide high-performance processing and typically take up more space and consume more power than CPUs. Because of their large number of small cores, GPUs are well suited for AI workloads, facilitating both neural network training and AI operation.
FPGA	Field Programmable Gate Array (FPGA), a technology that creates a chip with a set of logic elements, flip-flops, sometimes RAM and programmable electrical connections between them, consumes less power than CPUs and GPUs. Can be reprogrammed on site by engineers with programming experience.
ASICS	Application-specific integrated circuits (ASICs) are custom logic designed using manufacturer circuit libraries and offer the benefits of low power consumption, speed and small footprint. However, they are time-consuming to develop and more expensive than other options, so ASICs are recommended for products that will operate in very high volumes. ASICS Types: - Image processing units (VPUs), image and image processors, and coprocessors - Tensor processing units (TPUs), such as the first TPU developed by Google for its machine learning platform, TensorFlow. - Neural Compute Units (NCU), including from ARM
QPU	A quantum processing unit (QPU) is the central component of a quantum computer or quantum simulator. It is a physical or simulated processor that contains a series of interconnected qubits that can be manipulated to compute quantum algorithms. A QPU uses the behavior of particles such as electrons or photons to perform certain types of calculations much faster than the processors in modern computers. QPUs rely on behavior called superposition, the ability of a particle to be in many states at once, described in a relatively new branch of physics called quantum mechanics. In contrast, CPUs, GPUs, and DPUs apply the principles of classical physics to electrical currents. This is why modern systems are called classical computers.
Neuromorphic chips	Neuromorphic computing—brain-inspired computing—has emerged as a new technology that can process information at very low energy costs using electronic devices that mimic the electrical behavior of (biological) neural networks.
Photonic computing	Оптические вычисления или фотонные вычисления используют световые волны, создаваемые лазерами или некогерентными источниками, для обработки данных, хранения данных или передачи данных для вычислений. На протяжении десятилетий фотоны обещают обеспечить более высокую пропускную способность, чем электроны, используемые в обычных компьютерах (см. оптические волокна).



HARDWARE

- GPU
- TPU
- CPU
- FPGA
- Ascend
- etc.

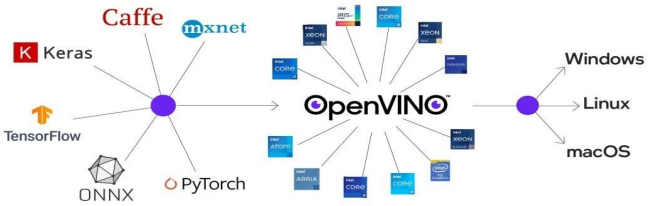
SOFTWARE

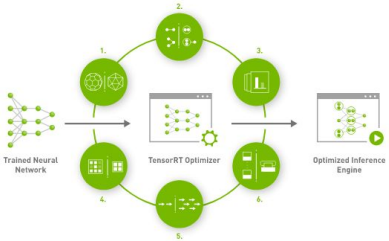
- CUDA
- Graph Compilers
- MKL - DNN
- etc.

ALGORITHMIC

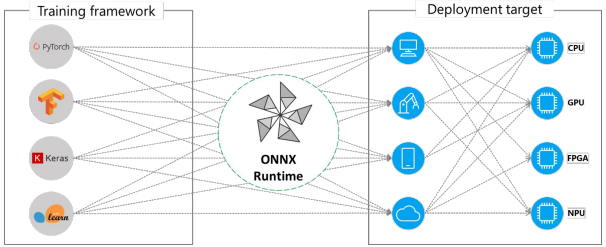
- PRUNING
- QUANTIZATION
- NAS
- etc.

Image source: deci.ai



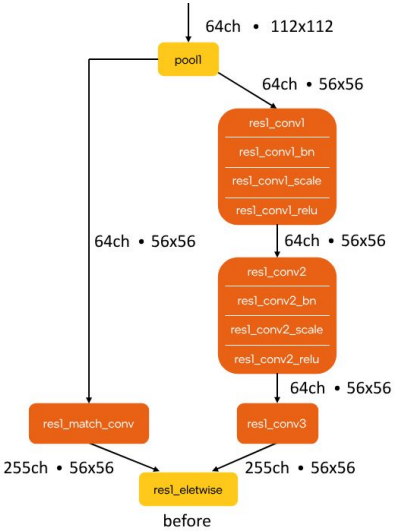


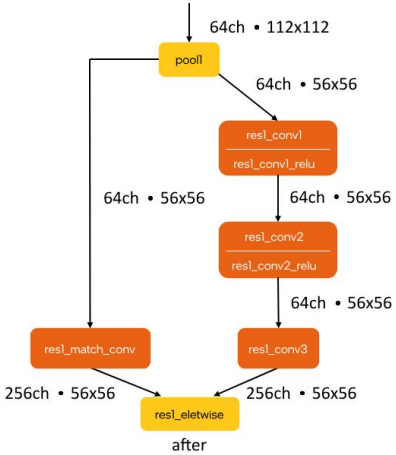
1. **Weight & Activation Precision Calibration**
   Maximizes throughput by quantizing models to INT8 while preserving accuracy
2. **Layer & Tensor Fusion**
   Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel
3. **Kernel Auto-Tuning**
   Selects best data layers and algorithms based on target GPU platform
4. **Dynamic Tensor Memory**
   Minimizes memory footprint and re-uses memory for tensors efficiently
5. **Multi-Stream Execution**
   Scalable design to process multiple input streams in parallel
6. **Time Fusion**
   Optimizes recurrent neural networks over time steps with dynamically generated kernels

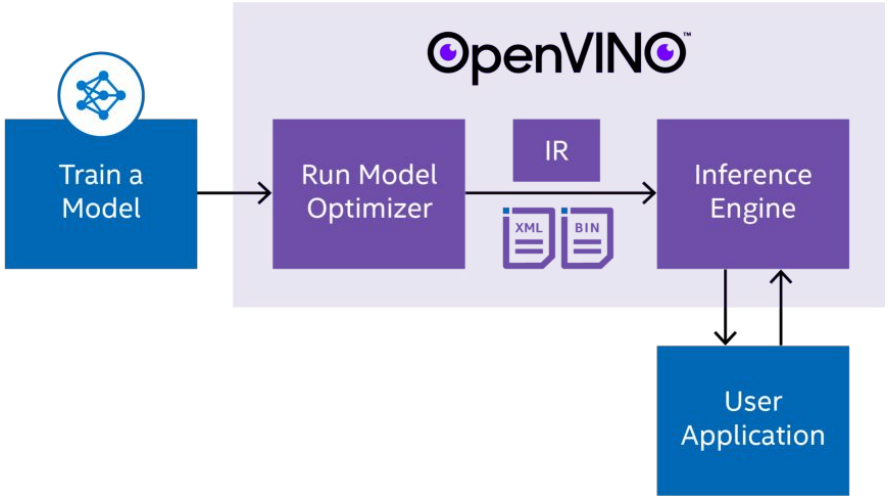


ONNX: This tool provides a standardized format for deep learning models, enabling them to be easily shared across different frameworks and platforms. In our project, ONNX used to ensure that our ResNet-50 model can be seamlessly transitioned between various environments.

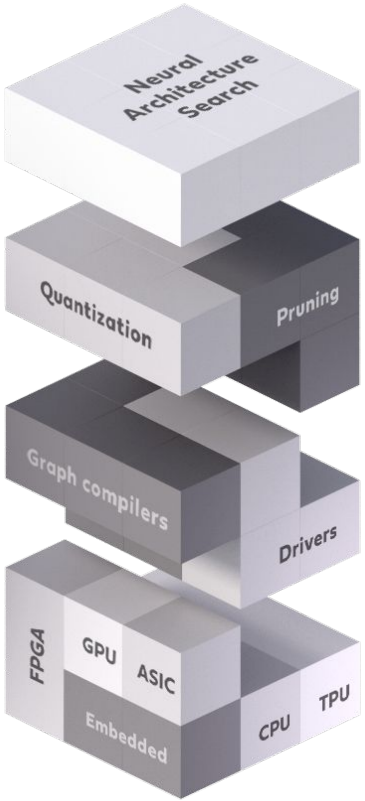
ONNX Runtime	Open source originally created by Microsoft and Facebook	Can be used as high level interface for TensorRT and OpenVino
TensorRT	Nvidia	This tool is specifically designed for NVIDIA GPUs, focusing on maximizing throughput and efficiency. It optimizes neural network models by fusing layers, selecting the most efficient data formats, and leveraging reduced precision (FP16) arithmetic where possible.
OpenVino	Intel	Developed by Intel, OpenVINO specializes in optimizing deep learning models for Intel hardware, particularly CPUs. It is a crucial tool for enhancing the performance of models where GPU resources are not available or limited.







To use quantization, some types of specification and various data formats we need specialized frameworks or we need to write low level solutions ourselves.



HARDWARE

- GPU
- TPU
- CPU
- FPGA
- Ascend
- etc.

SOFTWARE

- CUDA
- Graph Compilers
- MKL - DNN
- etc.

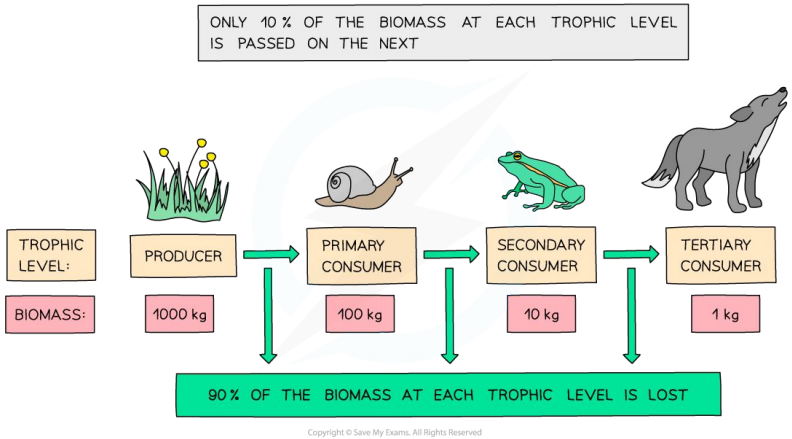
ALGORITHMIC

- PRUNING
- QUANTIZATION
- NAS
- etc.

This part will be covered in the future lectures

Image source: deci.ai

Adapted together



Brain consumes 20 joules energy per second

LLama-7B consumes 335 joules per request

Efficiency (Energy) loss at each level - each component was developed independently

Hardware → Software → Model

Image source: save my exams

We covered some of the problems - let's recall them

What are possible solutions?

Озвученные проблемы:

- Какие архитектуры нам нужны
- Мы хотим наши модели быть более эффективными
- Мы хотим более эффективно обучать модели
- Возможно, в будущем будут другие типы железа, но пока работаем с тем что есть

Решения:

- Hardware-Software-Model Co-Design: для этого нам надо понять как устроено железо и как оно дружит с софтом
- Нам надо научиться понимать, а где именно проблема и как работают модели под разной нагрузкой -, будем заниматься профилировкой моделей
- Автоматизация пайпланов обучение и поиска архитектур
- Экономия на вычислительных ресурсах через приближенные вычисление - квантизация и спарсификация
- Эффективность данных, не все данные одинаково важны

[1]: Codesign-NAS: Automatic FPGA/CNN Codesign Using Neural Architecture Search

>>> Thank you

Questions?

Any Questions?