Large language models and applications

Andrey Kuznetsov
PhD, Head of FusionBrain Lab, AIRI

Term 5. Modern Methods and Algorithms of Generative Artificial Intelligence

Outline

- Language modelling
- Transformer in details
- Open-source LLMs
- Empowering LLM
- Mixture-of-experts
- Pruning and Distillation
- Transformer insights

00

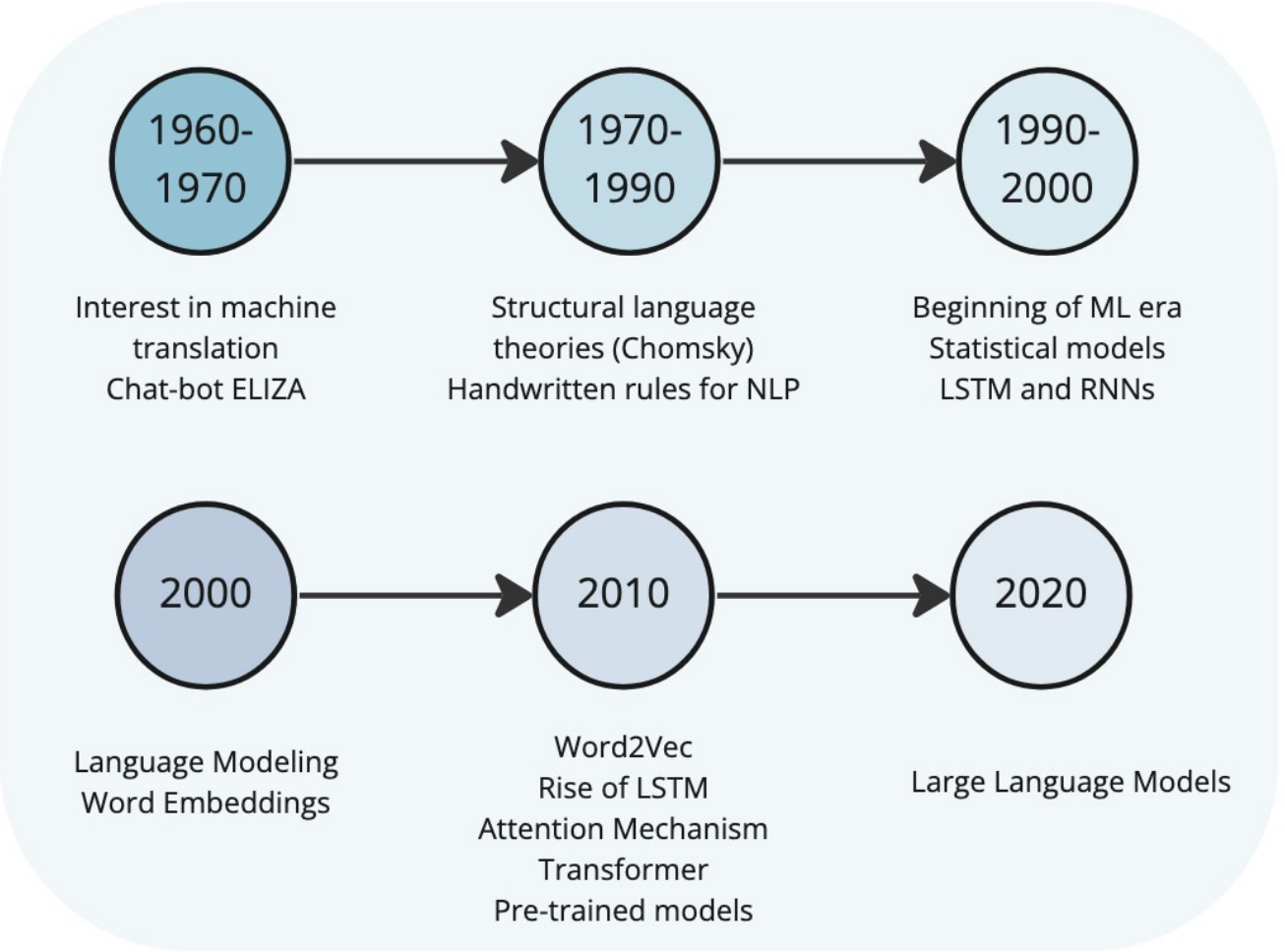
Language modelling

Modern Methods for NLP Research

Deep learning methods

- Vector representations (embeddings) — a word i assigned to the multidimensional vector: Word2Vec, GloVe, ...
- Embedding is a numeric vector that keeps some information about the word

Contextual embeddings are able to encapsulate information about a word in its context. Transformer-based architectures: BERT, RoBERTa, T5, GPT-2(3).

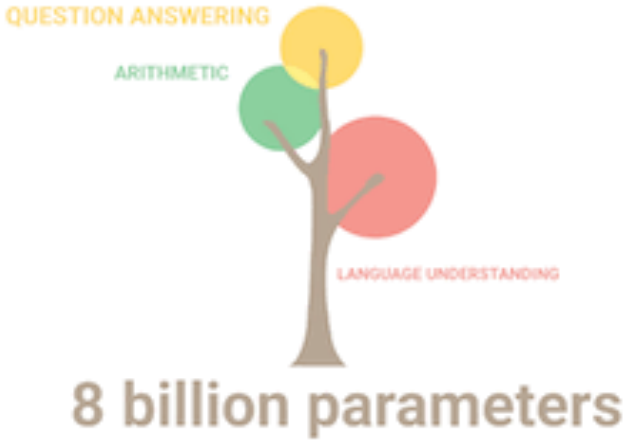


History of NLP

AIRI

Language Models Trend

- BERT (Devlin et al., 2018) – 340M
- GPT-2 (Radford et al. 2019) – 1.5B
- GPT-3 (Brown et al. 2020) – 175B
- MPT (MosaicML, 2023) – 7B-30B
- Llama-2 (Touvron et al. 2023) – 7B-70B
- Falcon (2023) - 7B-180B



Quality vs Quantity

Does the growth of the learnable parameters and the volume of training corpora lead to:

1

Understanding of the commonsense knowledge?

2

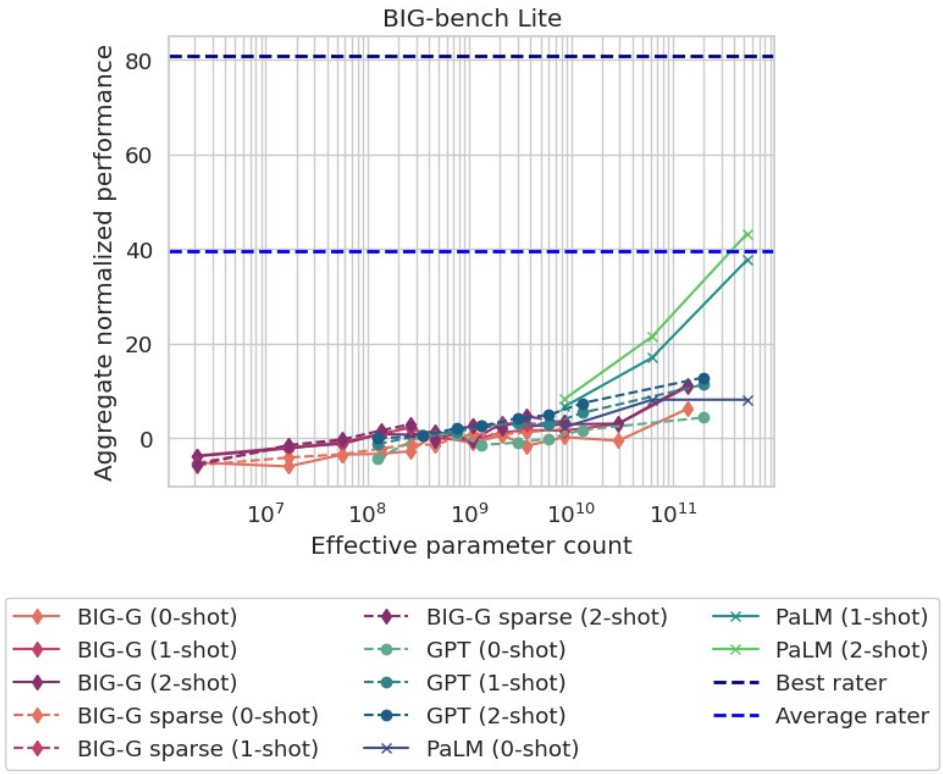
Understanding of the specific linguistic peculiarities?

J. Hwang, Ch. Bhagavatula, R. Bras, J. Da, K. Sakaguchi, A. Bosselut, and Ch. Yejin. 2020. Comet-atomic 2020: On symbolic and neural commonsense knowledge graphs.

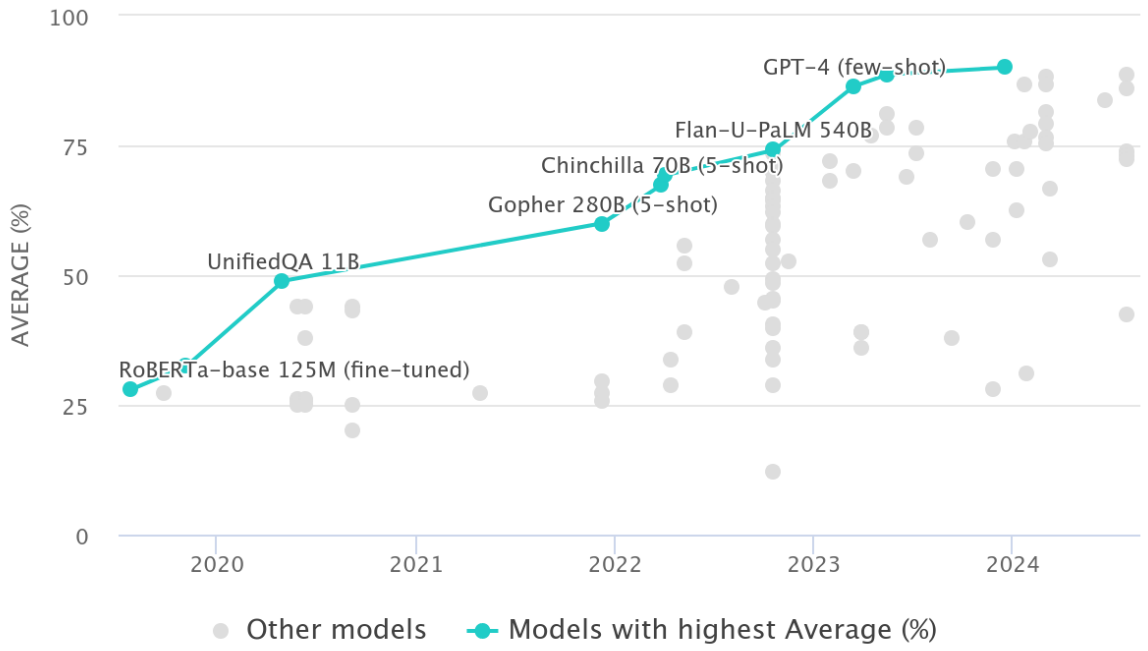
AIRI

Quality vs Quantity

1 BigBench



2 Massive Multitask Language Understanding (MMLU)



Emergent abilities

Emergent abilities encompass a wide range of tasks:

➔ Language transfer

➔ Writing programming code

➔ Understanding multimodal inputs

Few-shot learning LLMs acquire various skills through the **mere observation of recurring patterns in natural language** without explicit task-specific supervision.

01

Transformer in details

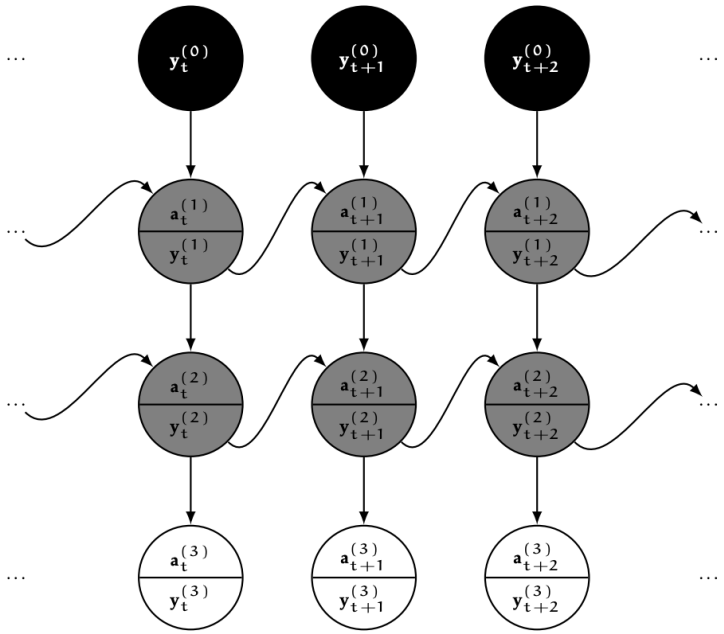
Do not go far into history. RNN

RNN is a class of neural networks which is able to handle sequential data by incorporating information from previous inputs

→ contains a hidden state — retain information from previous time steps

→ operates on sequential data by processing one input at a time

→ updates hidden state based on the current and previous hidden states



$$\mathbf{a}_t^{(k)} = f^{(k)}(\mathbf{y}_{t-1}^{(k)}, \mathbf{y}_t^{(k-1)})$$

$$\mathbf{y}_t^{(k)} = \sigma^{(k)}(\mathbf{a}_t^{(k)})$$

$$\mathbf{a}_t^{(k)} = \mathbf{y}_t^{(k-1)}\mathbf{W}_{in}^{(k)} + \mathbf{y}_{t-1}^{(k)}\mathbf{W}_{rec}^{(k)} + \mathbf{b}^{(k)}$$

AIRI

RNN. Pros and cons

- Process input of any length
- Model size do not increase with size of input
- Computation takes into account historical information
- Weights are shared across time

- Slow computation
- The further you go, the more you forget — vanishing gradient problem
- Cannot consider any future input for the current state

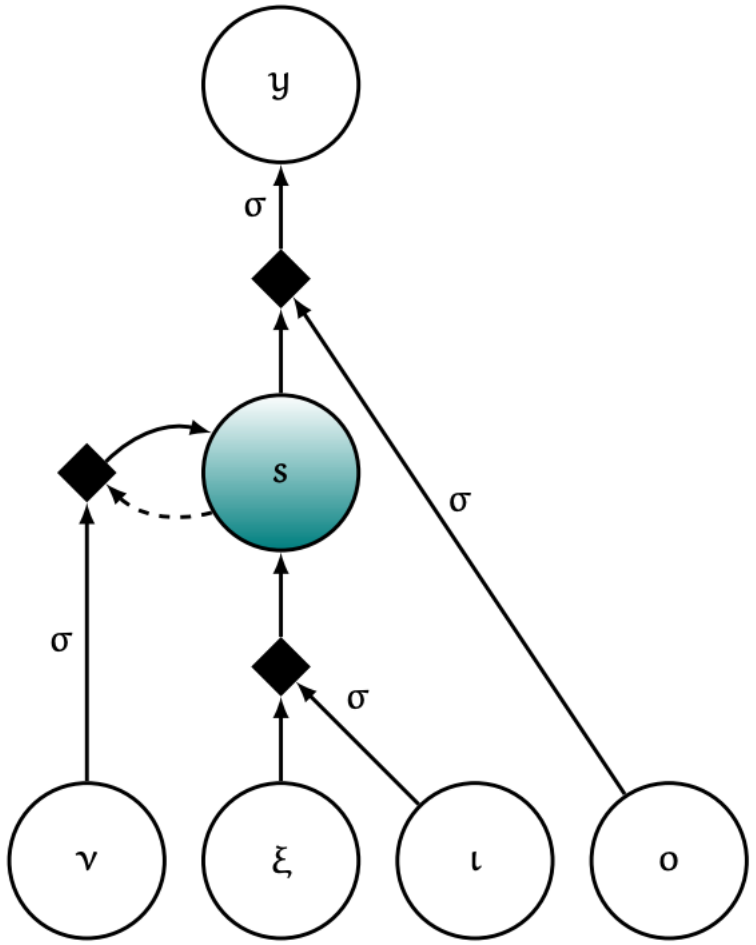
$$h_t = \sigma(wh_{t-1})$$

$$\frac{\partial h_{t'}}{\partial h_t} = \prod_{k=1}^{t'-t} w\sigma'(wh_{t'-k})$$

$$= \underbrace{w^{t'-t}}_{!!!} \prod_{k=1}^{t'-t} \sigma'(wh_{t'-k})$$

AIRI

LSTM



1

$$[\xi_t\ \iota_t\ \nu_t\ o_t] = a_t$$

$$s_t = \underbrace{\phi(\iota_t, \xi_t)}_{\text{input gate}} + \underbrace{\phi(\nu_t, s_{t-1})}_{\text{forget gate}}$$

$$y_t = \sigma(\underbrace{\phi(o_t, s_t)}_{\text{output gate}})$$

2

$$\begin{aligned}\frac{\partial s_t}{\partial s_{t-1}} &= \frac{\partial \phi(\nu_t, s_{t-1})}{\partial s_{t-1}} \\ &= \frac{\partial \sigma(\nu_t) s_{t-1}}{\partial s_{t-1}} \\ &= s_{t-1} \underbrace{\frac{\partial \sigma(\nu_t)}{\partial s_{t-1}}}_{=0} + \underbrace{\frac{\partial s_{t-1}}{\partial s_{t-1}}}_{=1} \sigma(\nu_t) \\ &= \sigma(\nu_t),\end{aligned}$$

3

$$\frac{\partial s_{t'}}{\partial s_t} = \prod_{k=1}^{t'-t} \sigma(\nu_{t+k}).$$

https://mediatum.ub.tum.de/doc/1256381/1256381.pdf

AIRI

LSTM. Pros and cons

- Mitigate the vanishing/exploding gradient problem in RNN
- With its Forget Gate architecture, LSTMs retain tokens that have higher value
- LSTMs are more suitable for longer sequences than RNN

- The additional gates in LSTM and classifying importance makes computations using LSTM more complex
- Due to complexity, LSTMs require higher computational cost and time

Get rid of RNNs in MT?

→ RNNs are slow, because not parallelizable over timesteps

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

Vaswani et al, 2017. Attention is all you need.

AIRI

Transformer

Attention is all you need =) 2017

Previously:

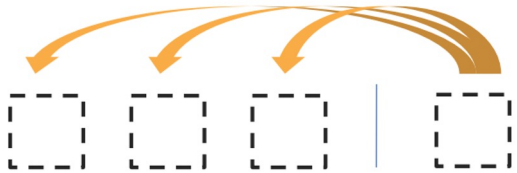
→ RNN encoder + RNN decoder, interaction via fix-sized vector

→ RNN encoder + RNN decoder, interaction via attention

NOW:

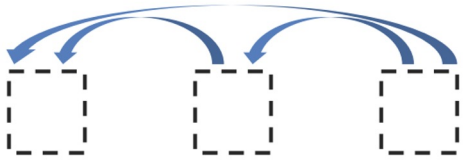
→ **attention + attention + attention**

AIRI



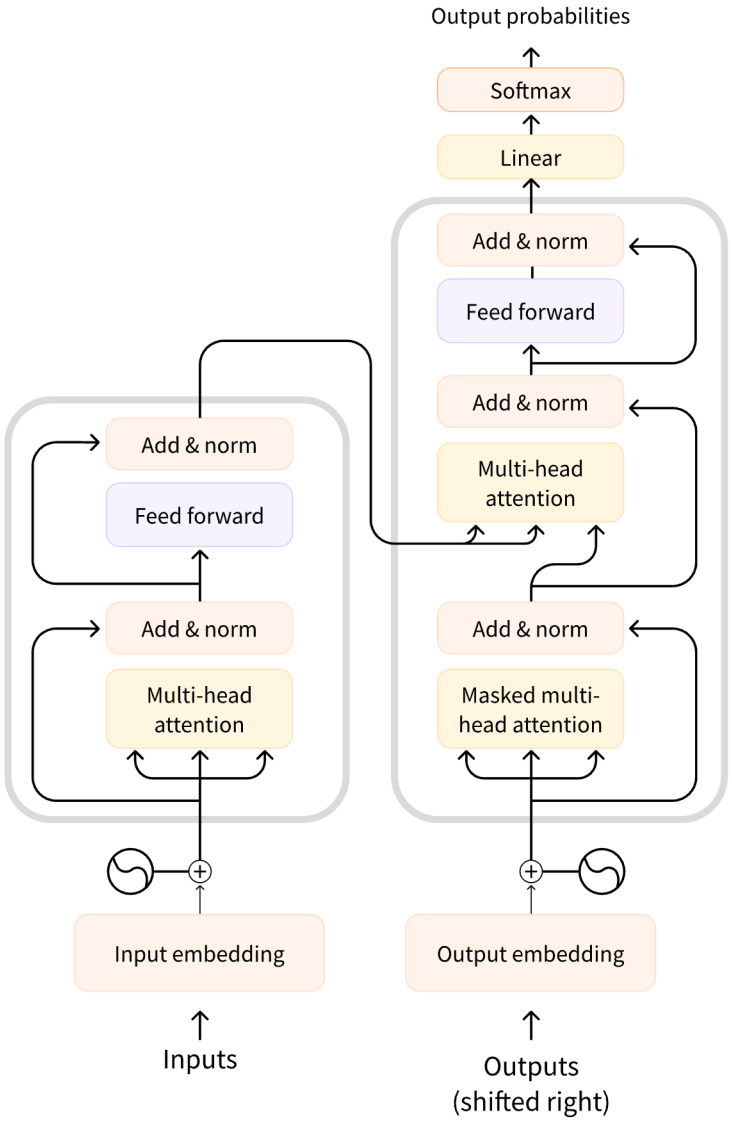
Encoder-Decoder Attention

Encoder Self-Attention



MaskedDecoder Self-Attention

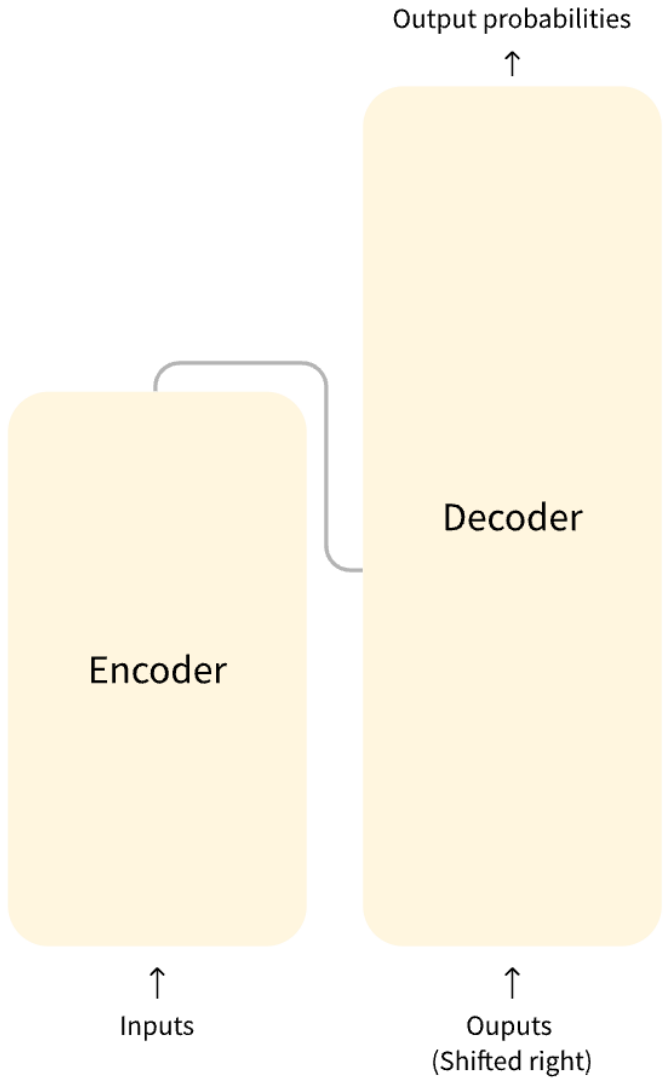
attention + attention + attention



Vaswani et al, 2017. Attention is all you need.

Transformer: high-level

- **Encoder**: The encoder receives an input and builds a representation of it (its embeddings). The model is optimized to acquire understanding from the input
- **Decoder**: The decoder uses the encoder's representation (embeddings) along with other inputs to generate a target sequence. The model is optimized for generating outputs



Transformer types

- **Encoder-Decoder Transformers** (*sequence-to-sequence transformers*)
  - Encode an input sequence and decode it into an output sequence
  - The original transformer model and the Text-to-Text Transfer Transformer (T5) model

- **Encoder-Only Transformers**
  - Only encode input and do not undertake decoding
  - Bidirectional Encoder Representations from Transformers (BERT) by Google, as well as its many variations like RoBERTa

- **Decoder-Only Transformers**
  - Specialize in decoding input into output
  - Generative Pre-trained Transformer (GPT) family of models by OpenAI, e.g., ChatGPT

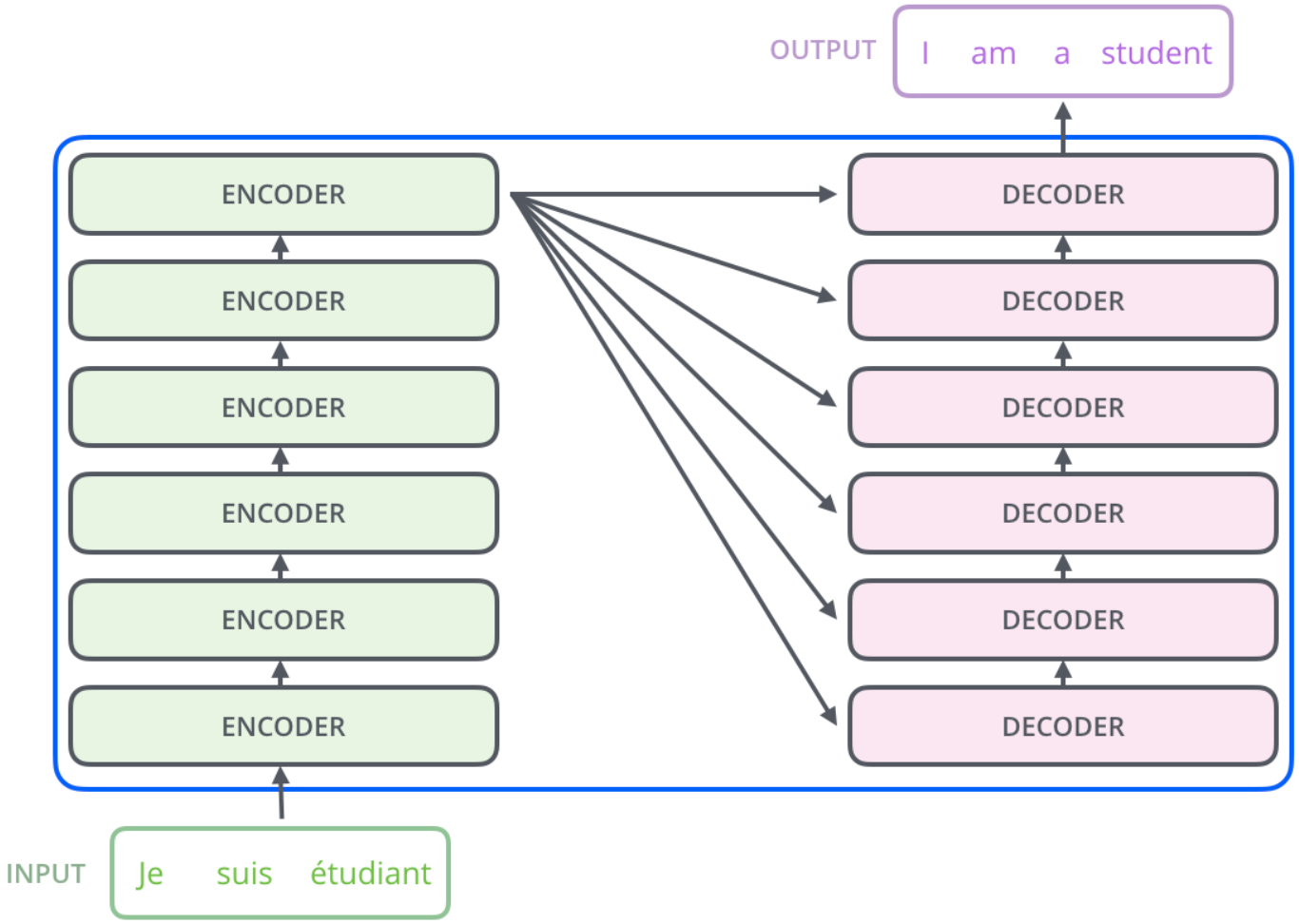
Transformer. Main blocks

- **Embedding Layer**
  - Input enters the transformer, which breaks it down into tokens
  - Transform tokens into embeddings

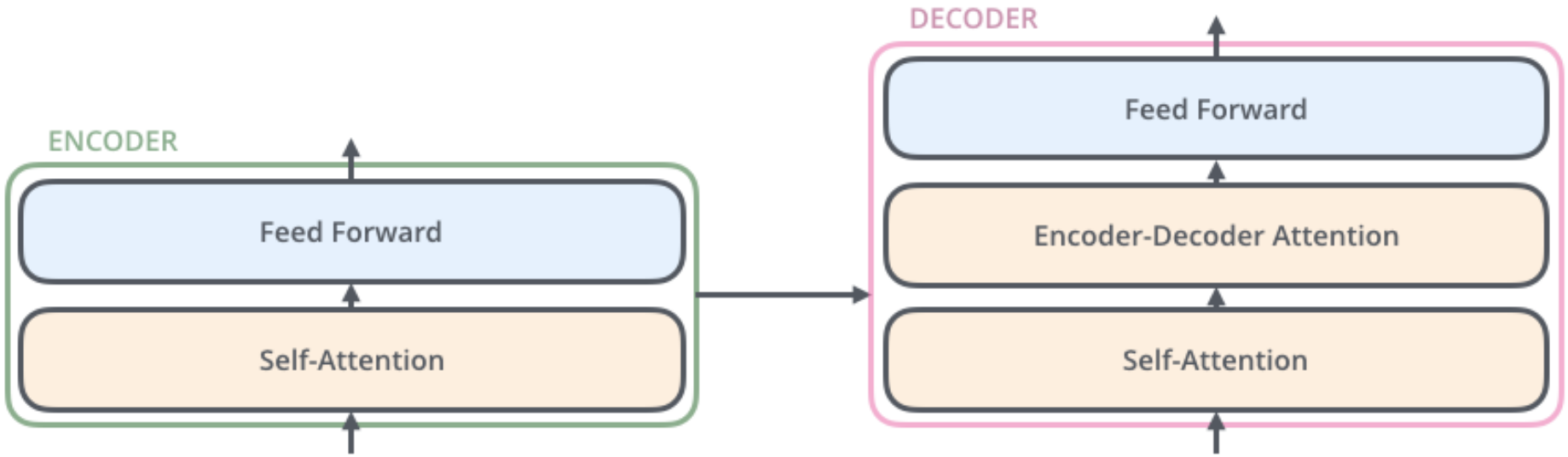
- **Positional Encoder**
  - Adds information to each token's embedding to indicate its position within the sequence – without recurrence or maintaining an internal state
  - Typically achieved by using an alternating set of sine and cosine functions to generate a unique positional signal for each token
  - Sine and cosine functions repeat their patterns over a regular interval, which is ideal for capturing sequential relationships, while being perpendicular to each other (preventing overlap)

- **Self-Attention Mechanism**
  - Systematically compares token embeddings against each other to determine their similarity and relevance
  - Result — a weighted representation of the input which captures the appropriate patterns and relationships between the tokens (further calculation of the most probable output)

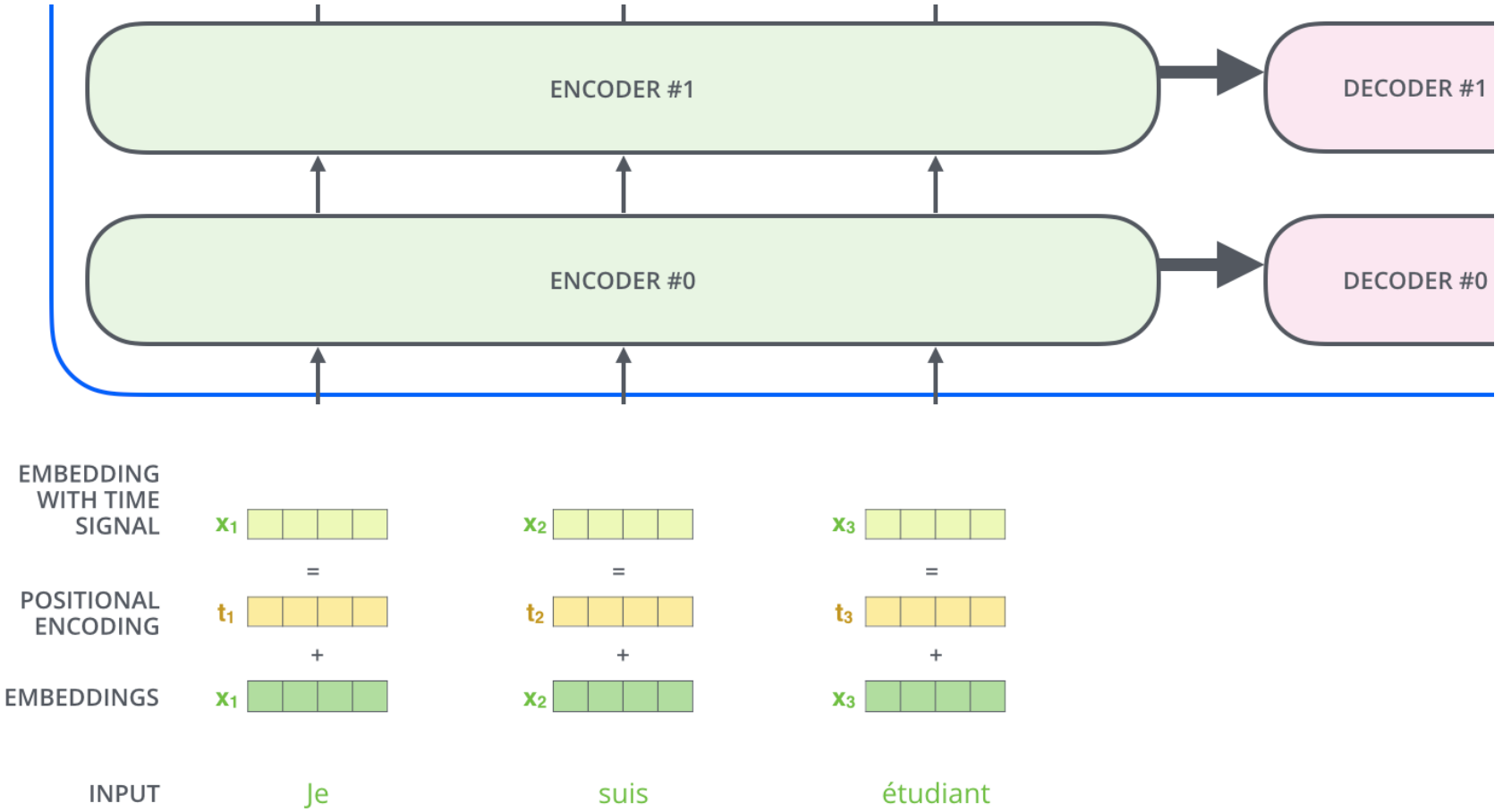
Transformer: high-level



Transformer: high-level

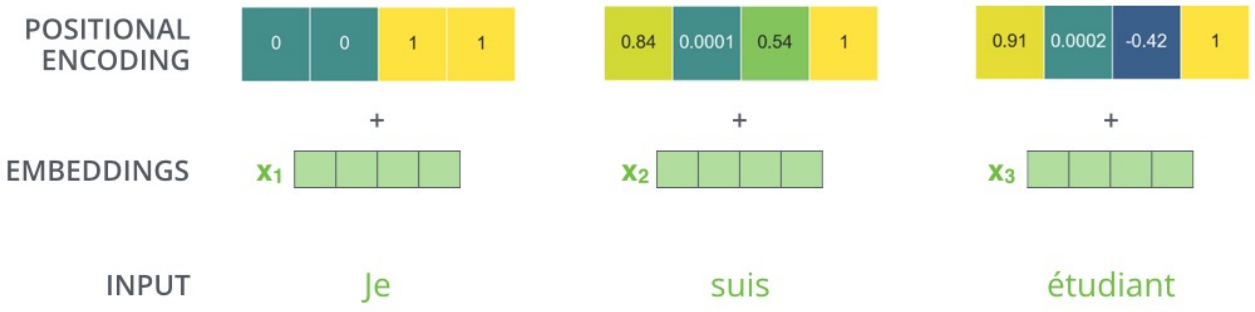


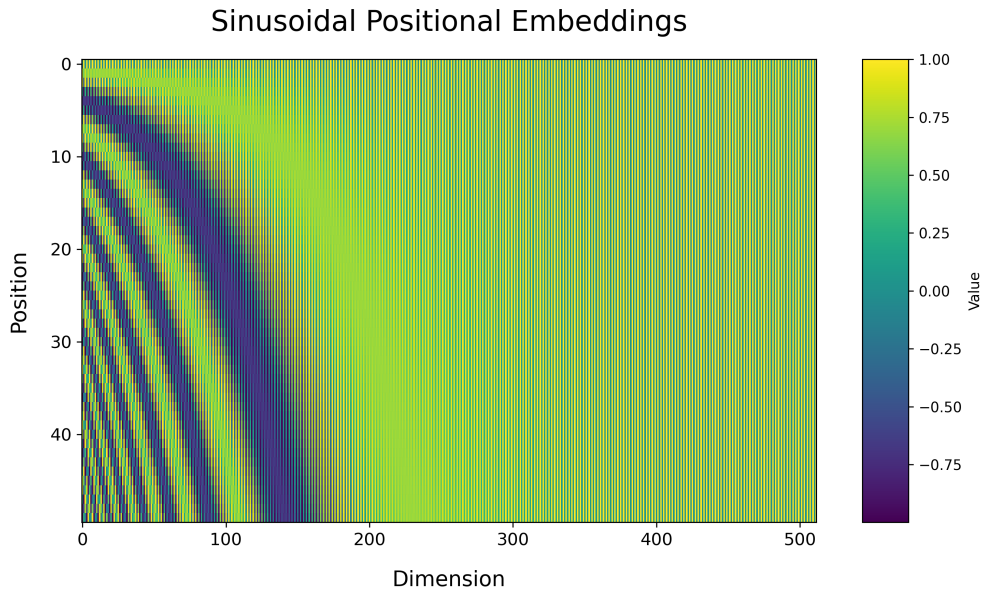
Transformer: high-level



Transformer. Positional Encoding

Positional encoding provides *order information* to the model





The fixed positional encodings used in Transformers

$$\mathrm{PE}(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

AIRI

Transformer. Positional Encoding

$$\mathrm{PE}(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

The original paper suggests using 10000 as base theta value, and the Llama 2 model uses this value. Newer versions of Llama (3 and higher) started to use 500000 as base theta value.

✓ Limitation: not effective with longer sequences than those seen during training

Rotary Positional Embeddings

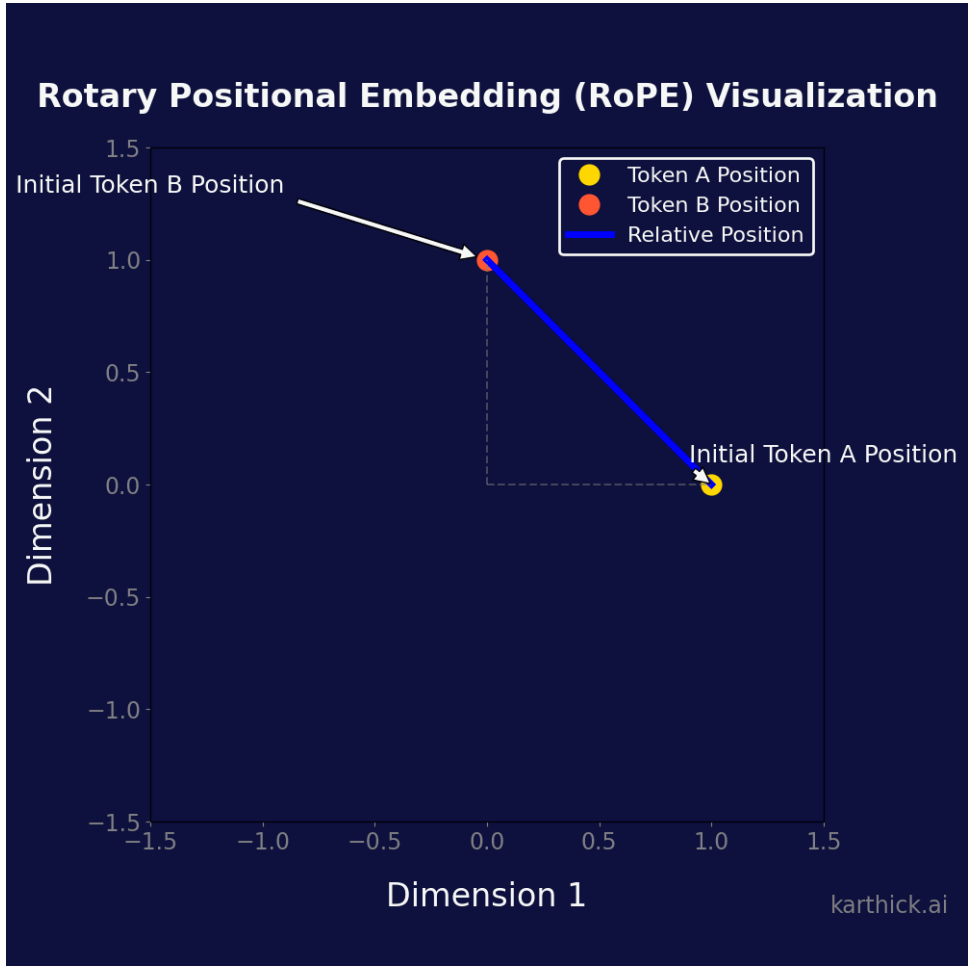
→ Encode positions by rotating the embedding vectors in a multidimensional space
→ Uses a rotation matrix to alter the representation of each token in a geometric way
→ A more natural way to represent sequential information

$$\mathrm{RoPE}(x_{pos}) = x_{pos} \cdot \cos(\theta_{pos}) + \hat{x}_{pos} \cdot \sin(\theta_{pos})$$

x_{pos} — original embedding vector
$\hat{x}_{pos}$ — x_{pos} rotated by the angle θ_{pos} in the embedding space
θ_{pos} — a predetermined function of the position, ensuring a unique rotation for each token

RoFormer: https://arxiv.org/abs/2104.09864v5

Transformer. RoPE



https://karthick.ai/blog/2024/Rotatory-Position-Embedding-(RoPE)

Transformer. RoPE. Advantages

- **Long-Range Context:** RoPE adeptly captures relationships between tokens across lengthy sequences, a challenge for conventional positional embeddings
- **Rotation Invariance:** By design, RoPE maintains effectiveness irrespective of sequence length, addressing a limitation of sine-cosine embeddings
- **Interpretability:** The rotational approach offers an intuitive geometric interpretation of how positional information influences the attention mechanism

Transformers. Self-attention

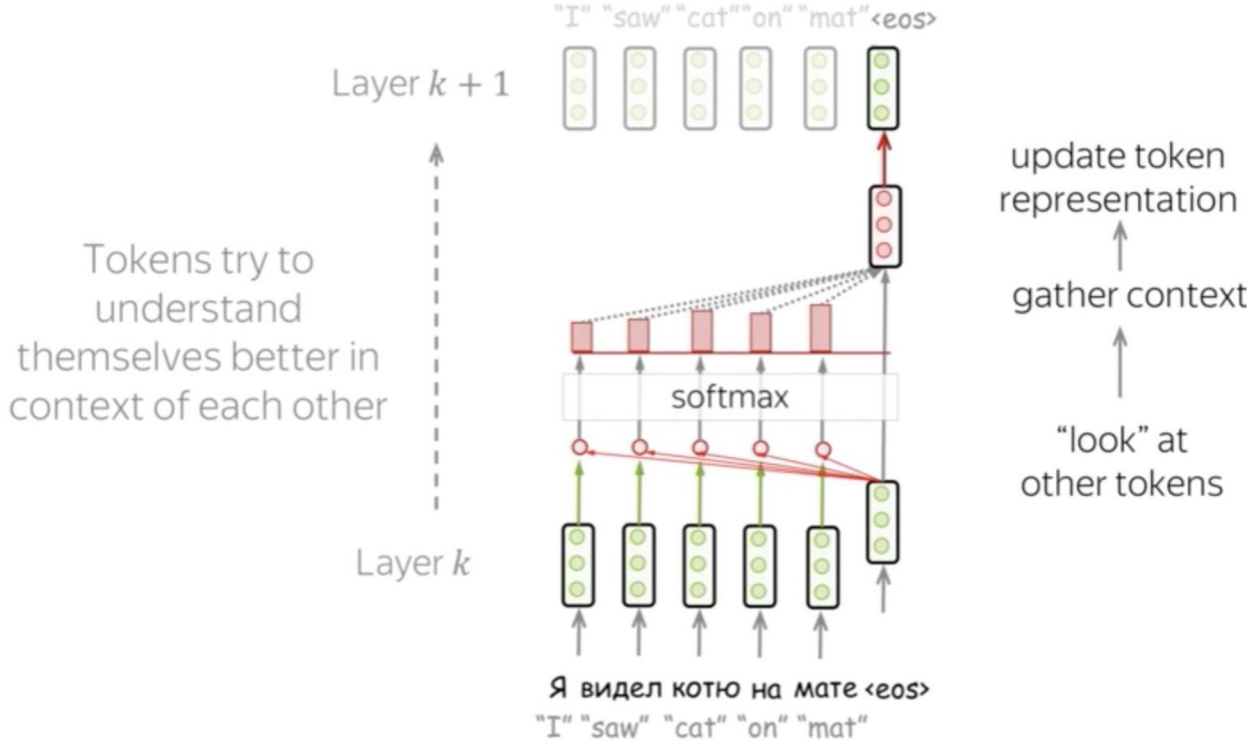
Previously: one decoder state looked at all encoder states

NOW: each state looks at each other states

Self-attention

- → tokens interact with each other
- → each token "looks" at other tokens
- → gathers context
- → updates the previous representation of "self"

In Parallel!



https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Query, Key and Value vectors

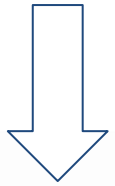
Query, Key and Value vectors

Each vector gets **three representations**:

- **Query** — asking for information
- **Key** — saying that it has some information
- **Value** — giving the information

These matrices allow different aspects of the x vectors to be used/emphasized in each of the three roles

Attention matches the key and query by assigning a value to the place the key is most likely to be.



Attention weights

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V}$$

AIRI

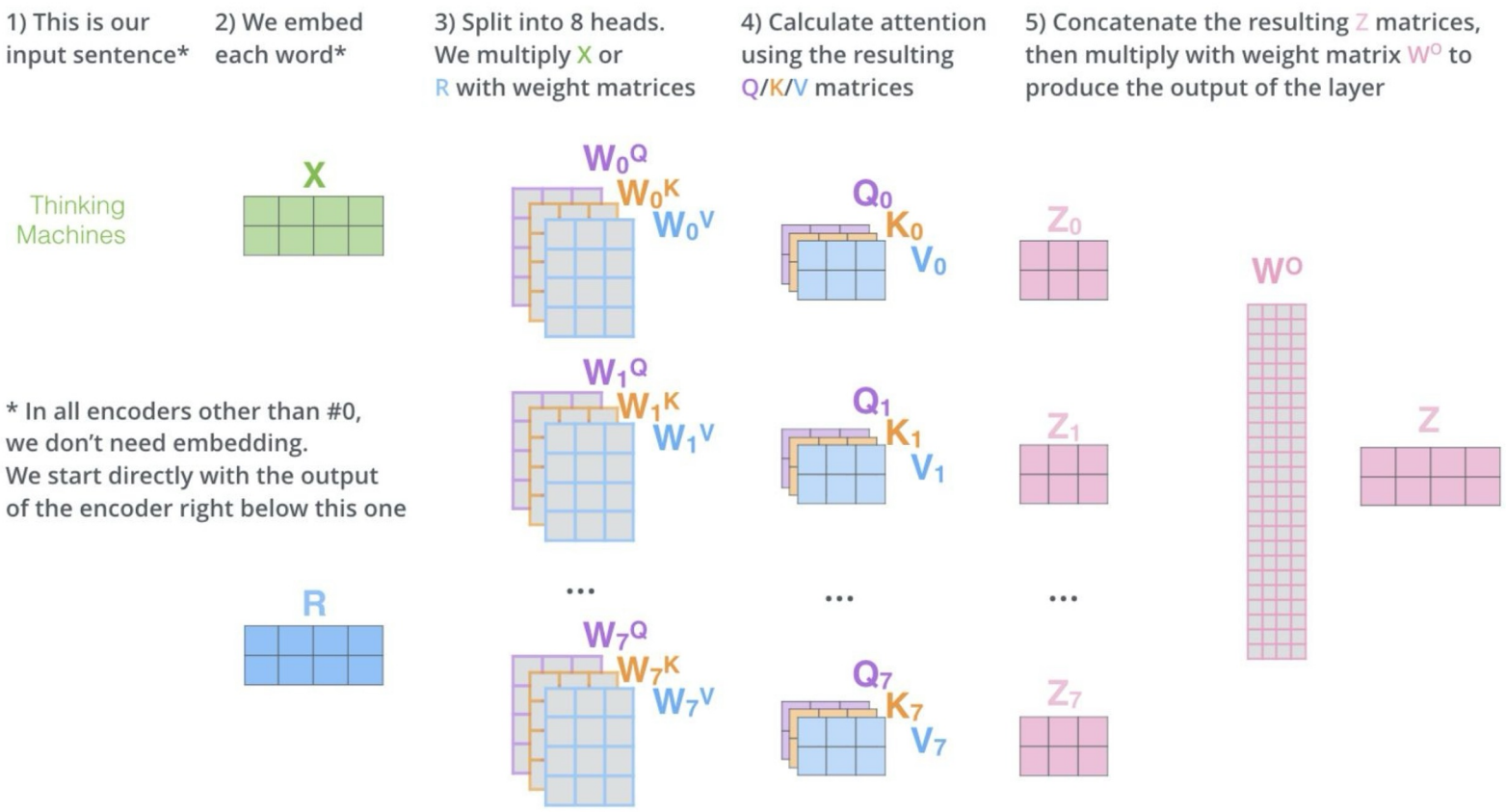
Masked self-attention

Decoder has different self-attention => **Masked self-attention**

- → we generate one token at a time => during generation, we don't know which tokens we'll generate in future
- → to enable parallelization we forbid the decoder to look ahead — future tokens are masked out (setting them to -inf) before the Softmax step in the self-attention calculation

Multi-head attention

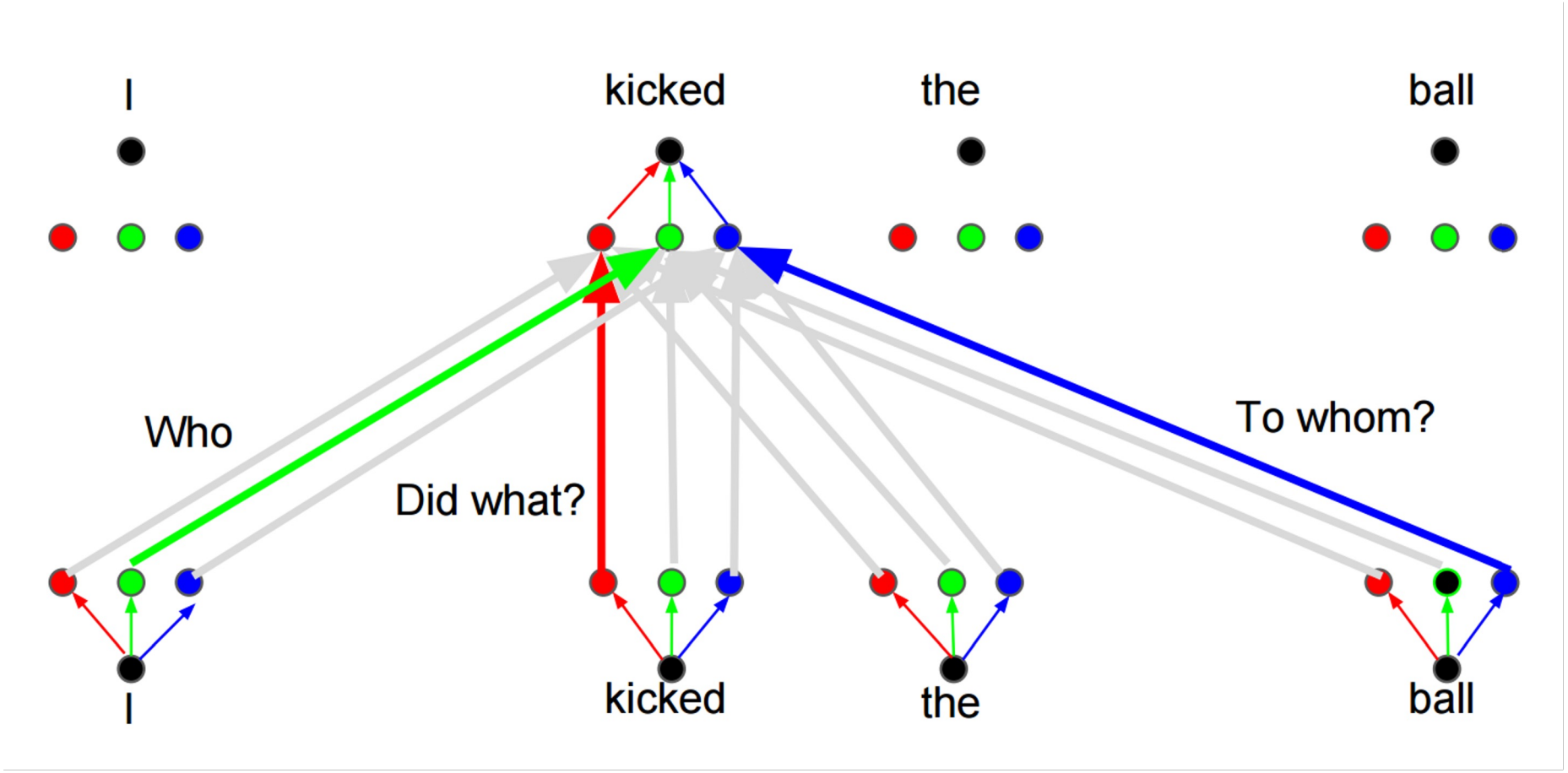
- We need to know different relationships between tokens in a sentence: syntactic relationships, lexical preferences, order, grammar issues like case or gender agreement
- Instead of having one attention mechanism, **multi-head attention** has several "heads" which **work independently** and focus on different things



$$\text{MultiHead}\,(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}\,(\text{head}_1,\ldots,\text{head}_h)\mathbf{W}^O$$

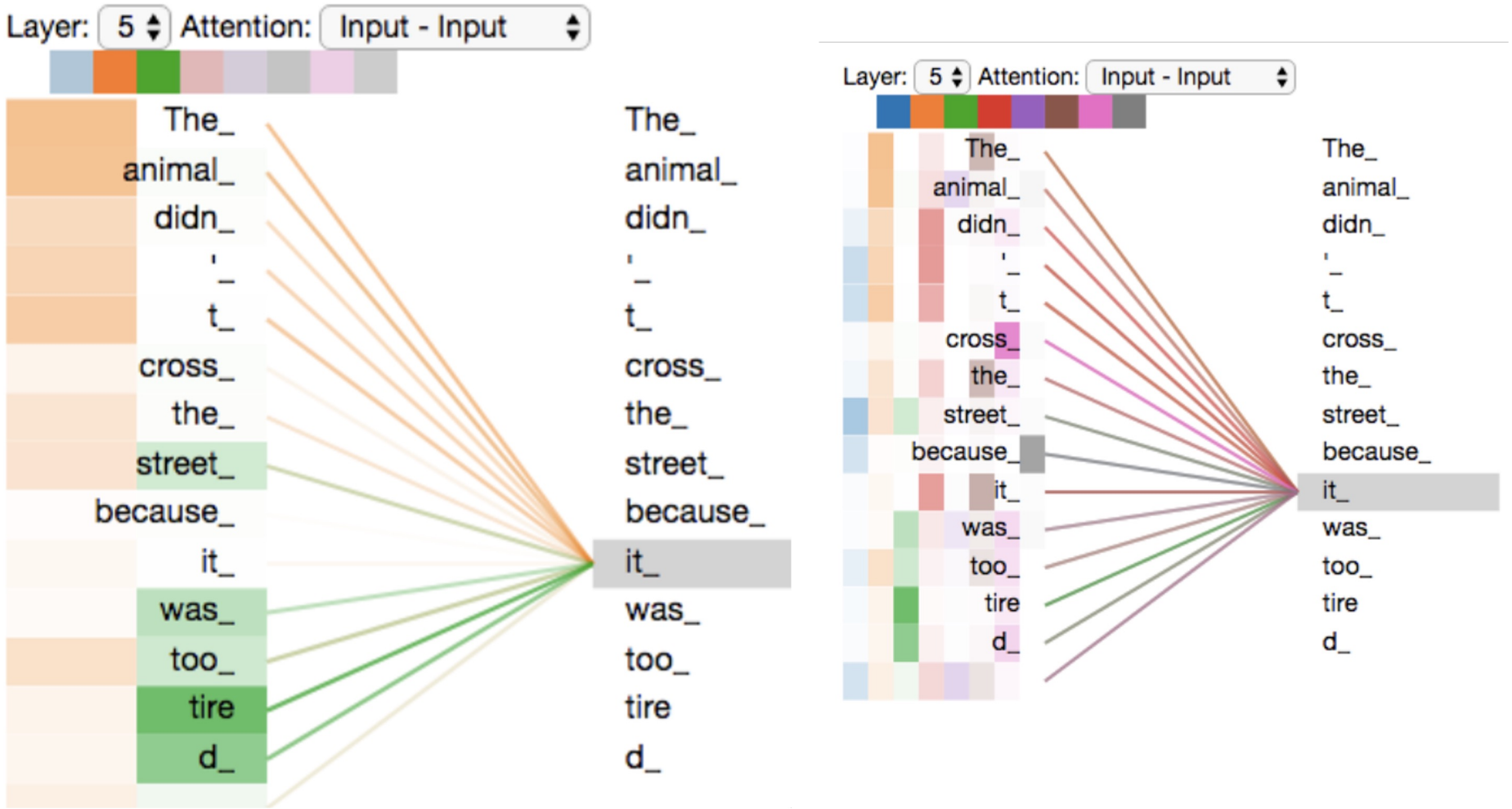
$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V\right)$$

Multi-head attention



Ashish Vaswani and Anna Huang. Self-Attention For Generative Models

Multihead self-attention in encoder



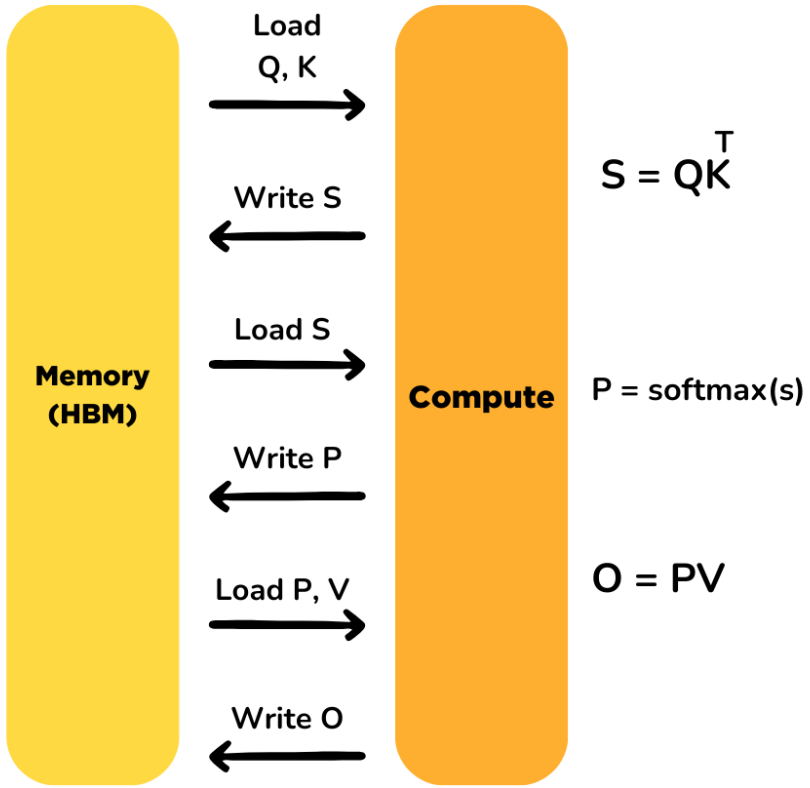
Jay Alammar. The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/

AIRI

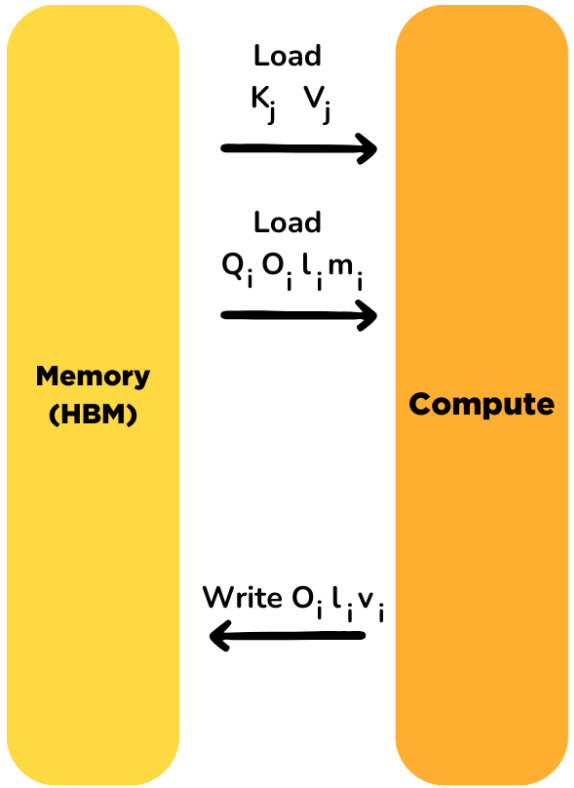
32

Flash attention

Standard Attention Implementation



Flash Attention



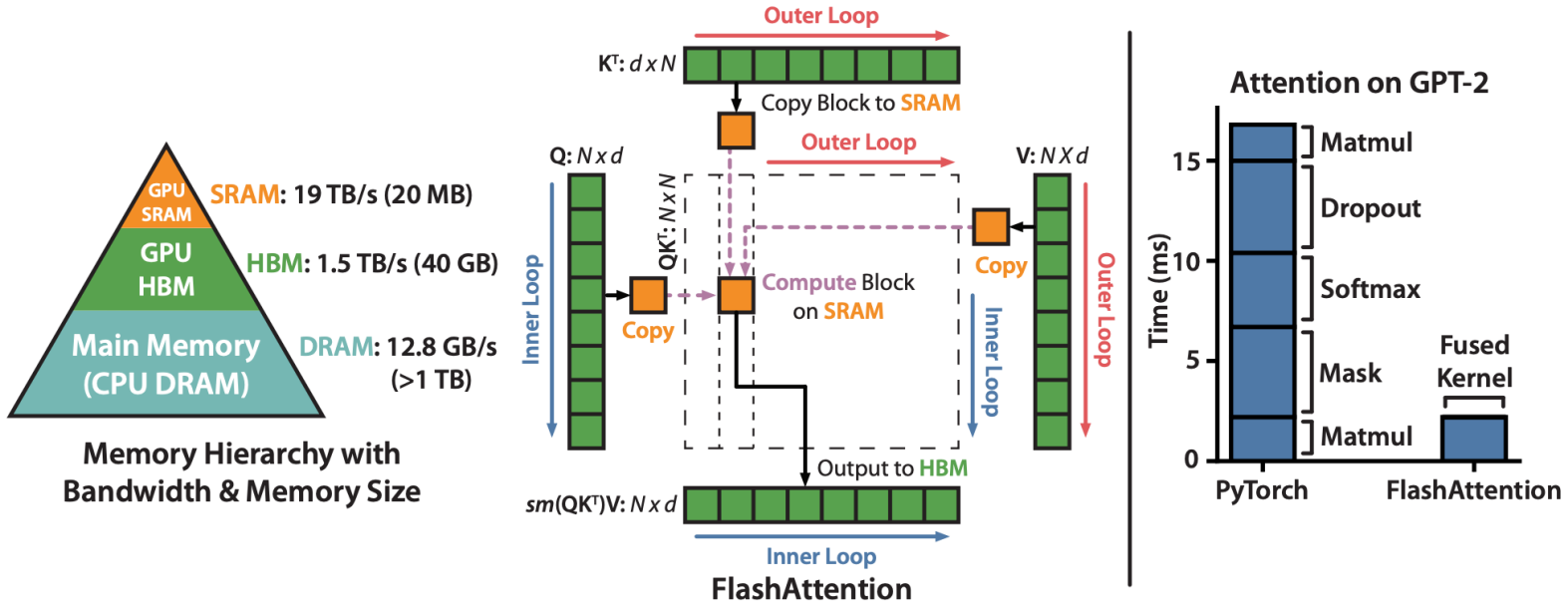
Kernel operations fused together, reducing reads & writes

$S_{ij} = Q_i K_j^T$
$\tilde{m}$ = rowmax of S
P = exp(s - m)
l = rowsum of P
$m = \max(\tilde{m}_{ij}, m)$
calculate O from l & m

Initialize O, l and m matrices with zeroes. m and l are used to calculate cumulative softmax. Divide Q, K, V into blocks (due to SRAM's memory limits) and iterate over them, for i is row & j is column.

AIRI

Flash attention (1, 2, 3, ...)



- Massive increase in LLM context length in the last two years: from 2-4K (GPT-3, OPT) to 128K (GPT-4) and even 1M (Llama 3)
- FA has yet to take advantage of new capabilities in modern hardware
- FA-2 achieve only 35% utilization of theoretical max FLOPs on the H100 GPU
- 3 main techniques to speed up attention
  - Overlap overall computation and data movement via warp-specialization
  - Interleave block-wise MatMul and Softmax operations
  - Incoherent processing that leverages hardware support for FP8 low-precision

Feed-forward block

- Goes after the self-attention mechanism and normalization
- Captures the input sequence's higher-level features to learn more intricate relationships from the data

It is composed of 3 layers:

- **Linear Transformation:** each token is multiplied by a weight matrix and added to a bias vector (both learned through training) allowing it to better fit the data and learn its more complex underlying relationships
- **Activation Function:** this introduces non-linearity into the network, further enabling it to model complex patterns that mirror relationships in real-world relationships – which are not simply linear. The most commonly used activation function within transformer architectures is the Rectified Linear Unit (ReLU), which works by directly outputting the input when it is a positive value while outputting zero if it is negative – creating a non-linear relationship between input and output
- **Linear Transformation:** similar to the first layer transformation, but with its own set of weights and biases

Feed-forward block

Feed-forward blocks

Each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

Residual connection (train better)

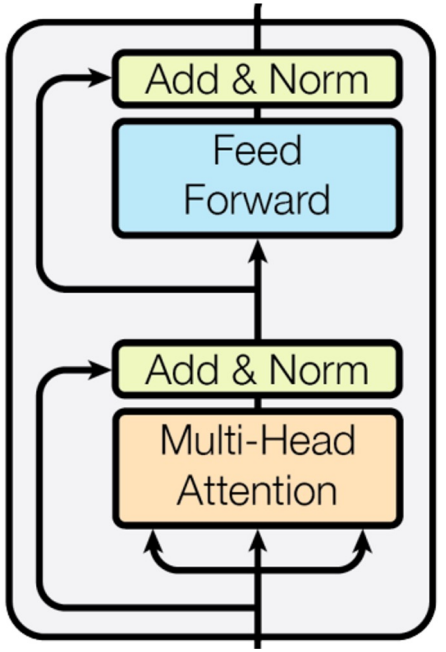
Residual connections => add an input of the block to its output
Ease the gradient flow through a network and allow stacking a lot of layers

Residuals

LayerNorm(x + dropout(Sublayer(x))

```
class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """
    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))
```



Rush. The Annotated Transformer. https://nlp.seas.harvard.edu/2018/04/03/attention.html

AIRI

Feed-forward block

Feed-forward blocks

Each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

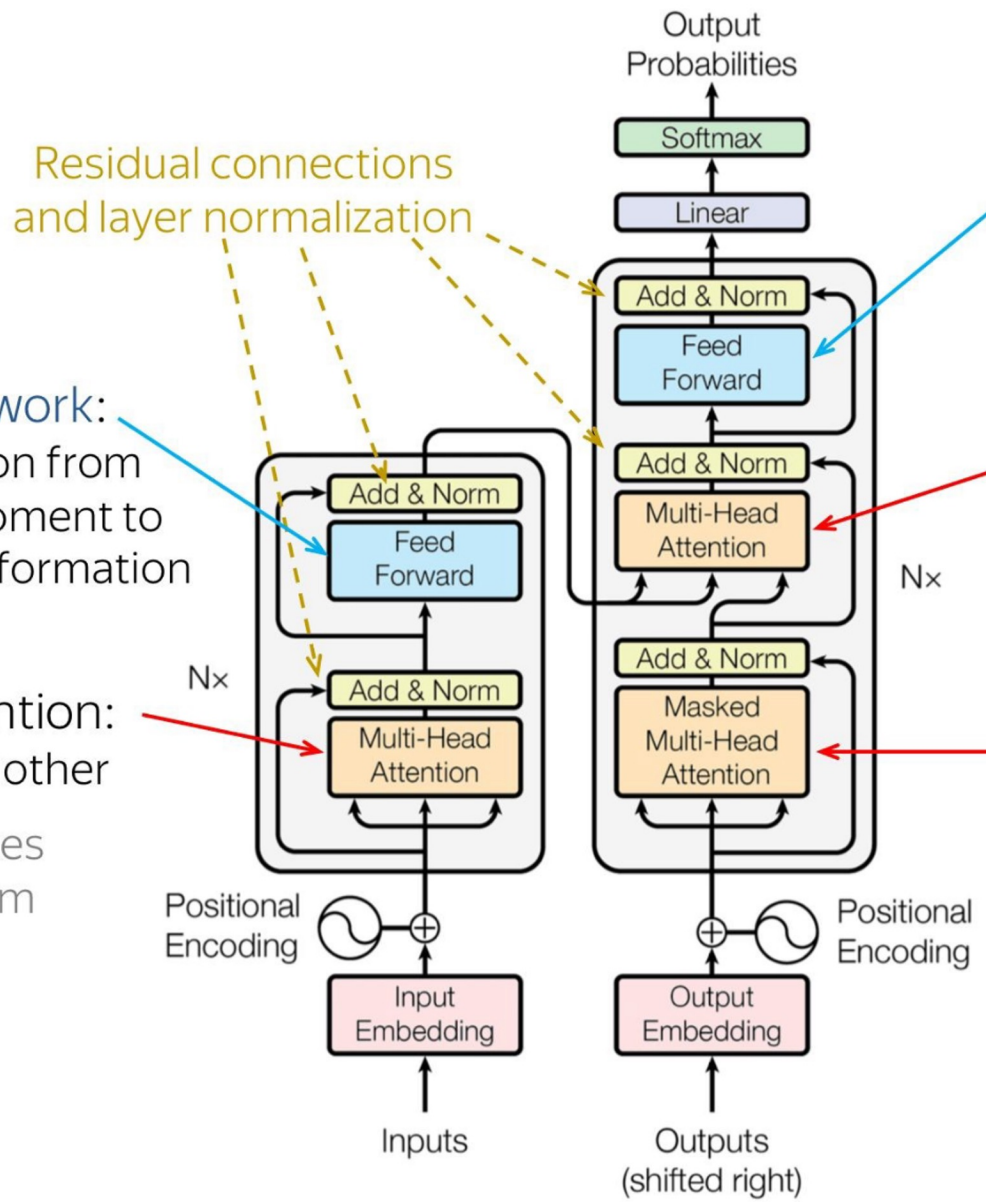
Residual connection (train better)

Residual connections => add an input of the block to its output
Ease the gradient flow through a network and allow stacking a lot of layers

Layer Normalization (train faster)

Improves convergence
Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer

AIRI



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

AIRI

Perplexity

Perplexity is defined as the exponential of the negative log-likelihood of a sequence

$$\text{Perplexity} = \exp\left(-\frac{1}{t}\sum_{i}^{t}\log p_\theta(x_i|x_{\text{context}})\right)$$

- → A lower perplexity is desirable
- → Since perplexity a relative metric with no upper bound, it is best used to compare similar models and configurations on a specific task and dataset to look for improvements in model performance

Limitations

- → Perplexity is sensitive to the size and diversity of datasets
- → Perplexity is hard to absolutely interpret. Unlike metrics such as BLEU or BERTScore, it only provides a measure of "confidence" rather than a measure of semantic or textual similarity
- → Perplexity does not capture the diversity of language usage (i.e. emotions and expressions). A model with low perplexity may still generate repetitive or uninteresting text, which isn't necessarily desirable

02

Open source LLMs

LLaMA

- A collection of foundation language models ranging from 7B to 65B parameters (*7B, 13B, 33B, 65B*)

- Based on the Transformer Decoder (aka GPT-3) with slight modifications

- Available upon request

- All models are pretrained on open source datasets

- Immense training dataset of 1.4T

Pre-normalization [GPT3]. To improve the training stability, we normalize the input of each transformer sub-layer, instead of normalizing the output. We use the RMSNorm normalizing function, introduced by Zhang and Sennrich (2019).

SwiGLU activation function [PaLM]. We replace the ReLU non-linearity by the SwiGLU activation function, introduced by Shazeer (2020) to improve the performance. We use a dimension of $\frac{2}{3}4d$ instead of $4d$ as in PaLM.

Rotary Embeddings [GPTNeo]. We remove the absolute positional embeddings, and instead, add rotary positional embeddings (RoPE), introduced by Su et al. (2021), at each layer of the network.

https://arxiv.org/abs/2302.13971

AIRI

LLaMA

- A collection of foundation language models ranging from 7B to 65B parameters
- LLaMA-13B outperformed GPT-3 (175B) on many tasks
- Model weights are available upon request

- Weights for the LLaMA models can be obtained from by filling out this form
- After downloading the weights, they will need to be converted to the Hugging Face Transformers format using the conversion script. The script can be called with the following (example) command:

```
python src/transformers/models/llama/convert_llama_weights_to_hf.py \
    --input_dir /path/to/downloaded/llama/weights --model_size 7B --output_dir
```

- After conversion, the model and tokenizer can be loaded via:

```
from transformers import LlamaForCausalLM, LlamaTokenizer

tokenizer = LlamaTokenizer.from_pretrained("/output/path")
model = LlamaForCausalLM.from_pretrained("/output/path")
```

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

https://arxiv.org/abs/2302.13971

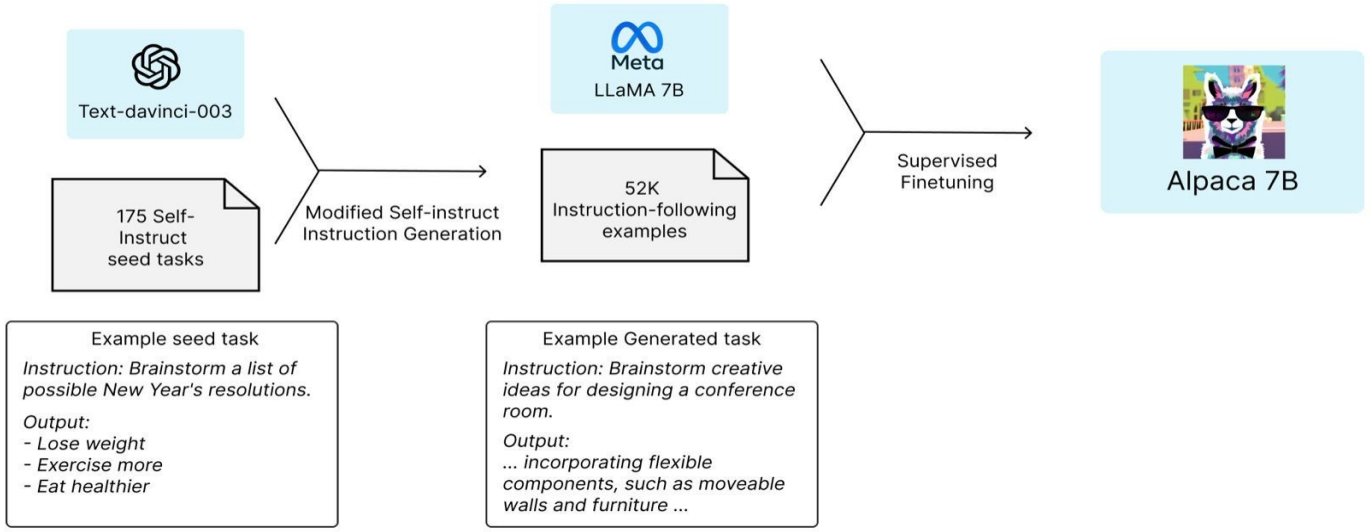
AIRI

Alpaca

Stanford Alpaca

- Fine-tuned from the LLaMA 7B model on 52K instruction-following demonstrations

- Train on 52K instruction-following demonstrations generated in the style of self-instruct using *text-davinci-003*



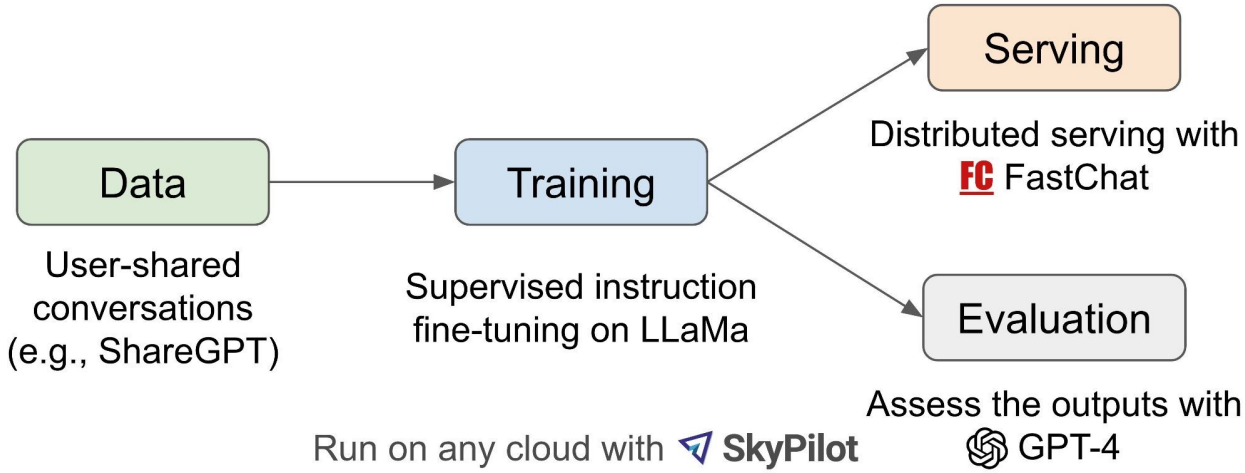
https://crfm.stanford.edu/2023/03/13/alpaca.html

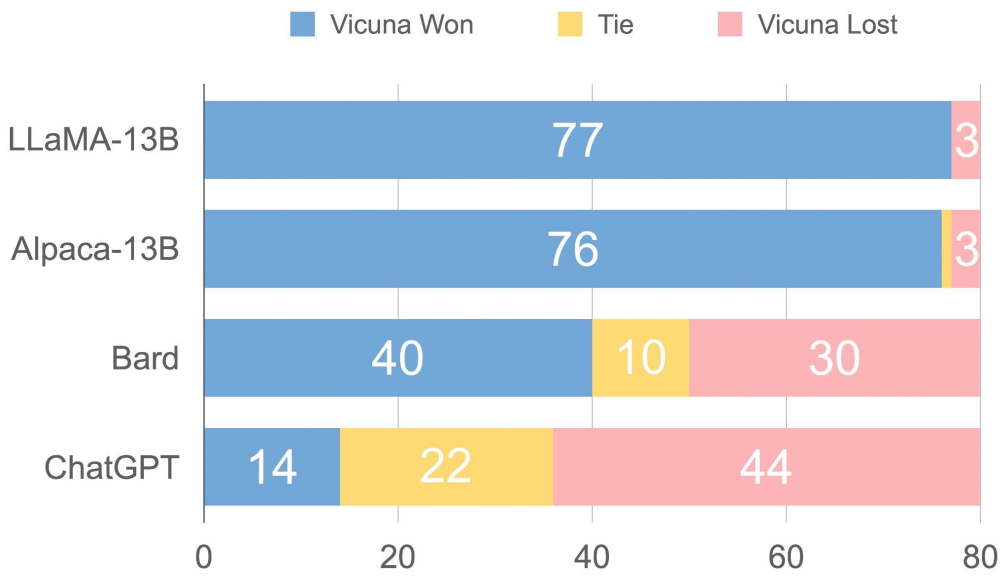
AIRI

Vicuna

- An open-source chatbot trained by fine-tuning LLaMA on user-shared conversations collected from ShareGPT

- Preliminary evaluation using GPT-4 as a judge shows Vicuna-13B achieves more than 90% quality of OpenAI ChatGPT and Google Bard





https://lmsys.org/blog/2023-03-30-vicuna/

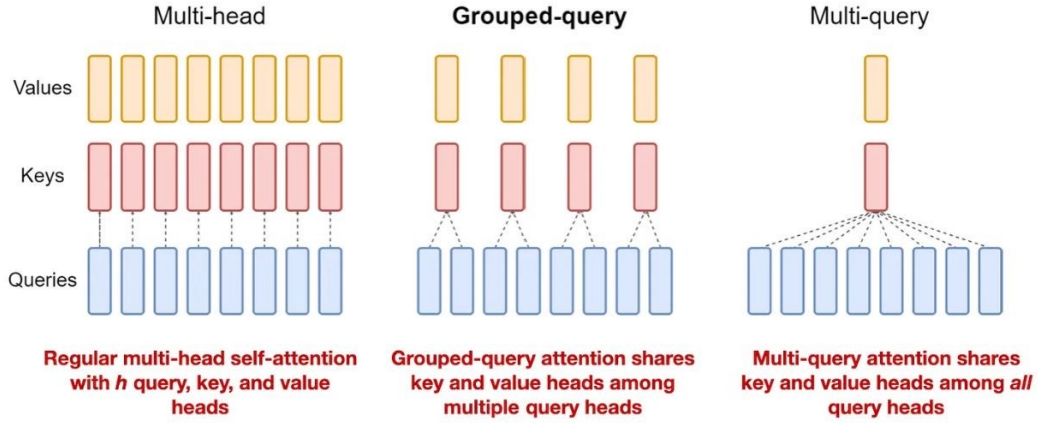
LLaMA 2

- A collection of foundation language models ranging from 7B to 70B parameters (7B, 13B, 34B, 70B)

- Available upon request of HF

- Based on LLaMA, with slight differences:
  - *Trained on cleaned dataset **40% larger** (2T tokens)*
  - *Doubled the context length of LLaMA (4 096 tokens),*
  - *Use grouped query attention for fast 70B model inference*

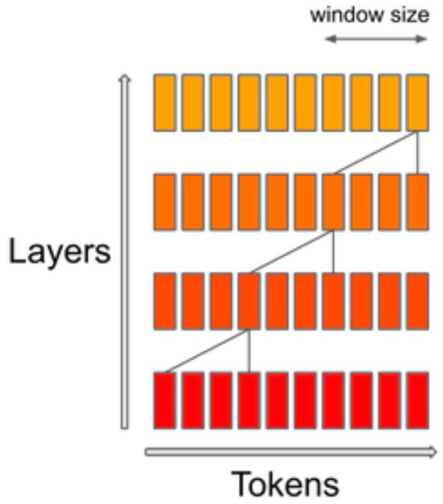
MODEL SIZE (PARAMETERS)	PRETRAINED	FINE-TUNED FOR CHAT USE CASES
7B	Model architecture:	Data collection for helpfulness and safety:
13B	Pretraining Tokens: 2 Trillion	Supervised fine-tuning: Over 100,000
70B	Context Length: 4096	Human Preferences: Over 1,000,000

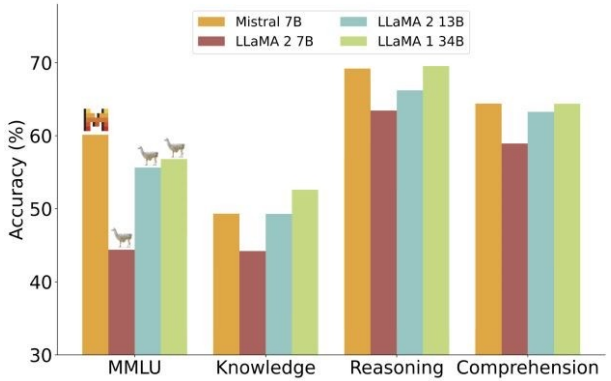


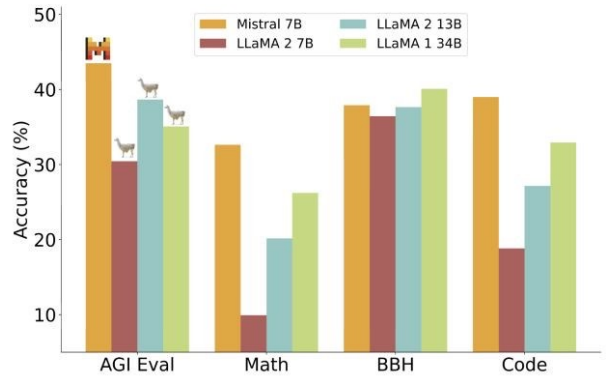
Mistral

Mistral 7B is a 7.3B parameter model based on the Transformer Decoder:

- Outperforms Llama 2 13B on all benchmarks
- Outperforms Llama 1 34B on many benchmarks
- Approaches CodeLlama 7B performance on code, while remaining good at English tasks
- Uses Grouped-query attention (GQA) for faster inference
- Uses Sliding Window Attention (SWA) to handle longer sequences at smaller cost







https://huggingface.co/docs/transformers/main/model_doc/mistral

Mistral

Three model versions:

- A base model Mistral-7B-v0.1 has been pre-trained to predict the next token on internet-scale data.

- An instruction tuned model (Mistral-7B-Instruct-v0.1) which is the base model optimized for chat purposes using supervised fine-tuning (SFT) and direct preference optimization (DPO).

- An improved instruction tuned model (Mistral-7B-Instruct-v0) which improves upon v1.

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("mistralai/Mistral-7B-v0.1", device_map="auto")
tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-v0.1")

prompt = "My favourite condiment is"

model_inputs = tokenizer([prompt], return_tensors="pt").to("cuda")
model.to(device)

generated_ids = model.generate(**model_inputs, max_new_tokens=100, do_sample=True)
tokenizer.batch_decode(generated_ids)[0]
```

```
"My favourite condiment is to ..."
```

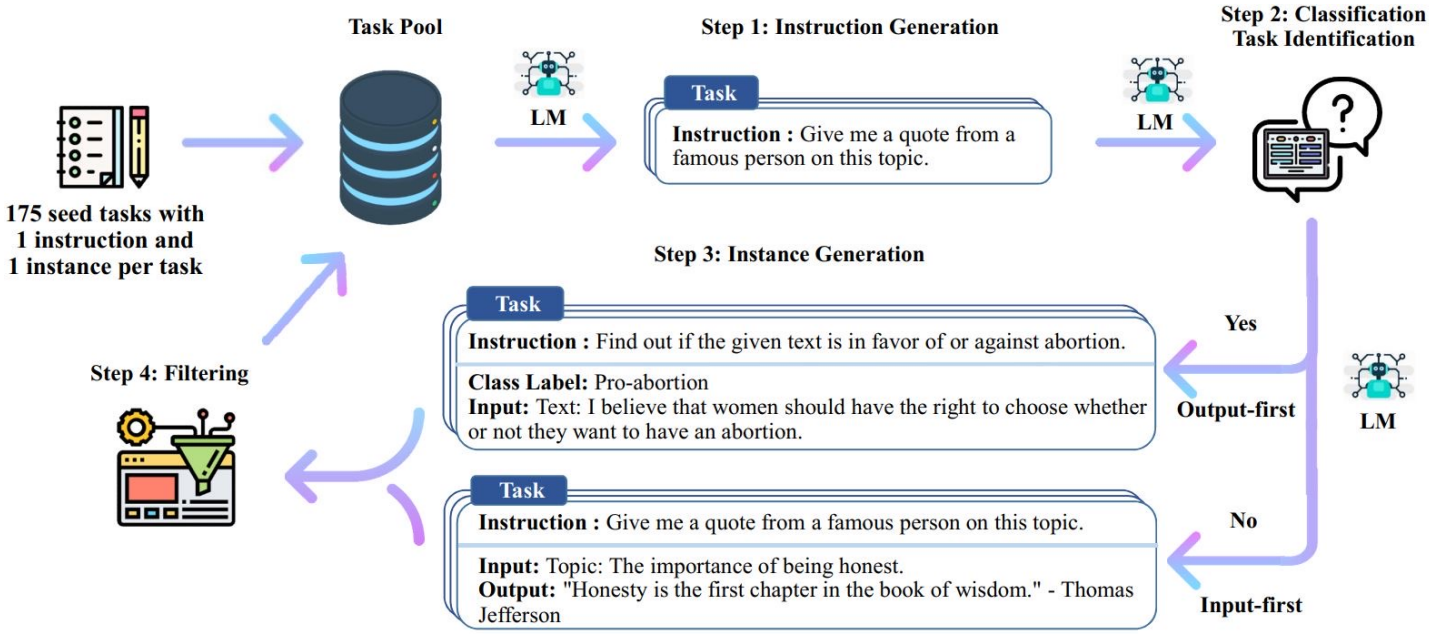
https://huggingface.co/docs/transformers/main/model_doc/mistral

03

Empowering LLM

Self-Instruct

- The Self-Instruct process is an **iterative bootstrapping** algorithm that starts with a seed set of manually-written instructions and uses them to prompt the language model to generate new instructions and corresponding input-output instances
- Generations are then filtered to remove low-quality or similar ones, and the resulting data is added back to the task pool
- This process can be repeated multiple times, resulting in a large collection of instructional data that can be used to fine-tune the language model to follow instructions more effectively



https://arxiv.org/abs/2212.10560

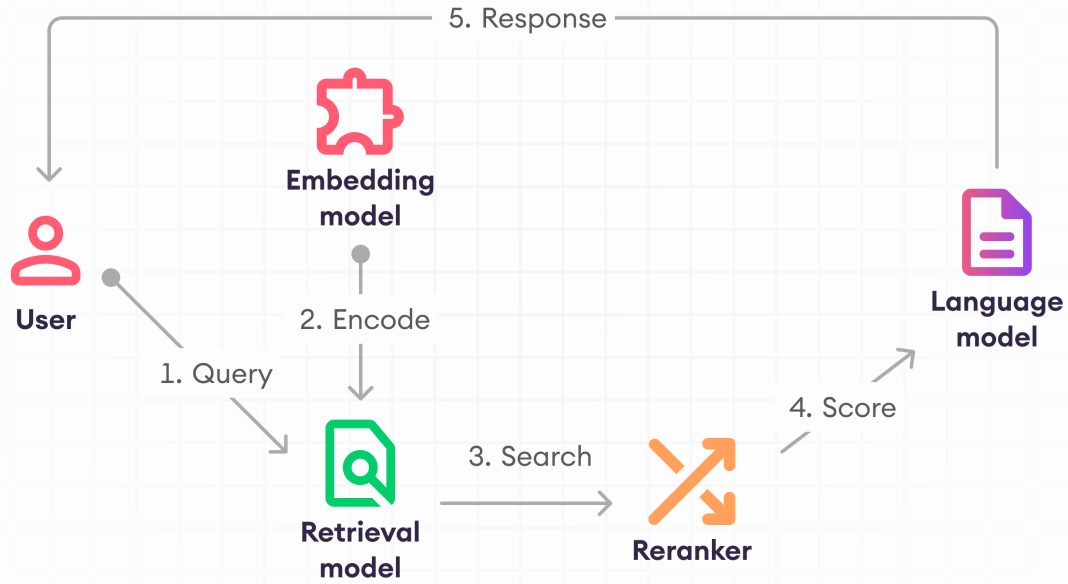
RAG

Retrieval Augmented Generation (RAG) combines the advanced text-generation capabilities of GPT and other large language models with information retrieval functions to provide precise and contextually relevant information.

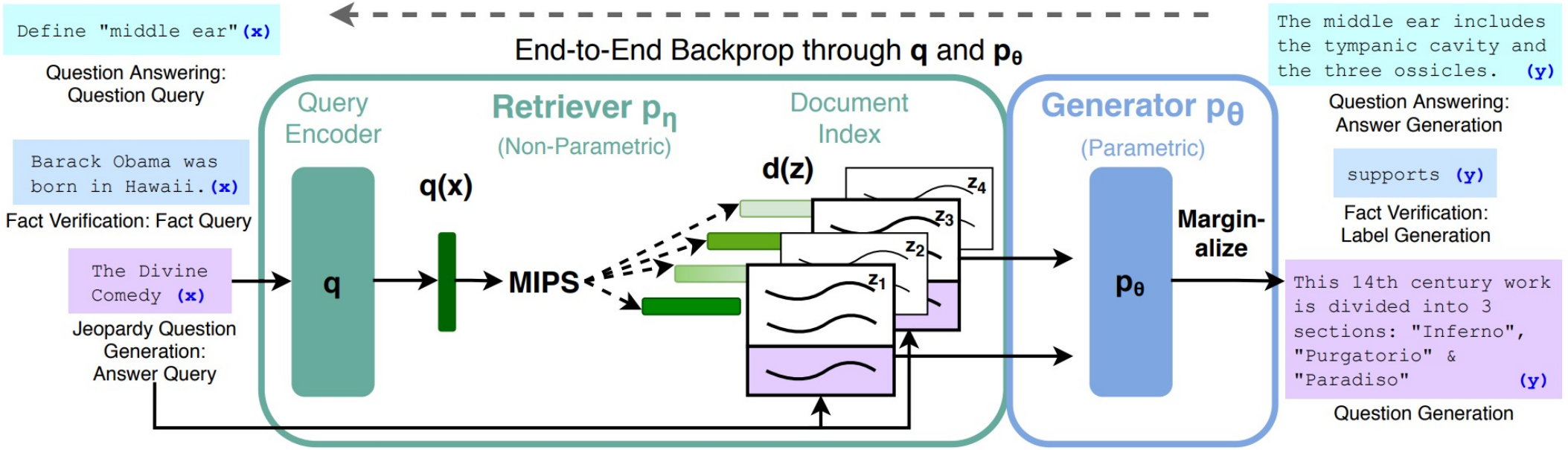
- → Improves language models' ability to understand and process user queries by integrating the latest and most relevant data
- → Prevents from hallucinating
- → Implements customization/personalization with specific data storages

RAG + LLM

- **Embedding model:** documents -> embeddings
- **Retriever:** the search engine within RAG, fetches the most relevant document vectors that match the query
- **Reranker (optional):** evaluates the retrieved documents to determine how relevant they are to the question at hand, providing a relevance score for each one
- **Language model:** takes the top documents provided by the retriever or reranker, along with the original question, and crafts a precise answer



RAG + LLM



A combination of a pre-trained **retriever** (*Query Encoder + Document Index*) with a pre-trained **seq2seq** model (*Generator*) followed by fine-tuning end-to-end. For query x, Maximum Inner Product Search (MIPS) is used to find the top-K documents z_i . For final prediction *y*, *z* is addressed as a latent variable and marginalized over **seq2seq** predictions given different documents.

https://arxiv.org/abs/2005.11401v4

RAG vs Fine-tuning

Feature Comparison	RAG	Fine-tuning
Knowledge Updates	Directly updates the retrieval knowledge base, ensuring information remains current without the need for frequent retraining, suitable for dynamic data environments.	Stores static data, requiring retraining for knowledge and data updates.
External Knowledge	Proficient in utilizing external resources, particularly suitable for documents or other structured/unstructured databases.	Can be applied to align the externally learned knowledge from pretraining with large language models, but may be less practical for frequently changing data sources.
Data Processing	Requires minimal data processing and handling.	Relies on constructing high-quality datasets, and limited datasets may not yield significant performance improvements.
Model Customization	Focuses on information retrieval and integrating external knowledge but may not fully customize model behavior or writing style.	Allows adjustments of LLM behavior, writing style, or specific domain knowledge based on specific tones or terms.
Interpretability	Answers can be traced back to specific data sources, providing higher interpretability and traceability.	Like a black box, not always clear why the model reacts a certain way, with relatively lower interpretability.
Computational Resources	Requires computational resources to support retrieval strategies and technologies related to databases. External data source integration and updates need to be maintained.	Preparation and curation of high-quality training datasets, definition of fine-tuning objectives, and provision of corresponding computational resources are necessary.
Latency Requirements	Involves data retrieval, potentially leading to higher latency.	LLM after fine-tuning can respond without retrieval, resulting in lower latency.
Reducing Hallucinations	Inherently less prone to hallucinations as each answer is grounded in retrieved evidence.	Can help reduce hallucinations by training the model based on specific domain data but may still exhibit hallucinations when faced with unfamiliar input.
Ethical and Privacy Issues	Ethical and privacy concerns arise from storing and retrieving text from external databases.	Ethical and privacy concerns may arise due to sensitive content in the training data.

https://twitter.com/rohanpaul_ai/status/1759649968955269170

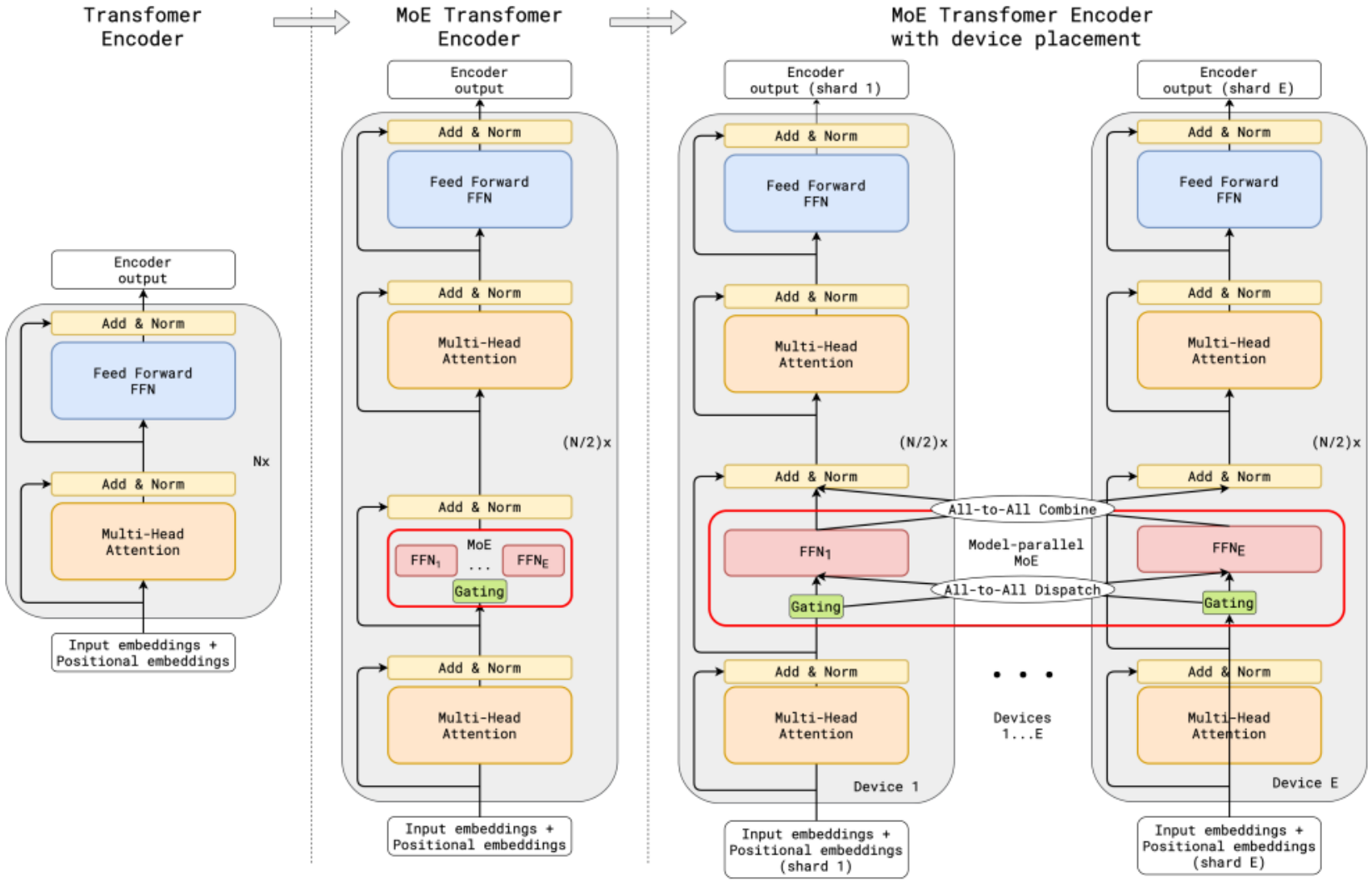
04

Mixture-of-experts

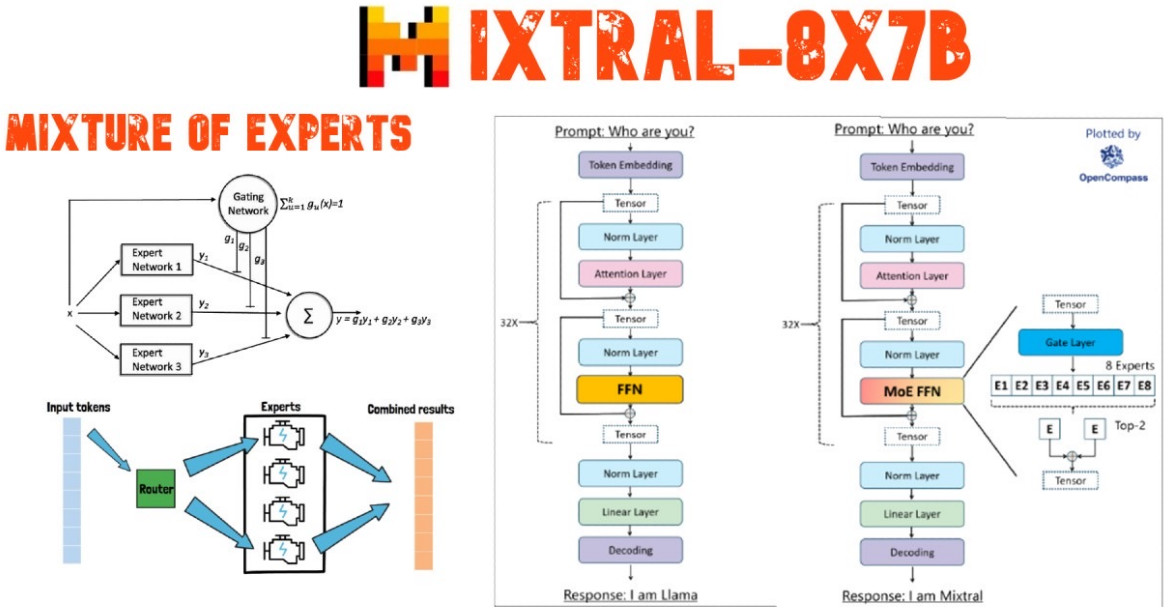
MoE

- Are **pretrained much faster** vs. dense models
- Have **faster inference** compared to a model with the same number of parameters
- Require **high VRAM** as all experts are loaded in memory
- Face many **challenges in fine-tuning** (sparse models are more prone to overfitting), but recent work with MoE **instruction-tuning is promising**

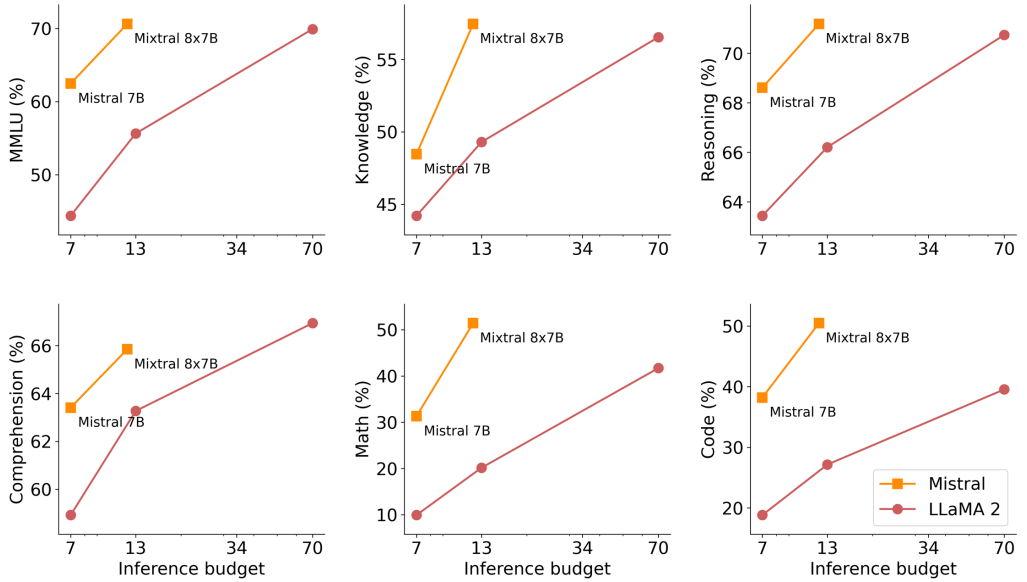
MoE


Transfomer Encoder
MoE Transfomer Encoder
MoE Transfomer Encoder with device placement
Encoder output
Add & Norm
Feed Forward FFN
Multi-Head Attention
Nx
Input embeddings + Positional embeddings
(N/2)x
MoE
FFN1
FFNE
Gating
Encoder output (shard 1)
Encoder output (shard E)
All-to-All Combine
All-to-All Dispatch
Model-parallel MoE
Devices 1...E
Device 1
Device E
Input embeddings + Positional embeddings (shard 1)
Input embeddings + Positional embeddings (shard E)

Mixtral



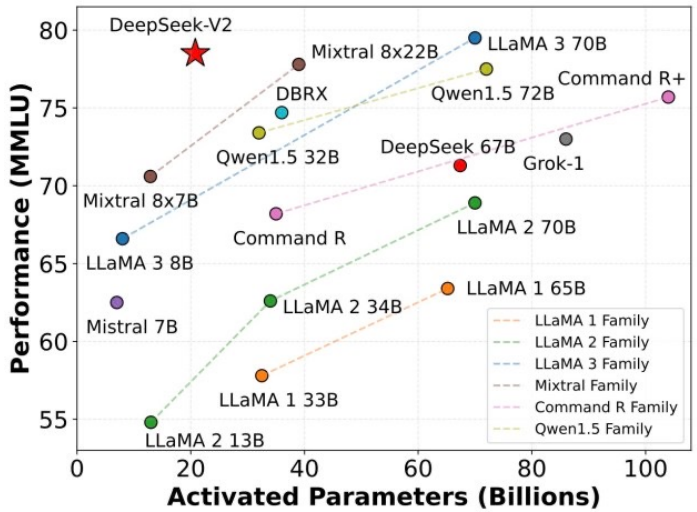
	LLaMA 2 70B	GPT - 3.5	Mixtral 8x7B
MMLU (MCQ in 57 subjects)	69.9%	70.0%	**70.6%**
HellaSwag (10-shot)	87.1%	85.5%	86.7%
ARC Challenge (25-shot)	85.1%	85.2%	**85.8%**
WinoGrande (5-shot)	**83.2%**	81.6%	81.2%
MBPP (pass@1)	49.8%	52.2%	**60.7%**
GSM-8K (5-shot)	53.6%	57.1%	**58.4%**
MT Bench (for Instruct Models)	6.86	**8.32**	8.30

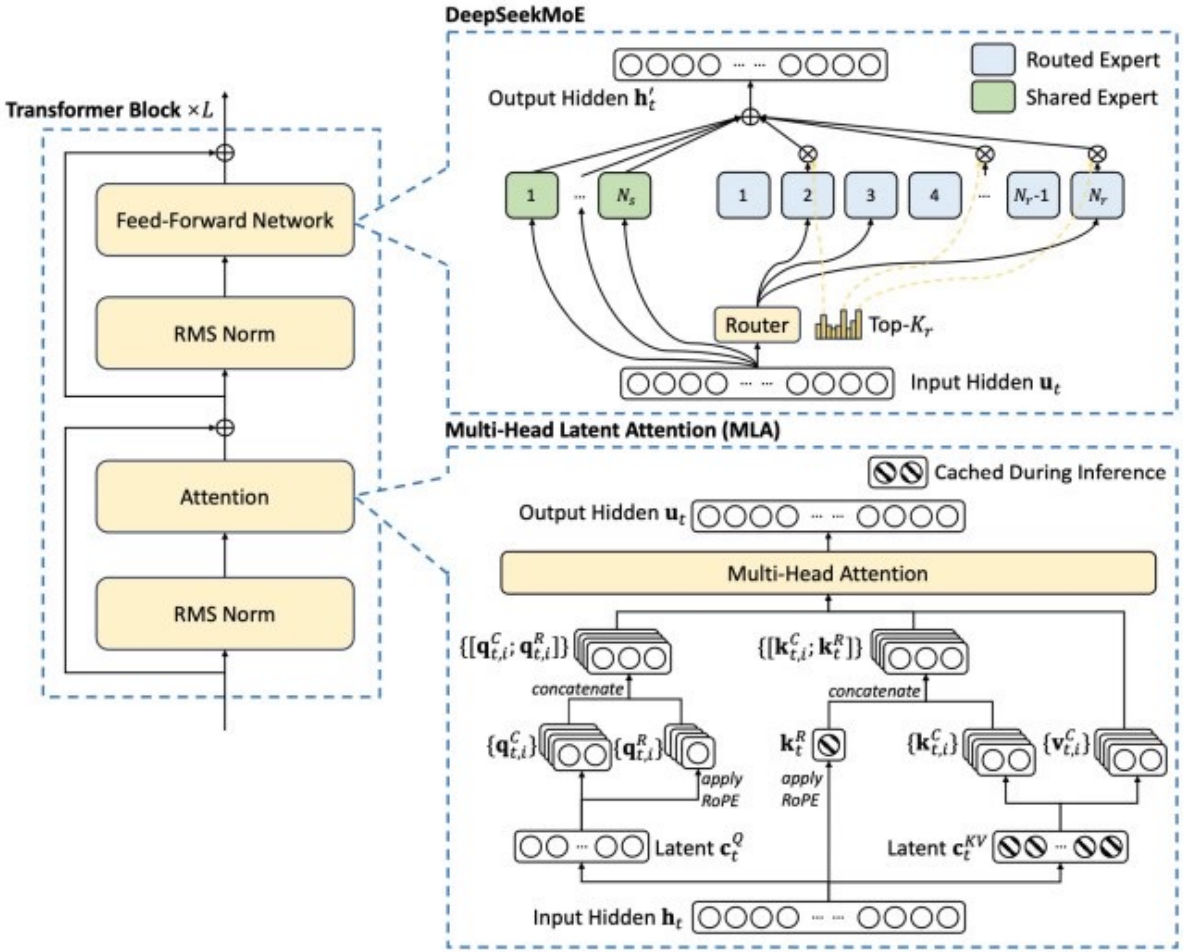


https://arxiv.org/abs/2401.04088

DeepSeek-V2

- Strong, open-source Mixture-of-Experts (MoE) language model
- Comprises 236B total parameters, with only 21B activated for each token
- Supports an extended context length of 128K tokens
- Delivers strong performance while being cost-effective and efficient





https://github.com/deepseek-ai/DeepSeek-V2

DeepSeek-V2 vs ...

- **Qwen1.5 72B:** DeepSeek-V2 demonstrates overwhelming advantages on most English, code, and math benchmarks, and is comparable or better on Chinese benchmarks
- **Mixtral 8x22B:** DeepSeek-V2 achieves comparable or better English performance, except for a few specific benchmarks, and outperforms Mixtral 8x22B on MMLU and Chinese benchmarks
- **LLaMA3 70B:** Despite being trained on fewer English tokens, DeepSeek-V2 exhibits a slight gap in basic English capabilities but demonstrates comparable code and math capabilities, and significantly better performance on Chinese benchmarks
- **Chat Models:** DeepSeek-V2 Chat (SFT) and (RL) surpass Qwen1.5 72B Chat on most English, math, and code benchmarks. They also exhibit competitive performance against LLaMA3 70B Instruct and Mistral 8x22B Instruct in these areas, while outperforming them on Chinese benchmarks

https://github.com/deepseek-ai/DeepSeek-V2

05

Pruning and Distillation

Existing approaches

→ LLM-Pruner
→ Wanda (Pruning by Weights and activations)
→ The Minitron Approach

LLM-Pruner

- A task-agnostic pruning technique
- Minimize the dependency on the original training corpus and preserving the linguistic capabilities of LLMs
- Iteratively examining each neuron within the model as a trigger for identifying dependency groups, thereby constructing the LLM's dependency graph.
- Assessment of the importance of these groups using both parameter-wise and weight-wise estimation

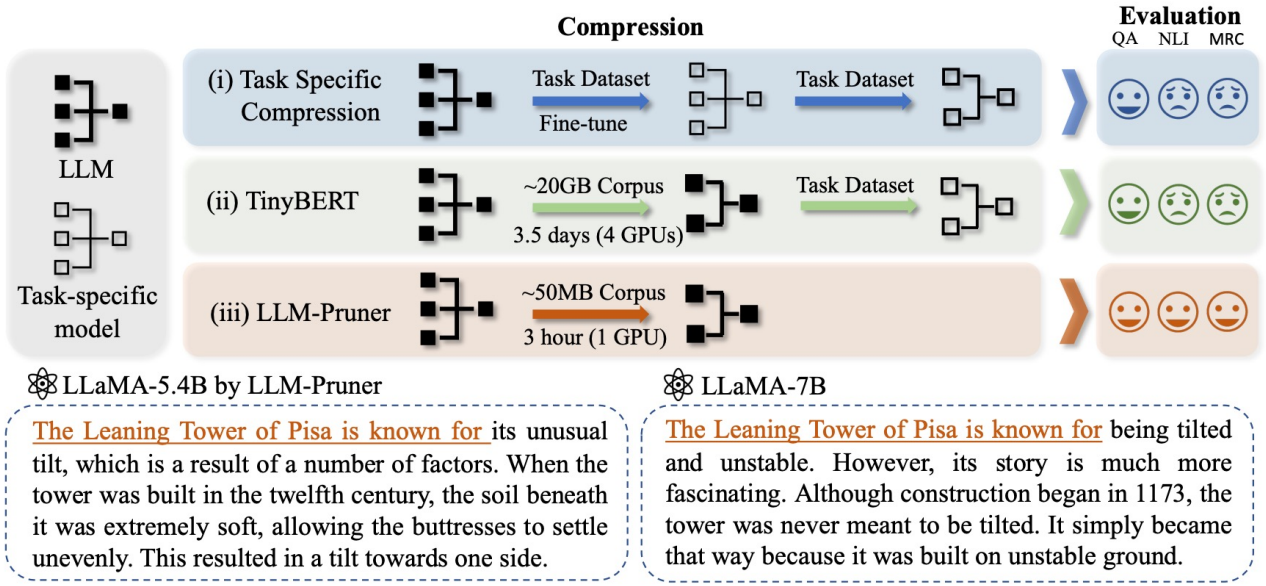


Figure 1: Illustration of LLM-Pruner. (i) Task-specific compression: the model is fine-tuned then compressed on a specific task. (ii) TinyBERT: First distill the model on unlabeled corpus and then fine-tune it on the specific task. (iii) LLM-Pruner: Task-agnostic compression within 3 hours.

Wanda

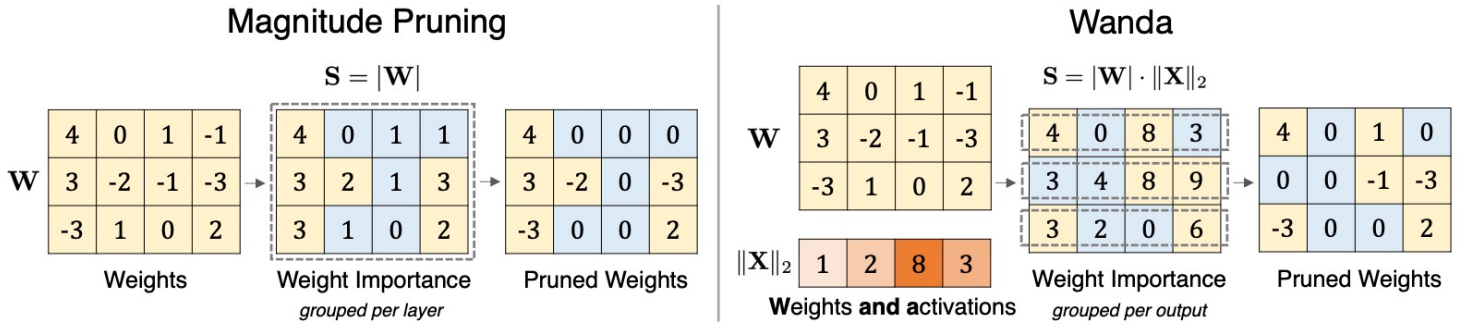
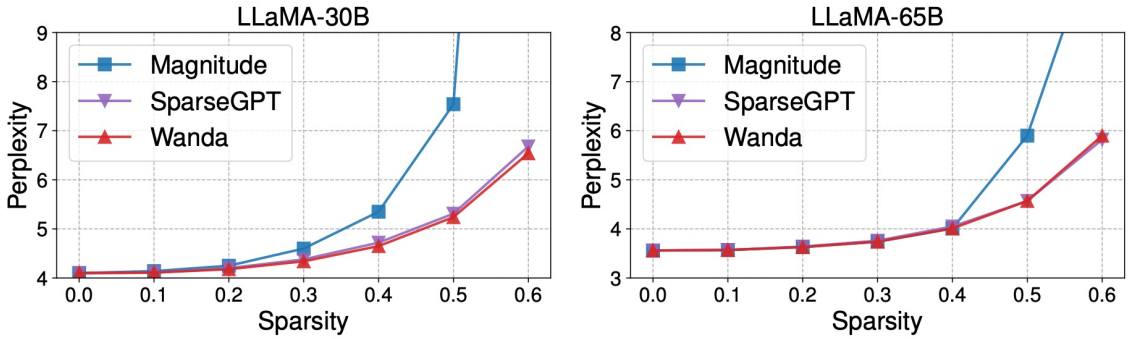


Figure 1: Illustration of our proposed method Wanda (Pruning by **W**eights **and** **a**ctivations), compared with the magnitude pruning approach. Given a weight matrix $\mathbf{W}$ and input feature activations $\mathbf{X}$, we compute the weight importance as the elementwise product between the weight magnitude and the norm of input activations ($|\mathbf{W}| \cdot \|\mathbf{X}\|_2$). Weight importance scores are compared on a *per-output* basis (within each row in $\mathbf{W}$), rather than globally across the entire matrix.

- Removes weights with the smallest magnitudes multiplied by the corresponding input activation norms
- Identifies effective sparse networks within pretrained LLMs



The Minitron Approach

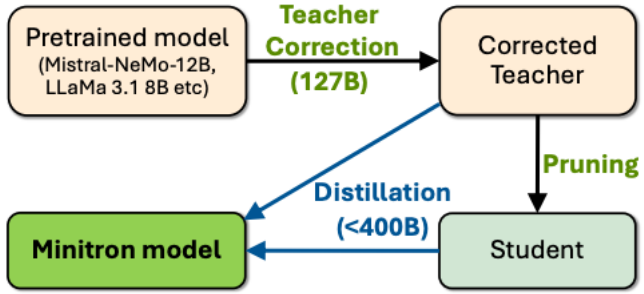
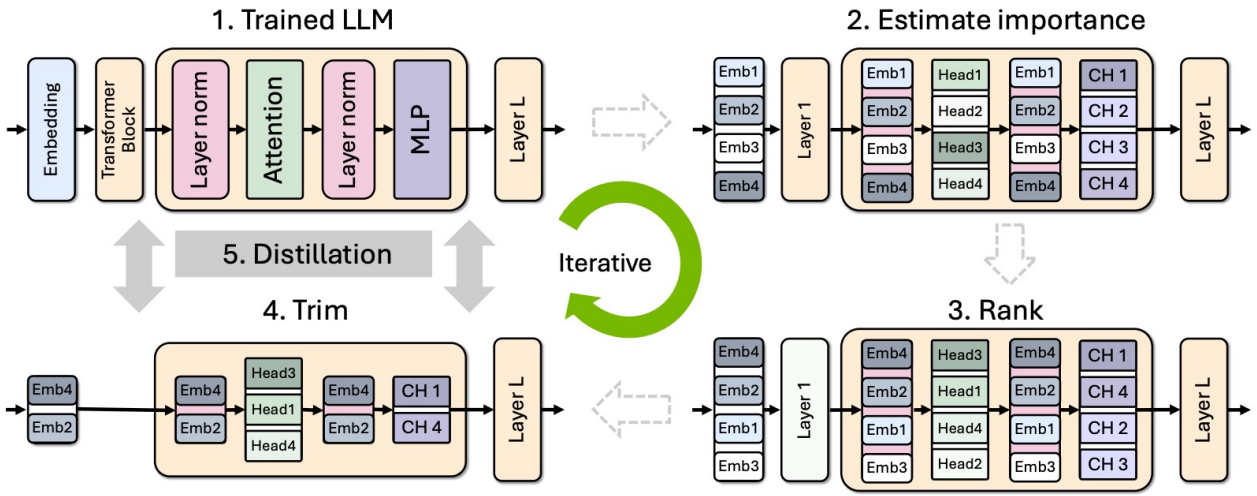
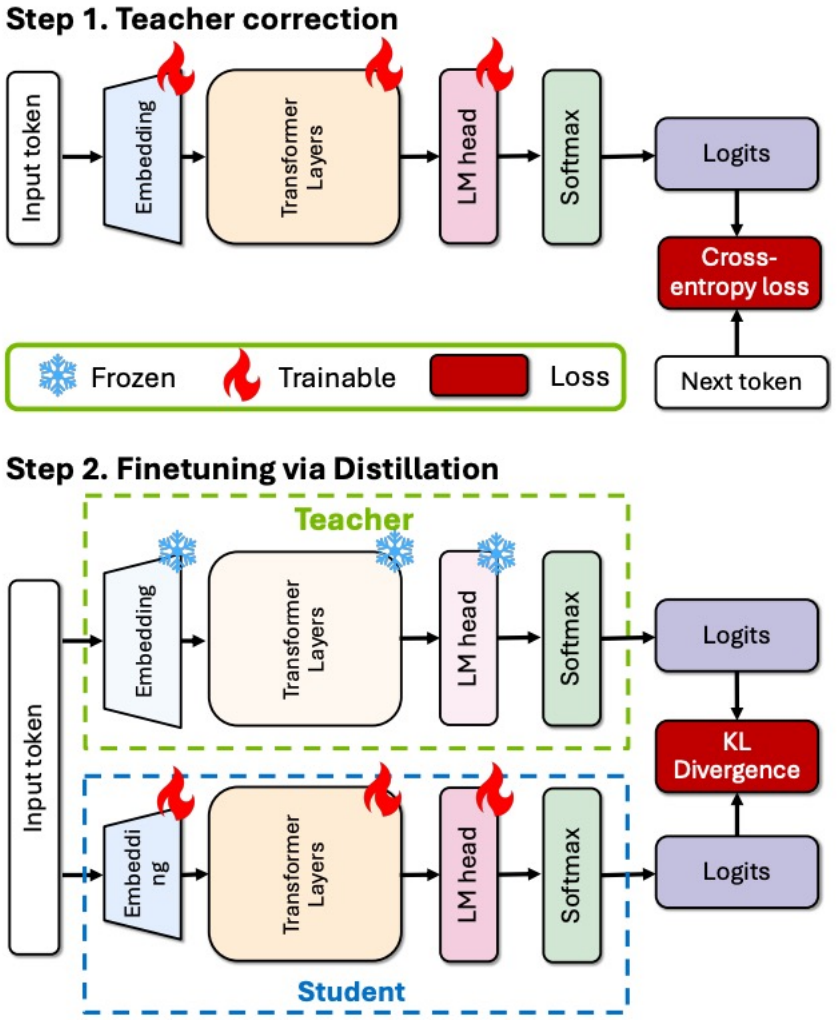


Figure 1 | High-level overview of our proposed pruning and distillation approach. The total number of tokens used for each step is indicated in parentheses.





Distillation approach

The Minitron Approach

Benchmarks(shots)	Gemma2 2B*	Minitron 4B	**Llama-3.1-Minitron 4B-Depth**	**4B-Width**	Gemma 7B	Mistral 7B	Llama 3.1 8B	**MN-Minitron 8B**	Mistral NeMo 12B-Base	12B-FT
Total Params	2.6B	4.2B	4.5B	4.5B	8.5B	7.3B	8B	8.4B	12.2B	12.2B
Non-Emb. Params	2B	2.6B	3.7B	3.7B	7.7B	7B	7B	7.3B	10.9B	10.9B
Training Tokens	2T	**94B**	**94B**	**94B**	6T	8T	15T	**380B**	-	+0.1T
Winogrande(5)	70.9	**74.0**	72.1	73.5	78	78.5	77.3	**80.4**	82.2	82.7
Arc_challenge(25)	55.4	50.9	52.6	**55.6**	61	60.3	57.9	**64.4**	65.1	62.3
MMLU(5)	51.3	58.6	58.7	**60.5**	64	64.1	65.3	**69.5**	69.0	70.1
Hellaswag(10)	73.0	75.0	73.2	**76.1**	82	**83.2**	81.8	83.0	85.2	85.3
GSM8k(5)	23.9	24.1	16.8	**41.2**	50	37.0	48.6	**58.5**	56.4	55.7
Truthfulqa(0)	-	**42.9**	38.2	**42.9**	45	42.6	45.0	**47.6**	49.8	48.3
XLSum en(20%) (3)	-	**29.5**	27.2	28.7	17	4.8	30.0	**32.0**	33.4	31.9
MBPP(0)	29.0	28.2	30.7	**32.4**	39	38.8	42.3	**43.8**	42.6	47.9
HumanEval(n=20)(0)	20.1	**23.3**	-	-	32.0	28.7	24.8	**36.2**	23.8	23.8

06

Transformer insights

Internal Representation Dynamics in Transformers

Key thesis

- Investigate transformer-based models
  → anisotropy dynamics
  → intrinsic dimension of embeddings
- Decoders anisotropy profile has a bell-shaped form
- Encoders anisotropy has a uniform distribution
- Embeddings intrinsic dimension increases in the initial phases of training
- Anisotropy value increases during training

The Shape of Learning: Anisotropy and Intrinsic Dimensions in Transformer-Based Models

Anton Razzhigaev[1,2], Matvey Mikhalchuk[2,4], Elizaveta Goncharova[2], Ivan Oseledets[1,2], Denis Dimitrov[2,3,4], and Andrey Kuznetsov[2,3,5]
[1]Skoltech, [2]AIRI, [3]SberAI,
[4]Lomonosov Moscow State University,
[5]Samara National Research University
razzhigaev@skol.tech

Abstract

In this study, we present an investigation into the anisotropy dynamics and intrinsic dimension of embeddings in transformer architectures, focusing on the dichotomy between encoders and decoders. Our findings reveal that the anisotropy profile in transformer decoders exhibits a distinct bell-shaped curve, with the highest anisotropy concentrations in the middle layers. This pattern diverges from the more uniformly distributed anisotropy observed in encoders. In addition, we found that the intrinsic dimension of embeddings increases in the initial phases of training, indicating an expansion into higher-dimensional space. Which is then followed by a compression phase towards the end of training with dimensionality decrease, suggesting a refinement into more compact representations. Our results provide fresh insights to the understanding of encoders and decoders embedding properties.

1 Introduction

a lens, through which we can study the orientation and concentration of embeddings (Ethayarajh, 2019; Biś et al., 2021). A higher degree of anisotropy suggests that vectors are more clustered or directed in specific orientations. In contrast, the intrinsic dimension offers a measure of the effective data dimensionality, highlighting the essence of information that is captured by the embeddings. Together, these metrics can serve as pivotal tools to probe into the black-box nature of the transformers.

Our investigation uncovers striking contrast in anisotropy dynamics between transformer encoders and decoders. By analyzing the training phases of various transformer models, we shed light on the consistent yet previously unrecognized patterns of the anisotropy growth. Even more, our analysis revealed a unique dynamic of the averaged intrinsic dimension across layers in decoders: an initial growth during the early stages of training is followed by a decline towards the end. This suggests a two-phase learning strategy, where the model initially tries to unfold information in higher di-

The Shape of Learning: Anisotropy and Intrinsic Dimensions in Transformer-Based Models A. Razzhigaev, M. Mikhalchuk, E. Goncharova, I. Oseledets, D. Dimitrov, A. Kuznetsov

A

EACL 2024

AIRI

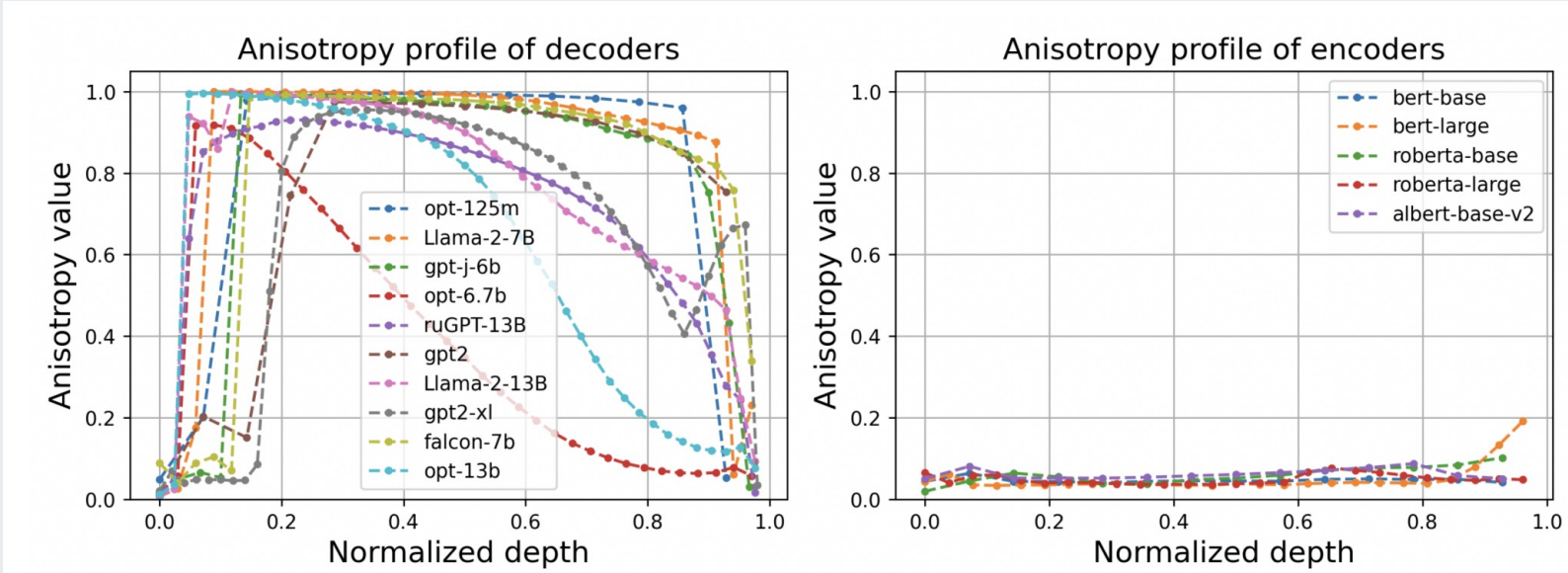
Anisotropy profile

$X \in \mathbb{R}^{n\_samples \times emb\_dim}$ — centered matrix of embeddings

σ_i — its singular values

$$\text{anisotropy}(X) = \frac{\sigma_1^2}{\sum_{i=1}^{k} \sigma_i^2}$$

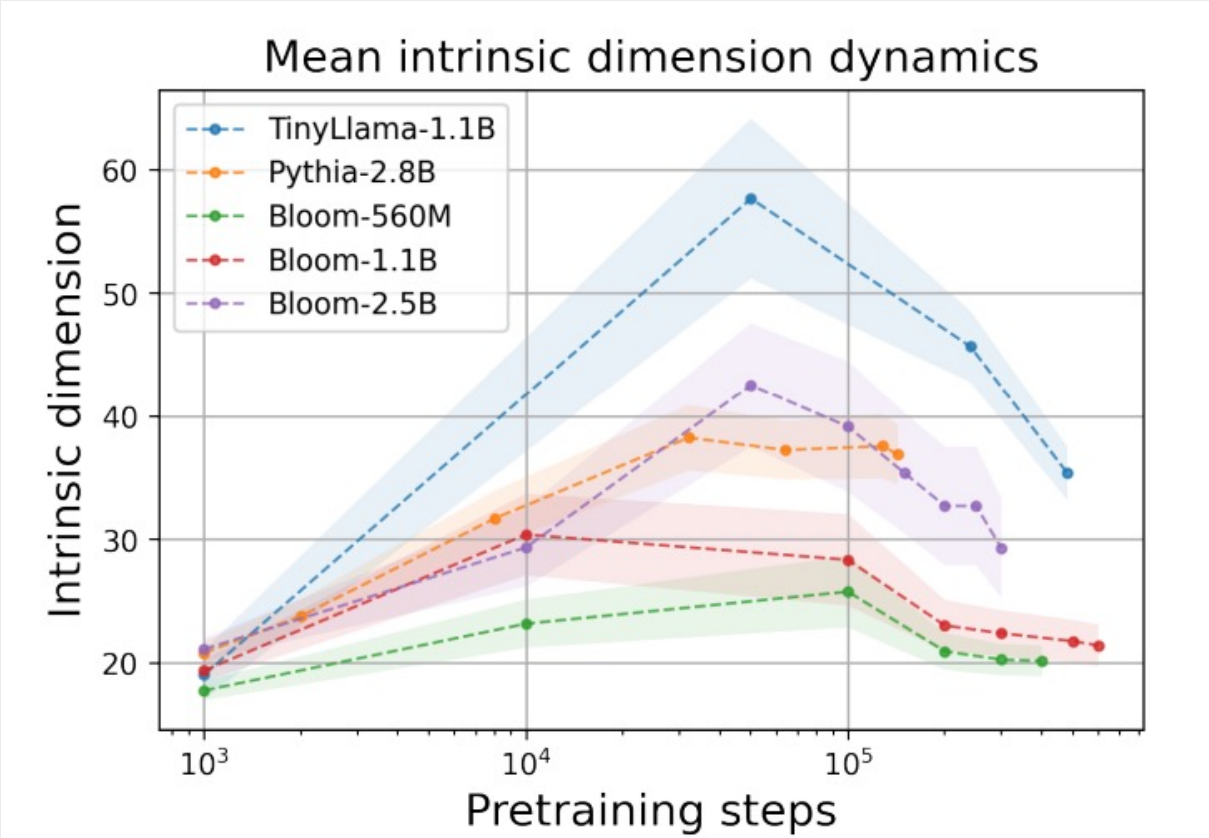
$$\text{average\_cosine} = \frac{2}{n(n-1)} \sum_{1 \le i < j \le n} \cos(X_i, X_j)$$



Different anisotropy profiles for transformer-based encoders and decoders

Intrinsic dimension

- Initial stages demonstrate sharp rise (attempt to map the information to higher dimensional spaces)
- Further training illustrates a notable decline (information compression)



Intrinsic dimension averaged across layers at different pretraining steps

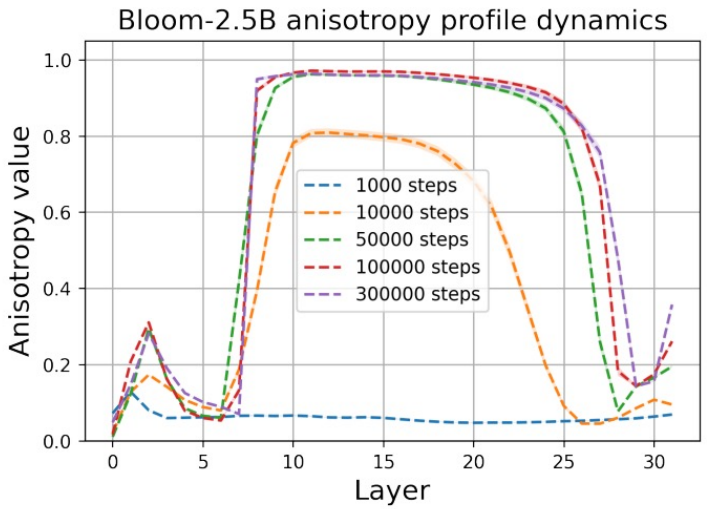
Architectural and training configurations

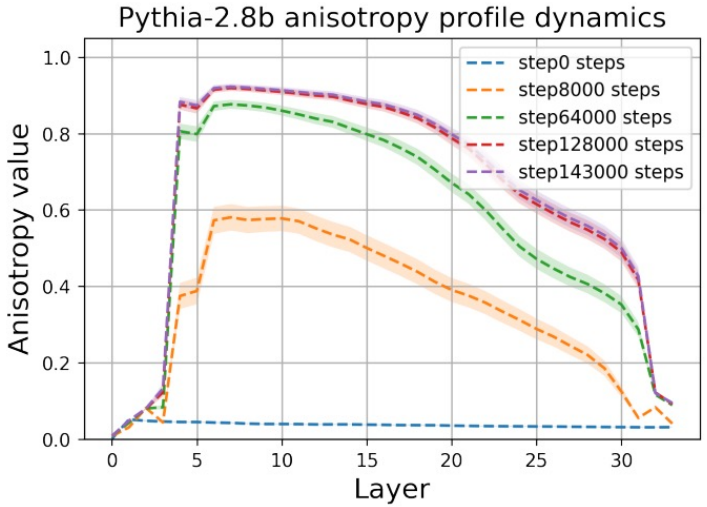
- The growth and the decline of the intrinsic dimension do not show correlation with the warmup period in the learning rate scheduler

	Bloom-560M	Bloom-1.1B	Bloom-3B	Bloom-7B	Pythia-2.8B	TinyLlama-1.1B
Architecture hyperparameters						
Layers	24	24	30	30	32	22
Hidden dim.	1024	1536	2560	4096	2560	2048
Attention heads	16	16	32	32	32	16
Activation	GELU				GELU	SwiGLU
Vocab size	250,680				50,257	32,000
Context length	2048				2048	2048
Position emb.	Alibi				RoPE	RoPE
Tied emb.	True				False	False
Pretraining hyperparameters						
Global Batch Size	256	256	512	512	1024	1024
Learning rate	3.0e-4	2.5e-4	1.6e-4	1.2e-4	1.6e-4	4.0e-4
Total tokens	341B				300B	3T
Warmup tokens	375M				3B	4B
Min. learning rate	1.0e-5				1.6e-5	4.0e-5

Results

- Specific attributes of transformer-based architectures were examined: anisotropy and intrinsic dimension
- Anisotropy profile is similar for different decoder models in terms of training steps dependency
- Intrinsic dimension pattern illustrates how the model operates with embedding dimension during training





Your transformer is secretly linear

Key thesis

- ➔ Study of the linearity properties of transformer decoder models
  → proposed a metric to measure the degree of linearity of transformer layers (linearity score)
  → measured linearity dynamics during pretraining and finetuning
- ➔ Proposed a new regularization approach for pretraining based on cosine similarity, designed to decrease layer linearity
- ➔ Proposed new algorithms for depth pruning of transformer decoders, allowing to remove the most linear layers
- ➔ Proposed a new distillation method that involves pruning, replacing certain layers with linear approximations, and then distilling layer-wise embeddings to preserve model performance

Your Transformer is Secretly Linear

Anton Razzhigaev[1,2], Matvey Mikhalchuk[1,5], Elizaveta Goncharova[1,4],
Nikolai Gerasimenko[3,5], Ivan Oseledets[1,2], Denis Dimitrov[1,3], and Andrey Kuznetsov[1,3]
[1]AIRI, [2]Skoltech, [3]SberAI, [4]HSE University,
[5]Lomonosov Moscow State University
razzhigaev@skol.tech

Abstract

This paper reveals a novel linear characteristic exclusive to transformer decoders, including models such as GPT, LLaMA, OPT, BLOOM and others. We analyze embedding transformations between sequential layers, uncovering a near-perfect linear relationship (Procrustes similarity score of 0.99). However, linearity decreases when the residual component is removed due to a consistently low output norm of the transformer layer. Our experiments show that removing or linearly approximating some of the most linear blocks of transformers does not affect significantly the loss or model performance. Moreover, in our pretraining experiments on smaller models we introduce a cosine-similarity-based regularization, aimed at reducing layer linearity. This regularization improves performance metrics on benchmarks like Tiny Stories and SuperGLUE and as well successfully decreases the linearity of the models. This study challenges the existing understanding of transformer architectures, suggesting that their operation may be more linear than previously assumed. [1]

Our investigation reveals a surprising discovery: the embedding transformations between sequential layers in transformer decoders exhibit almost linear properties. This observation is quantified using Procrustes similarity analysis, demonstrating a near-perfect linearity score of 0.99. Such a discovery not only challenges the traditional understanding of transformer architectures but also opens new opportunities for model optimization and efficiency.

Based on this insight, we introduce several new contributions to the field:

- Extensive analysis of the linearity properties of transformer decoders and its dynamics at the pretraining and fine-tuning stages.
- The development of new algorithms for depth pruning of transformer decoders, allowing to remove the most linear layers without a significant loss in performance.
- A novel distillation technique that involves pruning, replacing certain layers with linear approximations, and then distilling layer-wise embeddings to preserve model performance.

1 **Introduction**

Your Transformer is Secretly Linear A. Razzhigaev, M. Mikhalchuk, E. Goncharova, N. Gerasimenko, I. Oseledets, D. Dimitrov, A. Kuznetsov

A*

ACL 2024

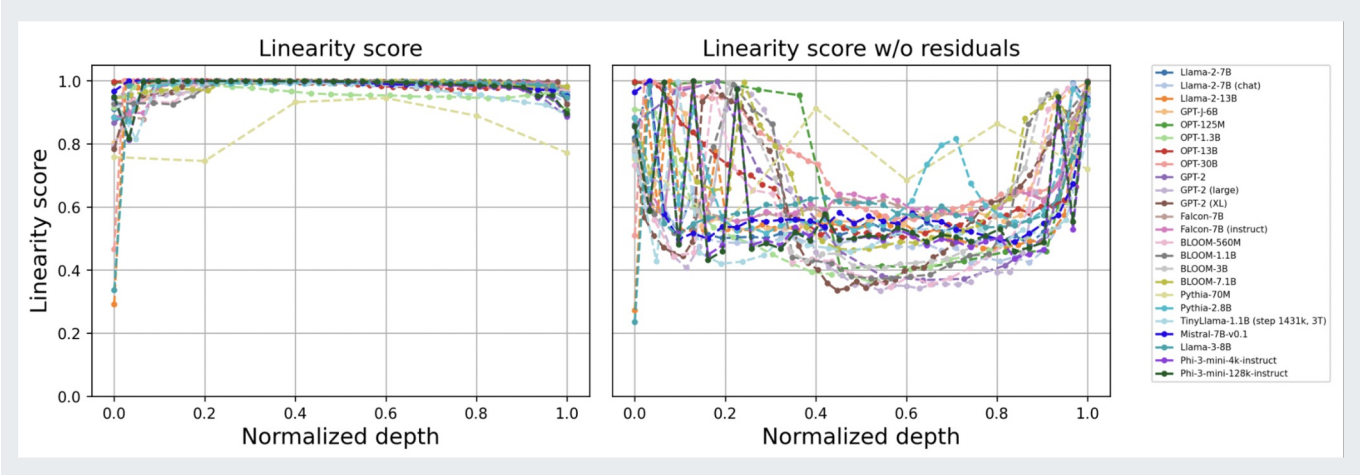
AIRI

Linearity score

- Let X, Y ∈ Mat(n, d) be a centered sets of embeddings.
  (X – input embeddings of the transformer layer, Y – corresponding
  layer outputs)
- To calculate linearity score we use normalized matrices
- To eliminate the influence of the residual connection, we also considered the linearity score between X and Y-X

$$\tilde{X} = X/||X||_2, \tilde{Y} = Y/||Y||_2$$

$$\text{linearity\_score} := 1 - \min_{A \in R^{d \times d}} ||\tilde{X}A - \tilde{Y}||_2^2$$



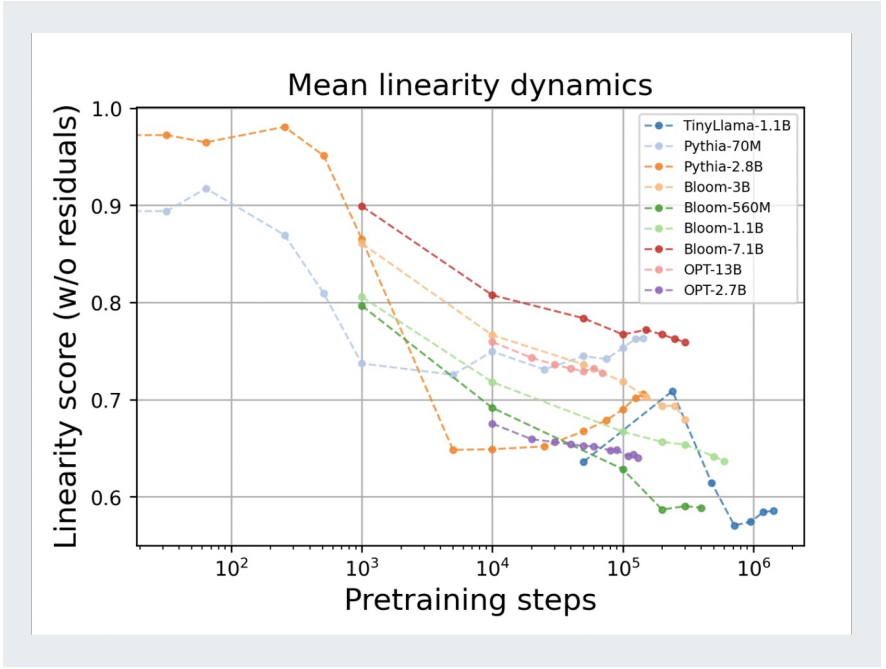
Linearity profiles for different open source models. Normalized depth is the layer index divided by the total depth.

Linearity score dynamics during pretraining and finetuning

- ➔ We measured the linearity dynamics of both open-source models with publicly available intermediate pretraining checkpoints and our custom models finetuned on small datasets
- ➔ As the models undergo pretraining, the linearity of the layers gradually decreases on average
- ➔ All studied models show an increase in linearity after finetuning

Model Name	Super_Glue/MultiRC	Super_Glue/BoolQ	Super_Glue/CB	Reward Modeling
OPT-125M	0.085 ± 0.008	0.217 ± 0.038	0.048 ± 0.009	0.060 ± 0.008
OPT-1.3B	0.055 ± 0.021	0.382 ± 0.004	0.088 ± 0.010	0.062 ± 0.007
OPT-2.7B	0.061 ± 0.025	0.356 ± 0.005	0.066 ± 0.029	0.054 ± 0.003
Llama2-7B	0.141 ± 0.006	0.051 ± 0.024	0.081 ± 0.070	0.194 ± 0.027
GPT2	0.085 ± 0.021	0.048 ± 0.016	0.004 ± 0.003	0.092 ± 0.013
GPT2-Large	0.049 ± 0.003	0.023 ± 0.008	0.025 ± 0.014	0.085 ± 0.008
GPT2-XL	0.040 ± 0.007	0.037 ± 0.007	0.028 ± 0.019	0.038 ± 0.008

Delta of linearity score w/o residuals after fine-tuning various tasks.

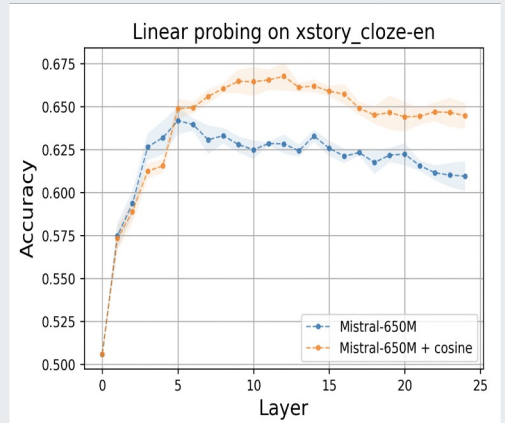


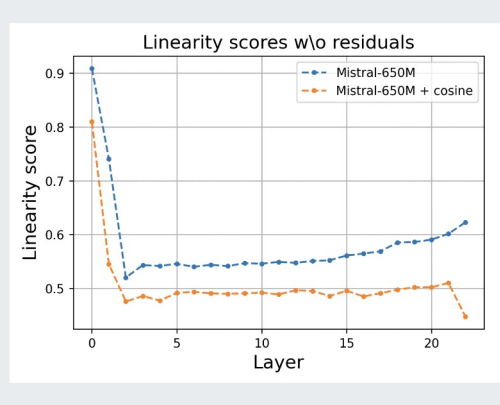
Linearity score (averaged across layers)
at different pretraining steps of open source models

Improving Linearity with cosine similarity regularization

$$L_{cosine} = \lambda \sum (1 - \cos(emb_i, emb_{i-1}))$$

- ➔ Experiments with pretraining of small models (Mistral architecture, 150/650M) on small datasets
- ➔ Proposed regularization that reduces the cosine distance between the input and output embeddings for each layer
- ➔ Surprisingly, after pretraining with such regularization, linearity scores become lower at each layer of the model
- ➔ Regularization improves performance on benchmark datasets such as SuperGLUE and TinyStories
- ➔ Experiments with linear probing demonstrated that such regularization improves the expressiveness of embeddings and improves performance on linear probing tasks



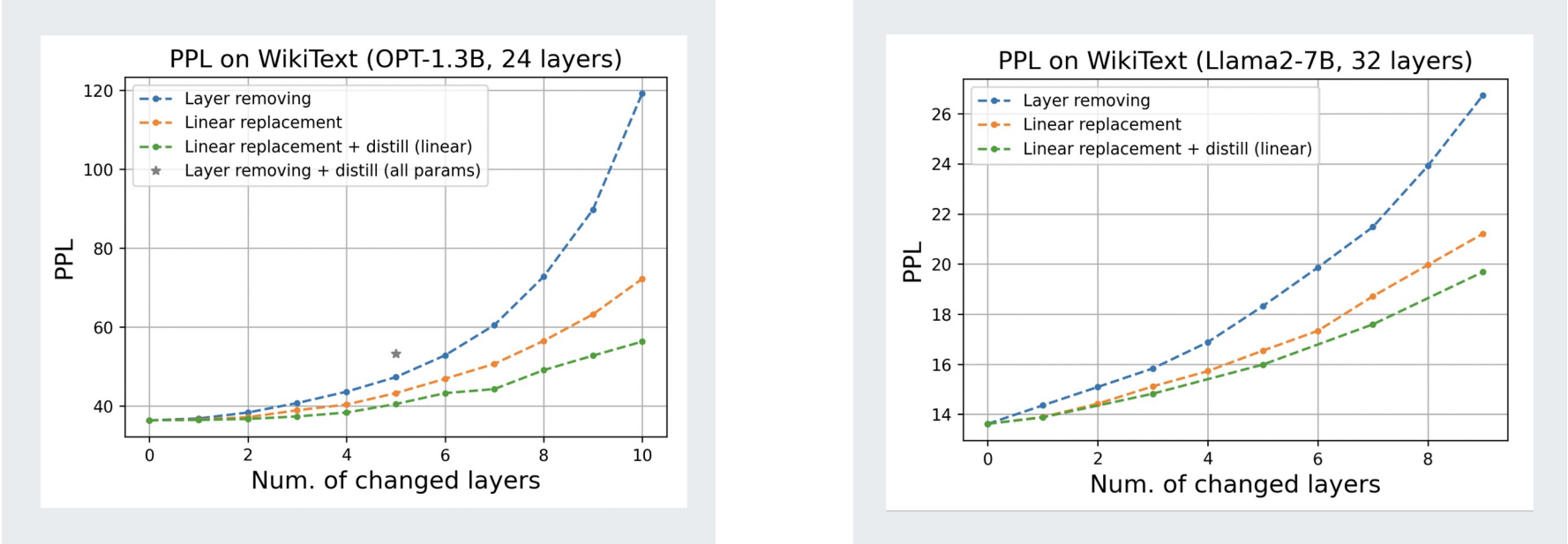


Mistral config	Grammar	Creativity	Consistency	Plot	Mean
650M	5.47	6.60	4.81	4.67	5.39
650M + cosine (0.5)	**6.07**	**7.02**	**5.74**	**5.48**	**6.08**
150M	4.88	6.51	4.16	3.88	4.86
150M + MSE (0.5)	**5.19**	6.70	4.47	4.20	5.14
150M + MSE (2.0)	5.00	6.81	4.56	4.29	5.17
150M + cosine (0.5)	5.14	**6.91**	**4.77**	**4.95**	**5.44**

TinyStories prompts completions evaluation via GPT-4

Exploiting Linearity for Pruning

➔ Removing the most linear layers of the model does not affect its performance much
➔ Replacing some layers with their linear approximations further improves the pruning
➔ Further finetuning of these linear layers all at once slightly reduces the perplexity increase



Perplexity on WikiText for various pruning and distillation methods (lower is better)

Contacts

Contacts

PhD,
Head of FusionBrain Lab, AIRI
Executive director on data science, Sber AI
kuznetsov@airi.net

@KUZNETSOFF87

@COMPLETE_AI