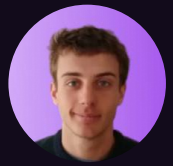
Pruna AI

Language Model Archirtectures



Bertrand Charpentier
Founder, President & Chief Scientist

Overview

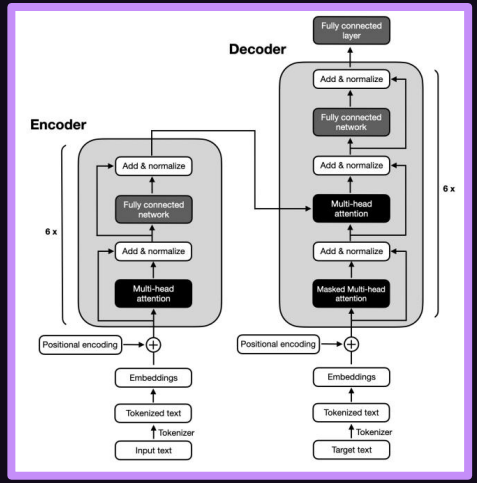
- Tokenizer
- Embeddings

Autoregressive LLMs
- Transformer block
- Attention (w/ QKV, (causal) masks, (layer) normalization, positional encoding, FCN, skip connection)
- Improvement: Multi-head attention
- Improvement: Flash attention
- Improvement: KV Cache
- Improvement: Paged attention
- Improvement: Positional Embeddings

State Space LLMs
- Continuous perspective
- Recursive perspective
- Convolution perspective

Diffusion LLMs
- Discrete diffusion

- Encoder only, encoder-decoder, decoder only architectures



Tokenizer

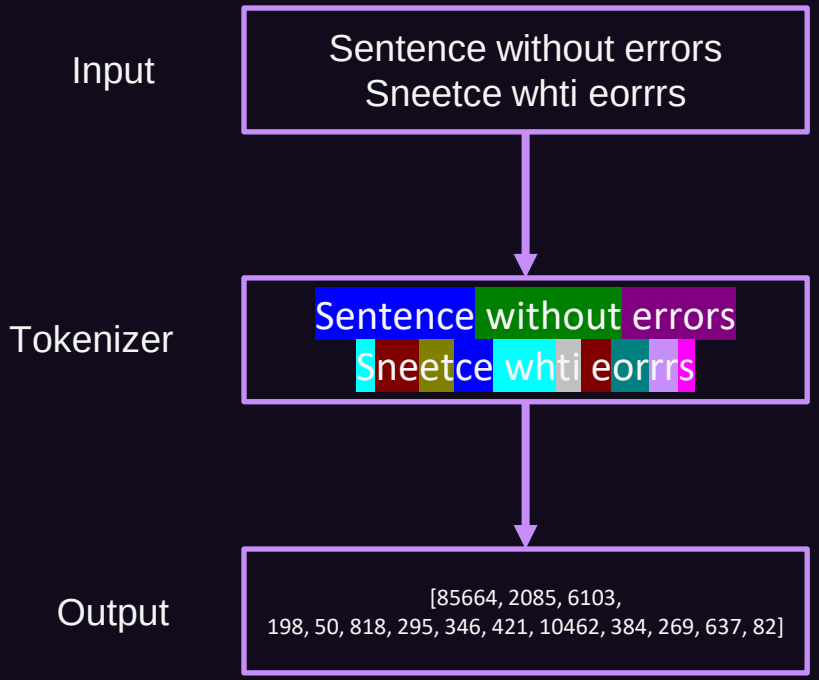
The tokenizer encodes a text sequence into a token ids sequence.

Input - Text
Ouptut - Tokens (w/ vocabulary size)
Algorithm

- Clean-up the text with *normalization (e.g. remove special characters…)*
- Split text into words with *pre-tokenization*
- Produce sequence of token via a *model (e.g. BPE…)*
- Adding special tokens with *post-processing (e.g. end of sentence.)*

Remarks

- A token represents a frequent sequence of characters.
- Some words are tokenized in multiple tokens.
- Rare sequences need more tokens.



Reference:

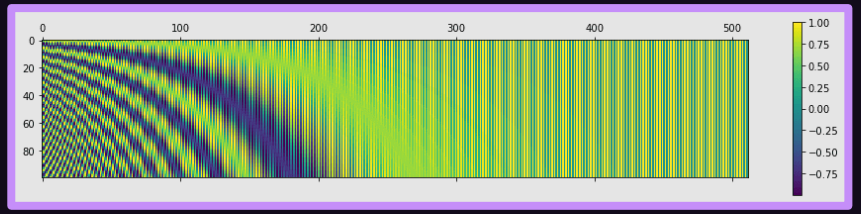
- https://huggingface.co/spaces/Xenova/the-tokenizer-playground
- https://github.com/SumanthRH/tokenization/tree/main

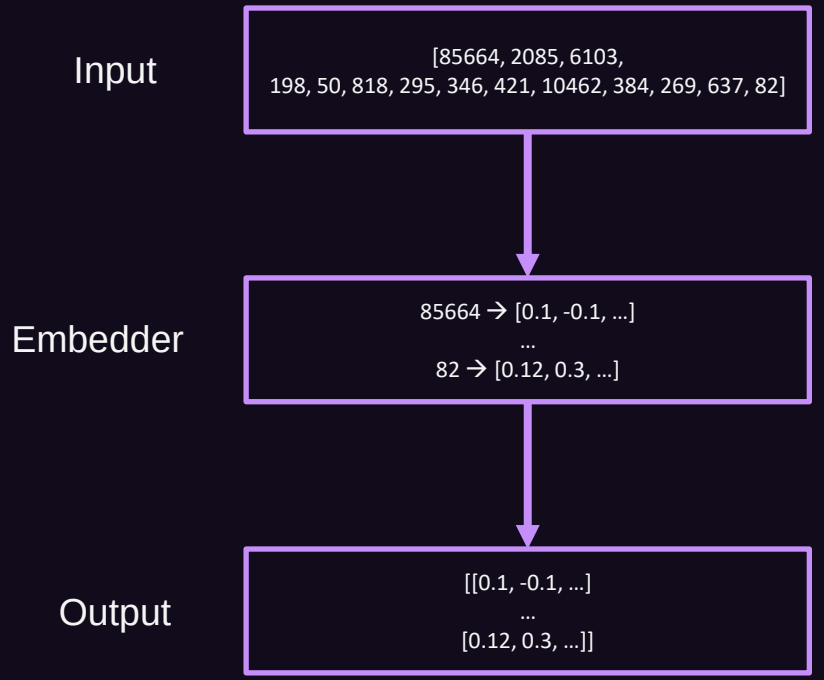
Embeddings

The embeddings encode token (and positional) information.

Input - Tokens
Output - Embeddings (w/ embedding size)
Algorithm
- A Look-up table associate a learned embedding to each token id.

Remarks
- One-hot encoding is not efficient. The embedding size would be the vocabulary size.
- Word2Vec, GloVe are popular word embedder.
- It can be trained end-to-end.
- The token position can be encoded with positional encoding.





Reference:
- https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Autoregressive LLMs

Transformer Block

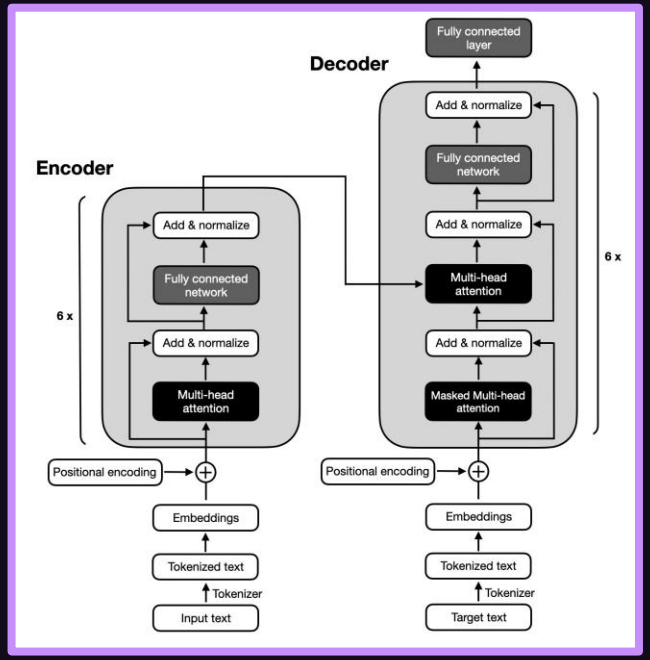
The transformer blocks models information of different tokens on each other.

Input - Embeddings
Output - Embeddings
Algorithm

- Attention to account for token dependencies.
- Normalization.
- Feed forward for element-wise non-linearities.

Remarks

- Transformer blocks is common building block in state-of-the-art models (incl. text, images, video, graphs). It repeats multiple times inside many architectures including LLMs.



Attention

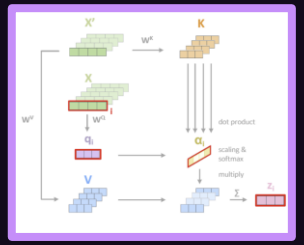
The attention mechanism determines the relative importance of each token relative to other tokens in the sequence. The token keys K and queries Q compute the attention map α which weigh the token values V.

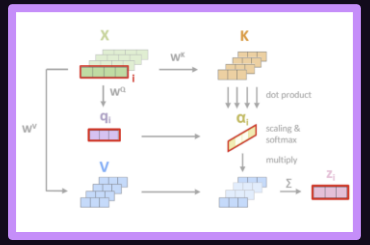
Input - Embeddings X (and X')
Output - Embeddings Y
Algorithm

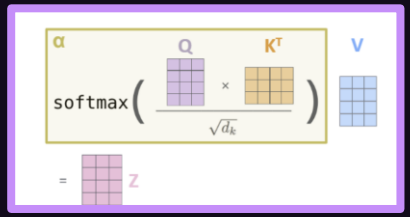
- **Compute keys/queries/values:** $K = W_K X$ (Self attention) or $K = W_K X'$ (Cross attention), $Q = W_Q X$, $V = W_V X$
- **Compute attention map (with Mask):** $\alpha = softmax\left((QK^T + M)/\sqrt{d}\right)$
- **Weight Values:** $Y = \alpha V$

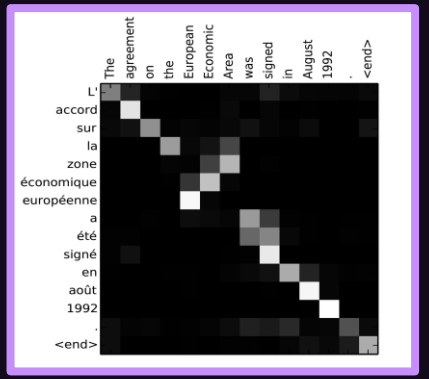
Remarks

- The attention map has n^2 time and memory complexity.
- The mask M is usually a lower/upper triangular causal mask. It guarantees that token attend only to tokens on their left.
- Encoding a sequence (like in training, and prompt processing) is fast since parallelizable.
- Decoding a sequence (like in text generation) is slow since sequential.









Reference:

- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- https://slds-lmu.github.io/seminar_nlp_ss20/attention-and-self-attention-for-nlp.html

Multi-Head Attention

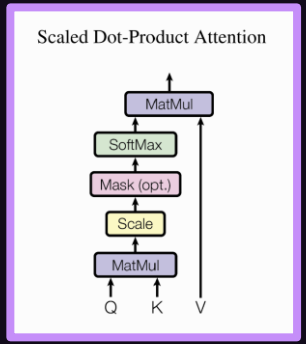
The multi-head attention makes an ensemble of attention mechanisms to increase capacity.

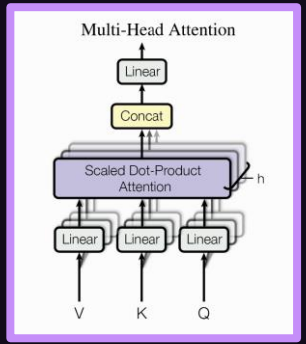
Input - Embeddings $\boldsymbol{X}$ (and $\boldsymbol{X'}$)
Output - Embeddings $\boldsymbol{Y}$
Algorithm

- $\boldsymbol{head_i = Attention_i(X)}$
- $\boldsymbol{Y = \mathbf{W}\ \mathbf{concat}(head_i, \dots, head_h)}$

Remarks

- Multiple heads give a diverse processing of the embeddings.
- One head with large embedding dimension d is (usually) less powerful than $\#heads$ of dimension $^{d}/_{\#heads}$.





Reference:

- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- https://arxiv.org/pdf/1706.03762

KV Cache

Problem

For generation with causal masks:

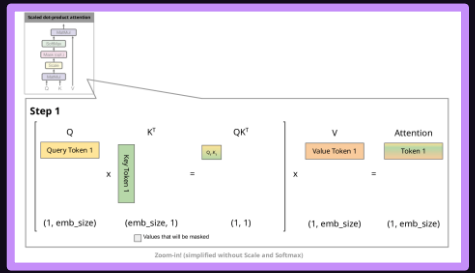
- Key/Values were already computed → No need to recompute KV from previous tokens.
- Query from previous tokens is not used → No need to store query from previous tokens.

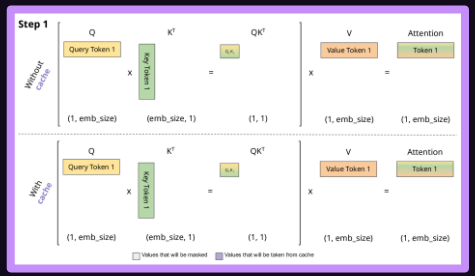
Solution

- Cache KV values from previous tokens.
- Do not use previous queries.

Remark

- Without KV cache → Computation and memory grow quadratically with the input sequence length.
- With KV cache → Computation and memory grow linearly with the input sequence length.
- More specifically $KV\ cache = batch\ size \times sequence\ length \times \#layers \times \#heads \times embedding\ dimension \times precision \times 2$ →Grouped Query Attention: Heads can share Keys and Values.





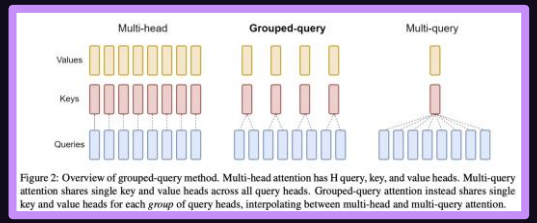


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Reference:

- https://medium.com/@joaolages/kv-caching-explained-276520203249

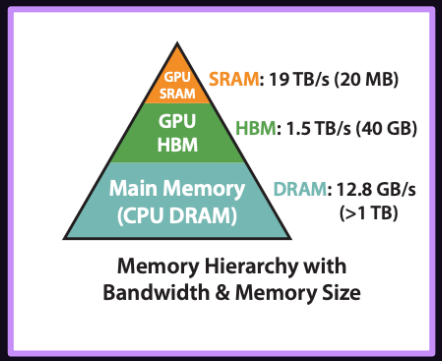
Flash Attention

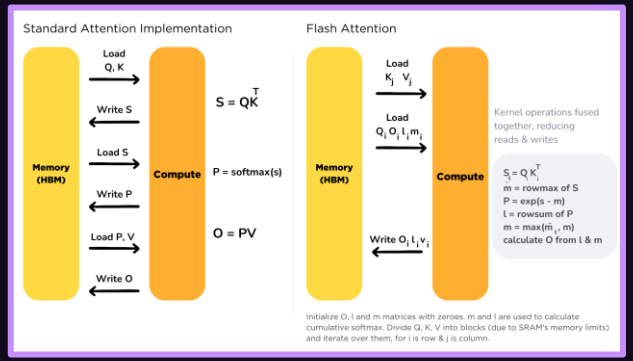
Problem

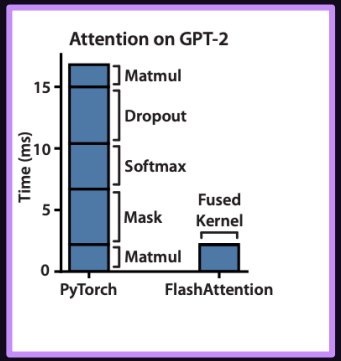
- After the model fits in the GPU High Bandwith Memory (HBM), the GPU SRAM requires memory accesses to HBM to perform model operations.
- A (naive) implementation of the attention operator would execute many sub-operators associated with many I/O memory accesses making the overall attention slow (i.e. the attention operator is memory bounded).

Solution

- Fuse many sub-operators into a single flash attention operator with less memory accesses → It loops over rows and columns to iteratively compute exact output without re-writing/re-loading dot product, softmax, or values.



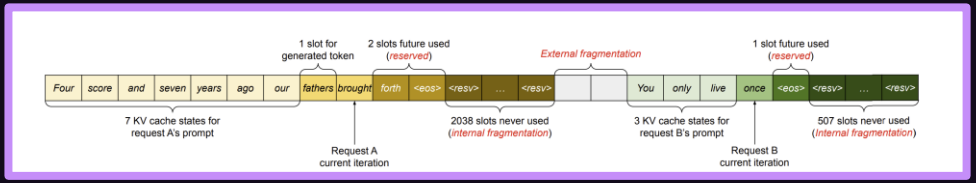




Reference:

- https://huggingface.co/docs/text-generation-inference/en/conceptual/flash_attention
- https://arxiv.org/abs/2402.07443
- https://towardsdatascience.com/flash-attention-fast-and-memory-efficient-exact-attention-with-io-awareness-a-deep-dive-724af489997b/

Paged Attention



Problem

For generation for batches:, memory is wasted for KV cache. Pre-allocation wrt max generated sequence requires (potentially wasted) contiguous memory.

- **Reserved memory**: memory allocated in the future is currently unused.
- **Internal fragmentation**: memory allocated for too long sequences but never used.
- **External fragmentation**: not allocated memory between allocated slots since too short for the predefined max sequence.

Solution

- Virtual memory acts as a codebook for KV cache enabling non-contiguous allocation on physical memory.
- Multiple generation can (sometimes) share KV cache.

Remark

- For batch inference, saving memory → larger batch size → higher throughput.

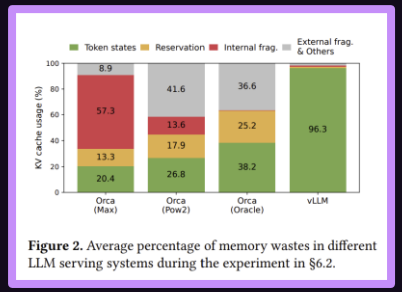


Figure 2. Average percentage of memory wastes in different LLM serving systems during the experiment in §6.2.

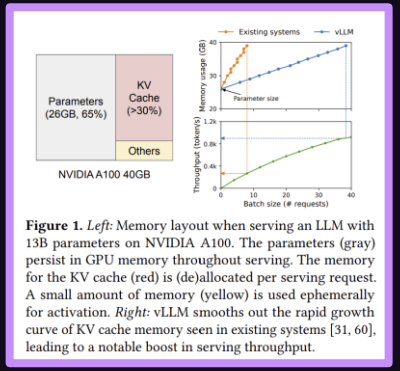


Figure 1. *Left:* Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. *Right:* vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.

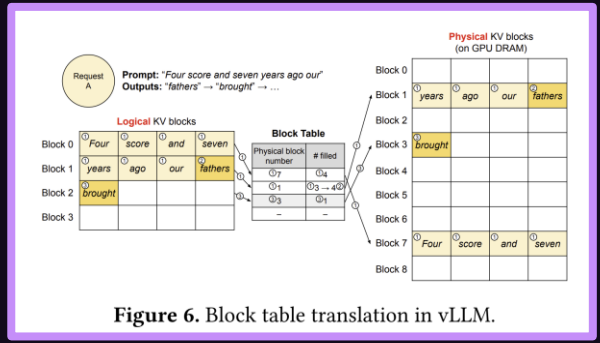


Figure 6. Block table translation in vLLM.

Reference:

- https://arxiv.org/pdf/2309.06180

Positional Embedding

Problem

- The position of the token is not encoded but contains important information.

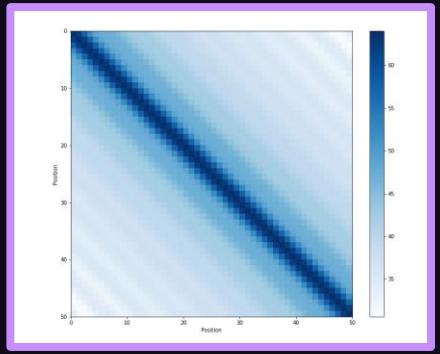
Solution 1

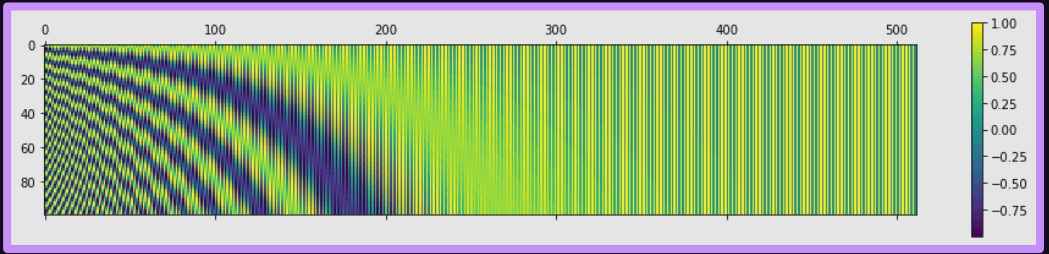
- **Positional embedding:** Concatenate or sum token embeddings and positional embeddings.

Remark

- Intuitively, tokens closer to each other in the sentence impact each other more.
- Positional encoding require to forward the information through the all model. It is okay because of skip connections.

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$





Reference:

- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- https://blog.eleuther.ai/rotary-embeddings/
- https://arxiv.org/pdf/2104.09864

Positional Embedding

Problem

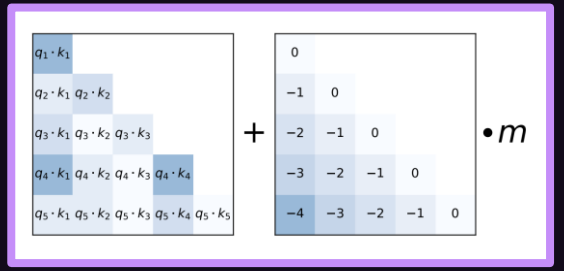
- The position of the token is not encoded but contains important information.

Solution 2

- **Relative positional encoding:** Encode position in attention maps depending on the relative distance between tokens.
  - Alibi add a shift to the attention maps.
  - Rotary embeddings rotate 2D blocks in QV.

Remark

- Intuitively, tokens closer to each other in the sentence impact each other more.
- We can extend to larger sequence length by e.g. selecting smaller θ_i.



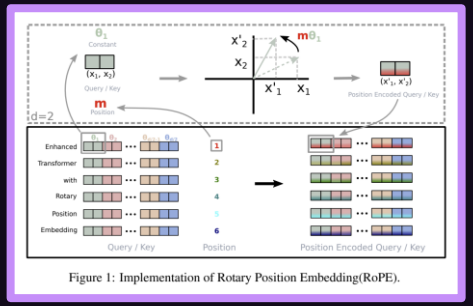


Figure 1: Implementation of Rotary Position Embedding(RoPE).

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m}\boldsymbol{W}_q\boldsymbol{x}_m)^\intercal(\boldsymbol{R}^d_{\Theta,n}\boldsymbol{W}_k\boldsymbol{x}_n) = \boldsymbol{x}^\intercal\boldsymbol{W}_q R^d_{\Theta,n-m}\boldsymbol{W}_k\boldsymbol{x}_n$$

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Reference:

- https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- https://blog.eleuther.ai/rotary-embeddings/
- https://arxiv.org/pdf/2104.09864

Feed Forward Network

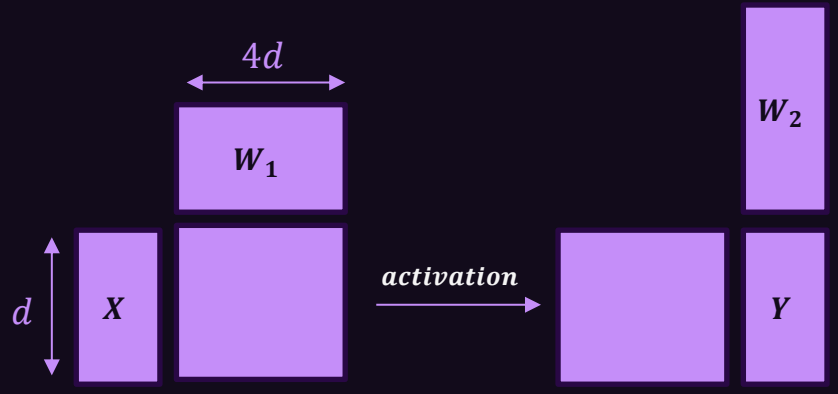
Feed forward Networks encode element-wise non-linearity to help to model complex features.

Input - Embeddings $\boldsymbol{X}$
Output - Embeddings $\boldsymbol{Y}$
Algorithm

- $\boldsymbol{Y = W_2 activation(W_1 X)}$

Remarks

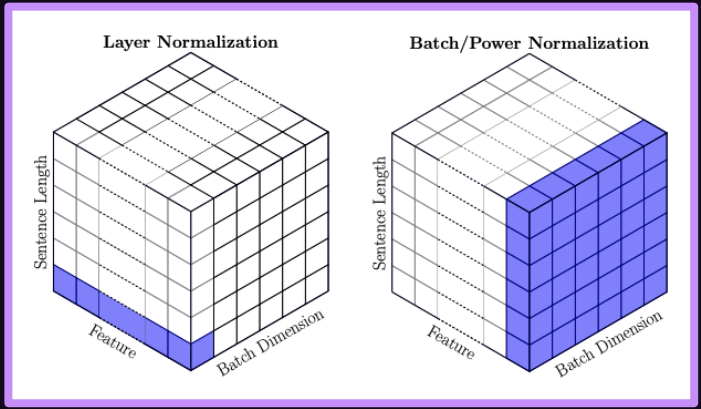
- The activation can be ReLU/GeLU.
- The intermediate dimension is often larger (e.g. $4 \times d$) → Inverted bottleneck



Layer Normalization

Input - Embeddings X
Output - Embeddings Y
Algorithm
- Normalize across features dimension (not batch dimension)

Remarks
- Batch normalization depends on batch size → Less suited for small batch size.
- Layer normalization does not depend on batch size → More suited for small batch size.
- Layer normalization applied before MHA and FFN show better stability.



Reference:
- https://proceedings.mlr.press/v119/shen20e/shen20e.pdf

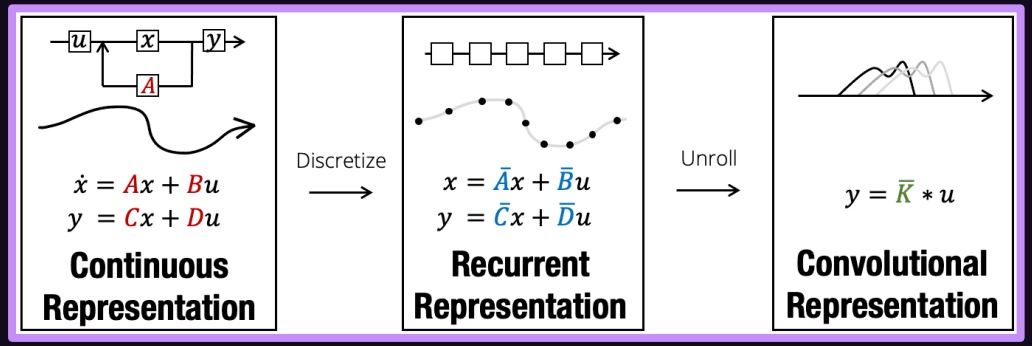
State Space LLMs

State Space Models

The States Spaces Models are traditionally used in control theory to model a dynamic system via state variables. It makes them suited to model sequences.

There are 3 state space representations:

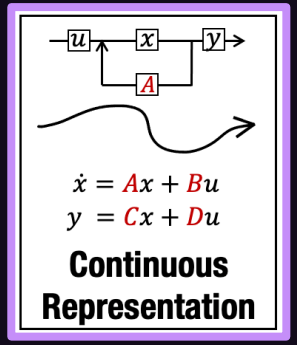
- **The continuous representation**
- **The recurrent representation**
- **The convolutional representation**



Reference:

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3
- https://arxiv.org/abs/2310.18780
- https://arxiv.org/pdf/2111.00396

Continuous Representation



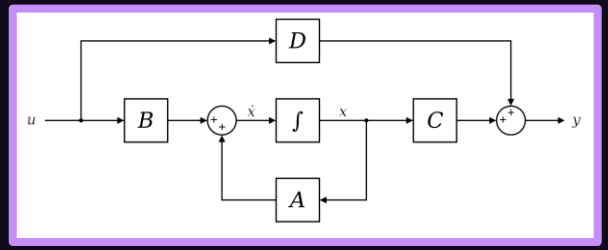
The continuous representation assumes continuous time to describe dependencies between the input, the state, and the output variables.

Input - The input $U(t) \in \mathbb{R}$**.**
Output - The output $Y(t) \in \mathbb{R}$**.**
Algorithm

- The matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ are learnable parameters.
- The variable $X(t) \in \mathbb{R}^p$ is the state variable.
- $X'(t) = \boldsymbol{A}X(t) + \mathbf{B}U(t)$
- $Y(t) = \boldsymbol{C}X(t) + \mathbf{D}U(t)$

Remark

- The state variable $X(t)$ can store long term information
- This representation is convenient for inherently continuous data (e.g. audio signals, time series…) which can have irregular timestamps.
- It requires integration which can be costly for both training and inference.
- It motivates/justifies discrete representations and matrix initializations (e.g. HiPPO matrices $\boldsymbol{A}$).



Reference:

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

Recursive Representation

The recursive representation discretizes the continuous representation to describe the current variables based on the previous time steps.

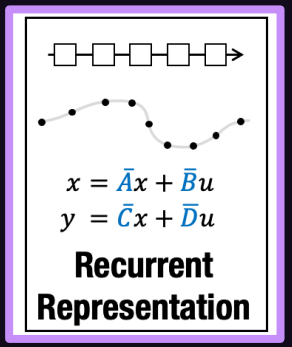
Input - The input $X_k \in \mathbb{R}$.
Output - The output $Y_k \in \mathbb{R}$.
Algorithm

- The matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ are learnable parameters.
- The variable $X \in \mathbb{R}^p$ is the state variable.
- The update is:

$$X_k = \overline{\boldsymbol{A}} X_{k-1} + \overline{\boldsymbol{B}} U_k$$
$$Y_k = \overline{\boldsymbol{C}} X_k + \overline{\boldsymbol{D}} U_k$$

Remarks

- It behaves like a simple RNN with input, hidden state, and an output.
- There are multiple formulation depending on the integration approximation. Integration should be accurate, causal, time-invariant, efficient.
- It can handle long range with theoretically unbounded context.
- It is fast for inference.
- It is slow for training since generation is sequential.



To discretize the continuous case, let's use the trapezoid method where the principle is to assimilate the region under the representative curve of a function f defined on a segment $[t_n, t_{n+1}]$ to a trapezoid and calculate its area $T : T = (t_{n+1} - t_n)\frac{f(t_n)+f(t_{n+1})}{2}$.

We then have: $x_{n+1} - x_n = \frac{1}{2}\Delta(f(t_n) + f(t_{n+1}))$ with $\Delta = t_{n+1} - t_n$.
If $x'_n = \mathbf{A}x_n + \mathbf{B}u_n$ (first line of the SSM equation), corresponds to f, so:

$$x_{n+1} = x_n + \frac{\Delta}{2}(\mathbf{A}x_n + \mathbf{B}u_n + \mathbf{A}x_{n+1} + \mathbf{B}u_{n+1})$$
$$\iff x_{n+1} - \frac{\Delta}{2}\mathbf{A}x_{n+1} = x_n + \frac{\Delta}{2}\mathbf{A}x_n + \frac{\Delta}{2}\mathbf{B}(u_{n+1} + u_n)$$
$$(*) \iff (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})x_{n+1} = (\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + \Delta\mathbf{B}u_{n+1}$$
$$\iff x_{n+1} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B}u_{n+1}$$

(*) $u_{n+1} \overset{\Delta}{\simeq} u_n$ (the control vector is assumed to be constant over a small Δ).

$$\overline{\boldsymbol{A}} = (\boldsymbol{I} - \frac{\Delta}{2} \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \frac{\Delta}{2} \cdot \boldsymbol{A})$$
$$\overline{\boldsymbol{B}} = (\boldsymbol{I} - \frac{\Delta}{2} \cdot \boldsymbol{A})^{-1}\Delta\boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$
$$\overline{\boldsymbol{D}} = \boldsymbol{D}$$

Trapezoid integration

$$\overline{\boldsymbol{A}} = e^{\Delta\boldsymbol{A}}$$
$$\overline{\boldsymbol{B}} = \boldsymbol{A}^{-1}(e^{\Delta\boldsymbol{A}} - \boldsymbol{I})\boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$
$$\overline{\boldsymbol{D}} = \boldsymbol{D}$$

Continuous SSM on step/held signal

$$\overline{\boldsymbol{A}} = \boldsymbol{I} + \Delta\boldsymbol{A}, \overline{\boldsymbol{B}} = \Delta\boldsymbol{B}.$$

Euler integration

Reference:

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3

Convolutional Representation

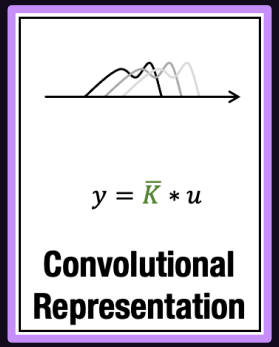
The recursive representation discretizes the continuous representation to describe the current variables based on the previous time steps.

Input - The input $X_k \in \mathbb{R}$.
Output - The output $Y_k \in \mathbb{R}$.
Algorithm

- The matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ are learnable parameters.
- The variable $X \in \mathbb{R}^p$ is the state variable.
- When unrolling the recurrence update, we obtain:

$$x_0 = \overline{B}u_0 \quad x_1 = \overline{AB}u_0 + \overline{B}u_1 \quad x_2 = \overline{A}^2\overline{B}u_0 + \overline{AB}u_1 + \overline{B}u_2 \quad \dots$$

Hidden state as a convolution



$$y_k = \overline{CA}^k\overline{B}u_0 + \overline{CA}^{k-1}\overline{B}u_1 + \cdots + \overline{CAB}u_{k-1} + \overline{CB}u_k$$

By extracting these coefficients into what we call the **SSM kernel** $\overline{K}$

$$\overline{K} = \left(\overline{CA}^i\overline{B}\right)_{i\in[L]} = (\overline{CB}, \overline{CAB}, \dots, \overline{CA}^{L-1}\overline{B})$$

Output as a convolution

Remarks

- The output attends to local information with a bounded context.
- It enables fast training since convolution is parallelizable.
- The context length of the convolution kernel can make the inference slower/faster.
- It is possible to take the best of recursive and convolutional representations by (1) estimating the the SSM from a convolution, or (2) expressing $\overline{A}$ in a specific basis where the convolution kernel is fast to compute.

Reference:
- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3
- https://arxiv.org/abs/2310.18780
- https://arxiv.org/pdf/2111.00396

Deep State Space Models

Deep State space models stack state space blocks and non-linearity to model complex dependencies between tokens.

Input - Often Texts (but could be other data types)
Output - Often Texts (but could be other data types)
Algorithm

- Tokenizer
- Embeddings: Token
- State space blocks
- Normalization
- Feed Forward

Remarks

- State space layers act on each feature independently.
- Feed forward layers mix features together.
- Deep state space models parametrize global filters over sequence length.
- SSM can attend long range information.
- They have constant compute and memory requirement with recurrent representation i.e. it can be fast.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4	**58.35**	**76.02**	**87.09**	**87.26**	**86.05**	**88.10**	**80.48**

Model	Params	Test ppl.	Tokens / sec
Transformer	247M	**20.51**	0.8K (1×)
GLU CNN	229M	37.2	-
AWD-QRNN	151M	33.0	-
LSTM + Hebb.	-	29.2	-
TrellisNet	180M	29.19	-
Dynamic Conv.	255M	25.0	-
TaLK Conv.	240M	23.3	-
S4	249M	**21.28**	**48K (60×)**

WikiText-103 (perplexity)

Reference:

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3
- https://arxiv.org/abs/2310.18780
- https://arxiv.org/pdf/2111.00396

Diffusion LLMs

Diffusion LLMs

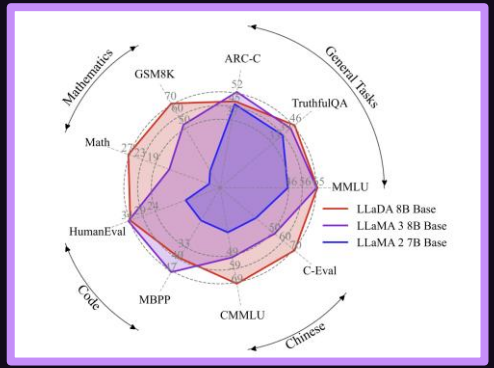
Diffusion LLMs models a diffusion process denoising tokens from fully masked to fully masked.

Input - Often Texts (but could be other data types)
Output - Often Texts (but could be other data types)
Algorithm

- Tokenizer
- Embeddings: Token
- Mask predictor blocks
  - The mask predictor predicts all the masked tokens.
  - Selected tokens (e.g. the least confident) are remasked.

Remark

- It can do semi-autoregressive masking. By predicting tokens block after tokens block.
- It can be fast by generating multiple tokens at once.
- It can naturally do in-context learning.



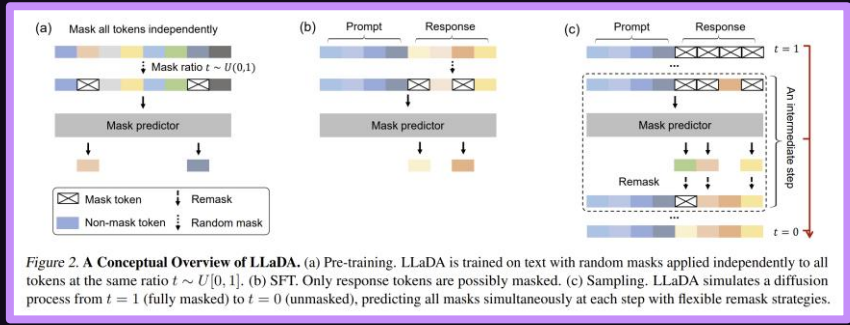


Figure 2. **A Conceptual Overview of LLaDA.** (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked)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

Reference:

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3
- https://arxiv.org/abs/2310.18780
- https://arxiv.org/pdf/2111.00396

Derivative LLMs Architectures

Mixture of Experts

Mixture of Experts is composed of a router network which assigned inputs to one of the expert networks.

Input - Embeddings
Output - Embeddings
Algorithm

- Router model that predicts which expert should be used.
- The selected network in the Mixture of expert networks which performs element-wise non-linearities.

Remarks

- MoE requires a lot of memory to fit all the expert networks on the hardware.
- MoE is compute efficient during training and inference since a subset of the parameters are active.
- MoE are often used for FFN in Autoregressive architectures.
- Each expert network specializes on some topic.
- Key examples are Llama 4, Mixtral.

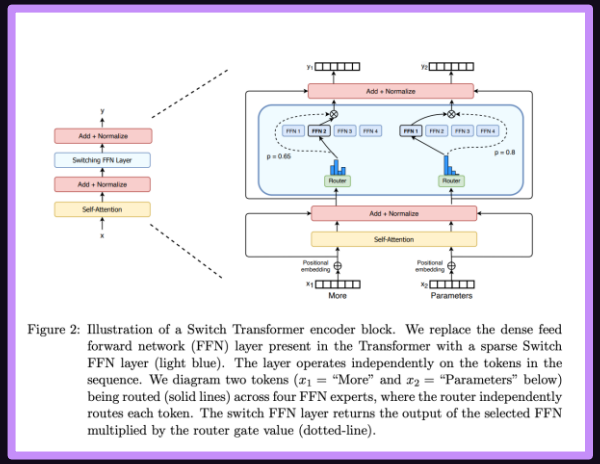


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters"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

Expert specialization	Expert position	Routed tokens
Sentinel tokens	Layer 1	been <extra_id_4><extra_id_7>floral to <extra_id_10><extra_id_12><extra_id_15> <extra_id_17><extra_id_18><extra_id_19>...
	Layer 4	<extra_id_0><extra_id_1><extra_id_2> <extra_id_4><extra_id_6><extra_id_7> <extra_id_12><extra_id_13><extra_id_14>...
	Layer 6	<extra_id_0><extra_id_4><extra_id_5> <extra_id_6><extra_id_7><extra_id_14> <extra_id_16><extra_id_17><extra_id_18>...
Punctuation	Layer 2	, , , , , , , , , " , , , , ,) .)
	Layer 6	, , , , , : ; , & , & & ? & - , , ? , , , . <extra_id_27>
Conjunctions and articles	Layer 3	The the the the the the the the the The the the the the The the the the
	Layer 6	a and and and and and and and and or and a and . the the if ? a designed does been is not
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died falling designed based disagree submitted develop
Visual descriptions *color, spatial position*	Layer 0	her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue
Proper names	Layer 1	A Mart Gr Mart Kent Med Cor Tri Ca Mart R Mart Lorraine Colin Ken Sam Ken Gr Angel A Dou Now Ga GT Q Ga C Ko C Ko Ga G
Counting and numbers *written and numerical forms*	Layer 1	after 37 19. 6. 27 11 Seven 25 4, 54 1 two dead we Some 2012 who we few lower each

Reference:

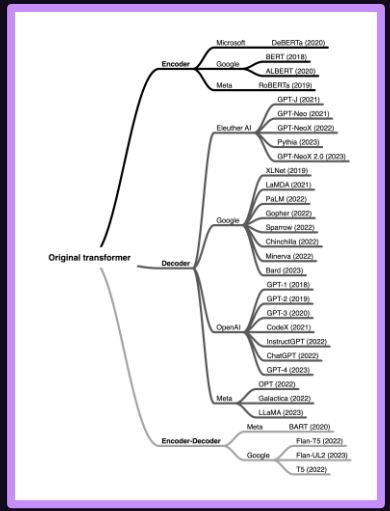
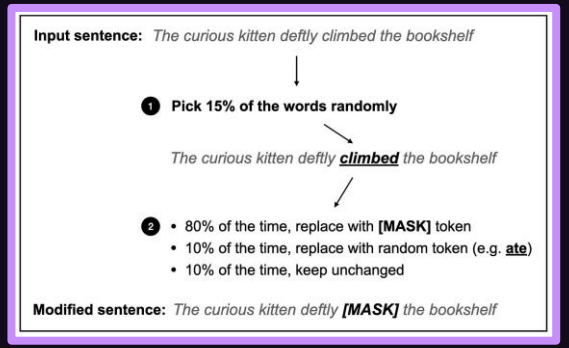
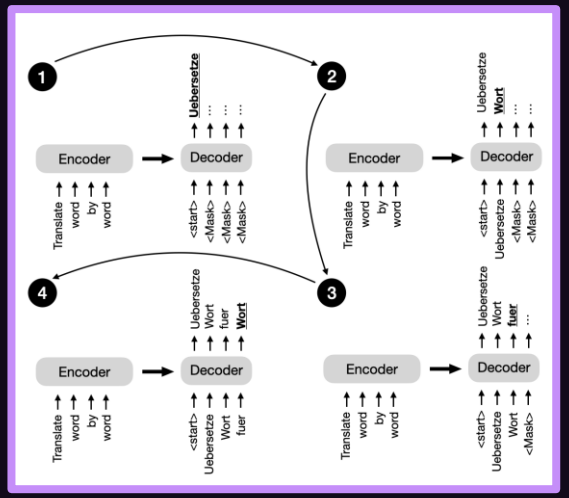
- https://arxiv.org/abs/2101.03961
- https://huggingface.co/blog/moe

Encoder & Decoder

LLMs building blocks can be arranged in different ways to solve their target tasks.

Encoder only
- Example: BERT, RoBERTa
- Use-case: learn (task specific) embeddings
- Training objective: predict masked inputs, sentence order

Encoder-Decoder
- Example: T5
- Use-case: generate texts
- Training objective: Next word prediction

Decoder only
- Example: GPT, Llama, Mistral…
- Use-case: generate texts
- Training objective: Next word prediction

Remarks
- Encoder can use no masks in the attention mechanisms.
- Decoder usually use causal masks.