UiT The Arctic University of Norway

BIO-AI LAB | ARCTIC LLM WORKSHOP 2023

Large Language Models

Day 2 - Session 5

Improvement in self-attention in LLMs

Nirwan Banerjee

nirwan.banerjee@uit.no

28. Oct 2023

Introduction

We have learnt much about the recent advancements with regards to Large Language Models (LLMs) that include its history, interpretability, finetuning and the various prompting techniques used in the field today. However, we have yet to have a peek at the magic that unfolds behind the veil of the fancy nomenclatures and architectures of LLMs.

Before we start

We want our model to assign a higher probability to sentences like "Japan is the home of cherry blossoms" and not "Food polar charger human". Language models can be divided into three types based on their training methodology.

Before we start

1. *Causal Language Modelling (CLM): These are also called autoregressive models. The model is trained to predict the next word/s in a sequence, given the knowledge of the previous words. Ex: GPT*

2. *Masked Language Modelling (MLM): Also known as 'fill in the blanks' technique, models are trained to fill in missing words in the training data. This gives them bi-directional context. Ex: BERT*

3. *Sequence-to-Sequence (Seq2Seq): In seq2seq modelling, the model is trained to generate output sequences given an input sequence. Ex: LSTM*

Before we start

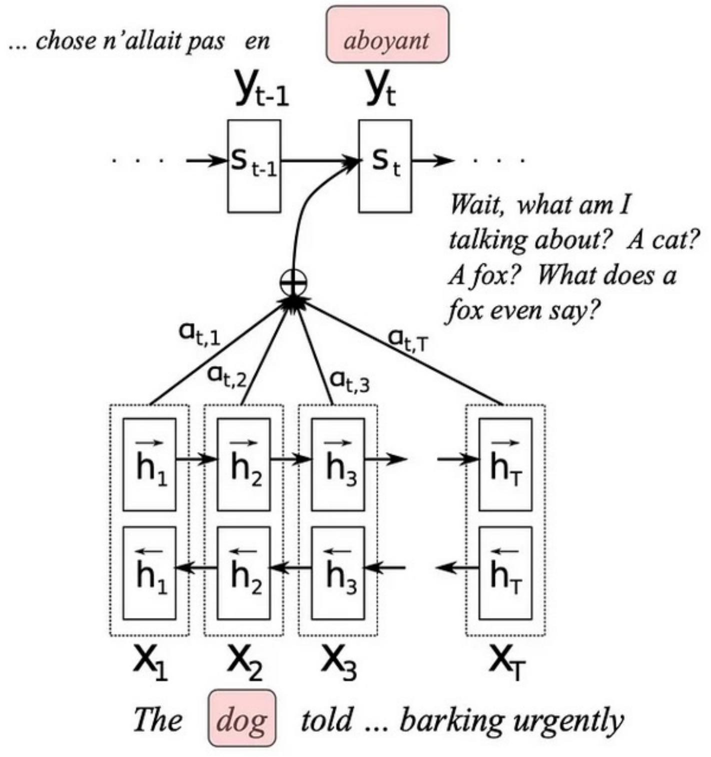
To put this another way, CLM language models try to estimate, “what word comes next” as opposed to MLM models, which “fill in the blanks”. Seq2Seq models try to estimate "what is the next sentence" as a contrast to CLMs.

Long range dependencies

Photo by Anusha Barwa on Unsplash

Let's consider the two sentences, "The dog told them something was amiss by barking urgently.", and "The cat told them something was amiss by meowing urgently". In these sentences, the word that follows "amiss by…" depends a lot on whether the second word was dog or cat.

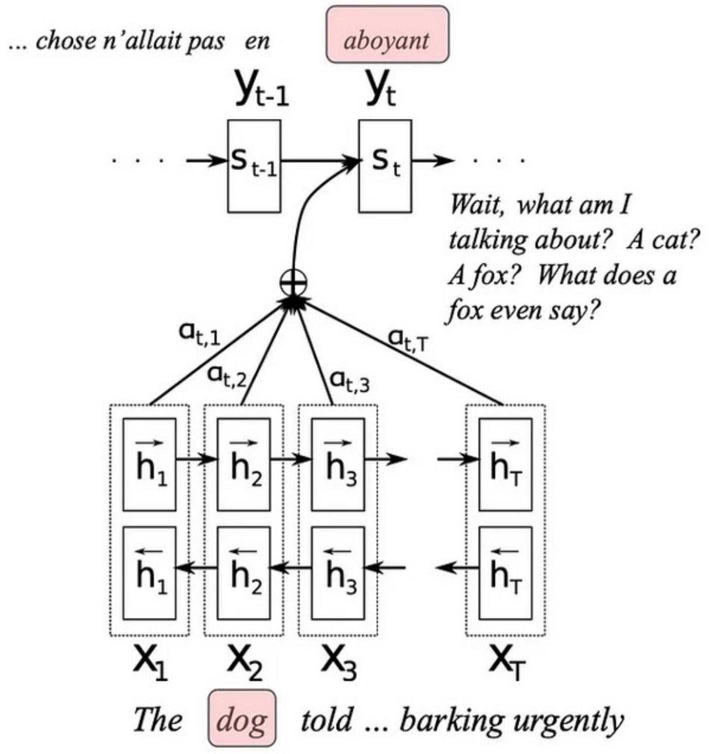
Before the age of transformers


… chose n'allait pas en
aboyant
y_{t-1}
y_t
s_{t-1}
s_t
Wait, what am I talking about? A cat? A fox? What does a fox even say?
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
x_1 x_2 x_3 x_T
The dog told … barking urgently

Adapted from "Neural Machine Translation By Jointly Learning to Align and Translate" by D Bahdanau et al, ICLR 2015.

Neural Machine Translation – Sutskever et al., in 2014, demonstrated the use of recurrent neural nets (RNNs) for machine translation, e.g., given an English sentence, "The dog told them something was amiss by barking urgently", produce a French translation, "Le chien leur a dit que quelque chose n'allait pas en aboyant d'urgence."

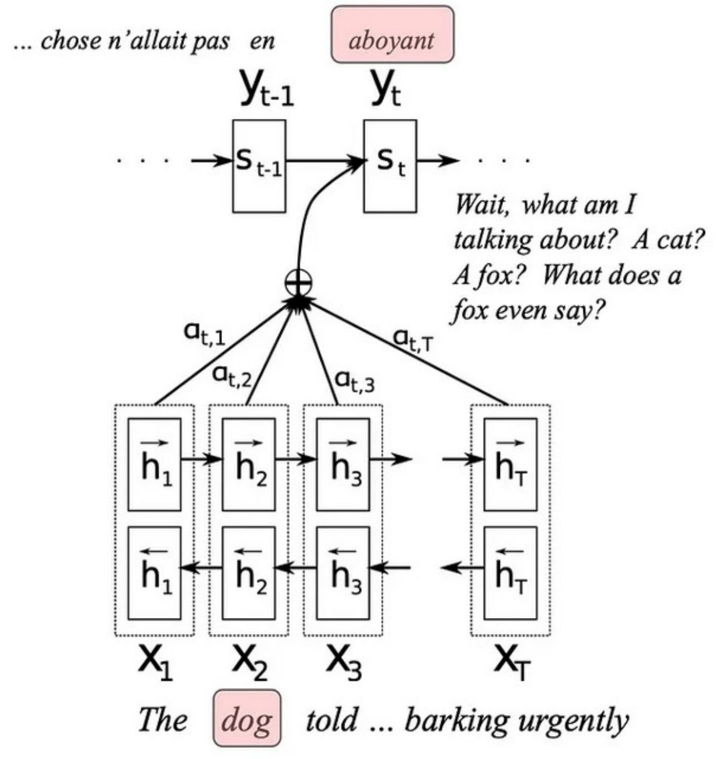
Before the age of transformers


... chose n'allait pas en
aboyant
y_{t-1}
y_t
s_{t-1}
s_t
Wait, what am I talking about? A cat? A fox? What does a fox even say?
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
x_1 x_2 x_3 x_T
The dog told ... barking urgently

Adapted from "Neural Machine Translation By Jointly Learning to Align and Translate" by D Bahdanau et al, ICLR 2015.

Neural Machine Translation – This issue could probably be solved if our model could focus on specific parts of the sentence while generating each French word in arbitrary order. Bahdanau et al., presented work that does exactly this.

Before the age of transformers



... chose n'allait pas en
aboyant
y_{t-1}
y_t
s_{t-1}
s_t
Wait, what am I talking about? A cat? A fox? What does a fox even say?
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
x_1 x_2 x_3 x_T
The dog told ... barking urgently

Adapted from "Neural Machine Translation By Jointly Learning to Align and Translate" by D Bahdanau et al, ICLR 2015.

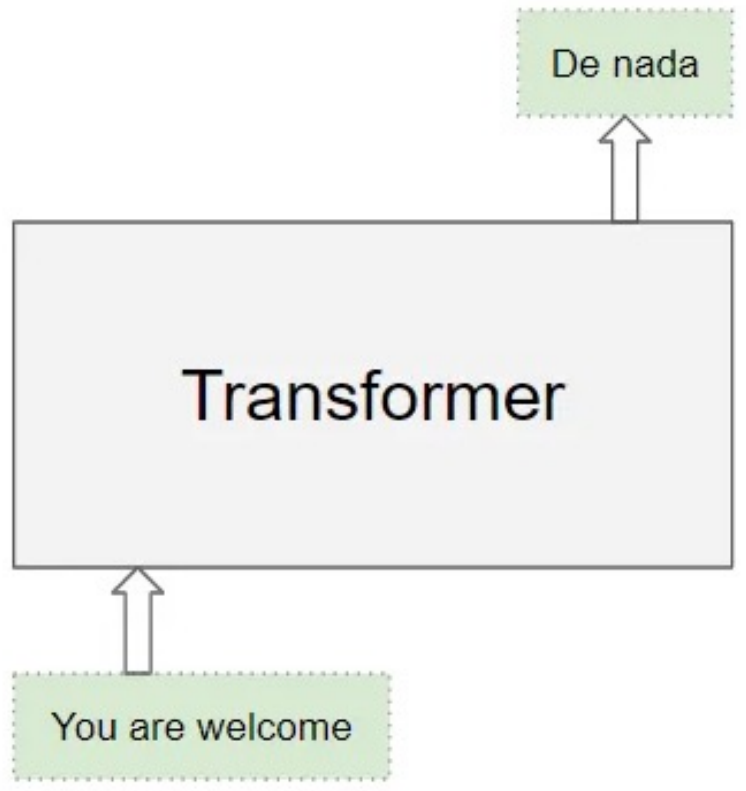
Neural Machine Translation and CLM models – NMT models have a correlation to CLM models. Both of them output one word at a time, in sequence, with the next word depending on previously generated words.

SECTION 2

WHAT ARE TRANSFORMERS?

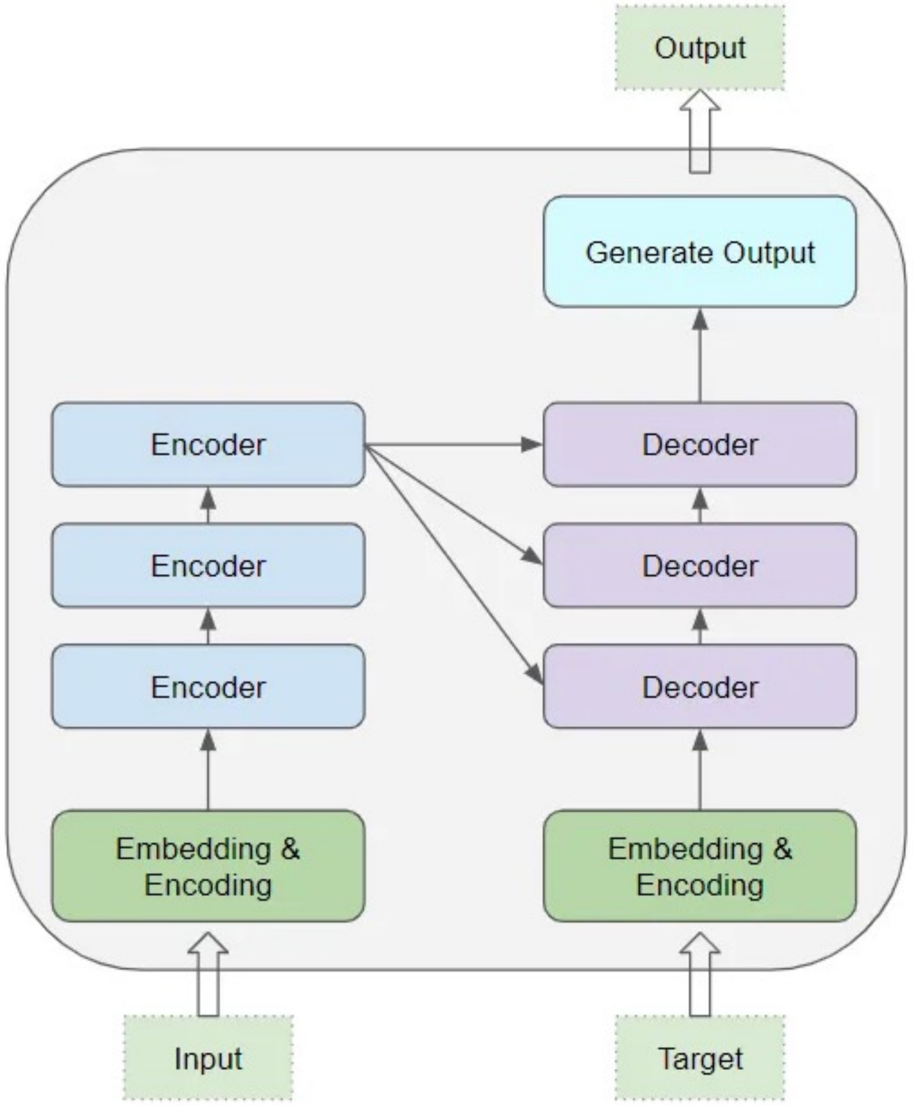
Attention is all you need

- In 2017, Viswani et al presented a new approach to machine translation that didn’t use RNNs at all — the Transformer.

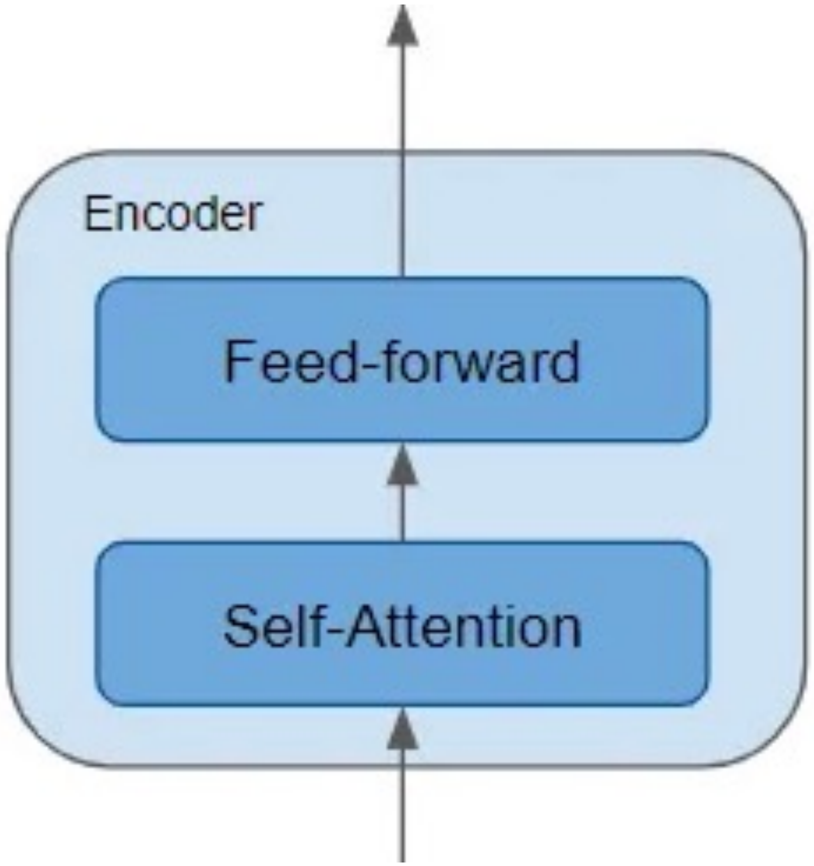


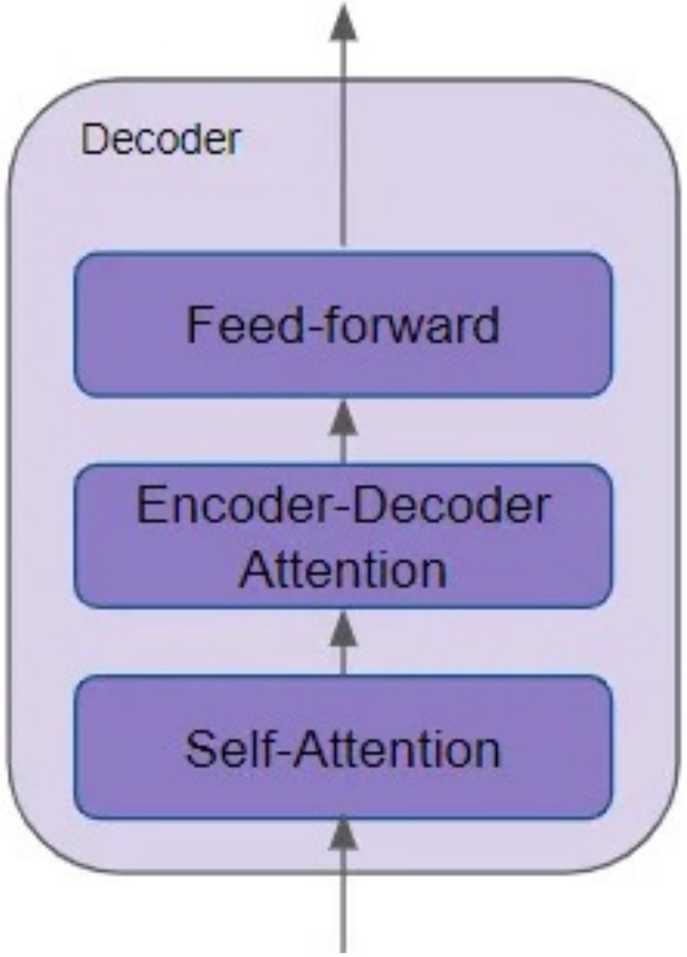
[1] *Lorem ipsum dolor sit amet, consectetur adipiscing elit.* “*Lorem ipsum dolor sit amet, consectetur adipiscing elit.*” *2020 25th International Conference on Pattern Recognition (ICPR),* pp. 7892-7899. IEEE, 2021.

A high level overview - Architecture

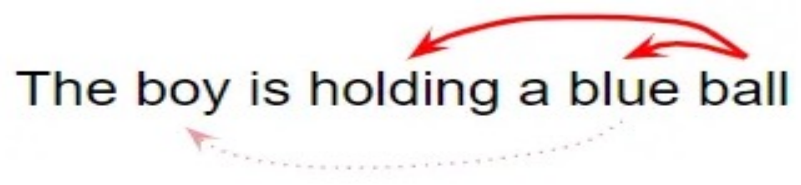


A high level overview – Encoder and Decoder



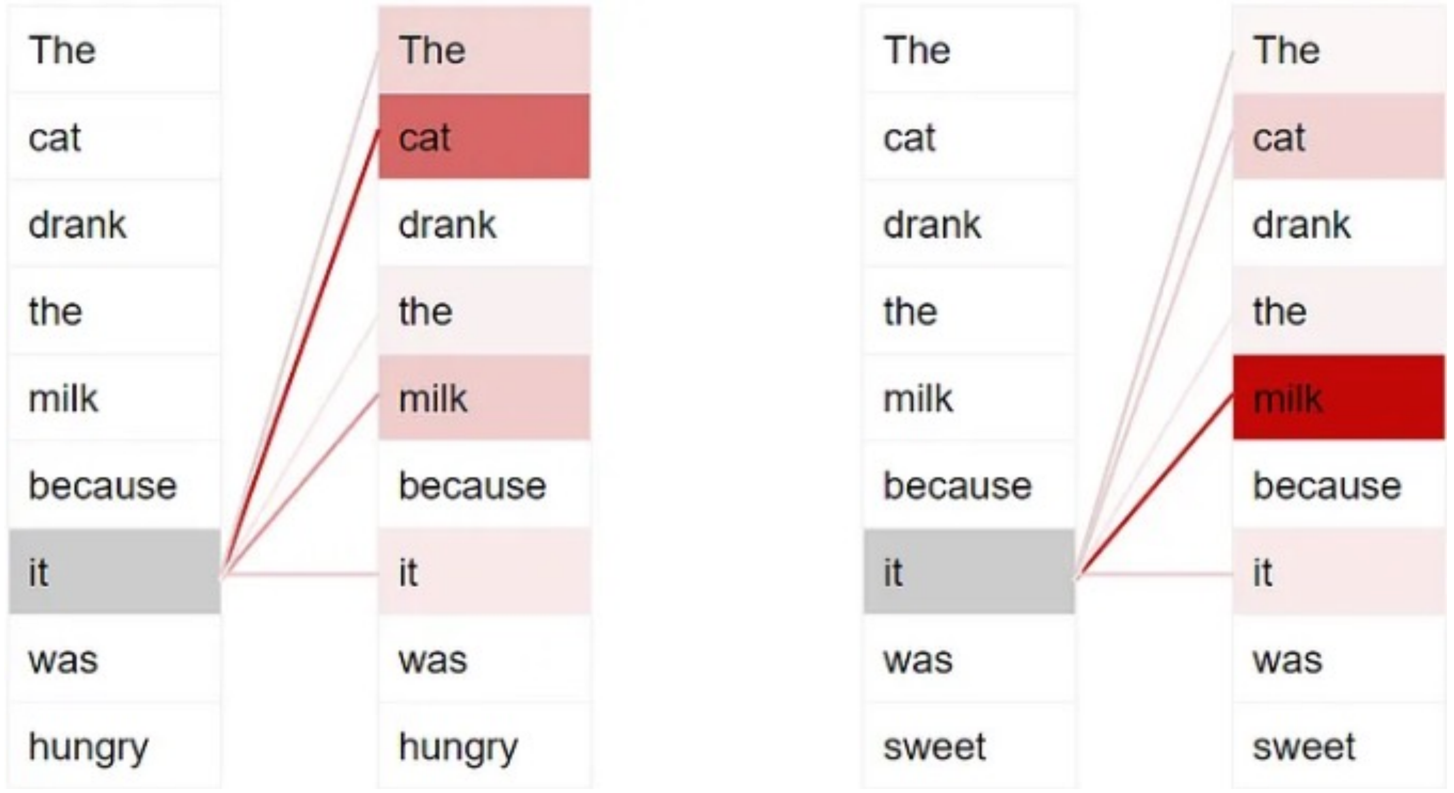


A high level overview - Attention



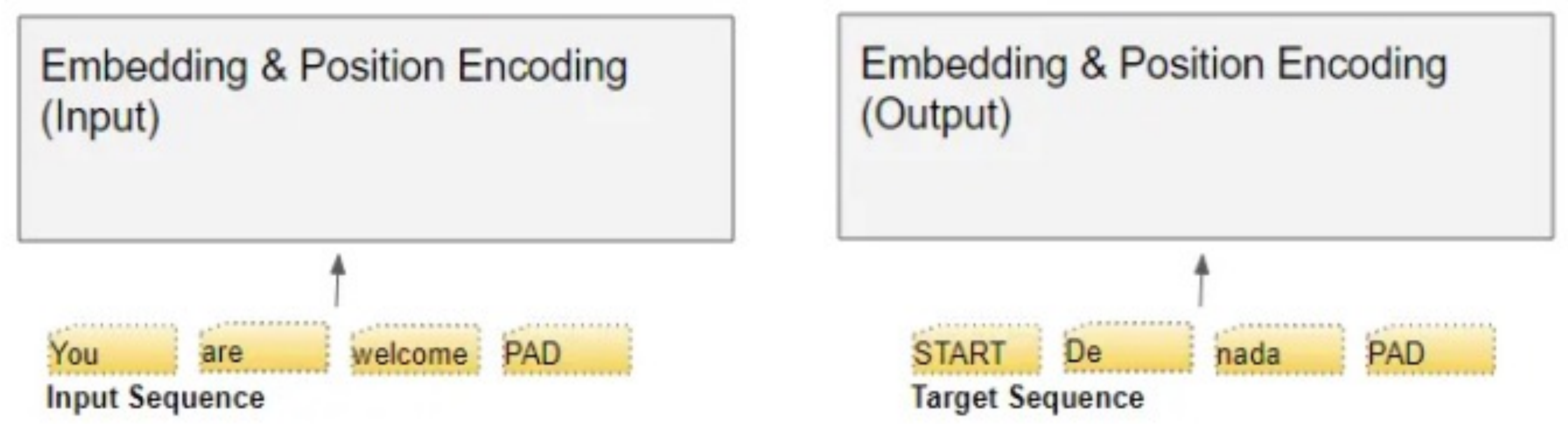
- While processing a word, Attention enables the model to focus on other words in the input that are closely related to that word.
- For example, in the above text, eg. ‘Ball’ is closely related to ‘blue’ and ‘holding’. On the other hand, ‘blue’ is not related to ‘boy’.
- The Transformer architecture uses self-attention by relating every word in the input sequence to every other word.

A high level overview - Attention

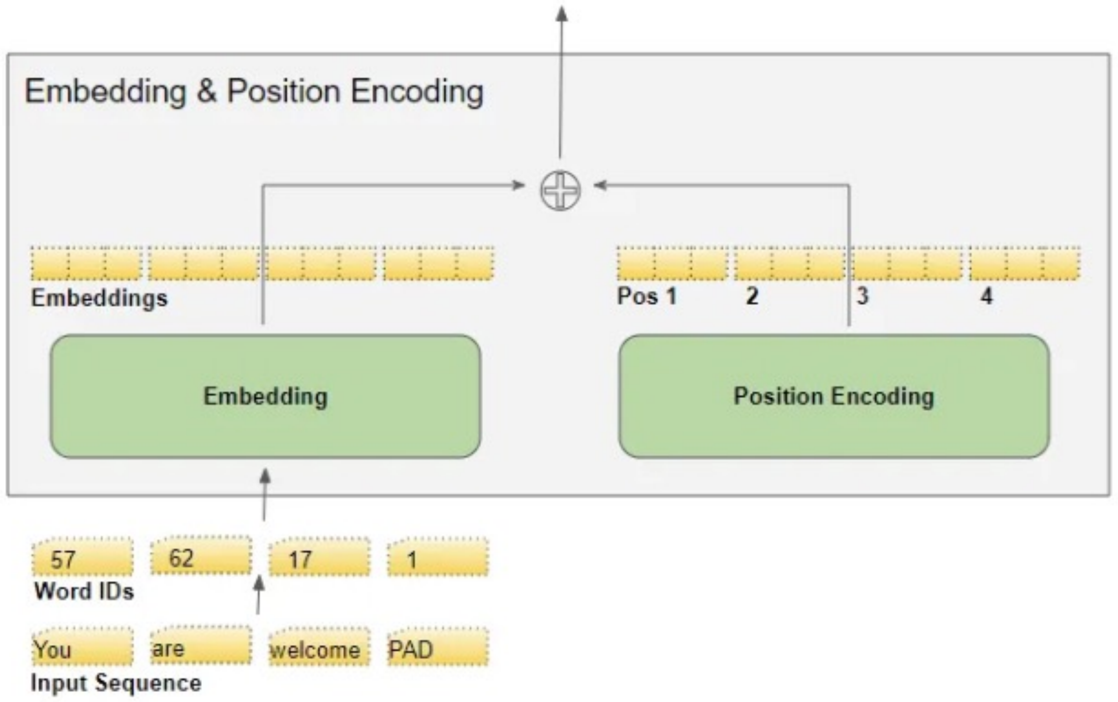


Consider two sentences:
1. The cat drank the milk because it was hungry.
2. The cat drank the milk because it was sweet.

A closer look – Embedding and Position Encoding


Embedding & Position Encoding (Input)
You
are
welcome
PAD
Input Sequence
Embedding & Position Encoding (Output)
START
De
nada
PAD
Target Sequence

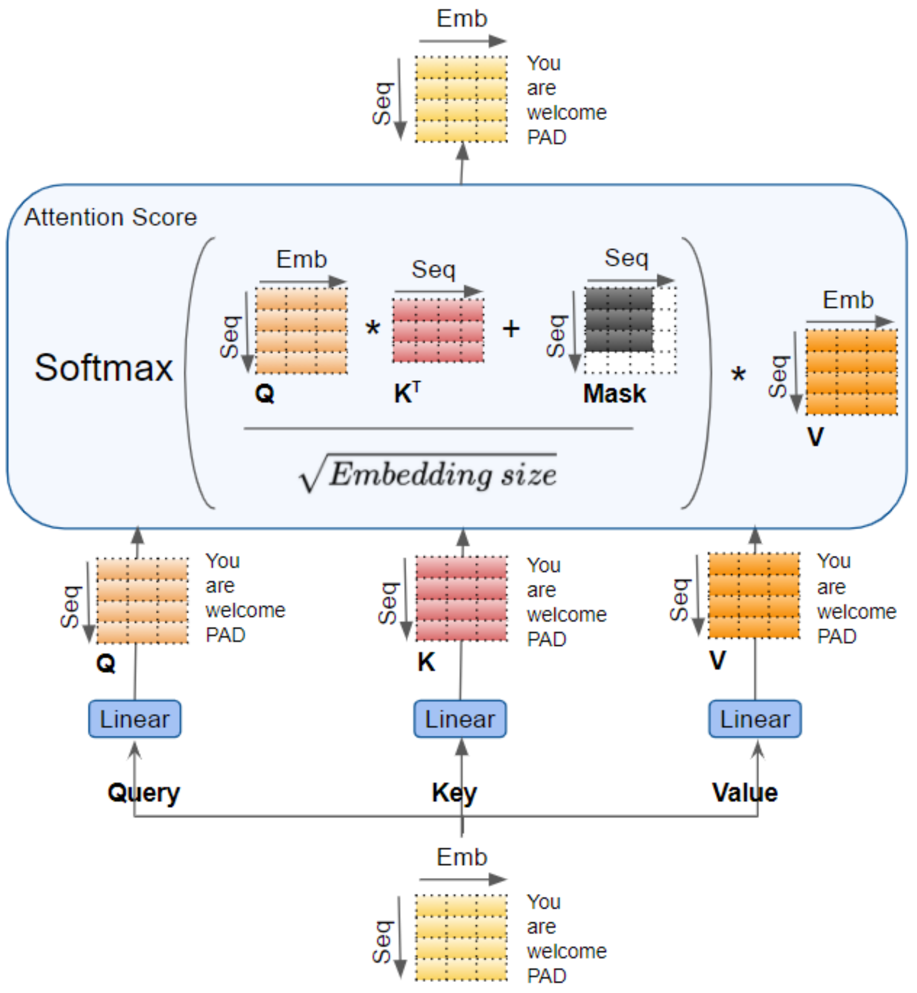
A closer look – Embedding and Position Encoding

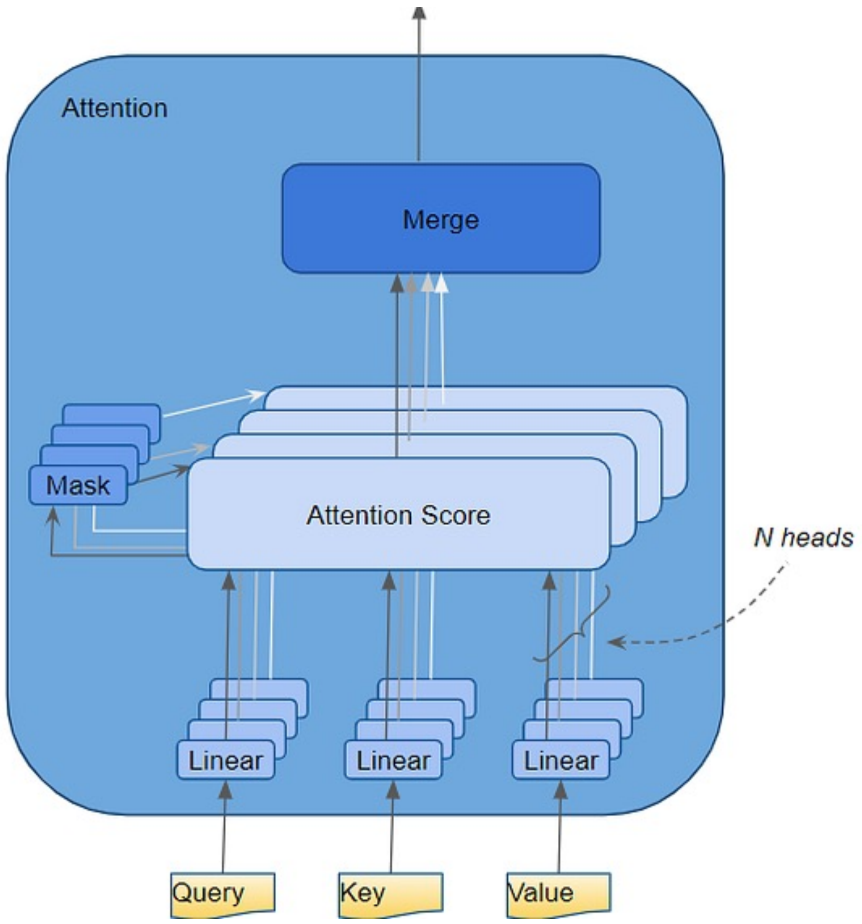


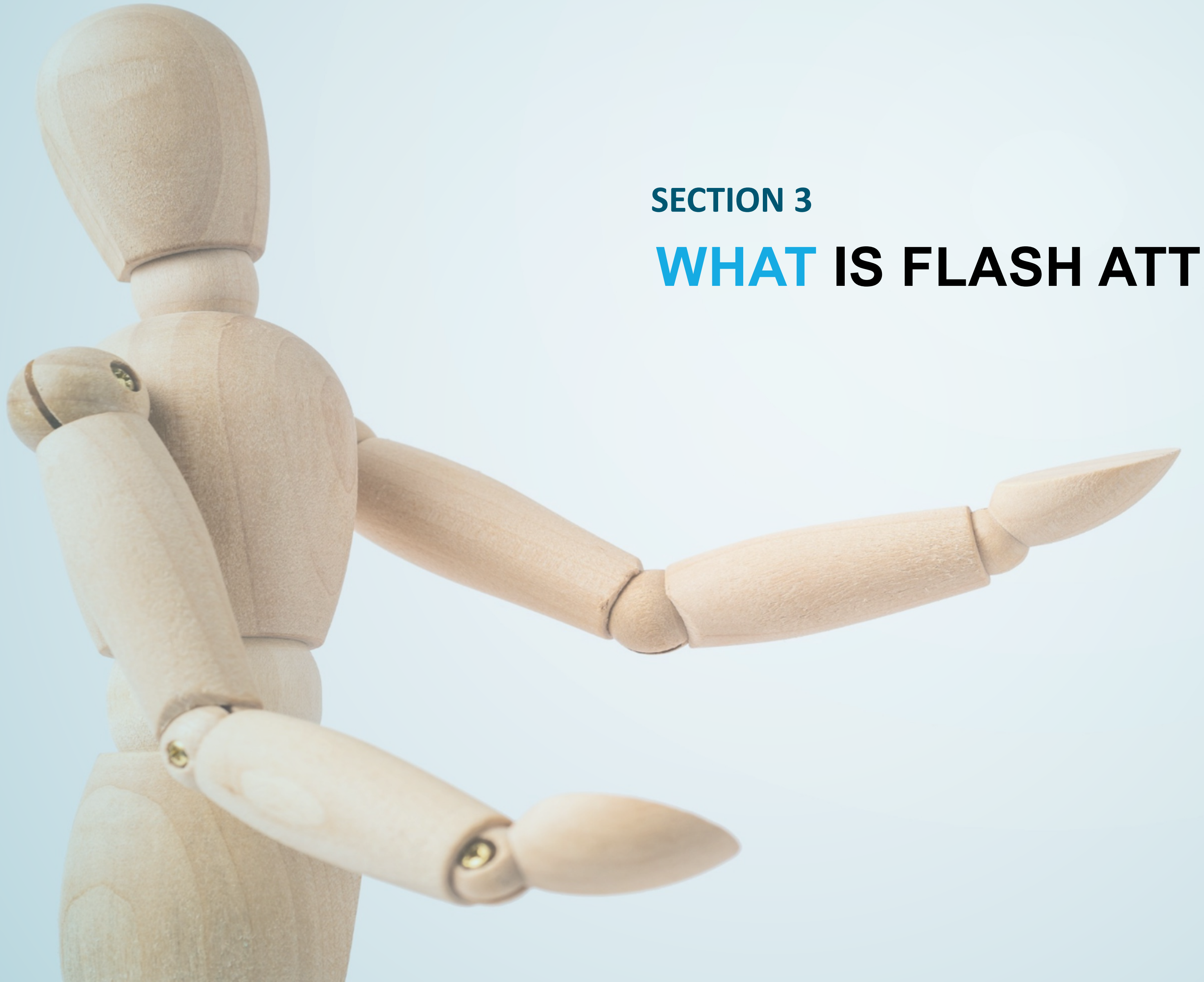
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- Transformers don’t use RNNs and all words in a sequence are input in parallel.
- The lost positional information is added later on through positional encoding.
- This is done through the equations show below where:
  pos is the position of the word in the sequence
  d_model is the length of the encoding vector
  i is the index value into the vector

A closer look – Self and multi-headed attention







SECTION 3

WHAT IS FLASH ATTENTION?

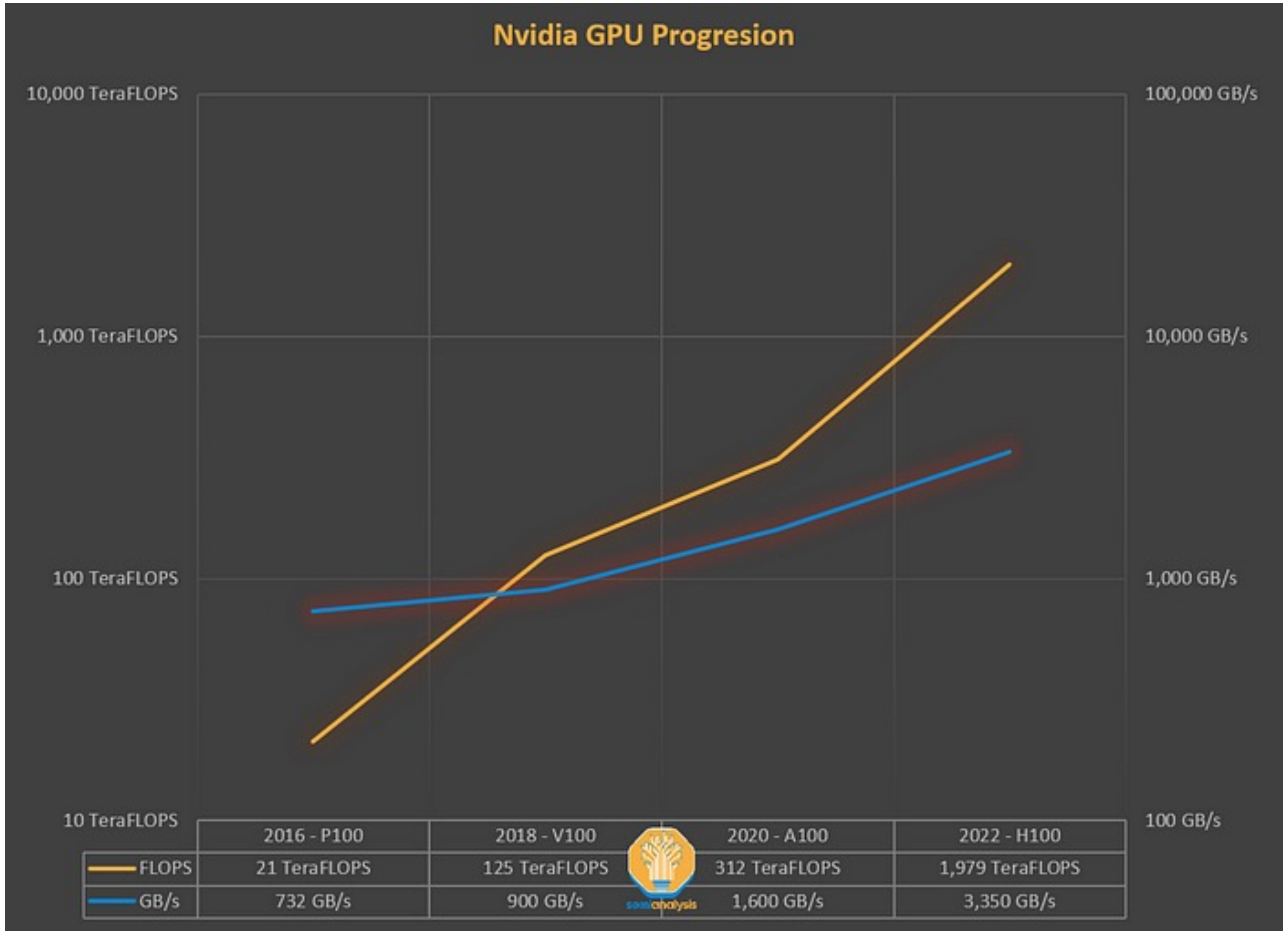
Flash Attention

- Normal self-attention is parallelizable
- However, it’s implementation on the hardware level is very inefficient
- It treats the GPU implementation as a blackbox
- Flash Attention introduces hardware level implementation changes that speeds up training and inference by leaps and bounds

Advantages of Flash Attention

- Fast – Upto 3 times as fast as normal self-attention
- Memory-Efficient – $O(N)$ vs $O(N^2)$
- Exact – Not an approximation
- IO Aware – It is “Sentient”

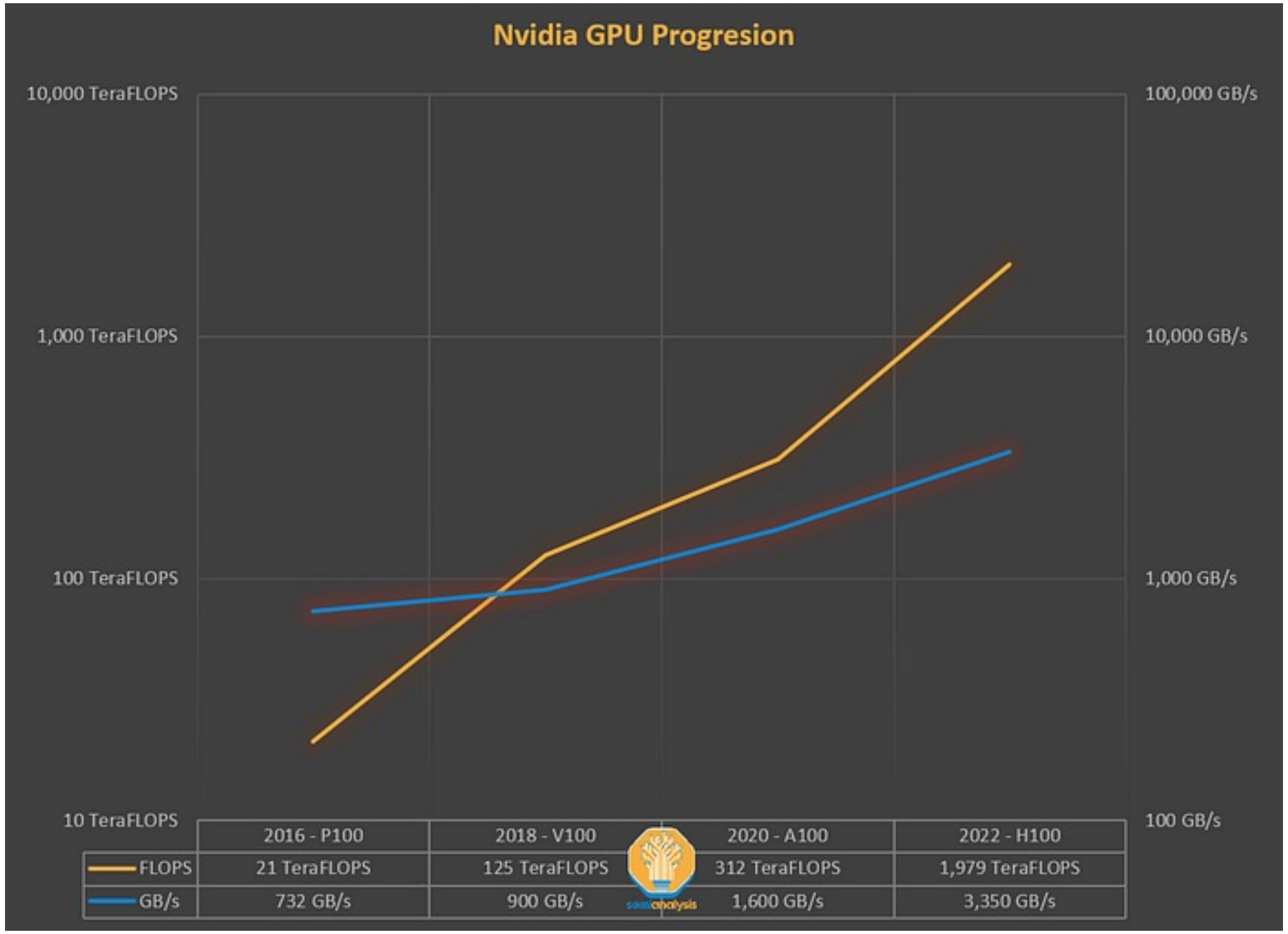
About the hardware



	2016 - P100	2018 - V100	2020 - A100	2022 - H100
FLOPS	21 TeraFLOPS	125 TeraFLOPS	312 TeraFLOPS	1,979 TeraFLOPS
GB/s	732 GB/s	900 GB/s	1,600 GB/s	3,350 GB/s

- GPUs have been adding compute capacity (FLOPS) at a faster pace than increasing the memory throughput (TB/s)
- It doesn’t matter if you can compute at exaFLOPS speeds if there is no data to be processed
- Since the hardware lost that balance we have to make our software compensate for it.

About the hardware

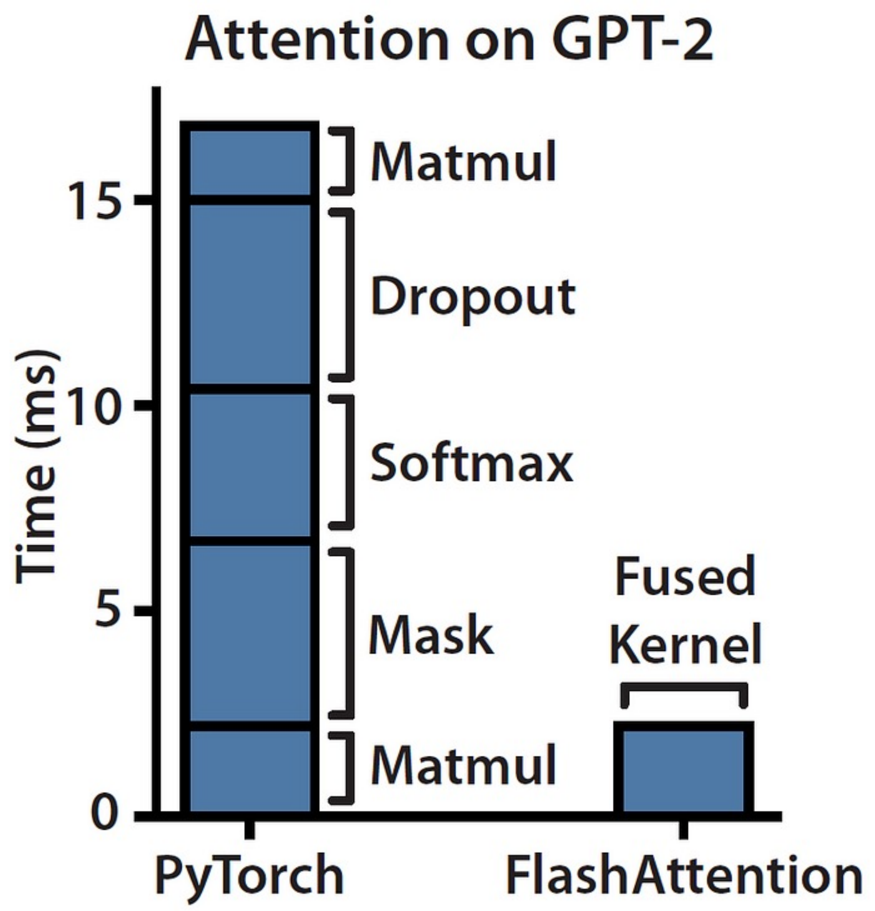


	2016 - P100	2018 - V100	2020 - A100	2022 - H100
FLOPS	21 TeraFLOPS	125 TeraFLOPS	312 TeraFLOPS	1,979 TeraFLOPS
GB/s	732 GB/s	900 GB/s	1,600 GB/s	3,350 GB/s

- Depending on this ratio, between computation and memory accesses, operations can be classified as either:
  - Compute bound – Matrix multiplication
  - Memory bound – elementwise ops (activation, dropout, etc)

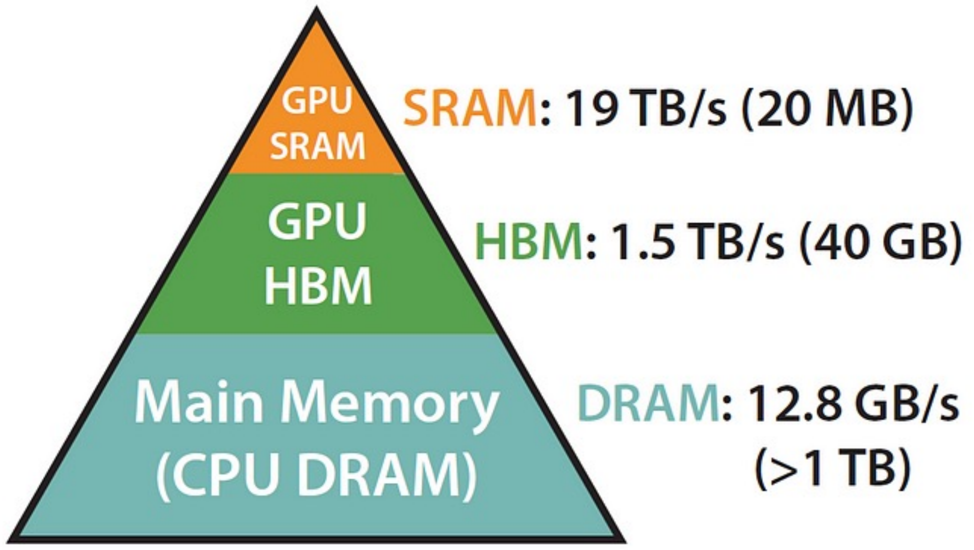
  Attention is memory-bound because it consists mostly of elementwise ops

About the hardware



- Masking, softmax and dropout take the bulk of the time for computation
- Not matrix multiplication
- Bulk of FLOPS is in matrix multiplication

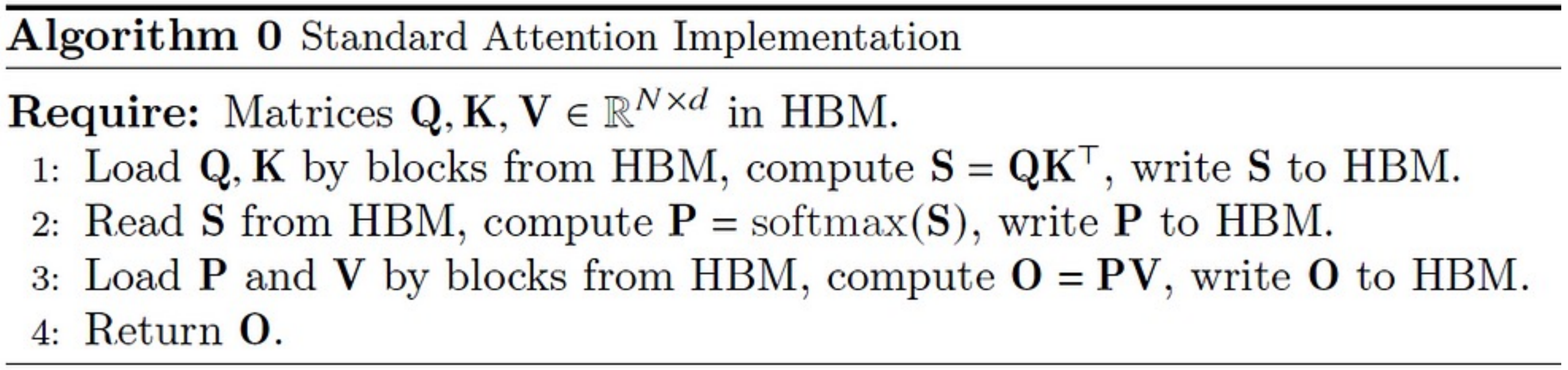
About the hardware



Memory Hierarchy with Bandwidth & Memory Size

- “IO-aware” in practice boils down to exploiting the fact that SRAM is so much faster than HBM
- A100 GPU has 40–80GB of high bandwidth memory
- A bandwidth of 1.5–2.0 TB/s and 192KB of on-chip SRAM per each of 108 streaming multiprocessors with bandwidth estimated around 19TB/s

About the hardware

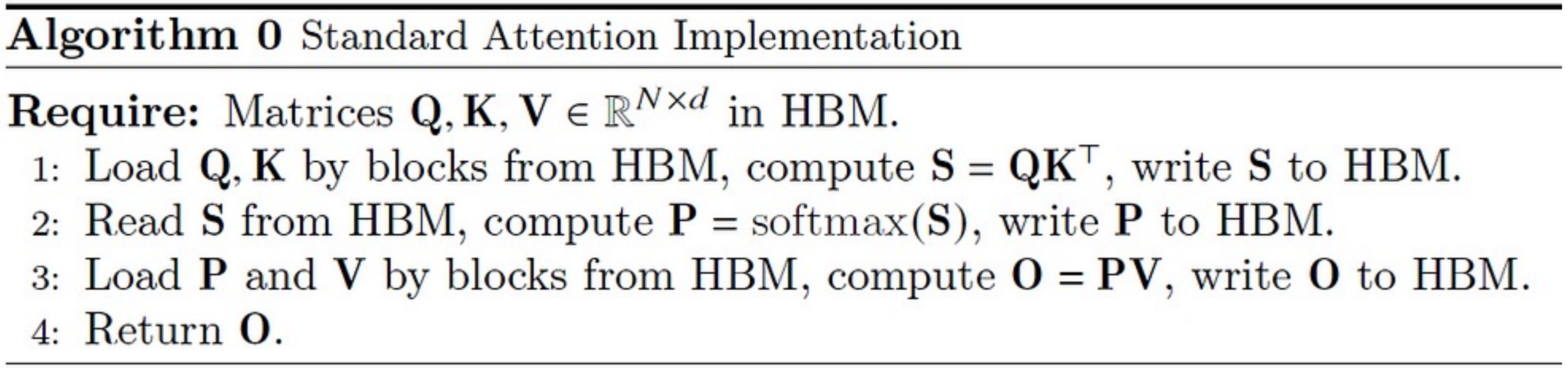


Algorithm 0 Standard Attention Implementation

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.
1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.
```

- Unnecessary step in step 1 by writing to HBM and then immediately reading it
- We can remove these redundant steps instead and do “kernel fusion”

About the hardware



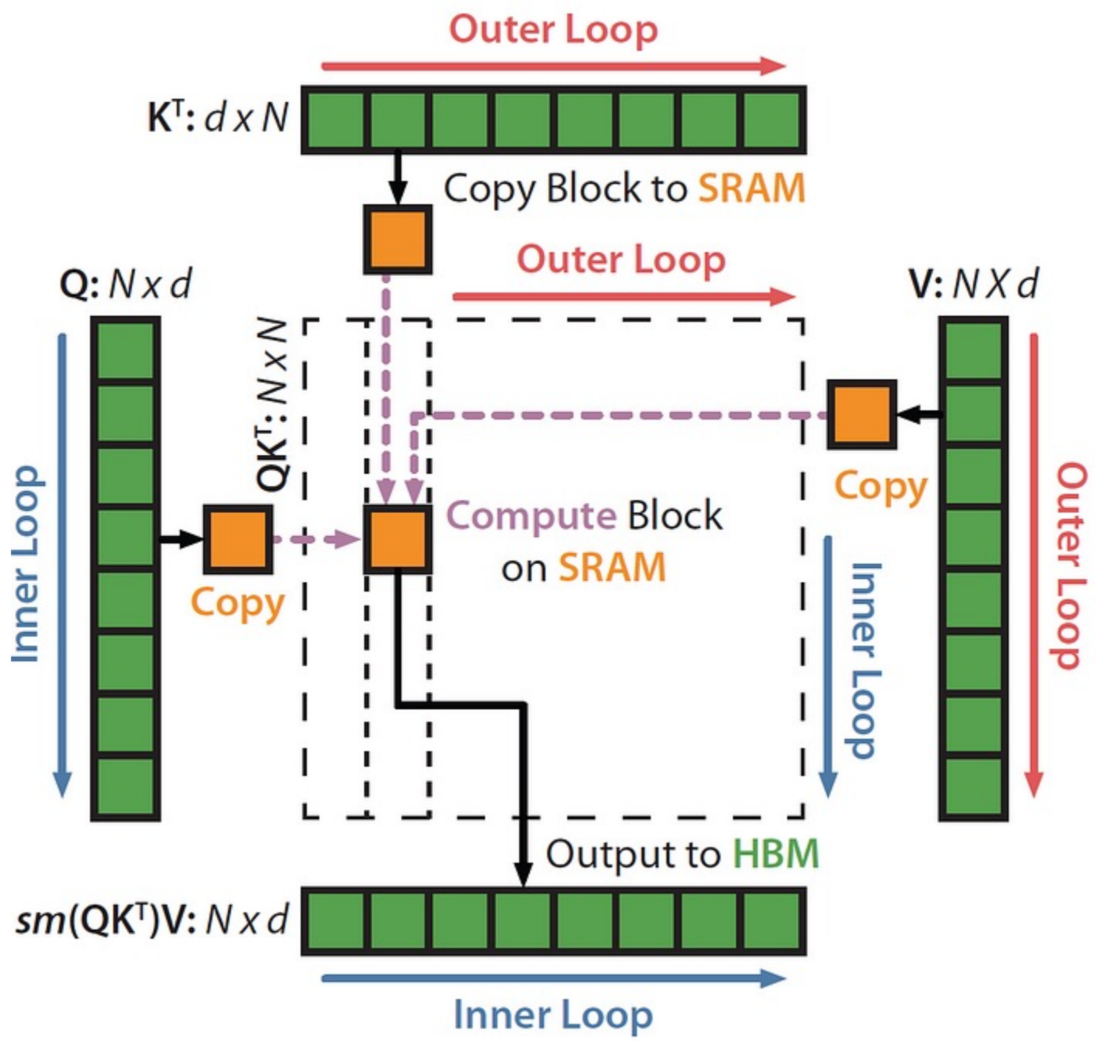
Algorithm 0 Standard Attention Implementation

```
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.
1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.
```

Flash Attention basically boils down to two steps

- Tiling
- Recomputation scores

About the hardware



Current scenario

The same thing that gives flash attention it’s power is also it’s drawback

Each new attention needs a new implementation

Consequently flash attention is only supported for a small subset of GPUs

Writing CUDA kernel code is messy and machine learning engineers who are mainly familiar with python don’t find it worthwhile to get their hands dirty.

Current scenario

- *OpenAI Triton*
- *Triton is basically a DSL (domain-specific language) between CUDA & other DSLs*
- *Needs to be adopted for frameworks like PyTorch*

PrivateGPT++

A completely secure and private way of using LLMs on your private documents

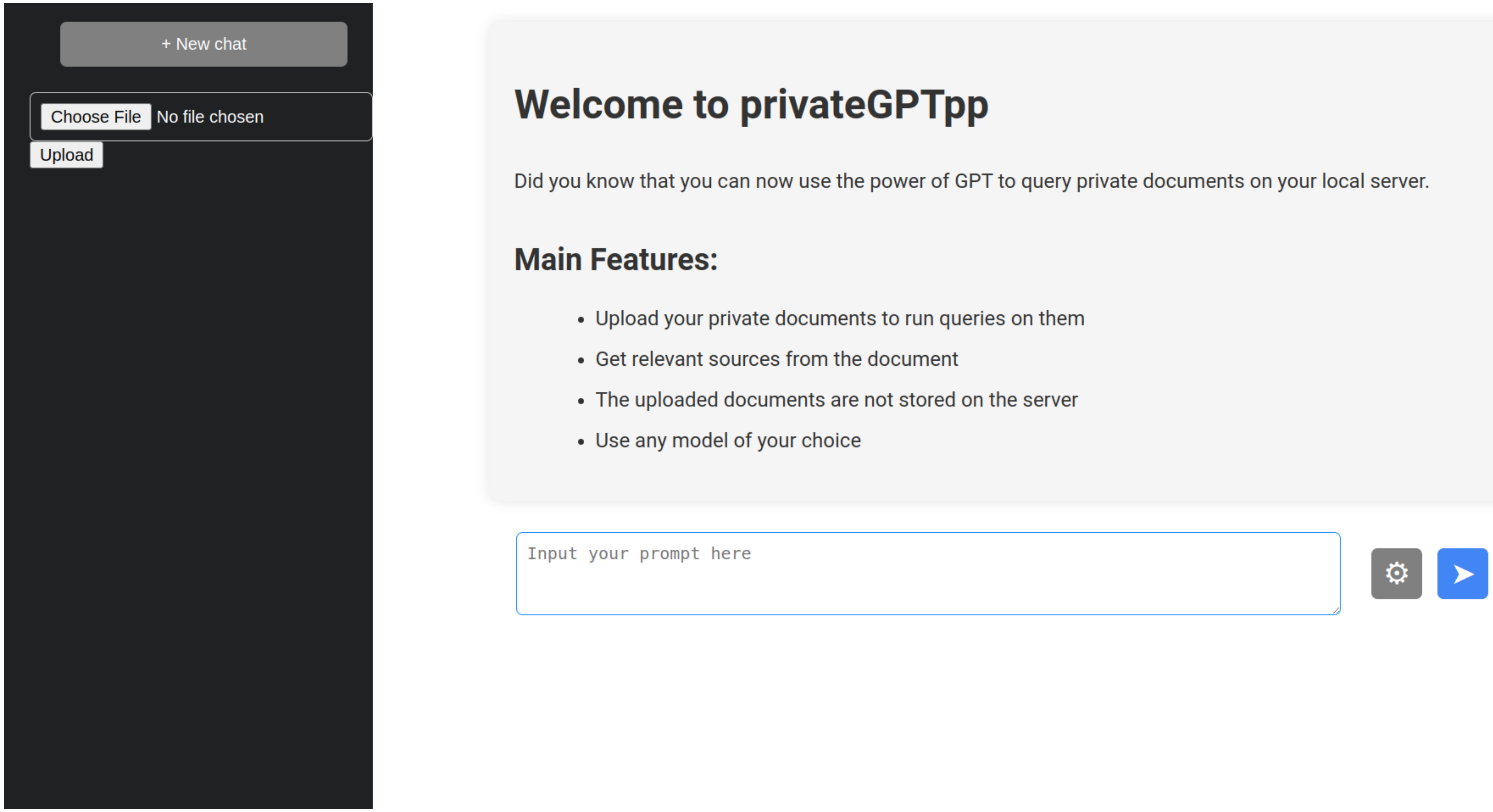
Runs on our own A100 server

Can use almost any model that is open source

Built for privacy, not optimized for speed

Use case scenarios: Understanding medical, legal and other types of sensitive documents.

PrivateGPT++



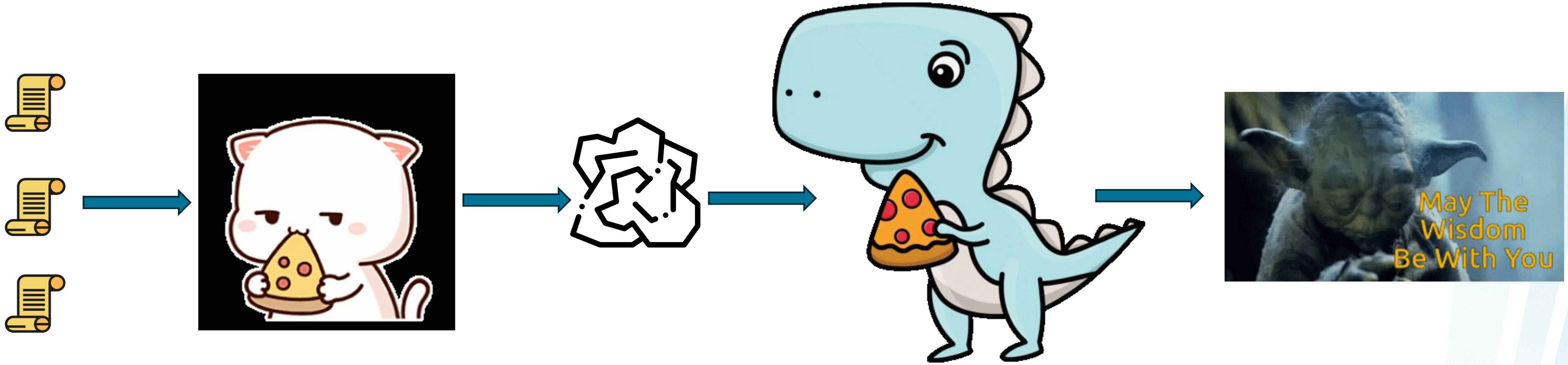
Welcome to privateGPTpp

Did you know that you can now use the power of GPT to query private documents on your local server.

Main Features:

- Upload your private documents to run queries on them
- Get relevant sources from the document
- The uploaded documents are not stored on the server
- Use any model of your choice

PrivateGPT++ - Pipeline



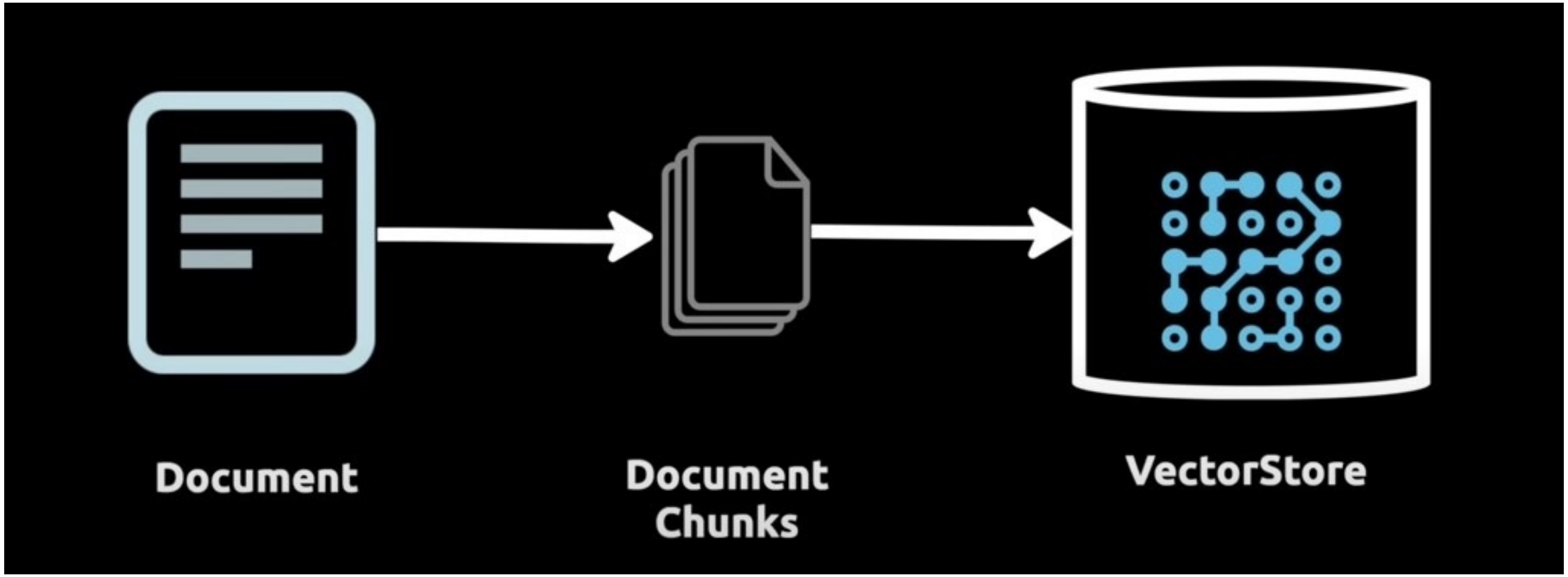
Ingestion

Processed Documents

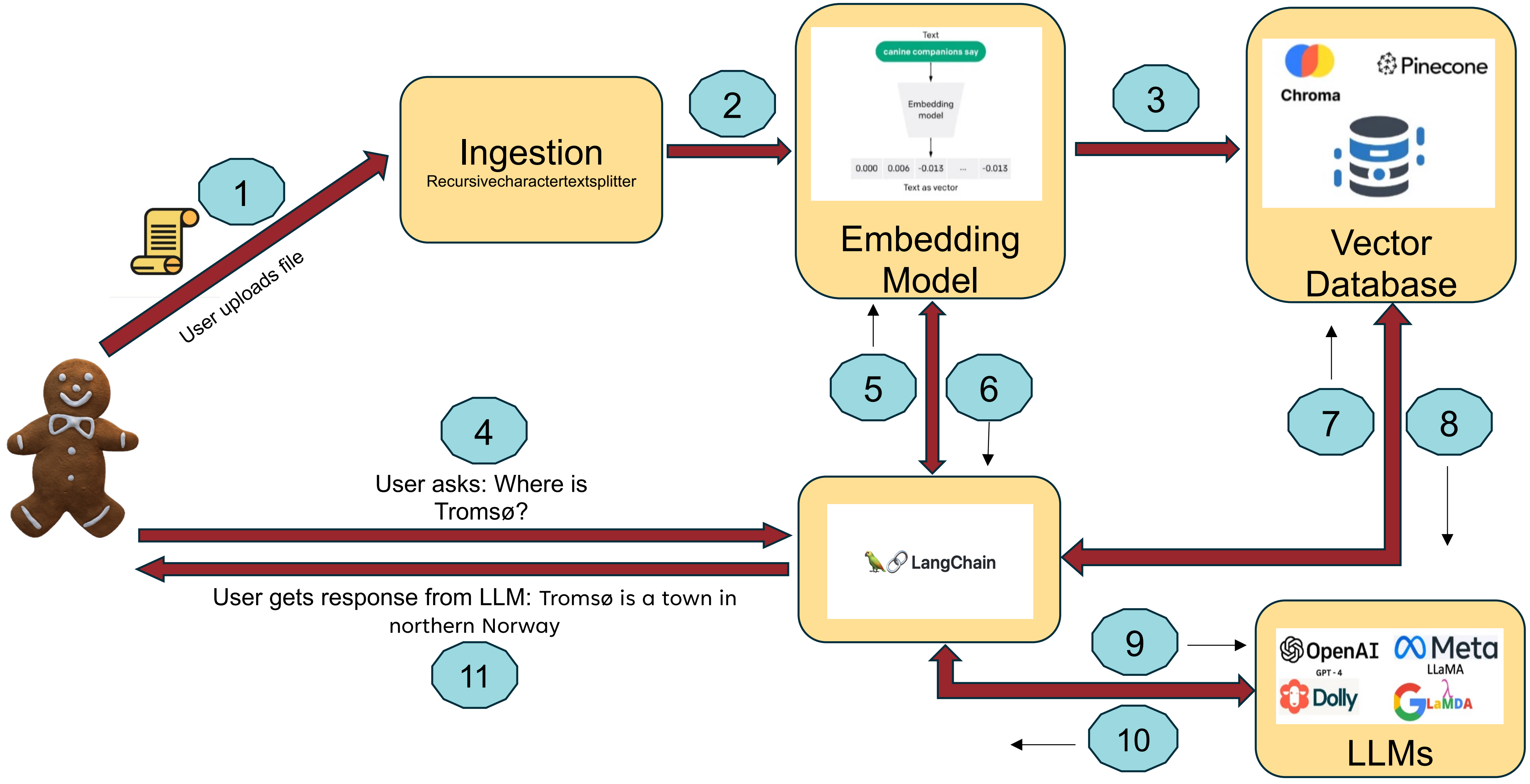
privateGPT++

Words of wisdom

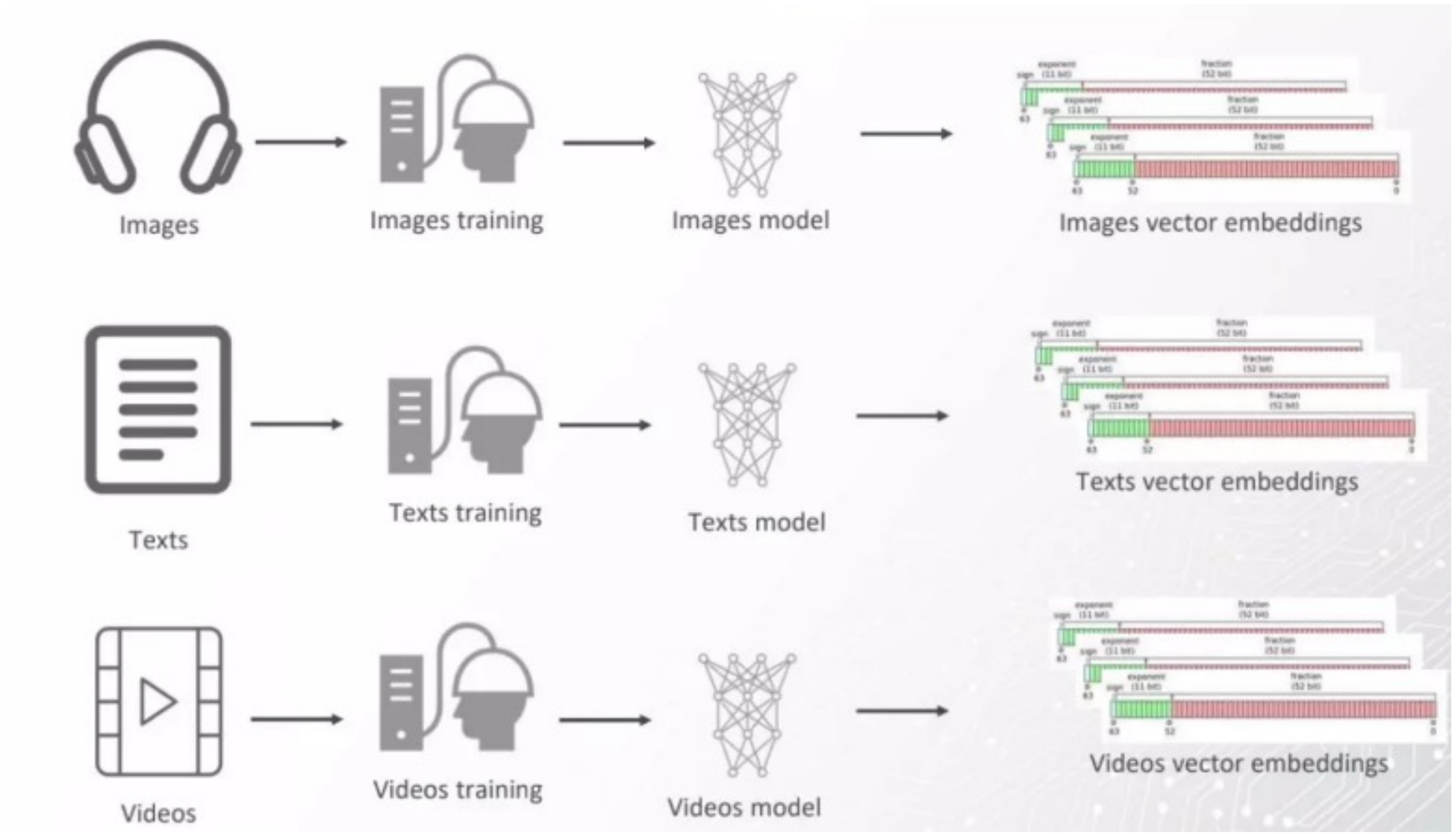
PrivateGPT++ - Processing Documents



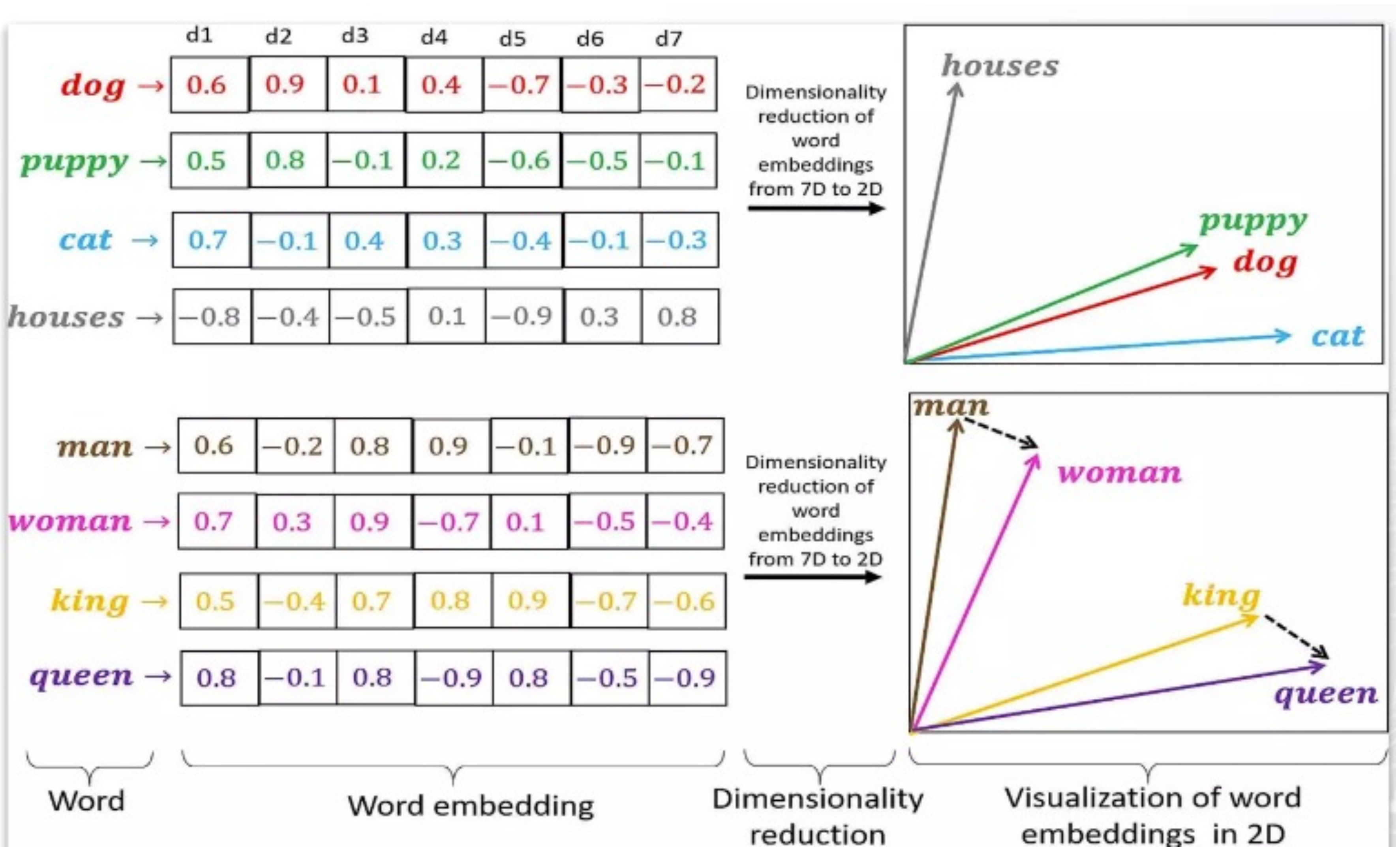
PrivateGPT++ - Flow diagram



PrivateGPT++ - Vector Embeddings



PrivateGPT++ - Vector Similarity

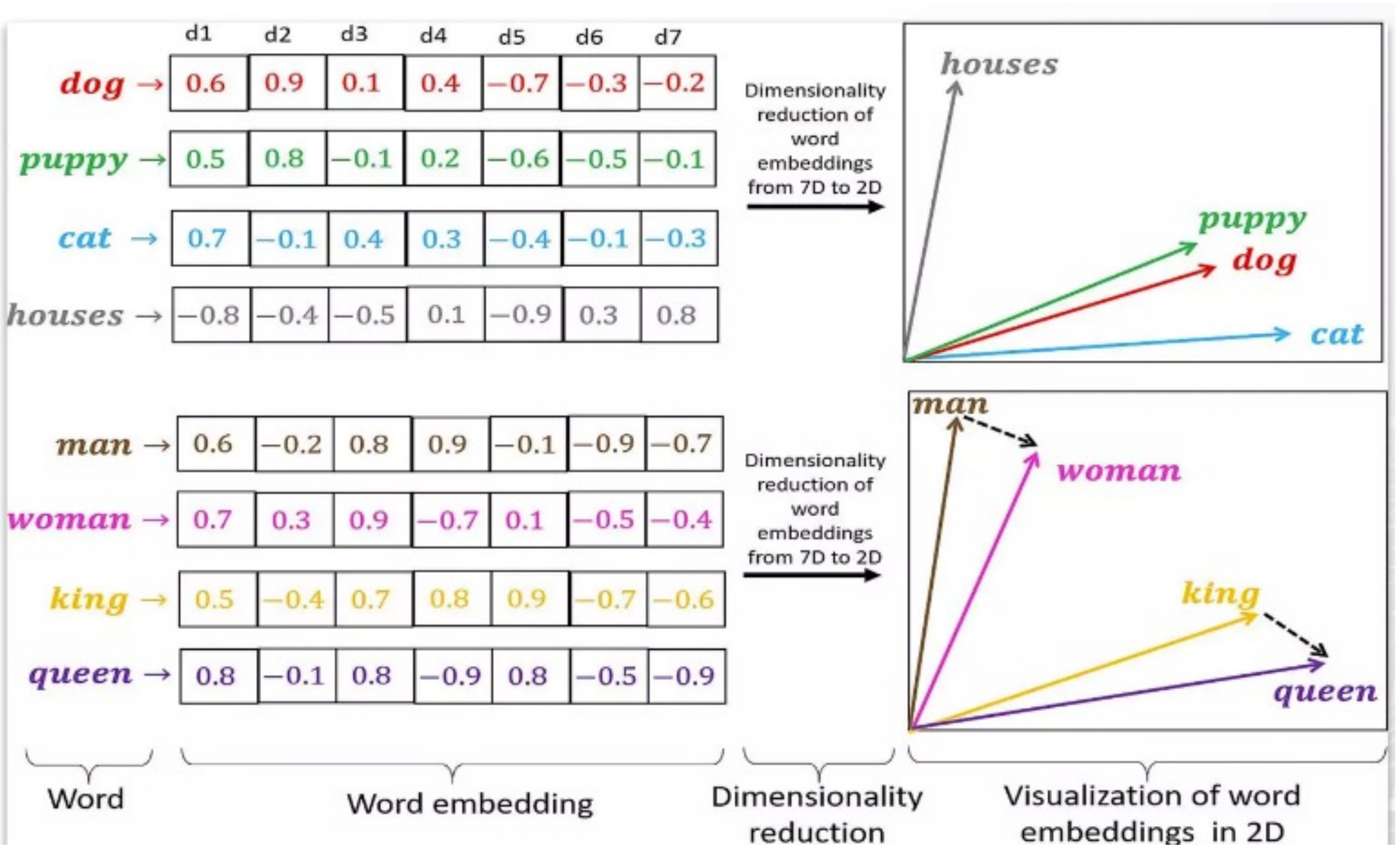


Word	d1	d2	d3	d4	d5	d6	d7
dog →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
puppy →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
cat →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

Word	d1	d2	d3	d4	d5	d6	d7
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
puppy
dog
cat
Dimensionality reduction of word embeddings from 7D to 2D
man
woman
king
queen
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

PrivateGPT++ - Vector Similarity

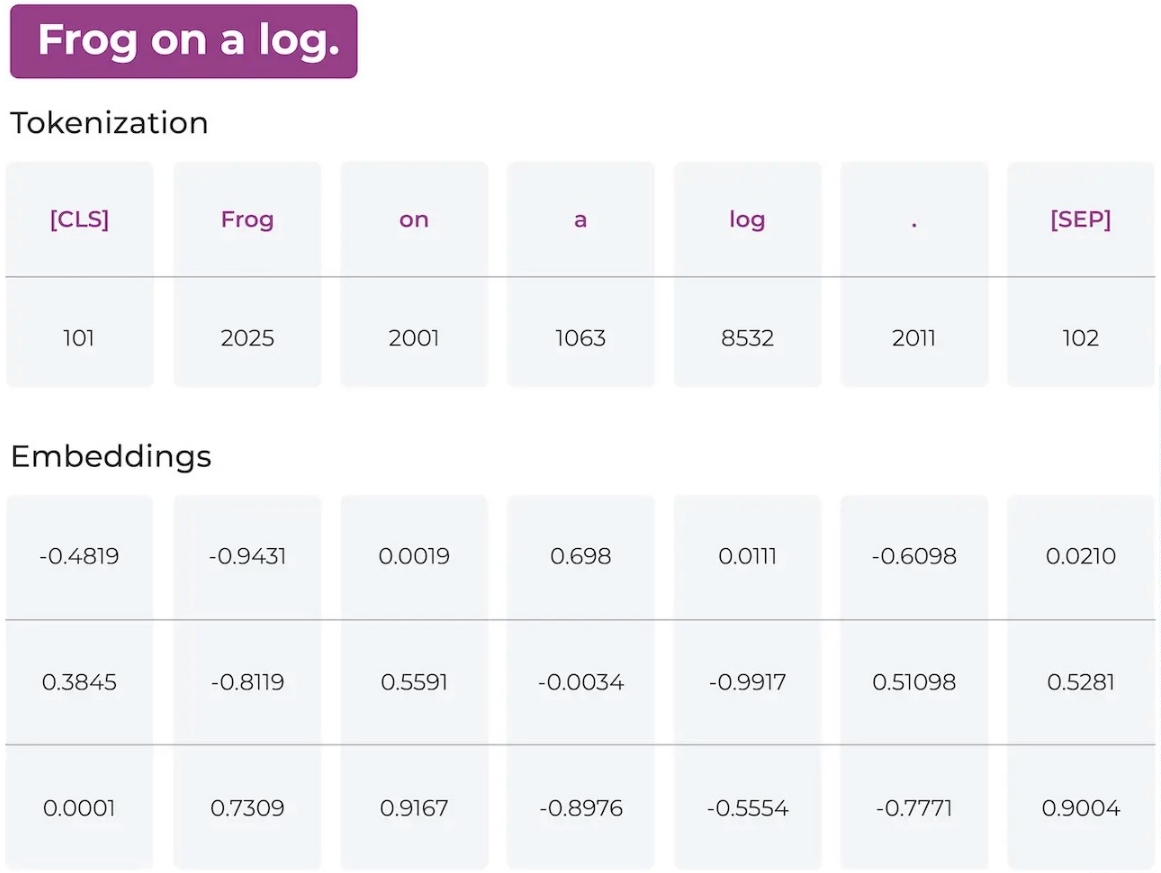


Word	d1	d2	d3	d4	d5	d6	d7
dog →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
puppy →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
cat →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
puppy
dog
cat
Dimensionality reduction of word embeddings from 7D to 2D
man
woman
king
queen
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

PrivateGPT++ - Tokenization and Embeddings

Difference between Tokenization and Embeddings.



Frog on a log.

Tokenization

[CLS]	Frog	on	a	log	.	[SEP]
101	2025	2001	1063	8532	2011	102

Embeddings

-0.4819	-0.9431	0.0019	0.698	0.0111	-0.6098	0.0210
0.3845	-0.8119	0.5591	-0.0034	-0.9917	0.51098	0.5281
0.0001	0.7309	0.9167	-0.8976	-0.5554	-0.7771	0.9004

https://en.uit.no/enhet/ifi

Thank You

Arctic LLM Workshop 2023
Dept. of Computer Science

www.bioailab.org