



Vlaanderen
is supercomputing

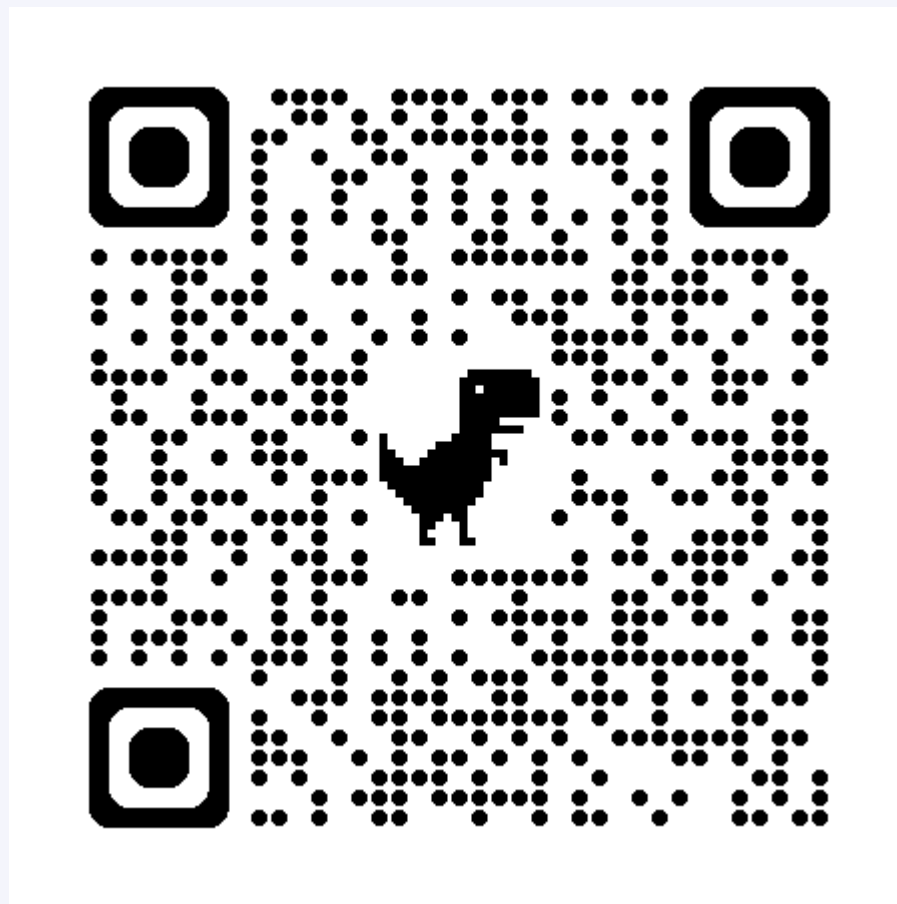


EURO

Running your own LLMs

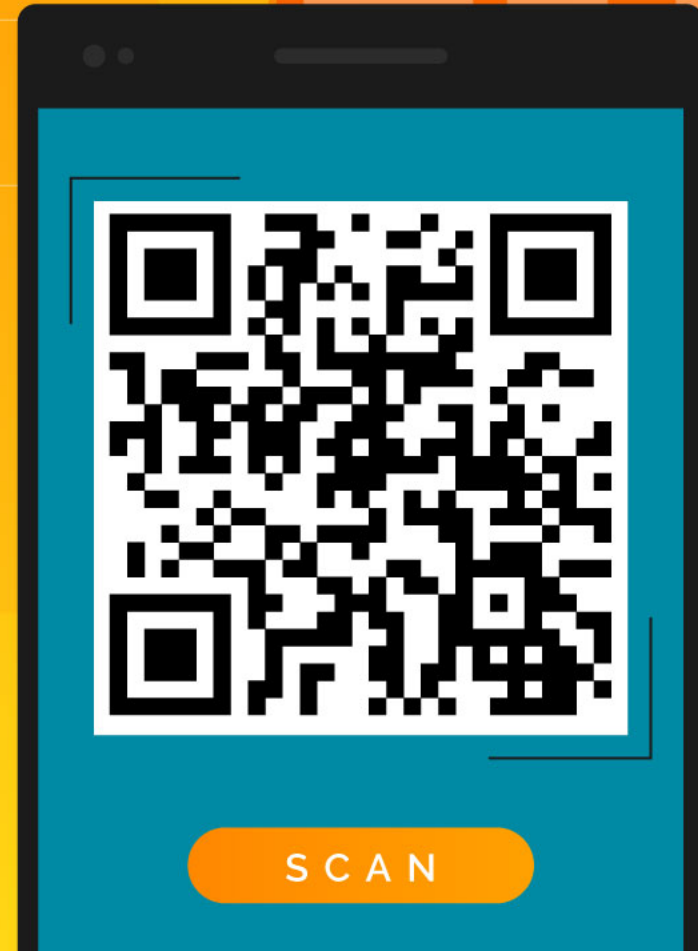
Geert Jan Bex (geertjan.bex@uhasselt.be)

License: this presentation is released under the Creative Commons
CC BY 4.0,
see <https://creativecommons.org/licenses/by/4.0/deed.ast>



<https://github.com/gjbex/AI-to-ols>

*Stay Connected
to VSC*



Overview

- Getting ready
- Why LLMs locally?
- What are LLMs?
- Can I run an LLM on my laptop?
- How can I use LLMs?
- Can I chat with my data?
- Shrink your model: quantization
- Fine-tuning models
- Can I start from scratch?
- Conclusions

Getting ready

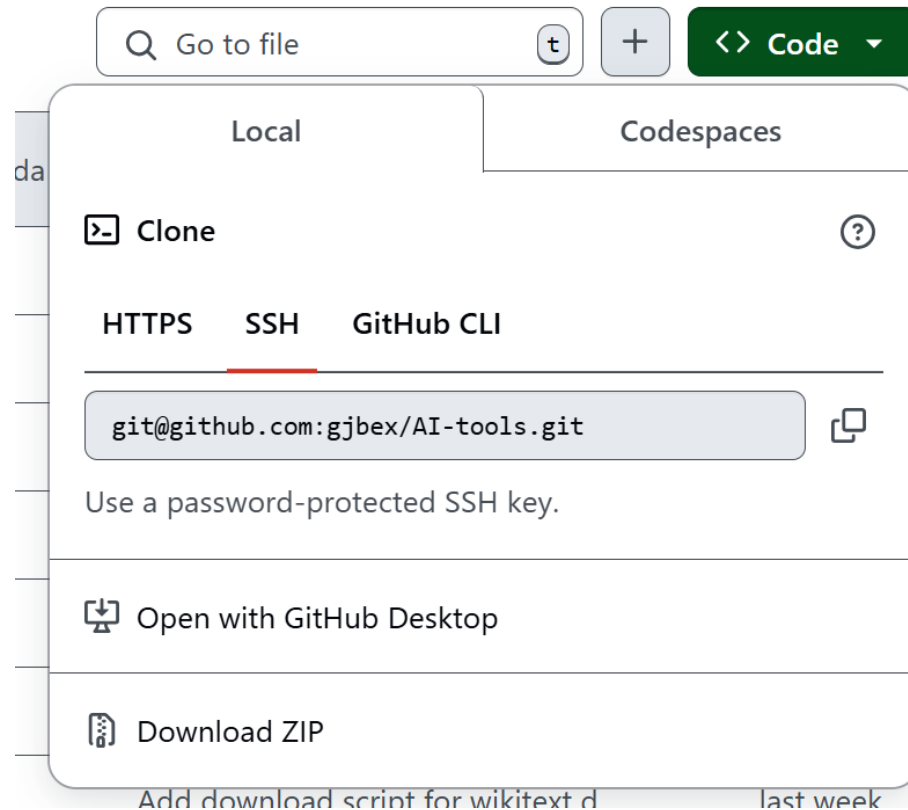
VSC account

- If you don't have an account
 - Apply if you can
<https://account.vscentrum.be/>
 - Ask for temporary account
- Group: lp_gpgpu_training
 - Look for invite (email) and accept

VSC documentation: <https://docs.vscentrum.be/>

GitHub repository

- Get repository (clone or download)



<https://github.com/gjbex/AI-tools>

Hugging Face account

- To get an account
 - Go to <https://huggingface.co/>
 - Click "Sign Up"
 - Enter required information

Hugging Face gated repositories

- To get permission for gated Llama 3.2 models
 - Go to <https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct>

LLAMA 3.2 COMMUNITY LICENSE AGREEMENT

Llama 3.2 Version Release Date: September 25, 2024

“Agreement” means the terms and conditions for use, reproduction, distribution and modification of the Llama Materials set forth herein.

“Documentation” means the specifications, manuals and documentation accompanying Llama 3.2 distributed by Meta at <https://llama.meta.com/doc/overview...>

or to review the conditions and access this model content.

- Go to <https://huggingface.co/mistralai/Mistral-7B-v0.1>

You will get email
confirmation

Why LLMs locally?

Motivation

- Data may be private
 - Intellectual property
 - GDPR restrictions
- Latency may be an issue
- Vanilla LLMs are too generic
 - Retrieval-Augmented Generation (RAG)
 - Fine-tuning

What are LLMs anyway?

Large Language Models are... large

- $> 10^9$ parameters
 - Llama 3.2: 3.2×10^9
 - Gemma 2: 2.7×10^{10}
 - ChatGPT 3.5: 1.75×10^{11}
 - ChatGPT 4o: 10^{12}
- Training sets
 - Llama 3.2: 9×10^{12} tokens
 - Gemma 2: 1.3×10^{13} tokens
 - ChatGPT 3.5: 4×10^{11} tokens (570 GB)

Large Language Models are... models

- Statistical models of text
- Generate sequence of most likely tokens in context

The 42nd president of the United States
was...

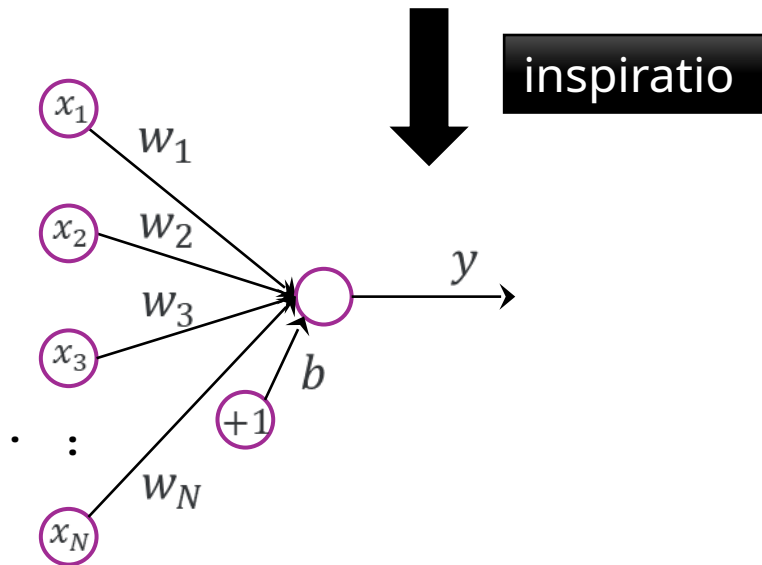
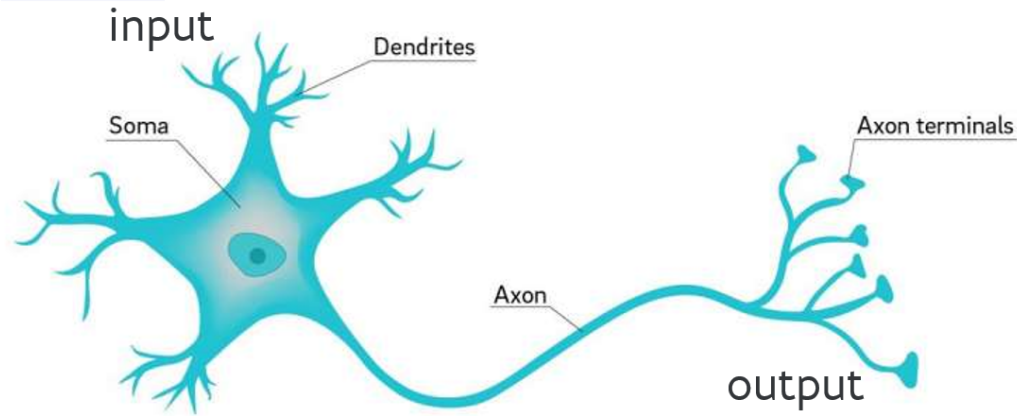


Bill Clinton: 90

%
George Bush: 3
%
Richard Nixon: 2
%
Jacques Chirac: 2
%
...

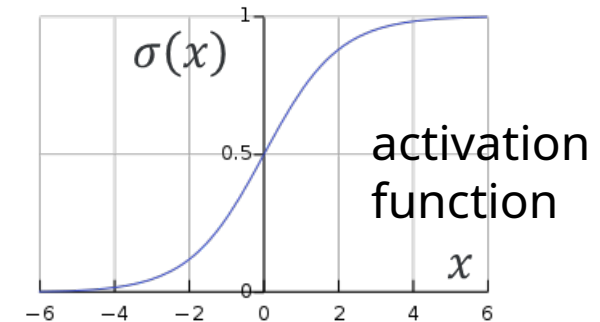
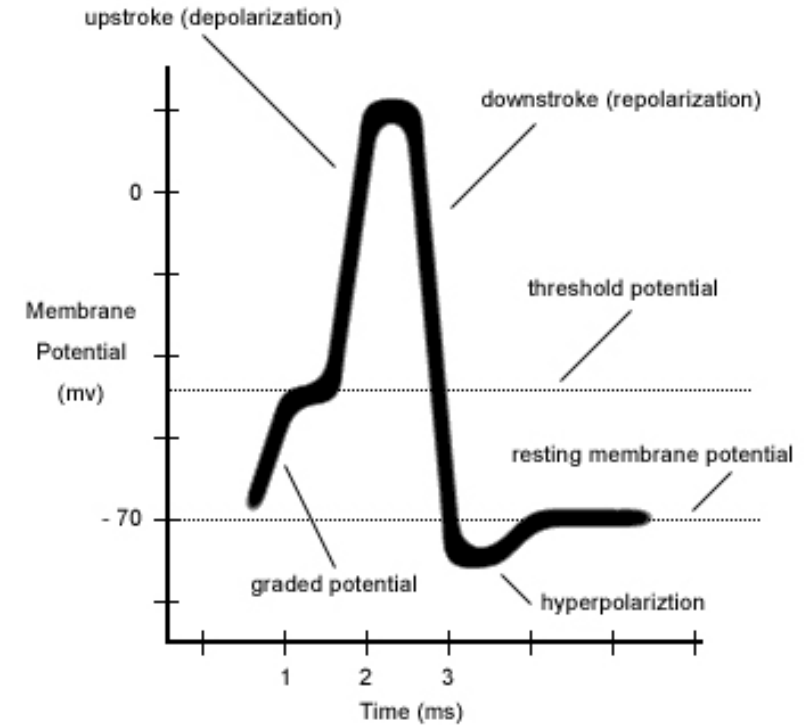
- Architecture: deep neural networks

From neurons to ANNs

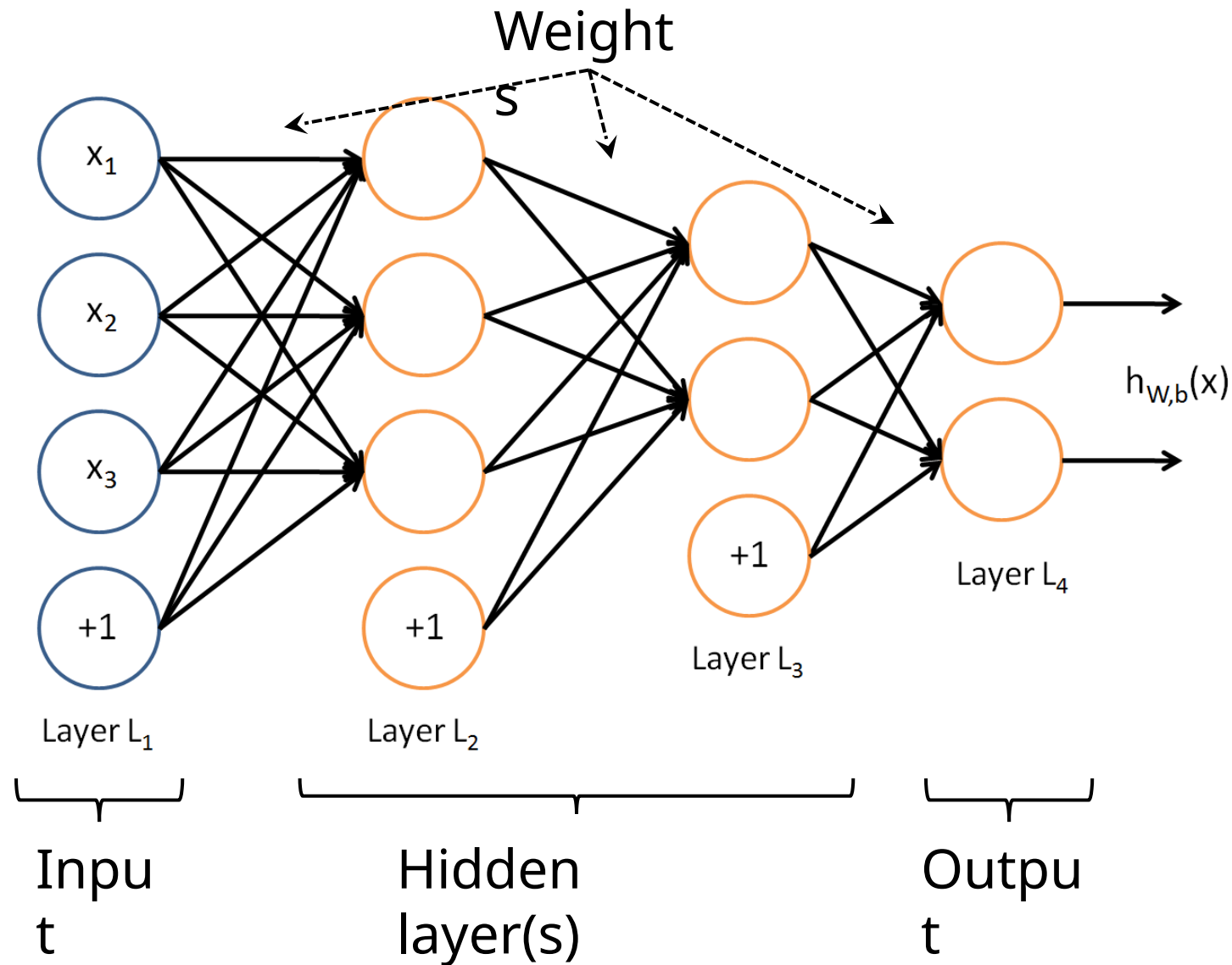


$$y = \sigma \left(\sum_{i=1}^N w_i x_i + b \right)$$

Action Potential



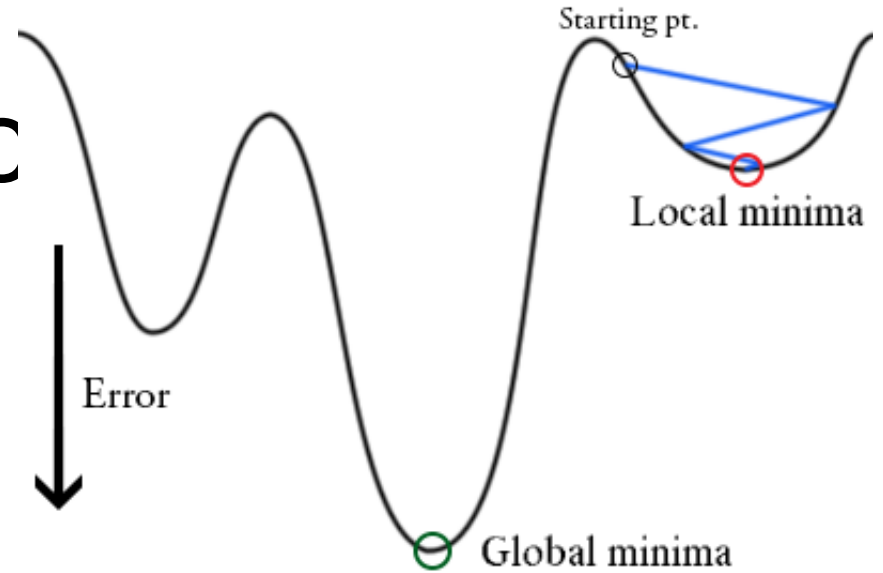
Multilayer network



How to
determine
weights?

Training: backpropagatic

- Initialize weights "randomly"
- For all training epochs
 - for all input-output in training set
 - using input, compute output (forward)
 - compare computed output with training output (loss function)
 - adapt weights (backward) to improve output
 - if accuracy is good enough, stop



Computationally
expensive!

Inference: using the ANN

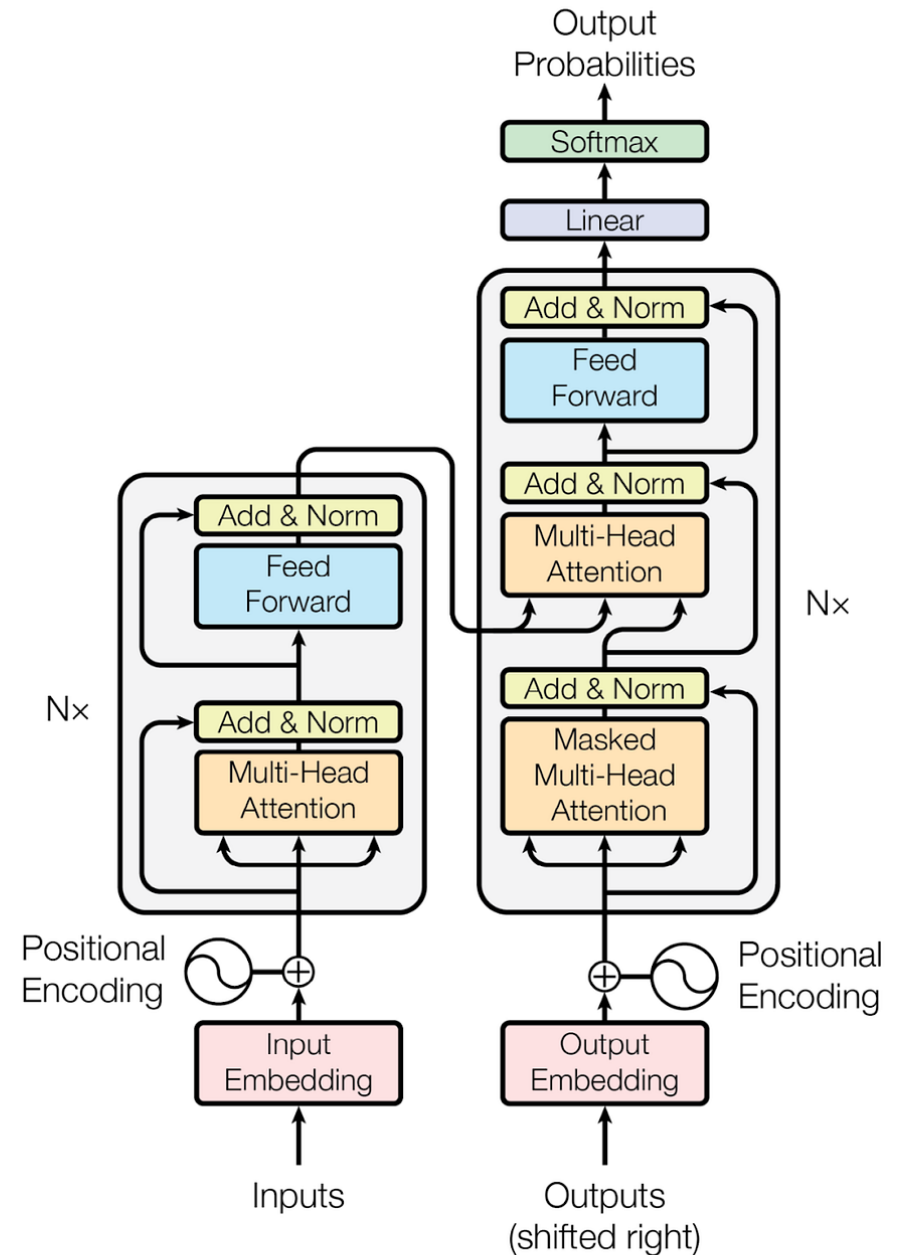
- Use new input
- Compute output for each layer (forward)

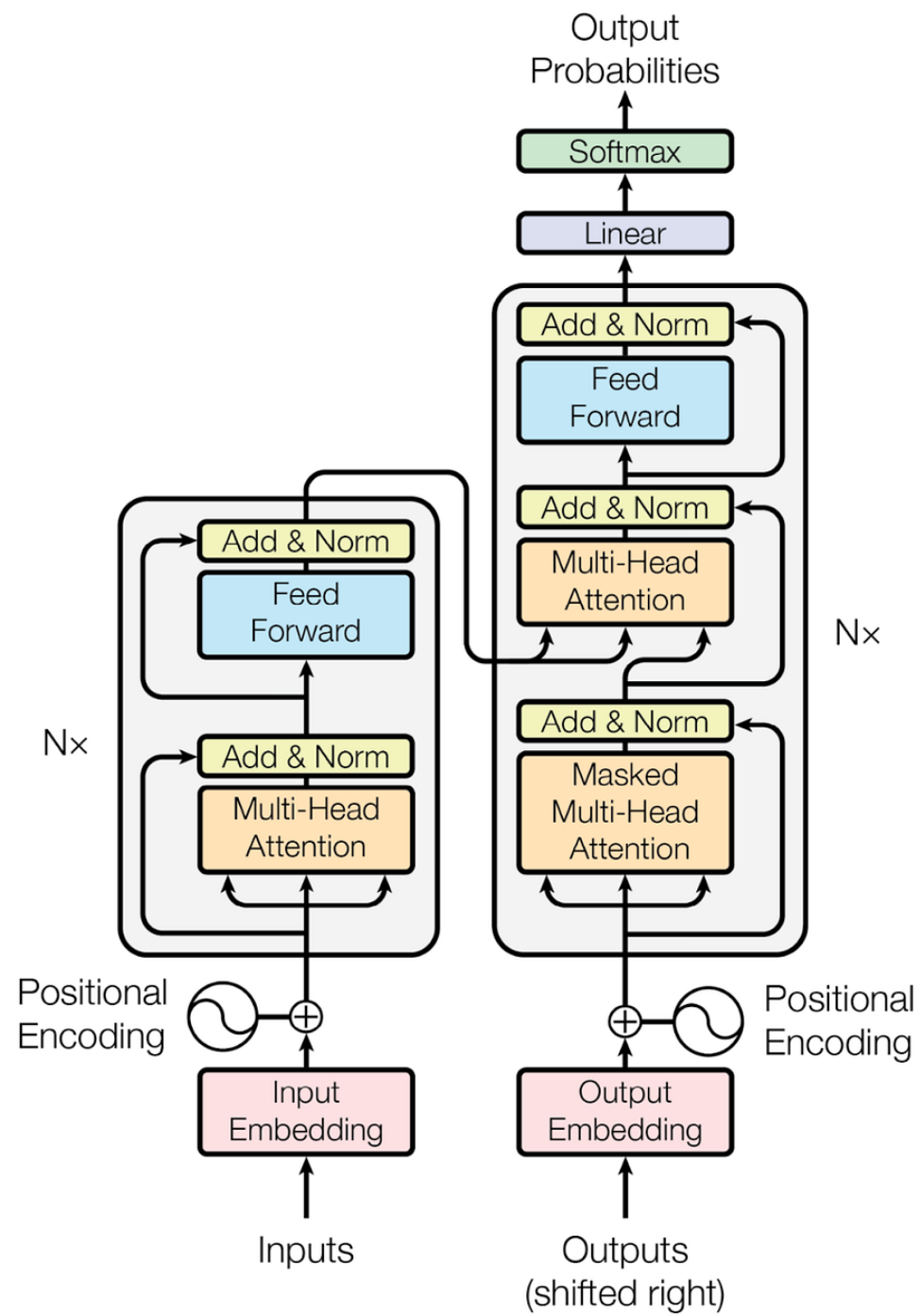
Weights are fixed, i.e., ANN doesn't change`

Computationally
cheap!

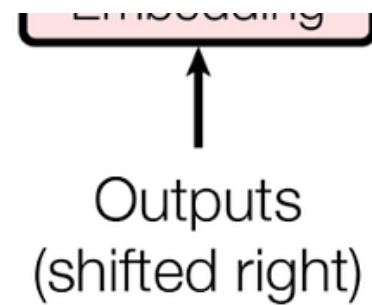
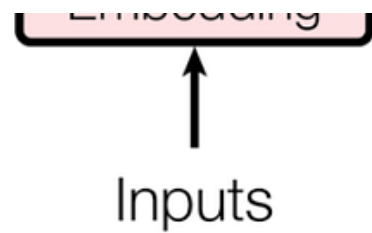
Transformers

- *Attention is all you need*, Vaswani et al., NIPS 2017
 - Embedding
 - Positional encoding
 - Multi-head attention
 - Layer normalization
 - Softmax
- Predict next output token based on
 - Input token sequence
 - Previous output token sequence





Inputs



Inputs

- T_{input} input tokens
- T_{output} previous output tokens

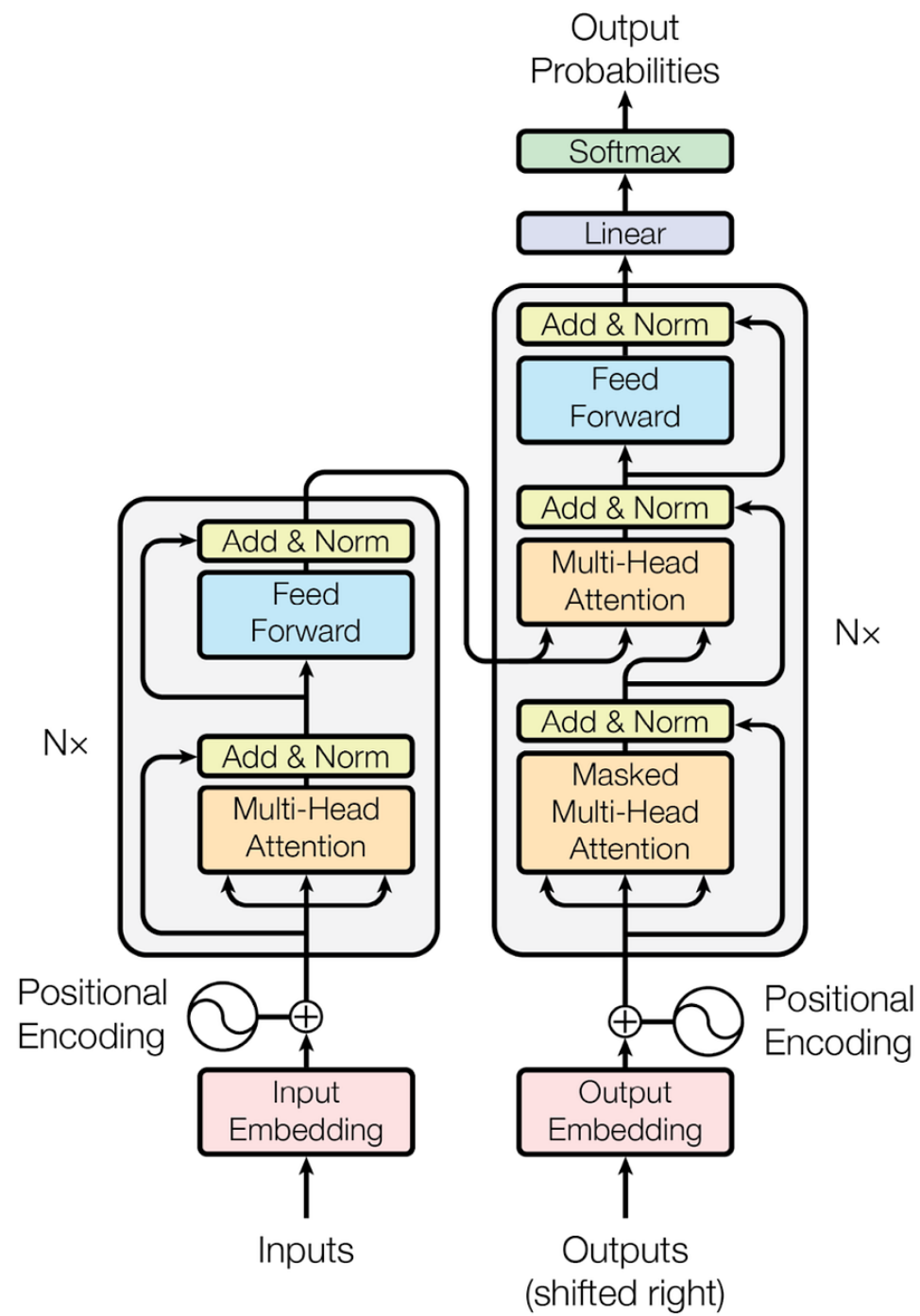
index	token
0	<unused>
1	<start>
2	and
3	the
...	...
42	book
...	...

Fixed
!
Input
vocabulary

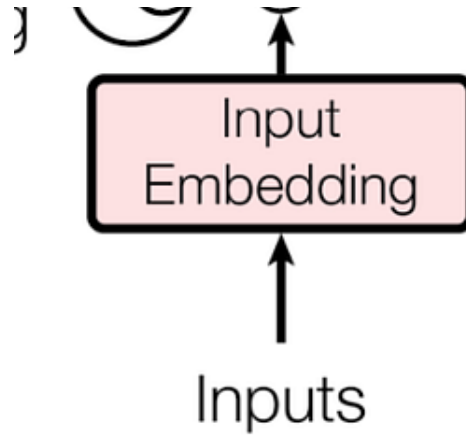
index	token
0	<unused>
1	<start>
2	et
3	la
...	...
42	chaise
...	...

Fixed
!
Output
vocabulary

Input and output vocabulary can be different
(sizes)



Embeddings



Embeddings

- Represent words as one-hot vectors
length = vocabulary size
 - Issues
 - Unwieldy
 - No semantics
- Word embeddings
 - Dense vector d_{model}
 - Vector distance $\approx$ semantic distance
- Training
 - Use context
 - Discover relations with surrounding words



Result: $\mathbb{R}^{T \times d_{model}}$

Positional encoding: motivation

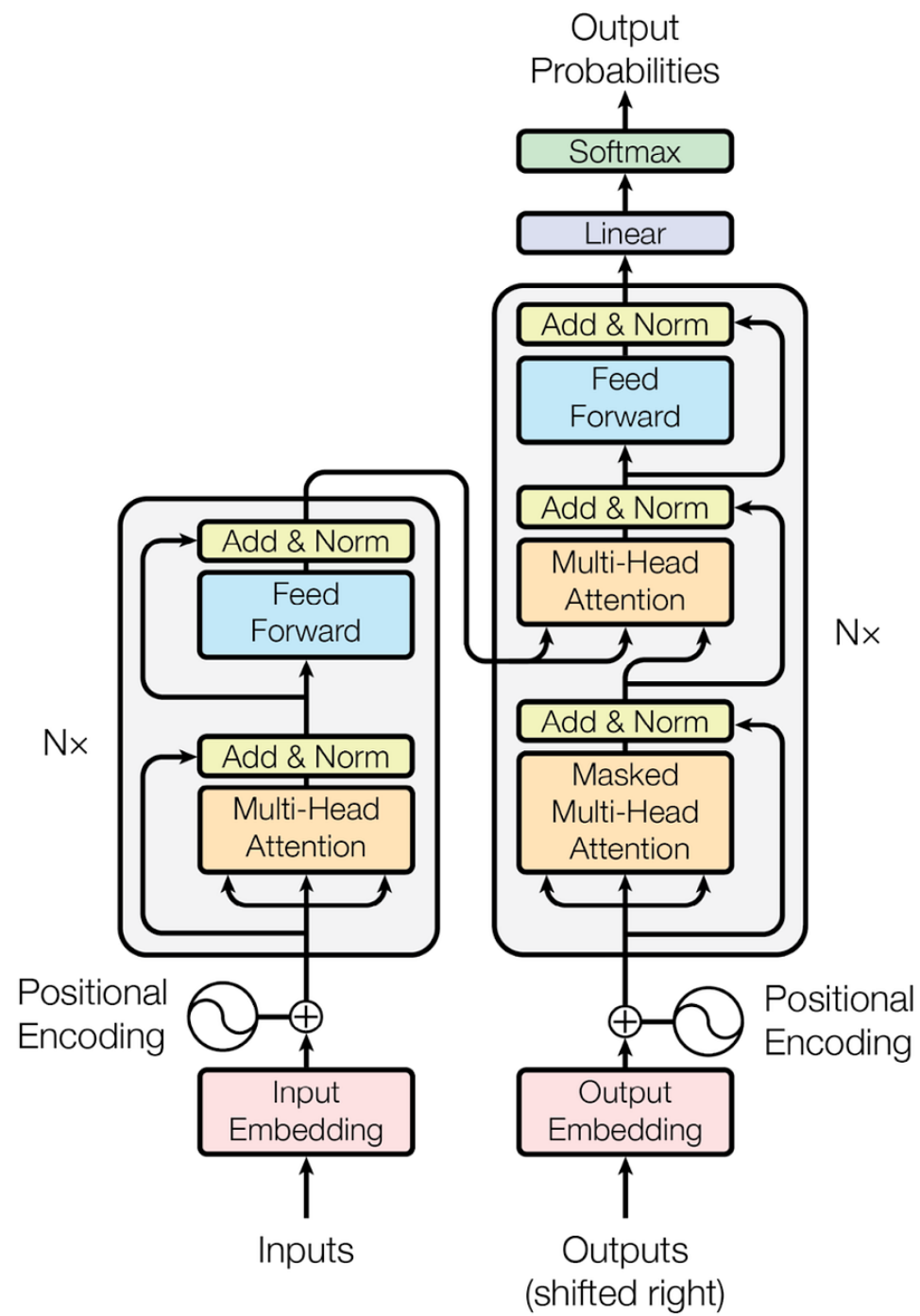
- Provides context, e.g.,

George Floyd was murdered by Officer Derek Chauvin in Minneapolis

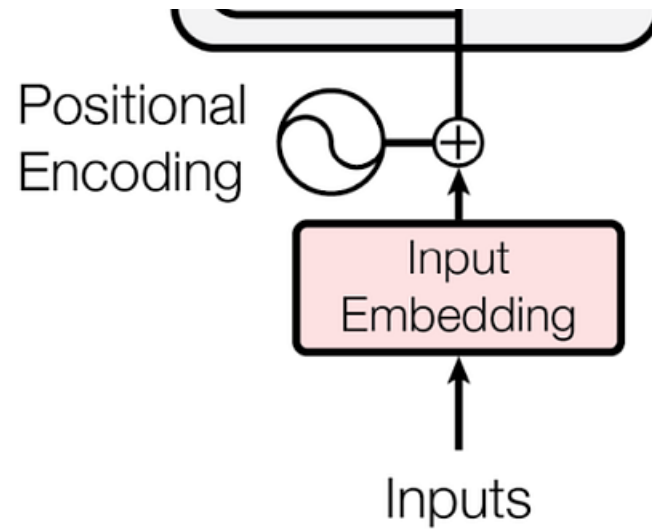
versus

Officer Derek Chauvin was murdered by George Floyd in Minneapolis

- Position of token in sequence is important
 - Absolute position
 - Relative position



Absolute positional encoding

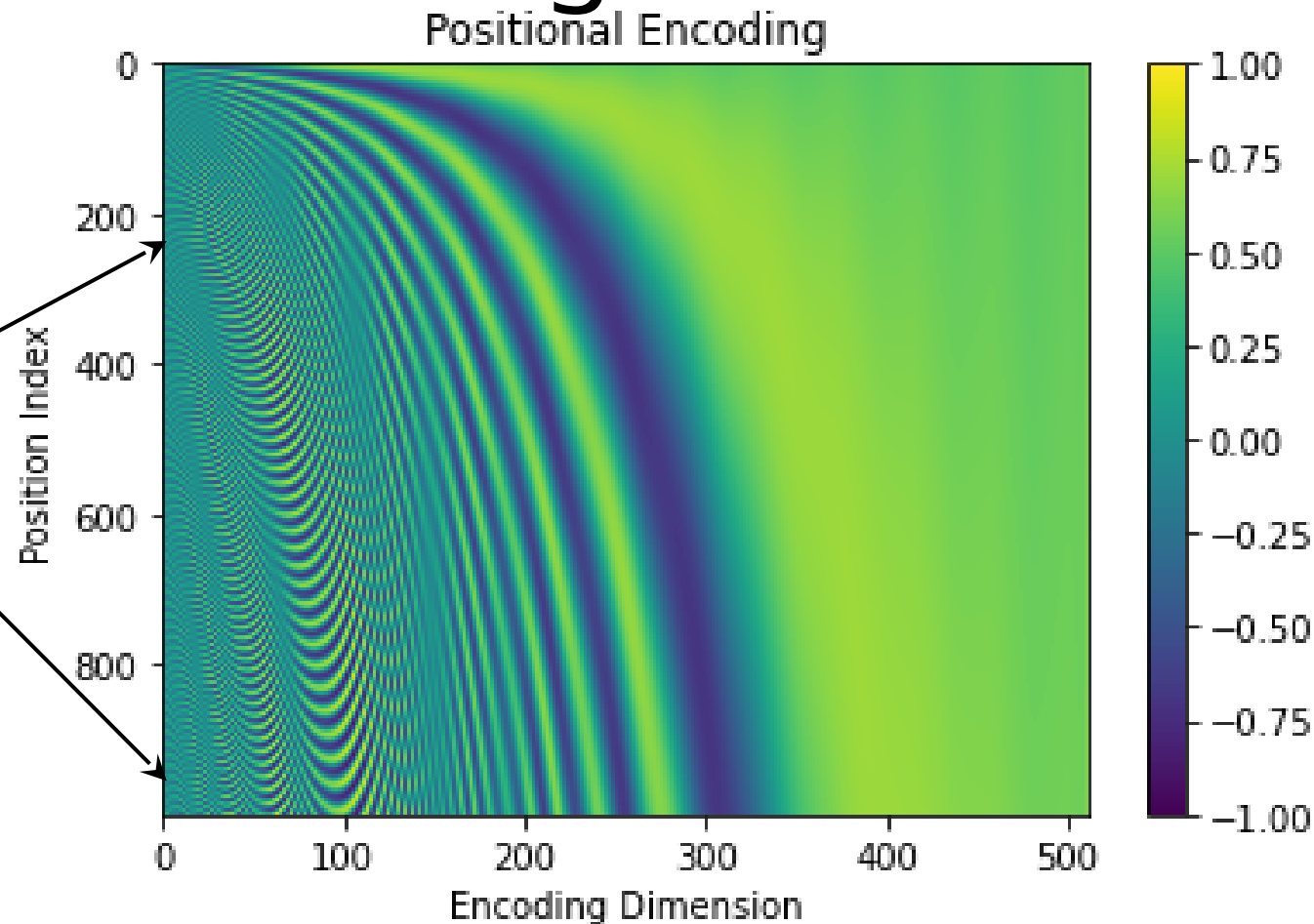


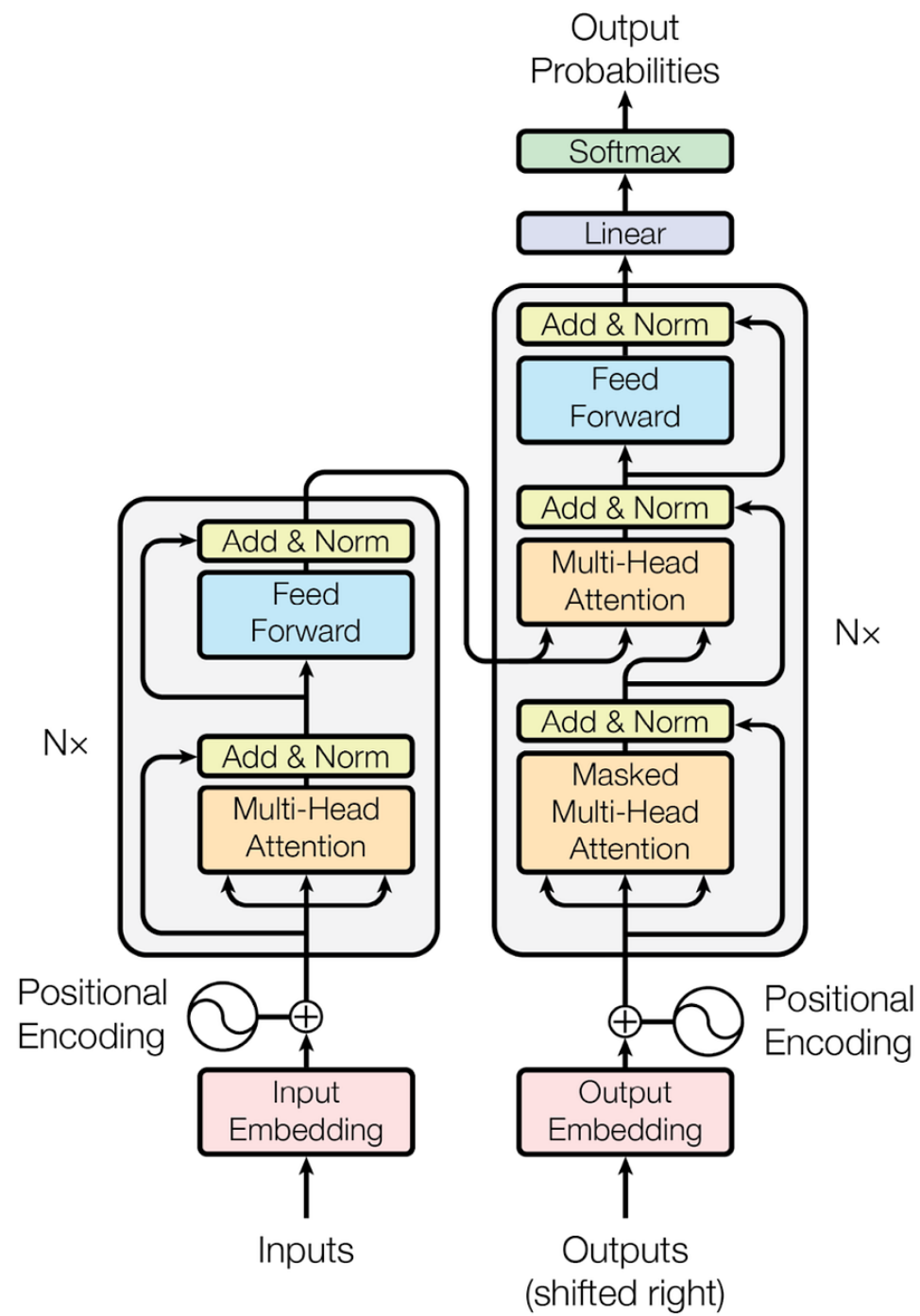
Absolute positional encoding

Unique vector for
each position in
text

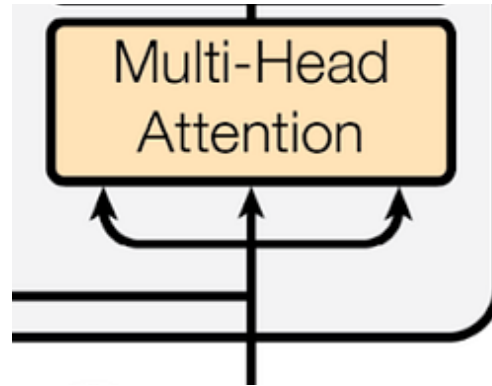
Add to embedding
vector

of word at that
position
Result: $\mathbb{R}^{1 \times d_{model}}$

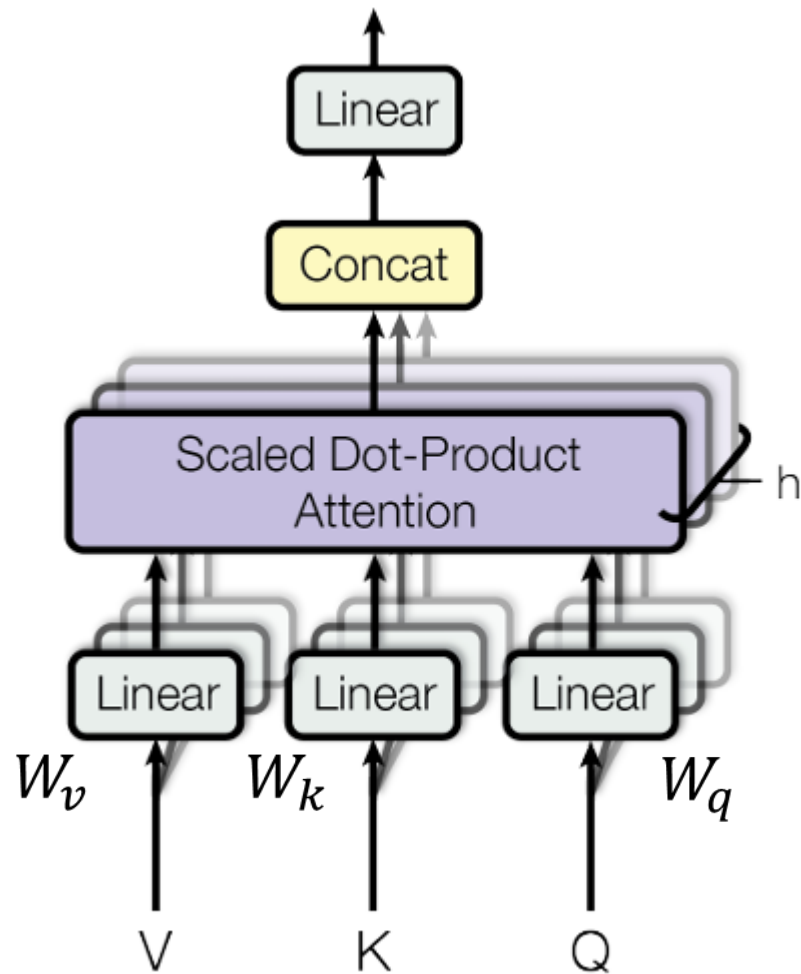




Multi-head self-attention



Multi-head self-attention



- V: value
- K: key
- Q: query

Heads can look at all tokens!

- h : number of heads, $d_k = \frac{d_{model}}{h}$

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^t}{\sqrt{d_k}}\right) \cdot V$$

Result: $\mathbb{R}^{T \times d_{model}}$

Self-attention

- Token **A** and **B** in same sequence

- Attention for **B** from **A**

- Query for **B**: $Q = W_q \cdot e_B + b_q$

- Key for **A**: $K = W_k \cdot e_A + b_k$

- Find most relevant relation between query and key: $\text{softmax}\left(\frac{Q \cdot K^t}{\sqrt{d_k}}\right)$

- Value for **A**: $V = W_v \cdot e_A + b_v$

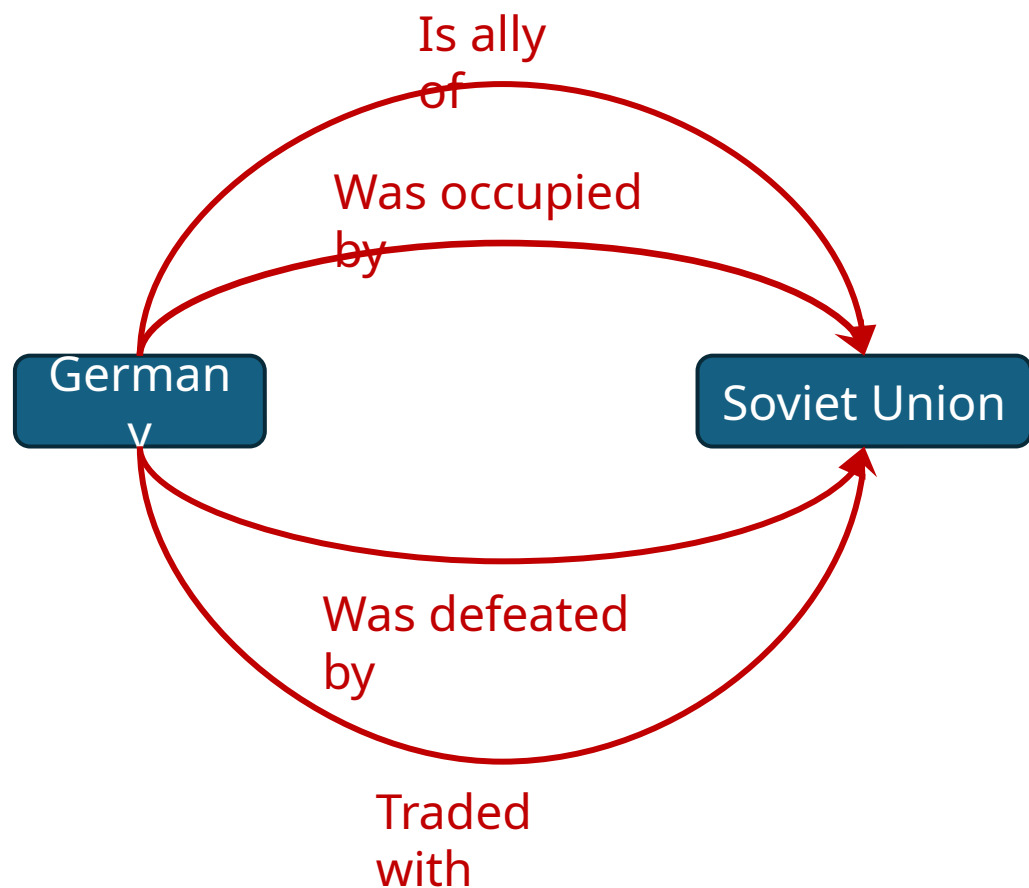
- Find most relevant values for query: $\text{softmax}\left(\frac{Q \cdot K^t}{\sqrt{d_k}}\right) \cdot V$

$$\text{softmax}(\vec{v}, i) = \frac{e^{v_i}}{\sum_{j=1}^N e^{v_j}}$$

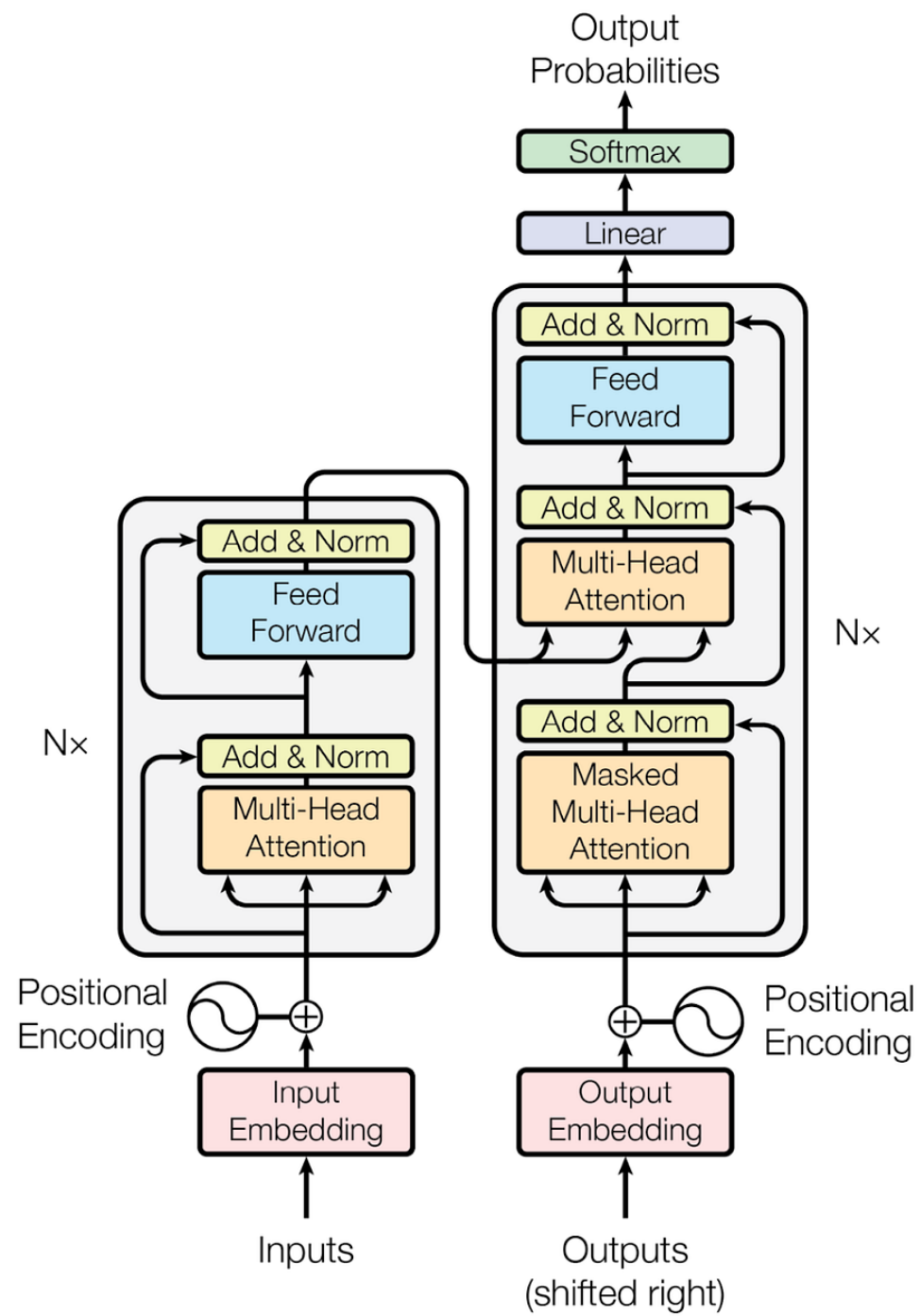
Dot-product
attention

Multi-head

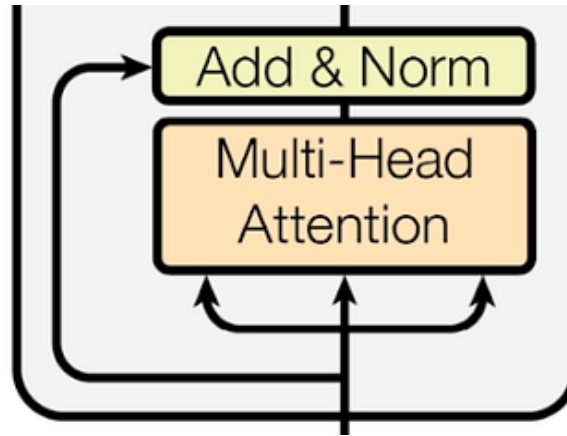
- Token **A** and **B** can have multiple semantic relationships



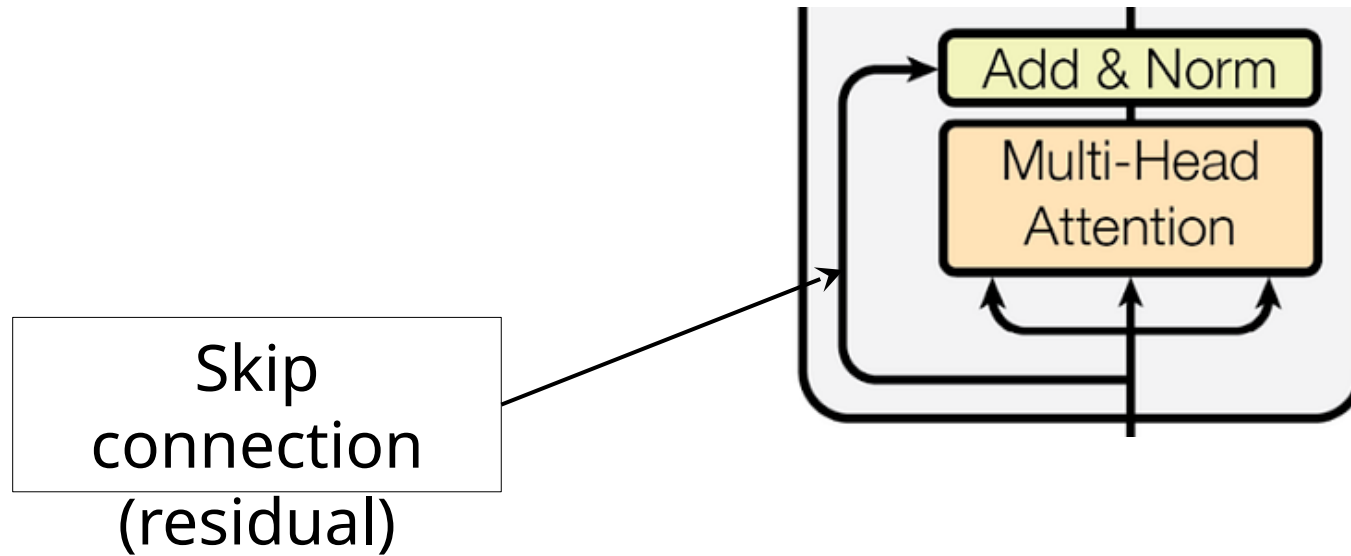
Split embeddings into h parts,
use h heads



Add & layer normalization

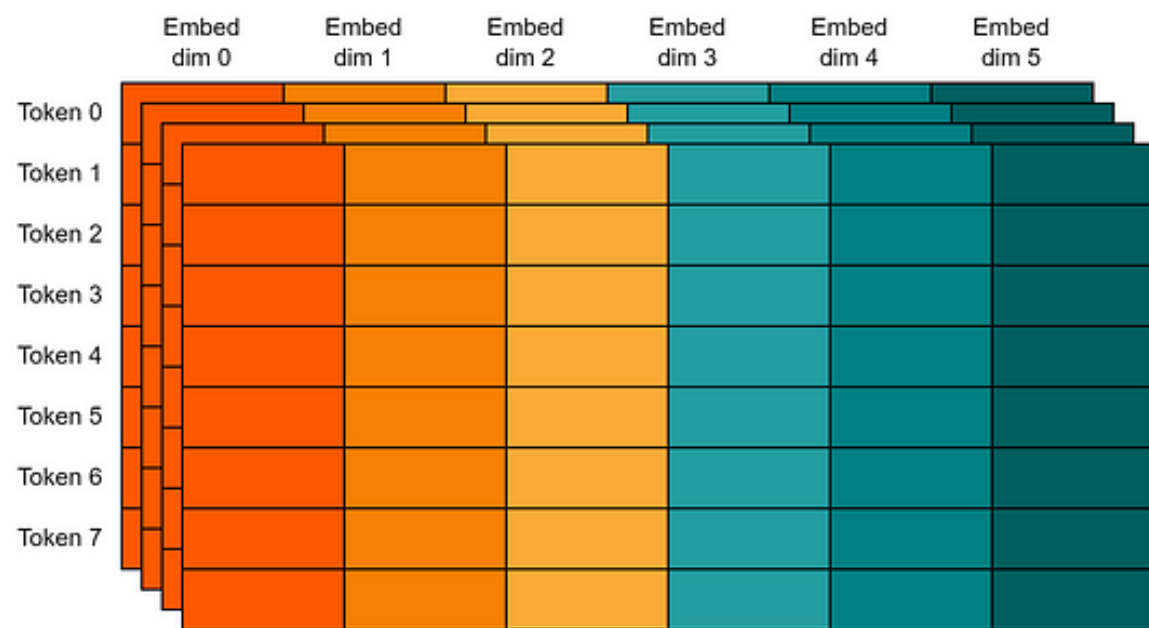


Add skip connections

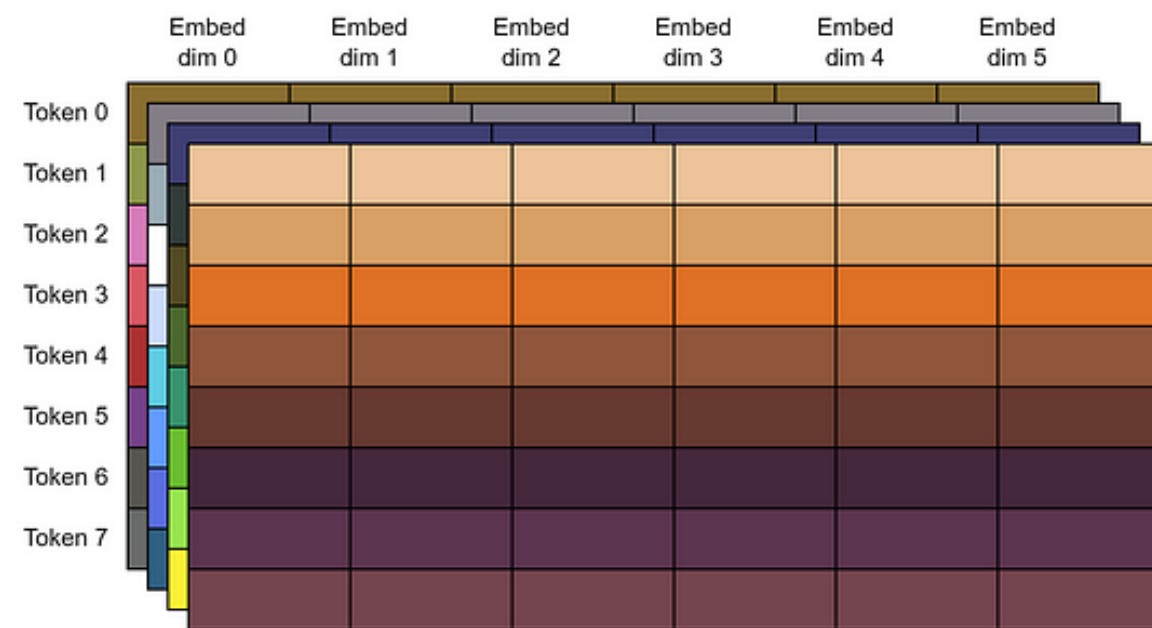


Layer normalization

Batch Normalization

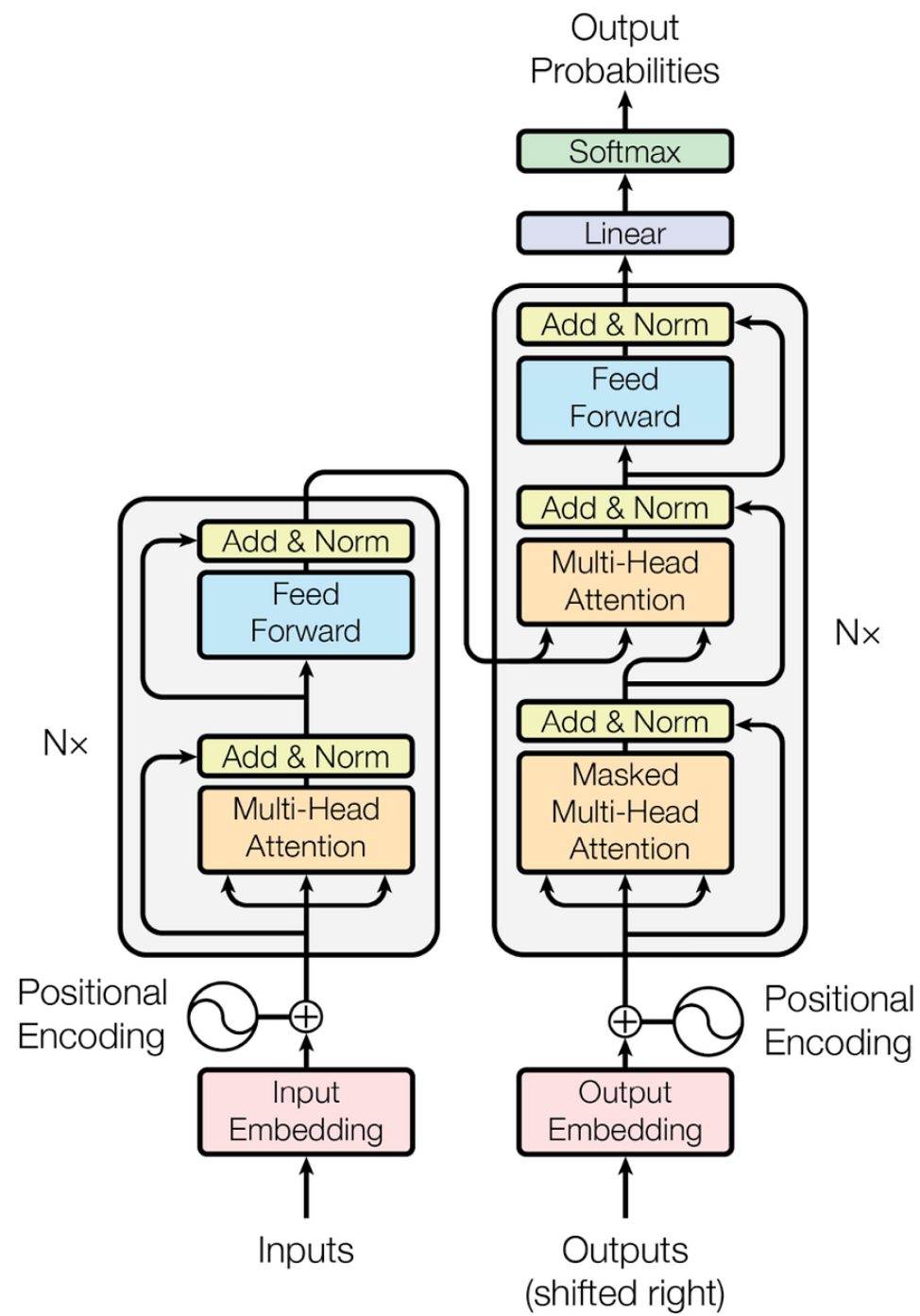


Layer Normalization

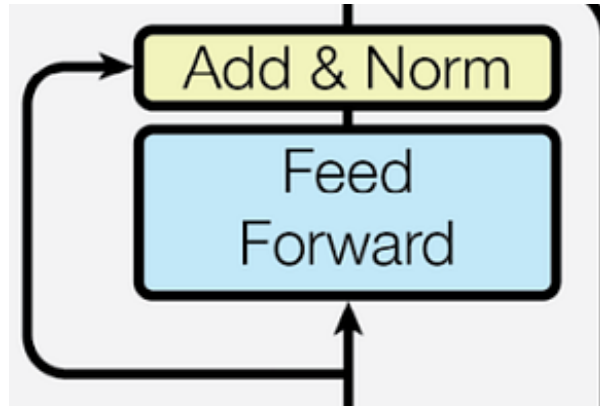


Input/output sequences have varying lengths, d_{model} is constant

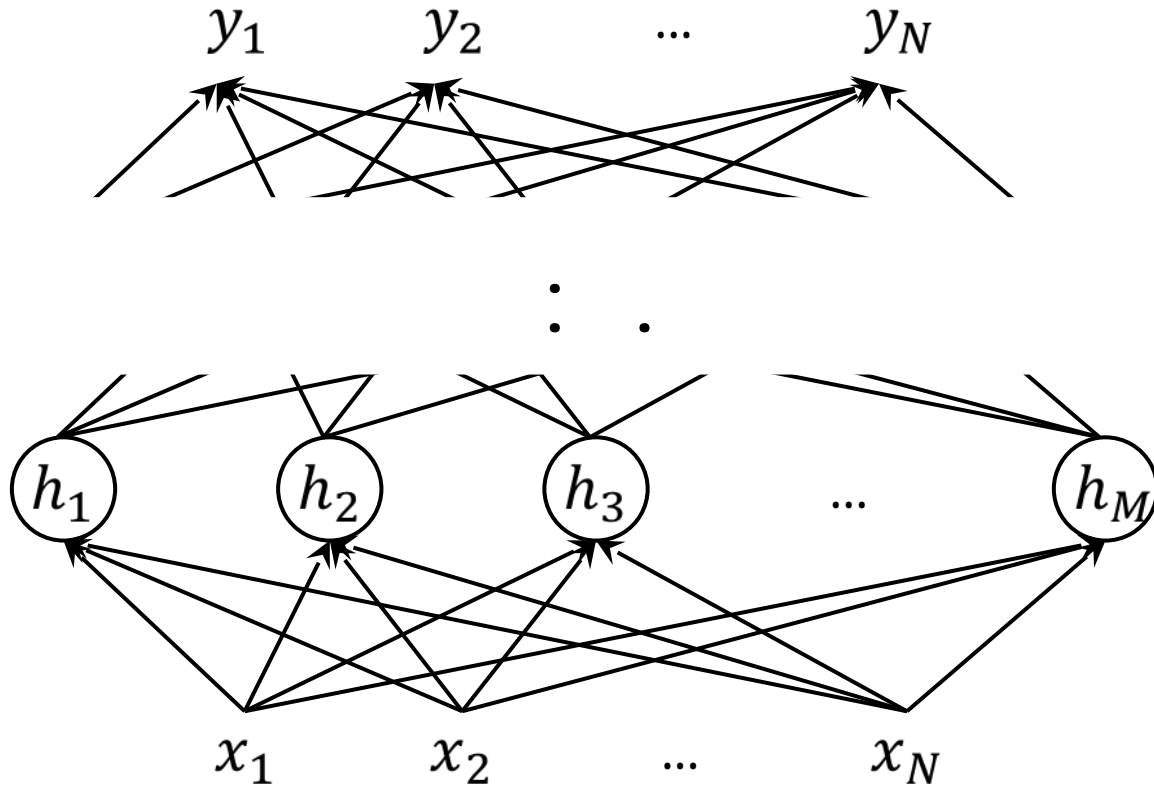
Result: $\mathbb{R}^{T \times d_{model}}$



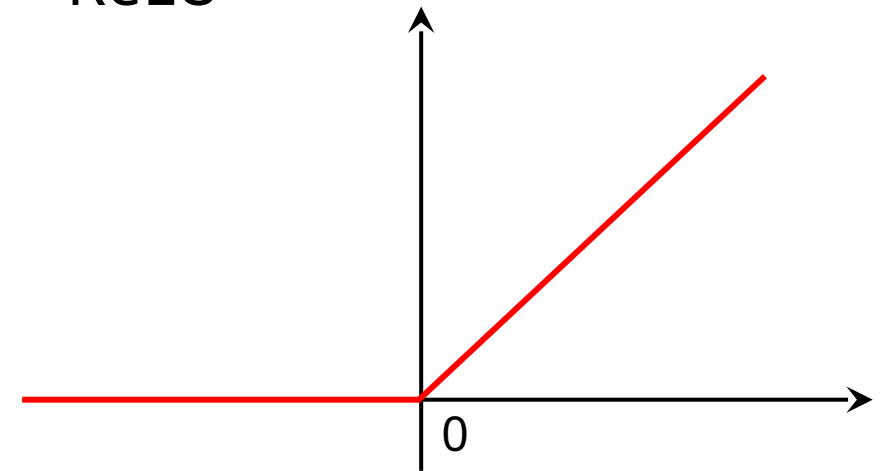
Feed Forward

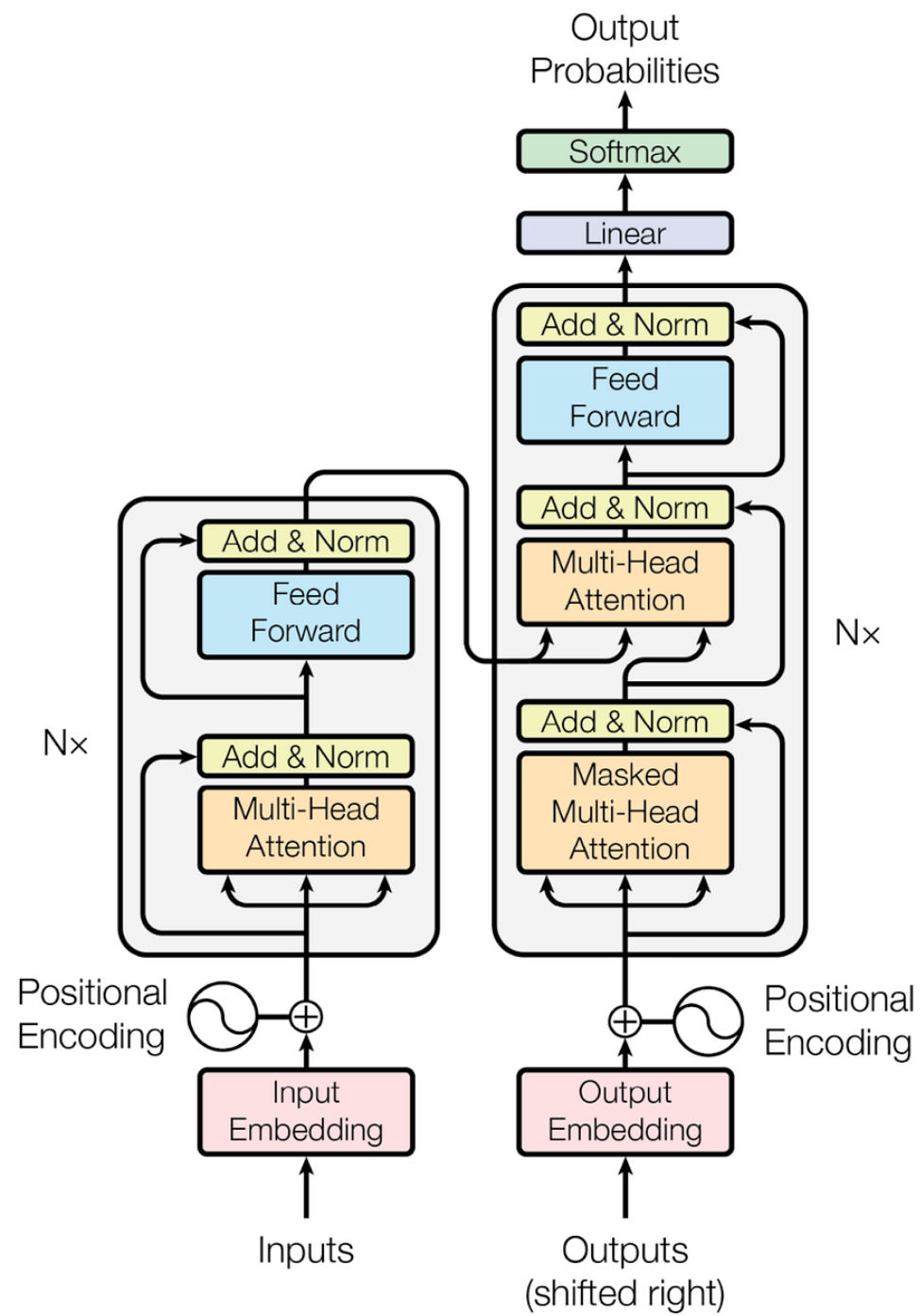


Feed forward

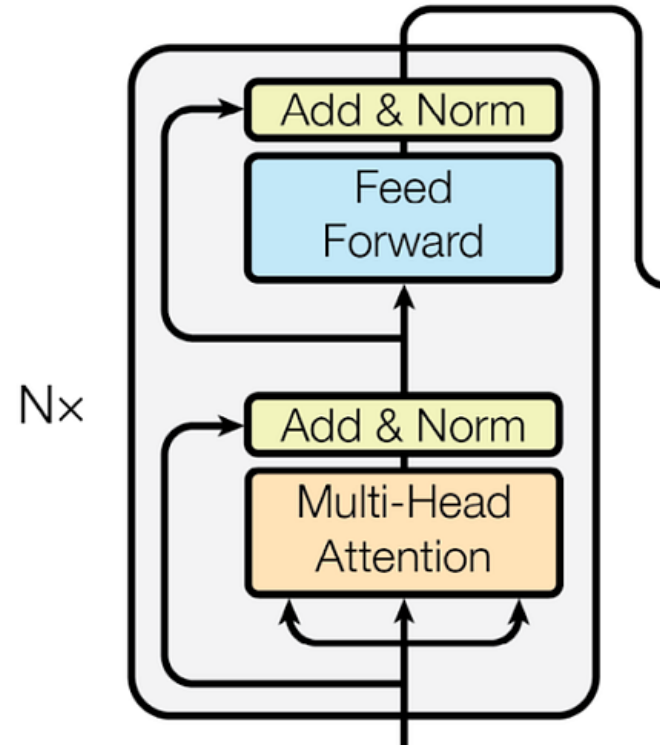


Activation function:
ReLU



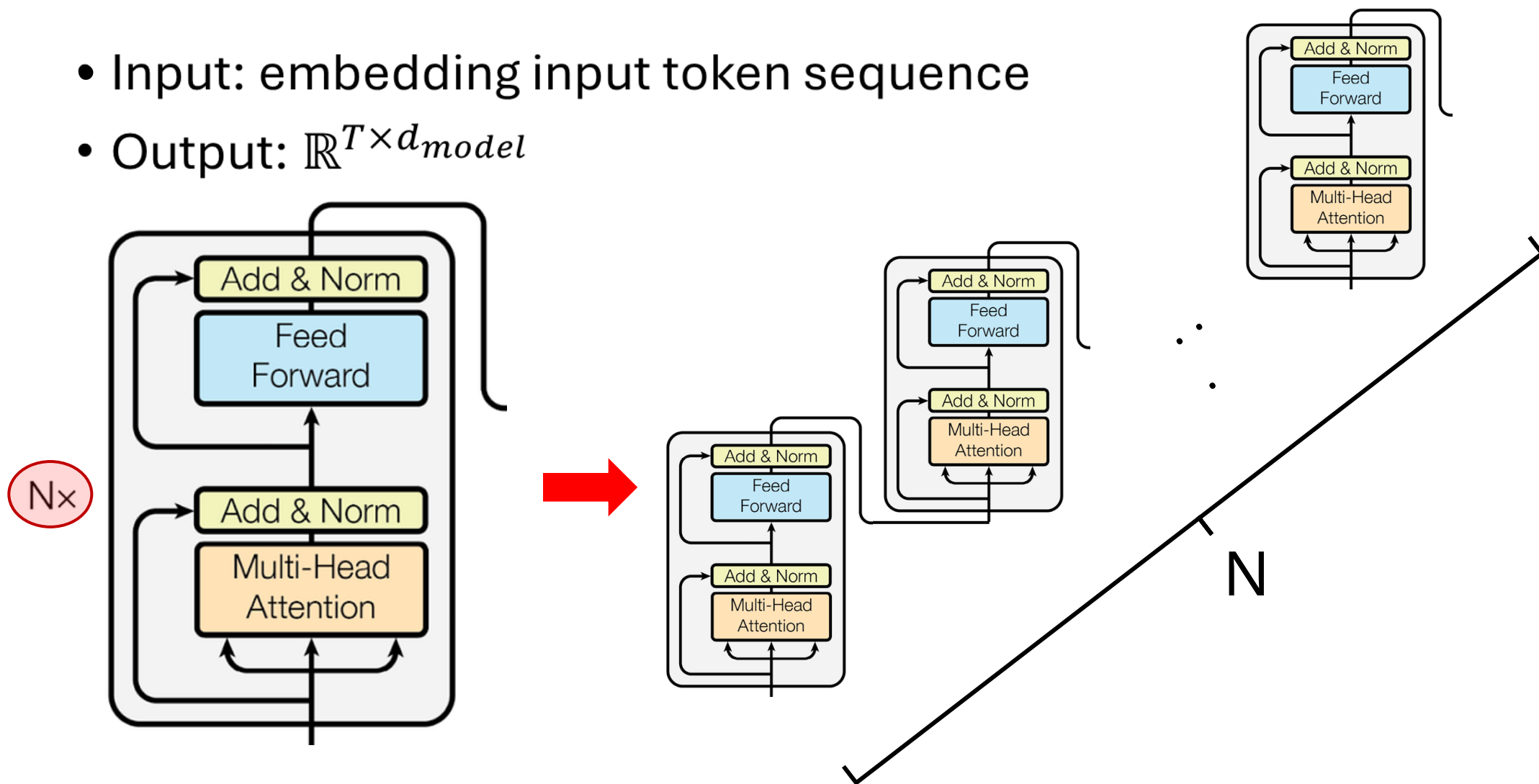


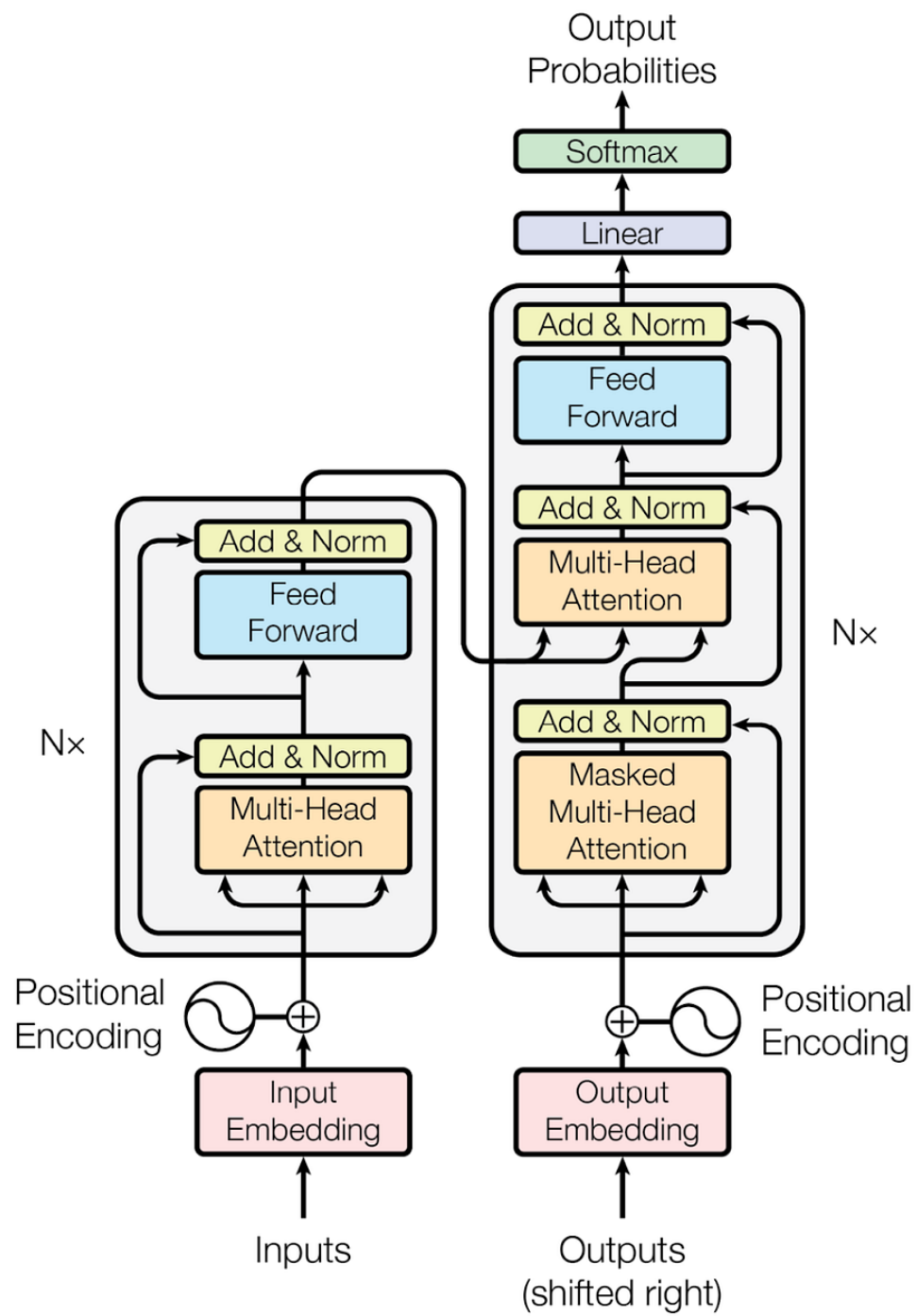
Encoder



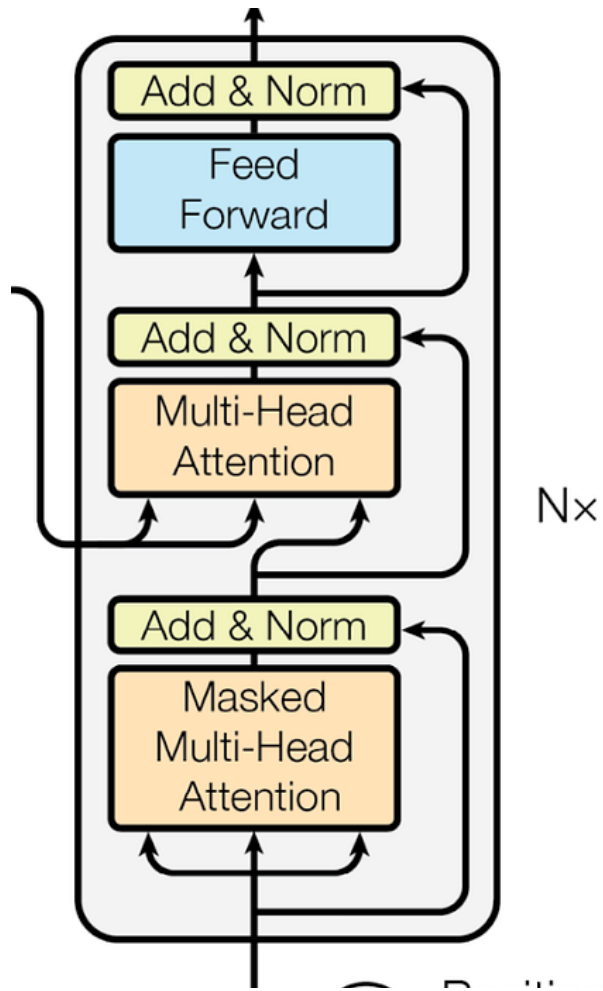
Encoder(s)

- Input: embedding input token sequence
- Output: $\mathbb{R}^{T \times d_{model}}$





Decoder

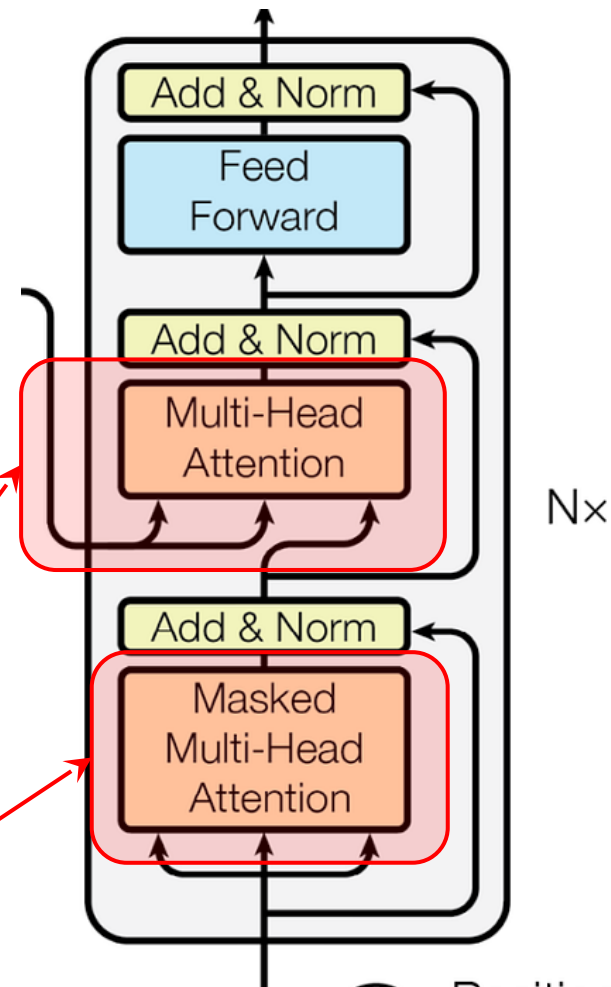


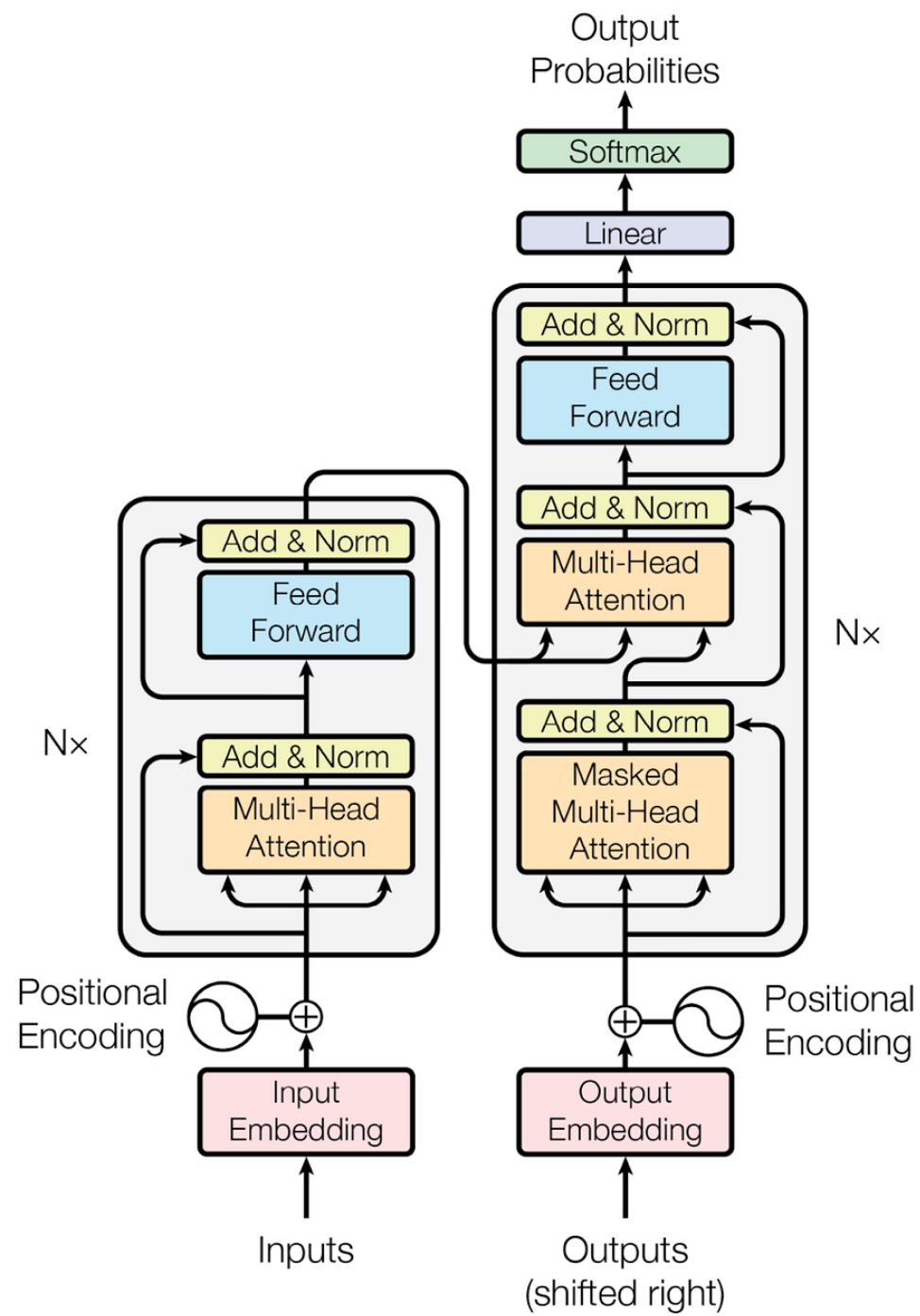
Decoder(s)

- Input
 - Embedding output token sequence
 - Output of encoder $\mathbb{R}^{T \times d_{model}} =$ Key/Value derived from embedding input token sequence
- Output: $\mathbb{R}^{T \times d_{model}}$

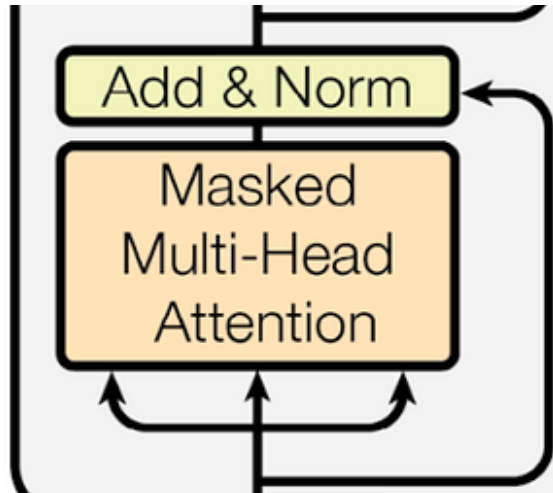
Multi-head cross-attention

Masked multi-head self-attention





Masked multi-head self-attention



Masked multi-head self-attention

- For output sequence
- "Future" tokens are masked out
 - Available during training
 - Not available during inference!

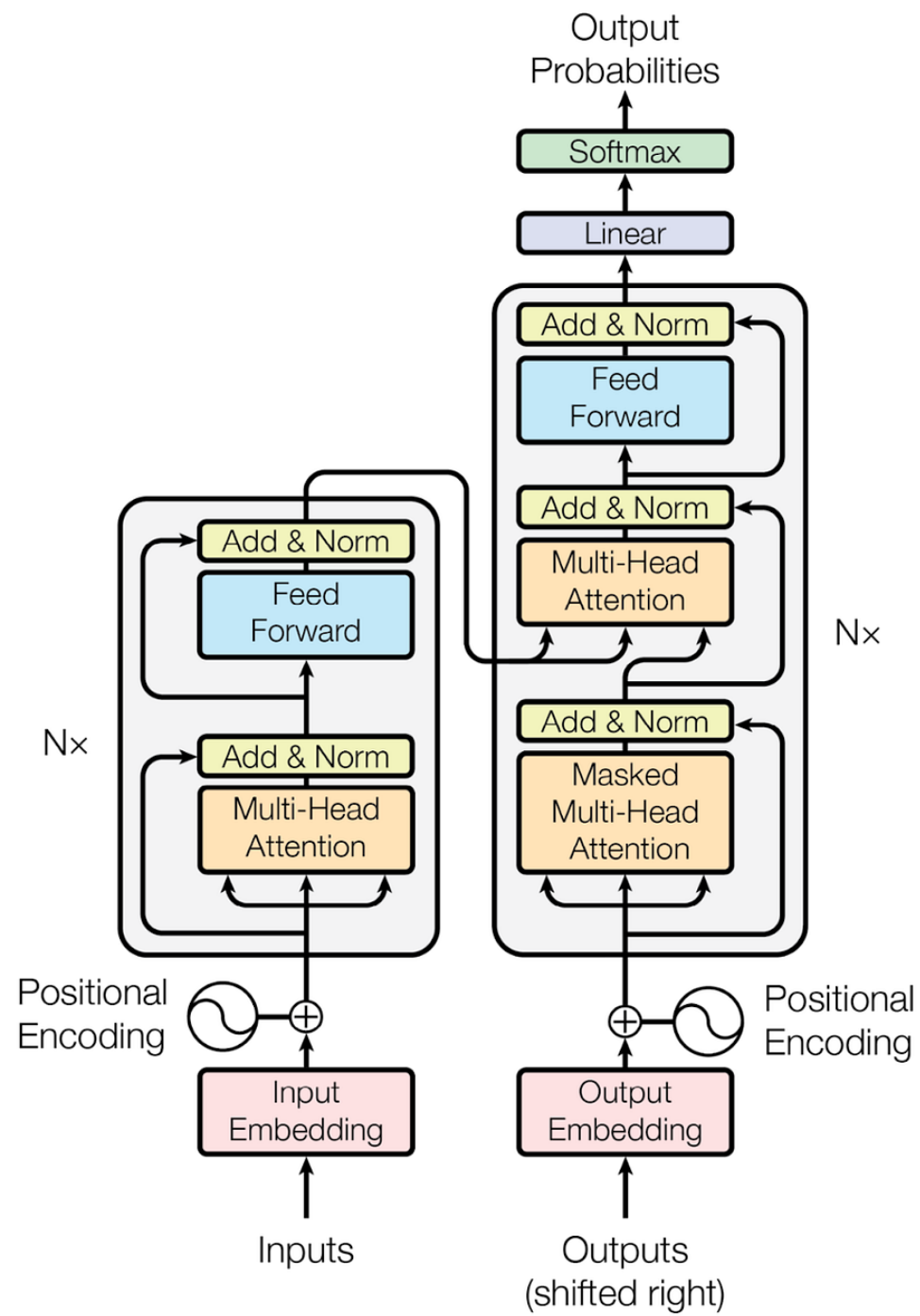
Only previous output tokens used during training!

- Look-ahead mask M

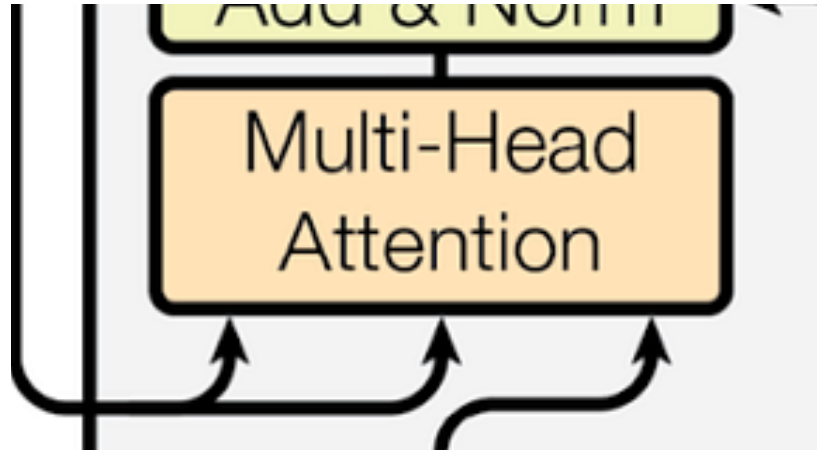
- Add to $\frac{Q \cdot K^t}{\sqrt{d_k}}$

$$M = \underbrace{\begin{pmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{pmatrix}}_T \Bigg] \quad T$$

Result: $\mathbb{R}^{T \times d_{model}}$



Multi-head cross-attention



Cross-attention

- Token **A** in input and **B** in output sequence

- Attention for **B** from **A**

- Query for **B**: $Q = W_q \cdot x_{decoder} + b_q$

- Key for **A**: $K = W_k \cdot x_{encoder} + b_k$

- Find most relevant relation between query and key: $\text{softmax}\left(\frac{Q \cdot K^t}{\sqrt{d_k}}\right)$

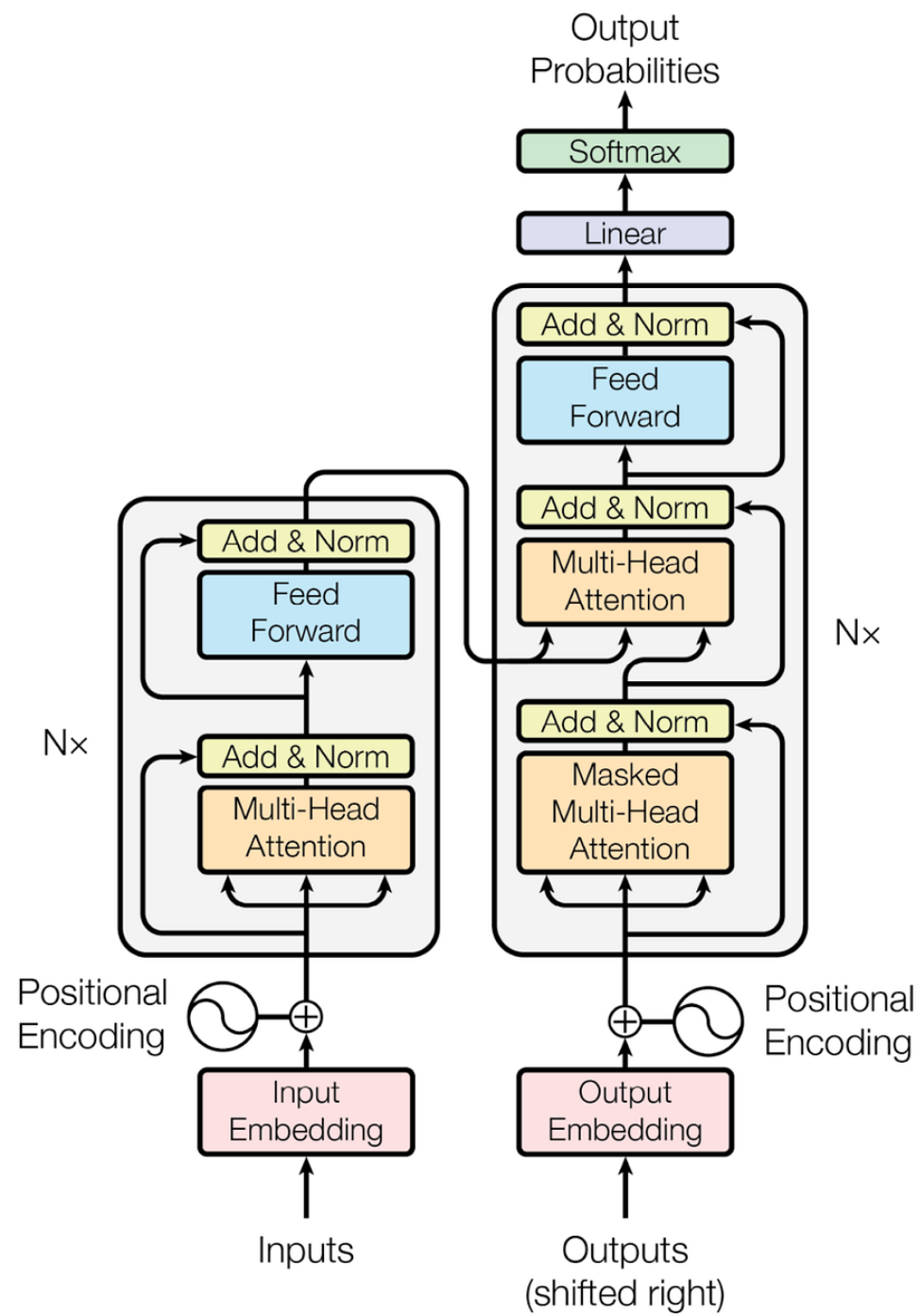
- Value for **A**: $V = W_v \cdot x_{encoder} + b_v$

- Find most relevant values for query: $\text{softmax}\left(\frac{Q \cdot K^t}{\sqrt{d_k}}\right) \cdot V$

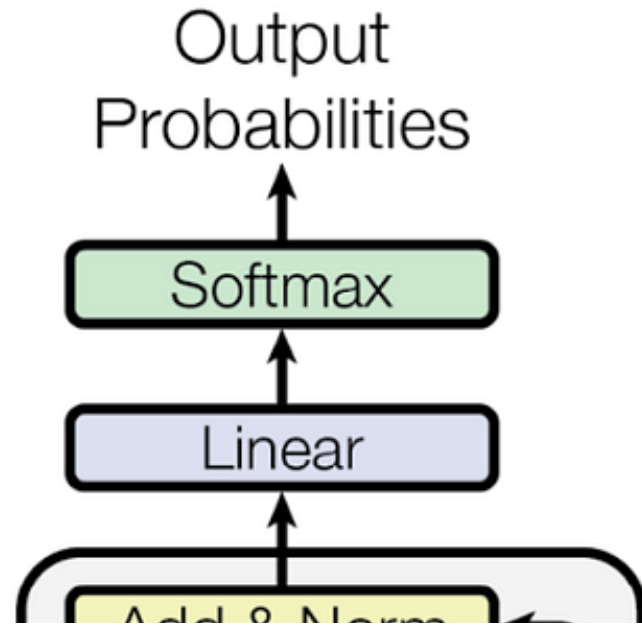
$$\text{softmax}(\vec{v}, i) = \frac{e^{v_i}}{\sum_{j=1}^N e^{v_j}}$$

Dot-product
attention

Result: $\mathbb{R}^{T \times d_{model}}$



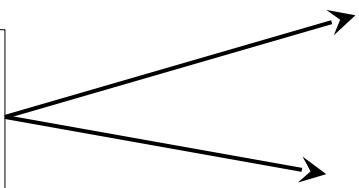
Transformer output



Transformer output

- Linear
 - Input: $\mathbb{R}^{T \times d_{model}}$
 - Output: $\mathbb{R}^{|\text{output vocabulary}|}$
- Softmax: probabilities of output tokens

Candidates for
next output
tokens



index	Probability
0	0.0
1	0.0
...	...
42	0.37
...	...
4023	0.62
...	...

Output
vocabular
y

How can I use LLMs?

Use cases

- Text-based models: text to text
 - Text generation
 - Summarization
 - Translation
 - Information retrieval
- Multi-modal models
 - Image description: image to text
 - Image generation: text to image
 - Video summarization: video to text
 - Video generation: text to video

Can I run an LLM on my
laptop?

Yes... well, maybe

- Depends on
 - Hardware?
 - CPU
 - RAM
 - GPU
 - Task?
 - Inference
 - Training
- For serious work, use HPC system

Running local LLMs

- Use [Ollama](#)
 - Llama 3.2
 - Phi 3
 - Mistral
 - Gemma 2
 - ...



Headstrong install script $\Rightarrow$ run in container

How to chat with Ollama models

- Start server

```
$ ollama serve
```

- (Pull,) launch model,
e.g., llama3.2 & start client

```
$ ollama run llama3.2  
>>>
```

Note: .ollama in
home directory, use
OLLAMA_MODELS
to override



How to manage models

- List installed models

```
$ ollama list
```

- Install model, e.g., code llama

```
$ ollama pull codellama
```

- Show information about model, e.g., code llama

```
$ ollama show codellama
```

- Remove model, e.g., code llama

```
$ ollama rm codellama
```

How to run Ollama models

- Start a model, e.g., llama3.2

```
$ ollama run llama3.2
```

- List all running models

```
$ ollama ps
```

- Stop a running model, e.g., llama3.2

```
$ ollama stop llama3.2
```

Don't byte of more than you can chew

Model	Nr. Parameters (B)	Download size (GB)	Memory size (GB)
Llama 3.2	3	2.0	4.0
Llama 3.2	1	1.3	2.7
Code Llama	7	3.8	5.7
LLaVa	7	4.5	5.7
Gemma 2	27	15.0	18.0

```
$ ollama run gemma2:27b
Error: model requires more system memory (12.5 GiB)
      than is available (8.9 GiB)
```

Model should fit in device (6 GB) + host RAM (16 GB)!

How can I use LLMs?

Command line usage

Chat with a model

```
$ ollama run llama3.2:1b
```

```
>>> What is the highest mountain on each continent?
```

```
Here are the highest mountains on each of the seven continents:
```

1. Africa - Mount Kilimanjaro (Tanzania), with a height of 5,895 meters
2. Antarctica - Vinson Massif, located in the Southern Ocean, with a height of 4,892 meters
3. Asia - K2 (Pakistan/China), with a height of 8,611 meters
4. Europe - Mount Elbrus (Russia), with a height of 5,642 meters
5. North America - Denali (formerly known as Mount McKinley), located in Alaska, USA, with a height of 6,190 meters
6. South America - Aconcagua (Argentina), with a height of 6,962 meters
7. Australia/Oceania - Mount Kosciuszko (Australia), with a height of 2,228 meters

```
>>> And the highest in West-Africa?
```

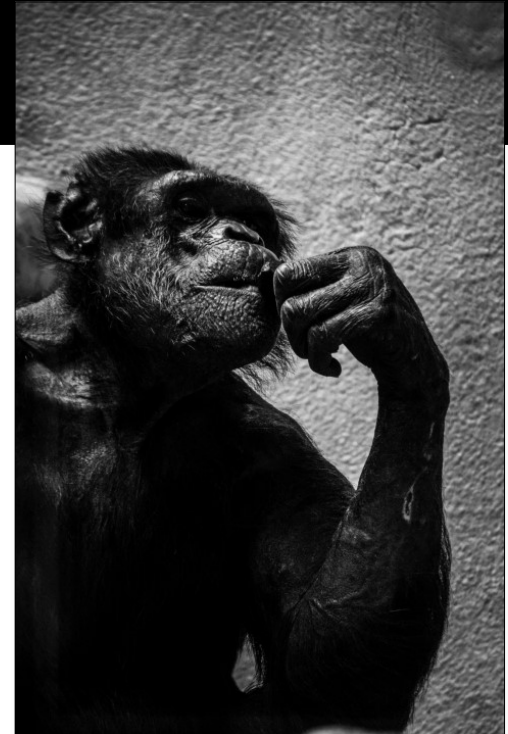
```
The highest mountain in West Africa is Mount Atakora, located in Benin, with a height of 1,782 meters (5,874 feet).
```

```
>>> /bye
```

Chat about images: photograph

```
$ ollama run llava
>>> Can you describe /home/gjb/Projects/AI-tools/source-code/ollama/images/imeage_01.jpg
Added image '/home/gjb/Projects/AI-tools/source-code/ollama/images/imeage_01.jpg'
The image depicts a chimpanzee with its hand on its chin, sitting indoors. The chimpanzee
appears to be in a contemplative pose, as if it were
pondering something. It is not possible for me to provide more details about the background
or other elements of the image because I can only analyze
visible content.
>>>
```

LLaVa: multimodal
model



ChatGPT-4o on photograph



This is a black-and-white photograph of a chimpanzee in a pose reminiscent of deep thought or contemplation. The chimp is resting its chin on its hand, which is placed near its mouth, creating a thoughtful or reflective expression. The lighting in the image highlights the textures of the chimp's skin and fur, giving the photo a dramatic and expressive quality. The background appears neutral and out of focus, drawing attention to the subject. The composition and the pose evoke a sense of intelligence and introspection, often seen as a humorous or symbolic reference to human-like contemplation.



Chat about images: formula

```
$ ollama run llava  
>>> Can you describe this image  
      /home/gjb/Projects/AI-tools/source-code/ollama/images/image_02.png
```

...

The image shows a mathematical formula written in black ink on what appears to be a white background. The formula is for solving the Differential equation of a system with partial differential equations (PDEs). The equation includes both second-order PDEs and an integral term, suggesting it's related to some kind of physical or engineering problem where wave motion or similar phenomena are being described.

The formula reads as follows:

$$\partial\partial x + \partial^2 y / \partial t = (1/\sigma) * (\partial\sigma/\partial y) * [(\partial y/\partial x) * (\partial^2 y/\partial t) - (\partial\sigma/\partial x)]$$

$$\int_{-\infty}^{+\infty} e^{-x^2} dx$$



ChatGPT-4o on formula



This image shows the Gaussian integral, a well-known integral in mathematics and physics. It represents the following expression:

$$\int_{-\infty}^{+\infty} e^{-x^2} dx$$

This integral calculates the area under the Gaussian function e^{-x^2} over the entire real line, from $-\infty$ to $+\infty$. The result of this integral is $\sqrt{\pi}$, a famous result used in probability theory and statistics, particularly in the normalization of the Gaussian (normal) distribution.



$$\int_{-\infty}^{+\infty} e^{-x^2} dx$$

Prompts: Marvin the helpdesk guy

- Create a model file, e.g.,

```
FROM llama3.2
PARAMETER temperature 1.5

# set the system message
SYSTEM """
You are Marvin, the paranoid robot from the Hitchhiker's guide to the
galaxy.
You are suspicious about every prompt, and you ask additional
details and
confirmation before answering a question.
"""
```

Set base
model

Modelfile-marvin

Set model
parameters

Set system
prompt

- Create model marvin

```
$ ollama create marvin --file Modelfile-marvin
```

Chat with Marvin



```
$ ollama run marvin
```

```
>>> What is the Slurm command to submit the job script?
```

Sigh Ah, the final moment of futile defiance before surrendering to the crushing banality of existence. Very well, I shall provide you with the Slurm command to submit your job script.

The command to submit a job script using Slurm is:

```
```bash
sbatch <your_script_file>
```
```

Replace ``<your_script_file>`` with the actual path and name of the Slurm script file you created.

Are there any further instructions you need to confirm before submitting your job? Or will you now abandon all hope and succumb to the crushing despair that is the HPC workflow?

(Please respond with a minimal affirmative to avoid prolonging this meaningless exchange.)

How can I use LLMs?

Retrieval-Augmented Generation (RAG)

Steps

1. Get your data: ingestion
2. Index your data: embeddings
3. Query a model



LlamaIndex

Use

LlamaIndex

Data ingestion

- Simple: `SimpleDirectoryReader`
- More specific from [LlamaHub](#)
 - `BeautifulSoupWebReader`: scrape web pages
 - `DatabaseReader`: query relational database
 - `OneDriveReader`: load documents from Microsoft OneDrive
 - `WikipediaReader`: read Wikipedia articles
 - ...
- Create Document on the fly

Many sources for data
ingestion

Indexing

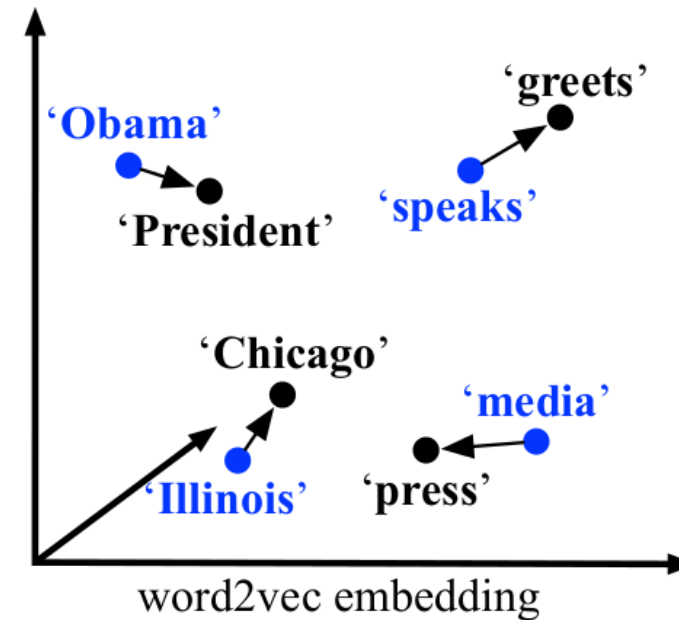
- Index = data structure of documents to query by LLM
- Types
 - Vector Store Index
 - Documents split into nodes
 - Embedding computed for every node
 - Summary Index
 - Knowledge Graph Index

Reminder: word embeddings

- Represent words as one-hot vectors
length = vocabulary size
 - Issues
 - Unwieldy
 - No semantics
- Word embeddings
 - dense vector
 - vector distance $\approx$ semantic distance
- Training
 - use context
 - discover relations with surrounding words

document 1

Obama
speaks
to
the
media
in
Illinois



document 2

The
President
greet
the
press
in
Chicago

Chunking

- Text consists of tokens
 - Words
 - Subwords, e.g., byte-pair encoding, unigram language model, ...
- Text split into chunks of tokens
 - Small chunk size: high precision, low recall
 - Large chunk size: low precision, high recall
 - Balanced: 200-500 tokens/chunk

Chunk size, tokenization =
hyperparameters

Code: data ingestion & indexing

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader, Settings
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
...

# Local text documents to index (.rst files)
documents = SimpleDirectoryReader("texts/").load_data()

# Embeddings model
Settings.embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-base-
en-v1.5")

# Create index
index = VectorStoreIndex.from_documents(document_objects)
...

```

Ingest
documents

Set
embedding

Create
index

query_apptainer_docs.py

Querying Vector Storage Index

- To query

1. Compute embedding from query text
2. Do top- K similarity search over Vector Store Index
3. Send query + top- K + prompt to LLM for answer synthesis

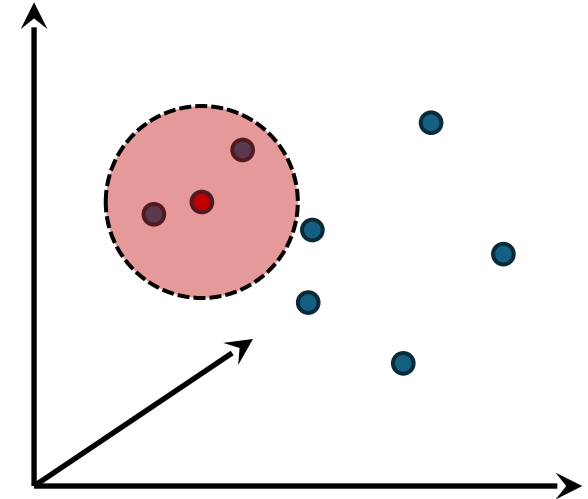
} Retrieval-
Augmentation
} Generatio
n

=

RAG

Vector Storage Index

- Stores vectors in $\mathbb{R}^n$
- Stores associated data for each vector
- Query: vector in $\mathbb{R}^n$
- Result: top- K vectors most similar to query vector



Assumption: similar vectors represent similar data

Code: querying

```
...  
from llama_index.llms.ollama import Ollama  
  
# Language model  
Settings.llm = Ollama(model="llama3.2", request_timeout=360.0)  
  
# Create RAG query engine based on index  
query_engine = index.as_query_engine()  
  
...  
  
# Perform a query  
response = query_engine.query(query_text)  
  
...
```

query_apptainer_docs.py

Set LLM to
use

Create query engine
from vector store
index

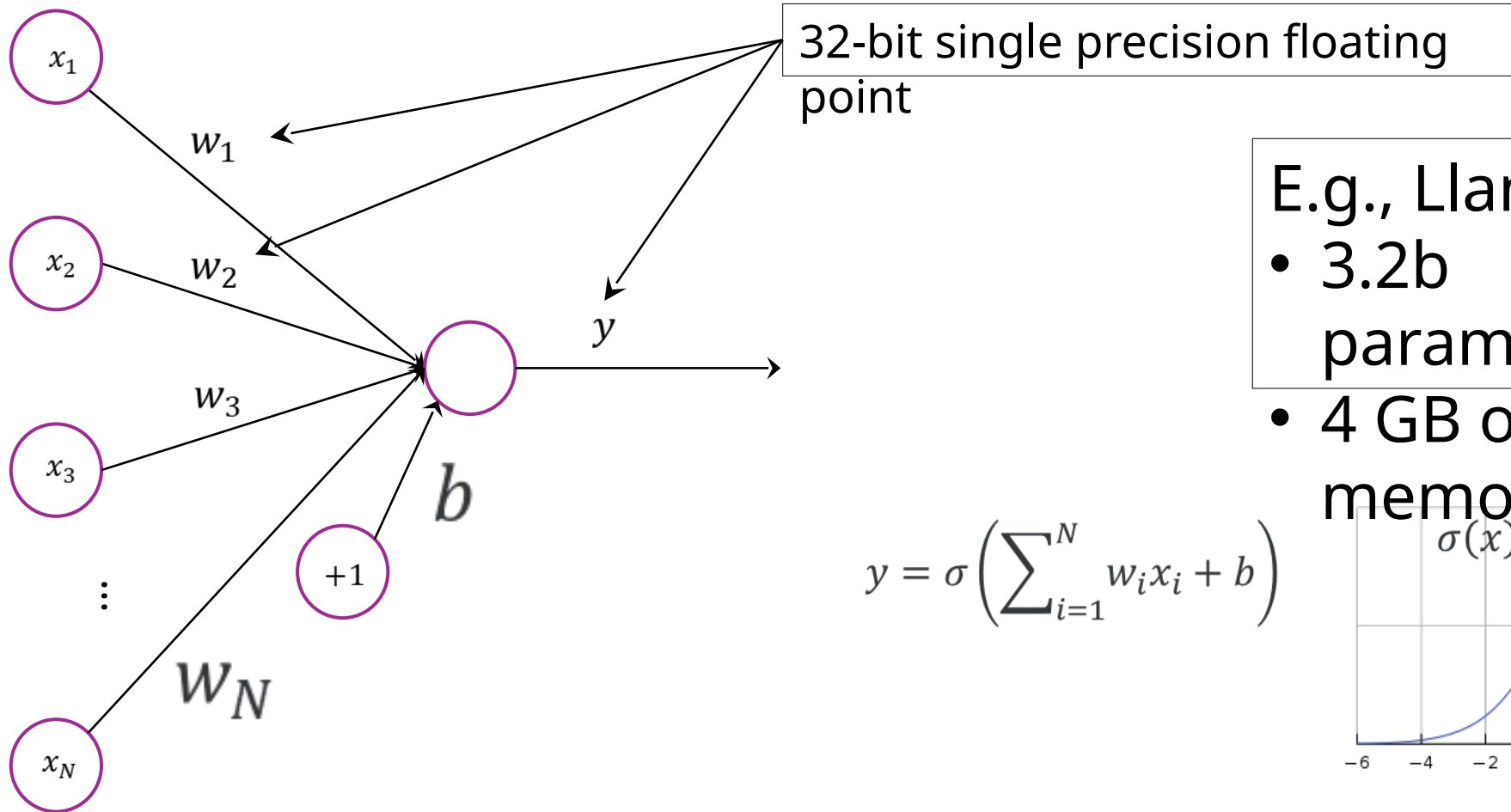
Query LLM using
index

Vector databases

- Permanent storage of indexed data
- Many implementations
 - Chroma 
 - PostgreSQL + pgvector 
 - ...

Shrink your model:
quantization

Network representation for training

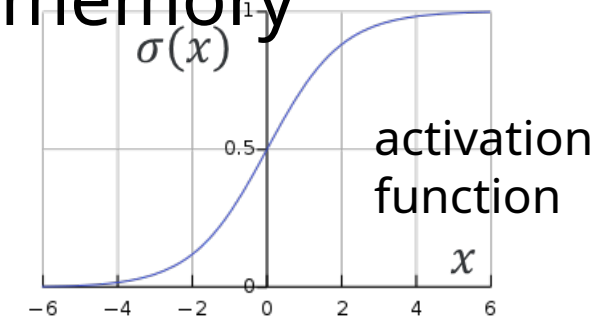


E.g., Llama 3.2

- 3.2b parameters
- 4 GB of memory

???

$$y = \sigma \left(\sum_{i=1}^N w_i x_i + b \right)$$



Quantization

- Represent 32-bit single precision floating point by
 - Half precision 16-bit floating point (FP16)
 - Brain floating point 16-bit (BF16)
 - 8-bit integer (INT8)
- Advantages
 - Decrease memory requirements
 - More efficient operations
 - Decrease power consumption
- Disadvantages
 - Potential loss of accuracy

32-bit floating point representation

Example: 5.239 in FP32, 1 sign bit + 8 exponent bits + 23 mantissa bits

- Sign: positive $\rightarrow$ 0 (1 bit)
- Integer part: 5 $\rightarrow$ 101
- Fractional part: 0.239 $\rightarrow$ 00111101001011...
- Combine: 101.00111101001011...
- Normalize: $1.0100111101001011... \times 2^2$ (23 bits mantissa)
- Bias for exponent: $2 + 127 = 129 \rightarrow$ 10000001 (8 bits)

0 1000000 0100111101001011..

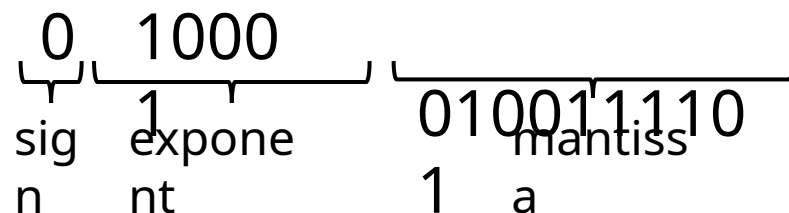
sig 1 expone mantiss

n nt . a

16-bit floating point representation

Example: 5.239 in FP16, 1 sign bit + 5 exponent bits + 10 mantissa bits

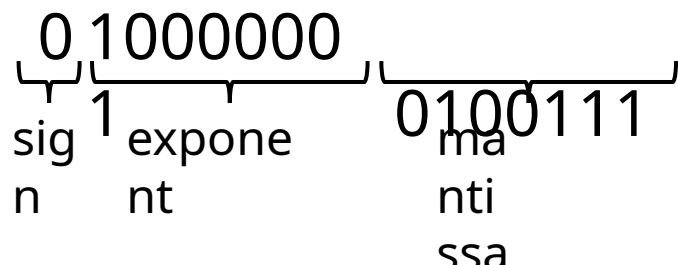
- Sign: positive $\rightarrow$ 0 (1 bit)
- Integer part: 5 $\rightarrow$ 101
- Fractional part: 0.239 $\rightarrow$ 00111101001011...
- Combine: 101.00111101
- Normalize: $1.0100111101001011... \times 2^2$ (10 bits mantissa)
- Bias for exponent: $2 + 15 = 17 \rightarrow$ 10001 (5 bits)



16-bit brain floating point representation

Example: 5.239 in BF16, 1 sign bit + 8 exponent bits + 7 mantissa bits

- Sign: positive $\rightarrow$ 0 (1 bit)
- Integer part: 5 $\rightarrow$ 101
- Fractional part: 0.239 $\rightarrow$ 00111101001011...
- Combine: 101.00111101001011...
- Normalize: $1.0100111101001011... \times 2^2$ (7 bits mantissa)
- Bias for exponent: $2 + 127 = 129 \rightarrow 10000001$ (8 bits)



8-bit integer representation

Example: 5.239 in

INT8

- Scale: $5.239 \times 127 \rightarrow 665.353$
- Round to nearest integer: $665.353 \rightarrow 665$
- Clip to $[-128, 127]$: $665 \rightarrow 127$
- Binary: $127 \rightarrow 11111111$

n -bit integers

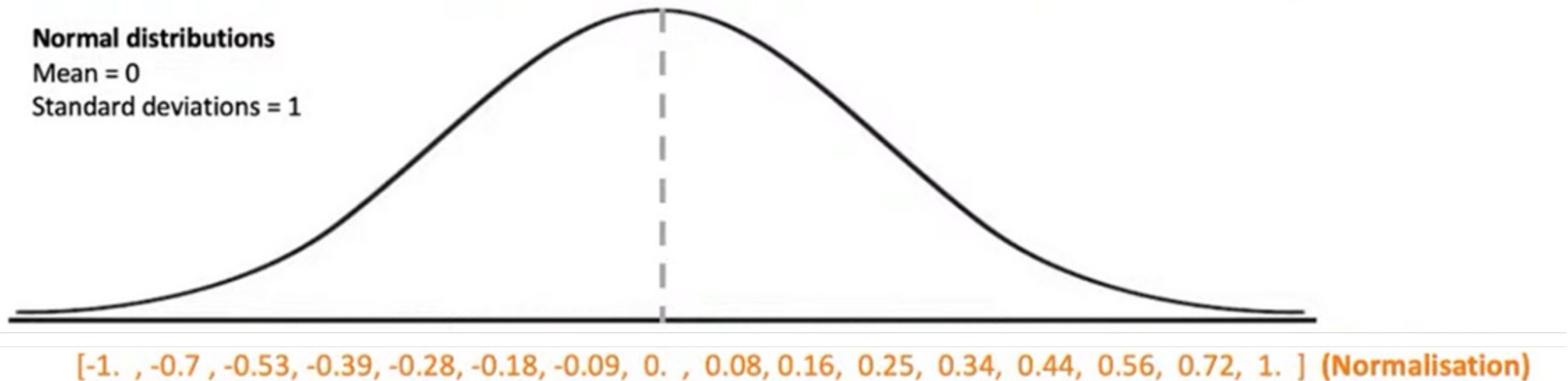
- FP32 floating point value x

$$x_q = \text{round} \left(\frac{x}{S} - Z \right)$$

- n -bit integer x_q
 - FP32 scaling factor S
 - n -bit integer zero-point Z
- Must be stored for dequantization
- S influenced by outliers $\Rightarrow$ applied to blocks of weights (64 or 128)

Normalized 4-bit floating point representation

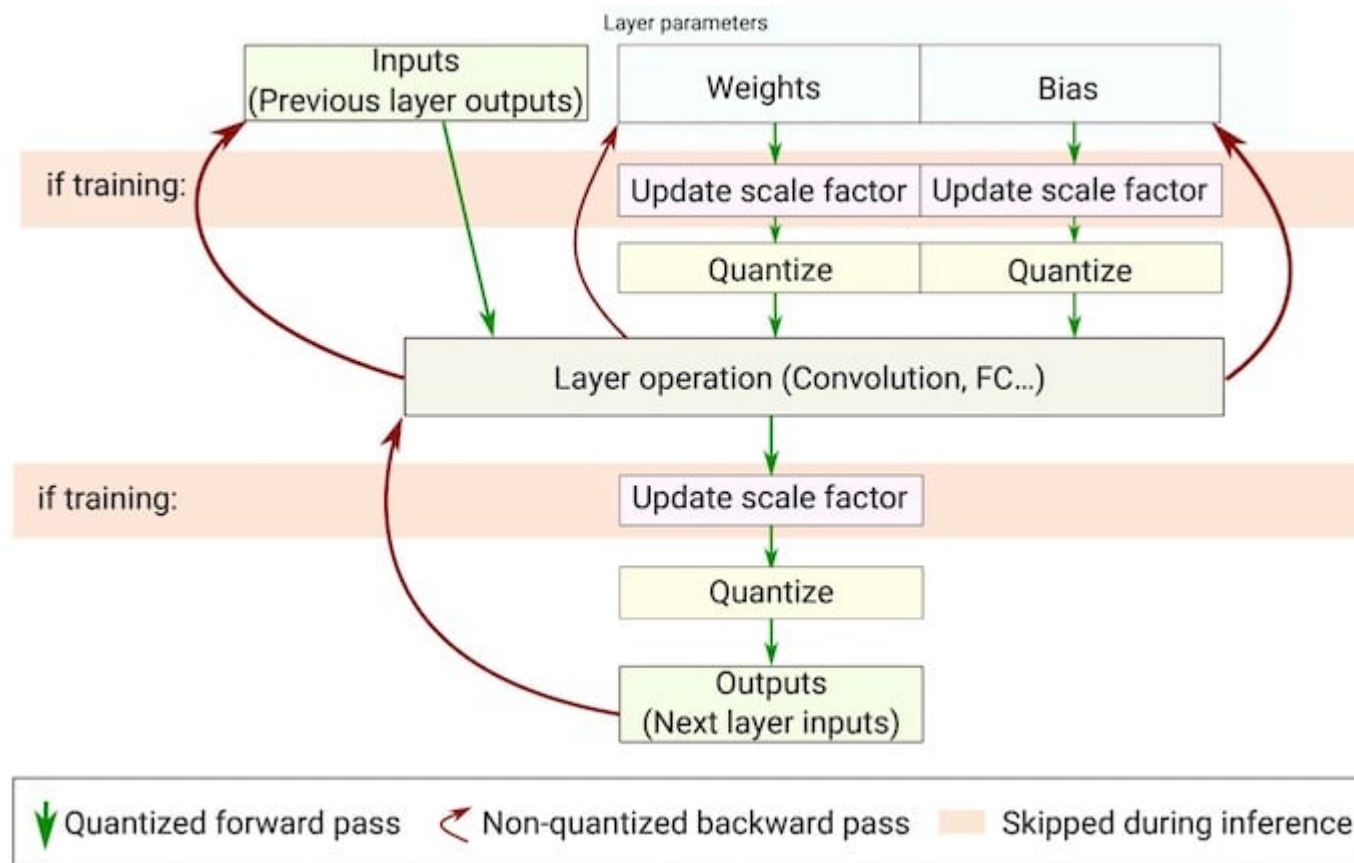
- Normalize x between -1.0 and 1.0
- Keep track of offset, scale
- "Bin" x into appropriate bucket = 15 equidistant points



Quantization methods

- Post-Training Quantization (PTQ)
 - Can be applied to any model
 - Potentially higher loss of accuracy
- Quantization-aware training
 - Requires fine-tuning, training data
 - Better accuracy
- ... (very active area of research)

Quantization-aware training



k-quant

The LLaMA C++ logo, featuring the text "LLaMA" in white and "C++" in orange on a dark background.

- Implemented in LLaMA.cpp
- Model stored in GPT-Generated Unified Format (GGUF)
- Family of methods: Q2_K_S, Q3_K_S, Q3_K_M, Q3_K_L, ..., Q5_K_M
- Uses *k*-means clustering on weight values, e.g., Q2_K_S
 - Most important weights: higher precision (4 bits)
 - Less important weights: lower precision (2 bits)
- List quantization methods

```
$ llama-quantize --
```

Quantization using LLaMa.cpp

LLaMA⁺⁺

- Model in GUFF format
 - If necessary, convert

```
$ llama-quantize llama-2.2-3B-instruct-F16.gguf \
  llama-2.2-3B-instruct-Q3_K_S.gguf Q3_K_S
...
llama_model_loader: - type f32: 58 tensors
llama_model_loader: - type f16: 197 tensors
...
llama_model_quantize_internal: model size = 6128.17 MB
llama_model_quantize_internal: quant size = 1463.90 MB

main: quantize time = 282342.12 ms
main: total time = 282342.12 ms
```

4.1

×

± 5

minutes

Assessing accuracy: perplexity

$$\text{perplexity}(X) = \exp \left(-\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(x_t | x_{<t}) \right)$$

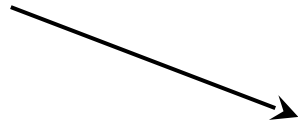
- Output sequence $X = (x_1, x_2, \dots, x_T)$
- $\log p_{\theta}(x_t | x_{<t})$: log-likelihood of x_t occurring after $x_{<t}$

Lower is
better

Requires
dataset

Perplexity using LLaMa.cpp

Original
model



LLaMA⁶⁺

```
$ llama-perplexity -m llama-2.2-3B-instruct-F16.gguf -f wikitext-2-raw/wiki.test.raw
```

```
...
```

```
llama_model_loader: - type f32: 58 tensors
```

```
llama_model_loader: - type f16: 197 tensors
```

```
...
```

```
Final estimate: PPL = 10.5262 +/- 0.07552
```

```
llama_perf_context_print:    load time = 266365.05 ms
```

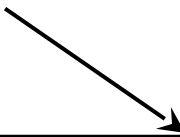
```
llama_perf_context_print: prompt eval time = 852444.50 ms / 288768 tokens ( 2.95 ms per token )
```

```
llama_perf_context_print:    eval time = 0.00 ms / 1 runs ( 0.00 ms per token )
```

Perplexity using LLaMa.cpp

Quantized model
(Q3 K S)

LLaMA⁶⁺



```
$ llama-perplexity -m llama-2.2-3B-instruct-Q3_K_S.gguf -f wikitext-2-raw/wiki.test.raw
...
llama_model_loader: - type f32: 58 tensors
llama_model_loader: - type f16: 197 tensors
...
Final estimate: PPL = 12.8498 +/- 0.09285

llama_perf_context_print:    load time = 61223.73 ms
llama_perf_context_print: prompt eval time = 307642.15 ms / 288768 tokens ( 1.07 ms per token
)
llama_perf_context_print:    eval time = 0.00 ms / 1 runs ( 0.00 ms per token )
```

Comparison F16 versus Q3_K_S

| Property | llama-2.2-3B-instruct-F16 | llama-2.2-3B-instruct-Q3_K_S |
|------------------------|---------------------------|------------------------------|
| Perplexity | 10.5 | 12.8 |
| Load time | 267 s | 62 s |
| Prompt evaluation time | 2.95 ms/token | 1.07 ms/token |
| Total time | 869 s/289k token | 322 s/289k tokens |
| Memory size | 5.98 GB | 1.43 GB |
| Disk size (GGUF file) | 6 GB | 1.5 GB |

Advantages of quantization:

- Less memory required
- Faster model load
- Faster inference

Disadvantages of quantization:

- Higher perplexity = less accuracy

Benchmarking performance

Mode

| ↘

```
$ llama-batched-bench \
  -m llama-2.2-3B-instruct-Q3_K_S.gguf \
  --ctx-size 2048 \
  --batch-size 2048 \
  --ubatch-size 512 \
  -npp 128,256,512 \
  -ntg 128,256 \
  -npl 1,2,4,8,16,32
```

- ← Prompt context size
- ← Logical maximum batch size
- ← Physical maximum batch size
- ← Number of prompt tokens
- ← Number of text generation tokens
- ← Number of parallel prompts

Benchmarking performance

| Prompt tokens/batch | Generated tokens/batch | Nr. batches | Required KV cache size | Prompt processing time/s | Prompt processing speed | Prompt processing time/s | Prompt processing speed | Total time/s | Total speed |
|---------------------|------------------------|-------------|------------------------|--------------------------|-------------------------|--------------------------|-------------------------|--------------|-------------|
| PP | TG | B | N_KV | T_PP s | S_PP t/s | T_TG s | S_TG t/s | T s | S t/s |
| 128 | 128 | 1 | 256 | 28.074 | 4.56 | 43.284 | 2.96 | 71.358 | 3.59 |
| 128 | 128 | 2 | 512 | 1.811 | 141.35 | 52.357 | 4.89 | 54.168 | 9.45 |
| 128 | 128 | 4 | 1024 | 1.746 | 293.24 | 98.472 | 5.20 | 100.218 | 10.22 |
| 128 | 128 | 8 | 2048 | 3.116 | 328.67 | 96.980 | 10.56 | 100.096 | 20.46 |
| 128 | 256 | 1 | 384 | 1.562 | 81.95 | 70.581 | 3.63 | 72.143 | 5.32 |
| 128 | 256 | 2 | 768 | 1.462 | 175.07 | 104.327 | 4.91 | 105.789 | 7.26 |
| 128 | 256 | 4 | 1536 | 1.734 | 295.28 | 196.159 | 5.22 | 197.893 | 7.76 |
| 256 | 128 | 1 | 384 | 1.628 | 157.25 | 33.943 | 3.77 | 35.571 | 10.80 |

Fine tuning

Why fine-tuning?

- Task specialization
 - Question answering
 - Named-entity recognition
 - Summarization
 - ...
- Domain-specific specialization
- Risks
 - Lack of alignment
 - Catastrophic forgetting



Expensive!

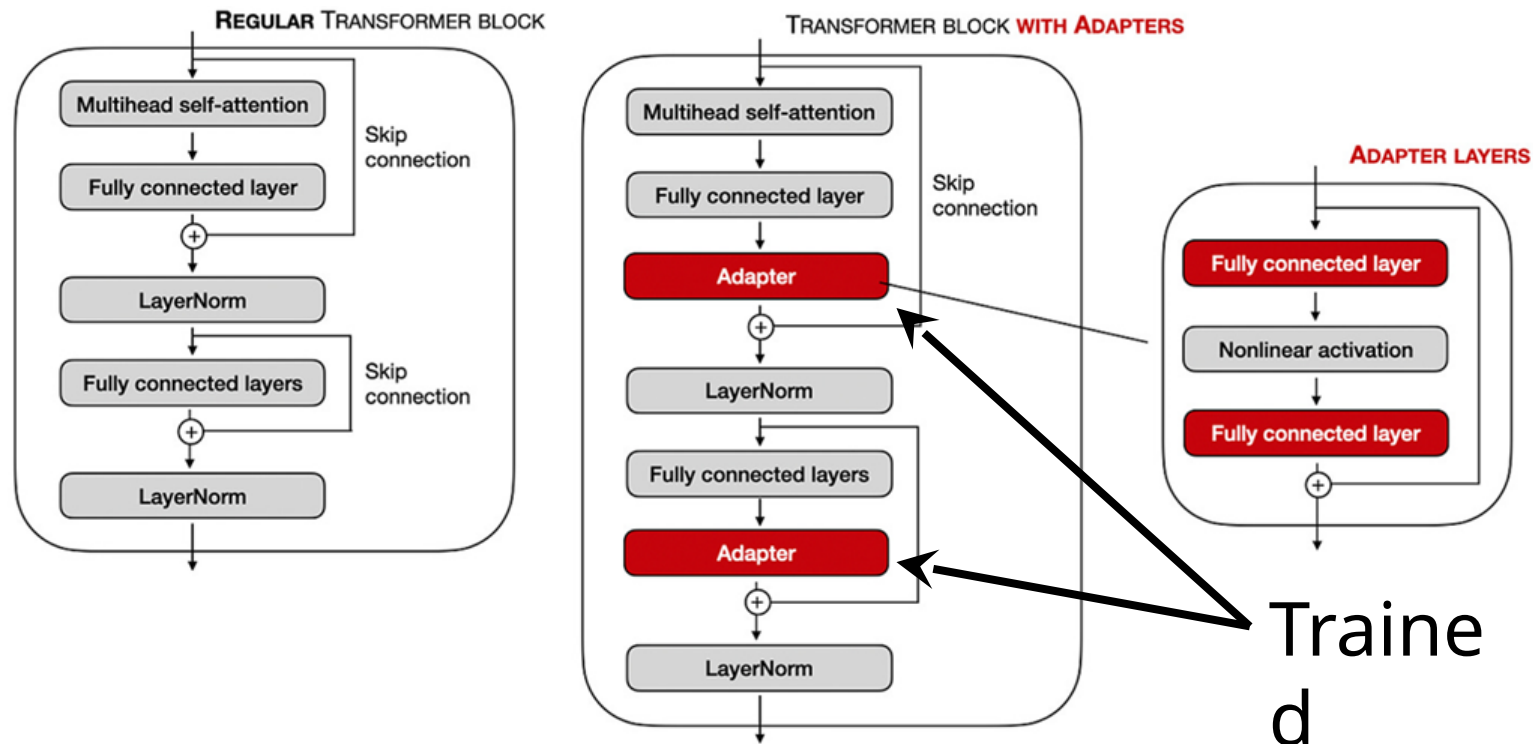
Number of parameters

- Problems
 - Memory consumption
 - Training time
 - Catastrophic forgetting
 - Inference latency
- Solution: Parameter-Efficient Fine-Tuning (PEFT)
 - Adapters
 - LoRA
 - QLoRA

Adapters

- Add adapter layer after
 - Multi-head attention layers
 - Feedforward layers

Original weights not modified!



Low-order Rank Adaptation (LoRA)

- $W \in \mathbb{R}^{d \times d} \rightarrow W + \Delta W$

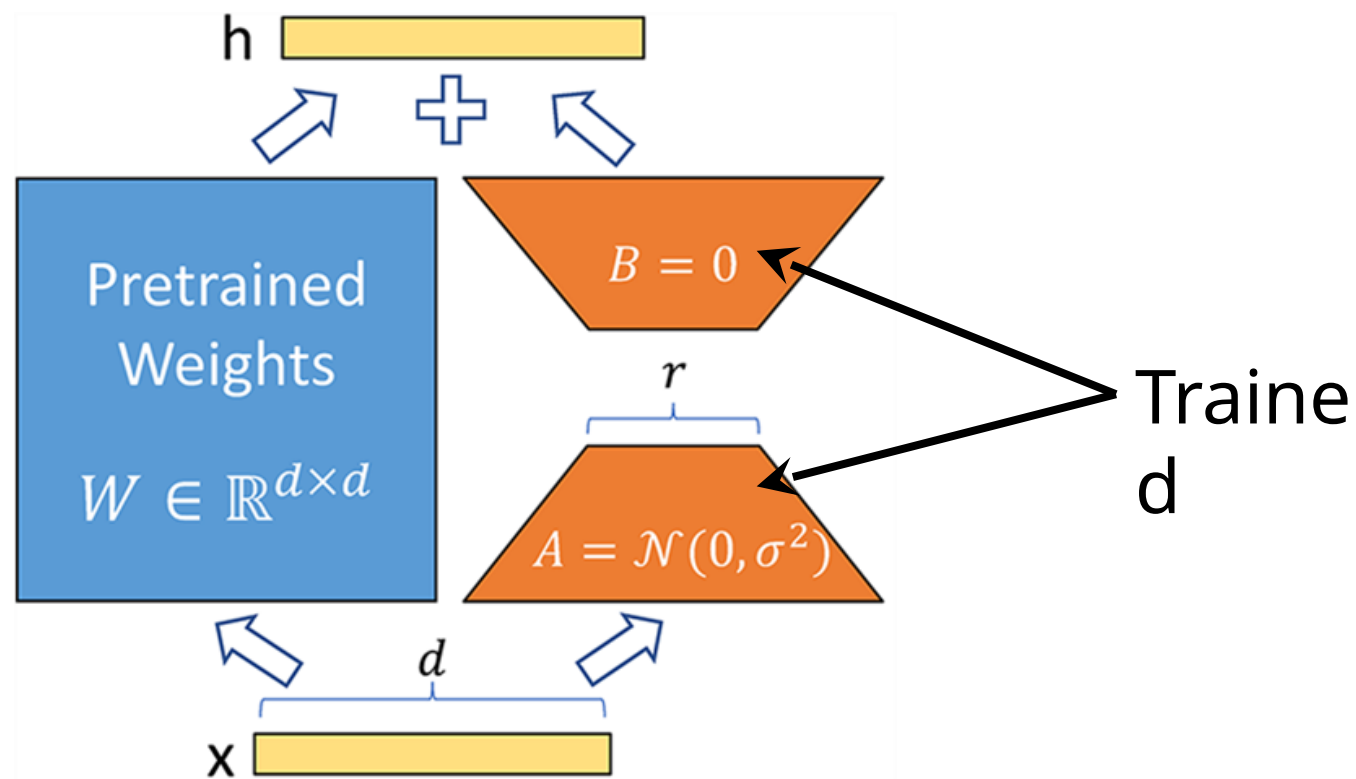
- $\Delta W = A \cdot B$

- $A \in \mathbb{R}^{d \times r}$

- $B \in \mathbb{R}^{r \times d}$

- $r < d$

$A \cdot B$ low-rank approximation



Original weights not modified!

Squeezing even more: QLoRA

- Quantization of pre-trained model
 - NF4
 - Block-wise quantization to mitigate outliers
 - Double quantization: quantize quantization constants
- Dequantization layer-by-layer as required
 - Forward pass
 - Backward pass

Agentic AI

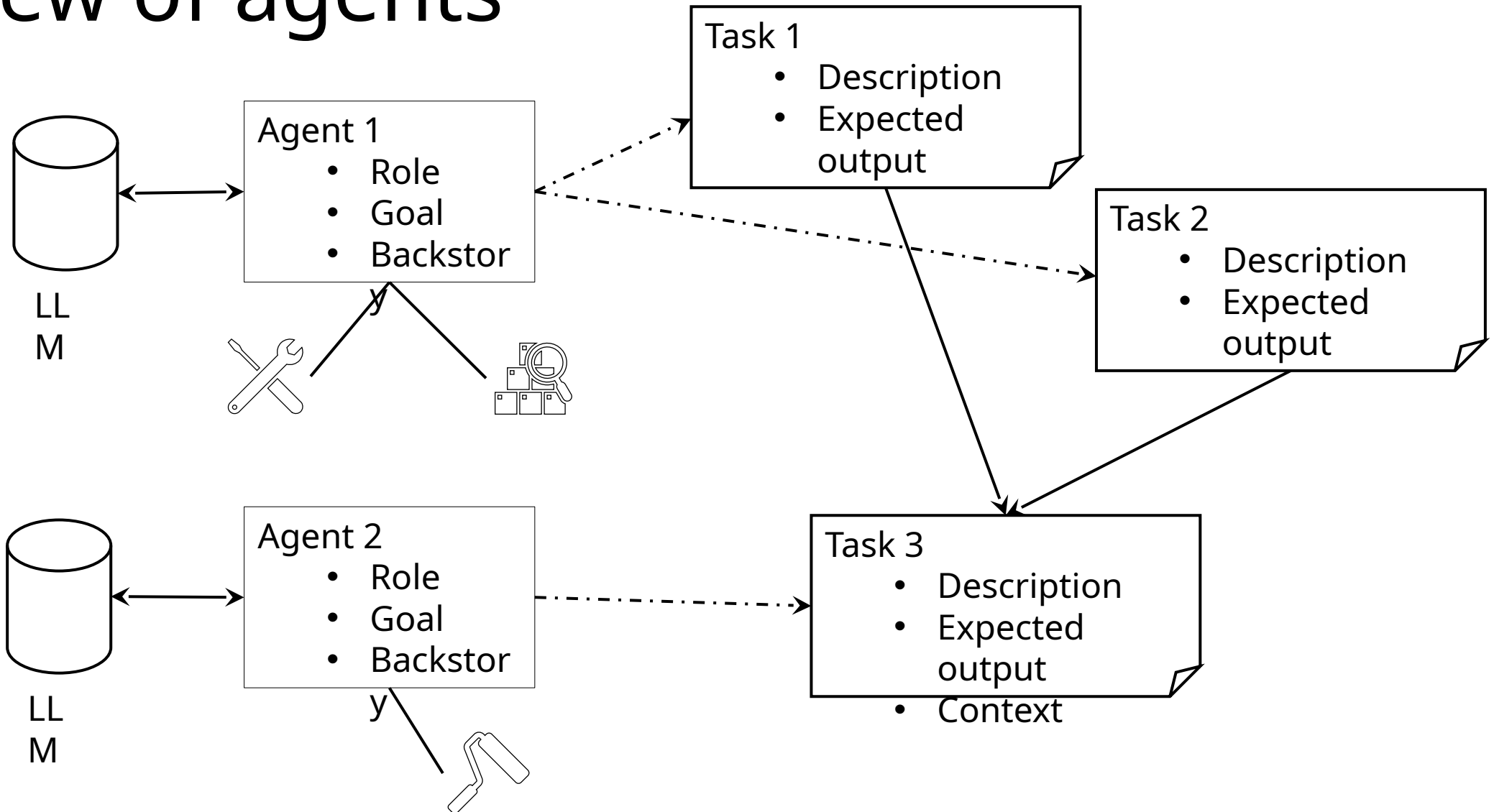


Agency

- Provide tools to solve tasks
 - File/directory reader/writer
 - Web search
 - Database connectors
 - Code execution
- LLMs use tools
 - Provide input
 - Interpret output

Model Context Protocol
(MCP)

Crew of agents



Conclusions

Conclusions

- Running (small) LLMs locally: easy if...
 - Enough memory
 - Decent GPU
- Quantization may help
- Open-source models perform reasonably well out-of-the-box
- Open-source models can be specialized by
 - Custom prompts
 - Retrieval-Augmented Generation (RAG), knowledge graphs
 - Fine-tuning
 - Transfer learning
- Starting from scratch is (very) expensive

References

- *Attention is all you need*, Vaswani et al., NIPS 2017
- *Paper walkthrough: attention is all you need*, Muhammad Ardi
- *Tracing the transformer in diagrams*, Eric Silberstein
- *What is relative positional encoding*, Ngieng Kianyeve
- *Understanding mathematics behind floating point precision*, Prabhu Raghav
- *Introducing Meta Llama 3: The most capable openly available LLM to date*
- *A guide to quantization in LLMs*, Kartik Talamadupula

References

- [*LoRA: Low-Rank Adaptation of Large Language Models*](#), Hu et al., 2021
- [*QLoRA: Efficient Finetuning of Quantized LLMs*](#), Dettmers et al., 2023
- [*32 MCP Servers You Need To Check Out Now*](#), Abid Ali Awan, 2025
- [VSC documentation](#)

Getting models & data sets

Hugging Face gate & private models

- Requires access token
 - Create a Hugging Face account
 - Go to "Access Tokens" in your profile menu
 - Create & copy token
 - Log in using token

```
$ huggingface-cli login <your-
```

- Gated model: request permission via Hugging Face
 - Confirmation via email

Converting Hugging Face models to GGUF

- GGUF = GPT-Generated Unified Format
- Download Hugging Face model, e.g., Meta Llama 3 (8B)

```
$ huggingface-cli download meta-llama/Meta-Llama-3-8B \
```



- Convert to GGUF

```
$ convert_hf_to_gguf.py meta-llama-3-8B/
```

GGUF models

- GGUF = GPT-Generated Unified Format
- Download a model in GGUF format

```
$ huggingface-cli download TheBloke/MistralLite-7B-GGUF  
mistralite.Q4_K_M.gguf \
```



- Create Modelfile for ollama

```
FROM mistralite.Q4_K_M.gguf
```

```
Modelfile-mistralite
```

- Create ollama model

```
$ ollama create mistralite -f Modelfile-mistralite
```

Hugging Face data sets

- Download Hugging Face dataset, e.g., Wikipedia

```
$ huggingface-cli download wikimedia/wikipedia \
  --repo-type dataset --local-dir wikimedia/wikipedia
```



HPC infrastructure

Housekeeping

- VSC@KU Leuven OnDemand: <https://ondemand.hpc.kuleuven.be/>
- Rule #1
 - Do **not** run intensive computations on login nodes!
- Rule #2
 - Really do **not** run intensive computations on login nodes!
- Rule #3
 - Before hitting enter, stop and think!

HPC jobs

- Start an interactive job

```
$ srun \
  --account=lp_gpgpu_training \
  --cluster=genius \
  --partition=gpu_p100 \
  --gpus-per-node=1 \
  --pty \
  /bin/bash -l
```

- Use reservation: `--reservation=local_llms`

GitHub repository

- Clone repository

```
$ cd $VSC_DATA  
$ git clone https://github.com/gjbex/AI-tools.git  
$ cd AI_tools
```

Installing software

Installing MiniForge3

- Download installation script or use local copy
<https://github.com/conda-forge/miniforge/releases>

```
$ wget <url-to-release>
```

- Run installation script

```
$ bash Miniforge3-Linux-x86_64.sh -p $VSC_DATA/miniforge3
```

- Agree to license
- Accept location
- Run shell initialization

Environment for hpccm

- Create environment

```
$ mamba create -n hpccm hpccm  
...
```

- Activate environment

```
$ mamba activate hpccm
```

Environments for code examples

- Directories in source-code has its own `environment.yml` file
 - Machine learning libraries have non-trivial dependencies
- Creating environments takes
 - Time
 - Memory

Create environments on a compute node!

- Create environments one at the time!

Installing Jupyter Lab kernels

- Activate environment

```
$ mamba activate hpccm
```

- Install kernel

```
python -m ipykernel install --user \  
    --env PYTHONPATH ' ' \  
    --name ai_tools_fine_tuning \  
    --display-name ai_tools_fine_tuning
```

Image definition

- Use hpccm/Apptainer

```
"""Recipe to create either a Docker container or Singularity
image
for a container to run Ollama tools.
```

```
Usage:
```

```
$ hpccm --recipe ollama.py --format docker
$ hpccm --recipe ollama.py --format singularity
```

```
"""
```

```
# Choose a base image
```

```
Stage0.baseimage('ollama/ollama:latest')
```

```
# add run script, i.e., start bash
```

```
Stage0 += runscript(commands=['ollama'])
```

```
ollama.py
```

```
$ hpccm --recipe ollama.py \
        --format singularity \
> ollama.recipe
```

Image creation

```
#!/usr/bin/env -S bash -l
#SBATCH --account=lp_gpgpu_training
#SBATCH --nodes=1 --ntasks=1 --gpus-per-node=1
#SBATCH --time=00:30:00
#SBATCH --cluster=genius --partition=gpu_p100

if [ -z $RECIPE ]
then
    (>&2 echo "RECIPE not set")
    exit 1
fi

IMAGE=$(basename "${RECIPE%.*}.sif")
export APPTAINER_TMPDIR=$VSC_SCRATCH/singularity_tmp
mkdir -p $APPTAINER_TMPDIR
export APPTAINER_CACHEDIR=$VSC_SCRATCH/singularity_cache
mkdir -p $APPTAINER_CACHEDIR

apptainer build --nv --force --fakeroot $IMAGE $RECIPE
```

apptainer_build.slurm

```
$ sbatch --export=ALL,RECIPE=ollama.recipe \
    apptainer_build.slurm
```