CS 696 Applied Large Langauge Models
Spring Semester, 2025
Doc 6 Cluster, Embedding, Attention
Jan 30, 2025

These slides use material from
Acknowledgments

Wikipedia
https://en.wikipedia.org/wiki/Byte_pair_encoding

Building a Large Language Model (from Scratch), Sebastian Raschka

Hands on Large Language Models, Jay Alammar and Maarten Grootendorst

Gemini Pro

RoFormer: Enhanced transformer with Rotary Position Embedding, Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, Yunfeng Liu, Neurocomputing, Volume 568, 1 February 2024, 127063

SDSU GPU Cluster

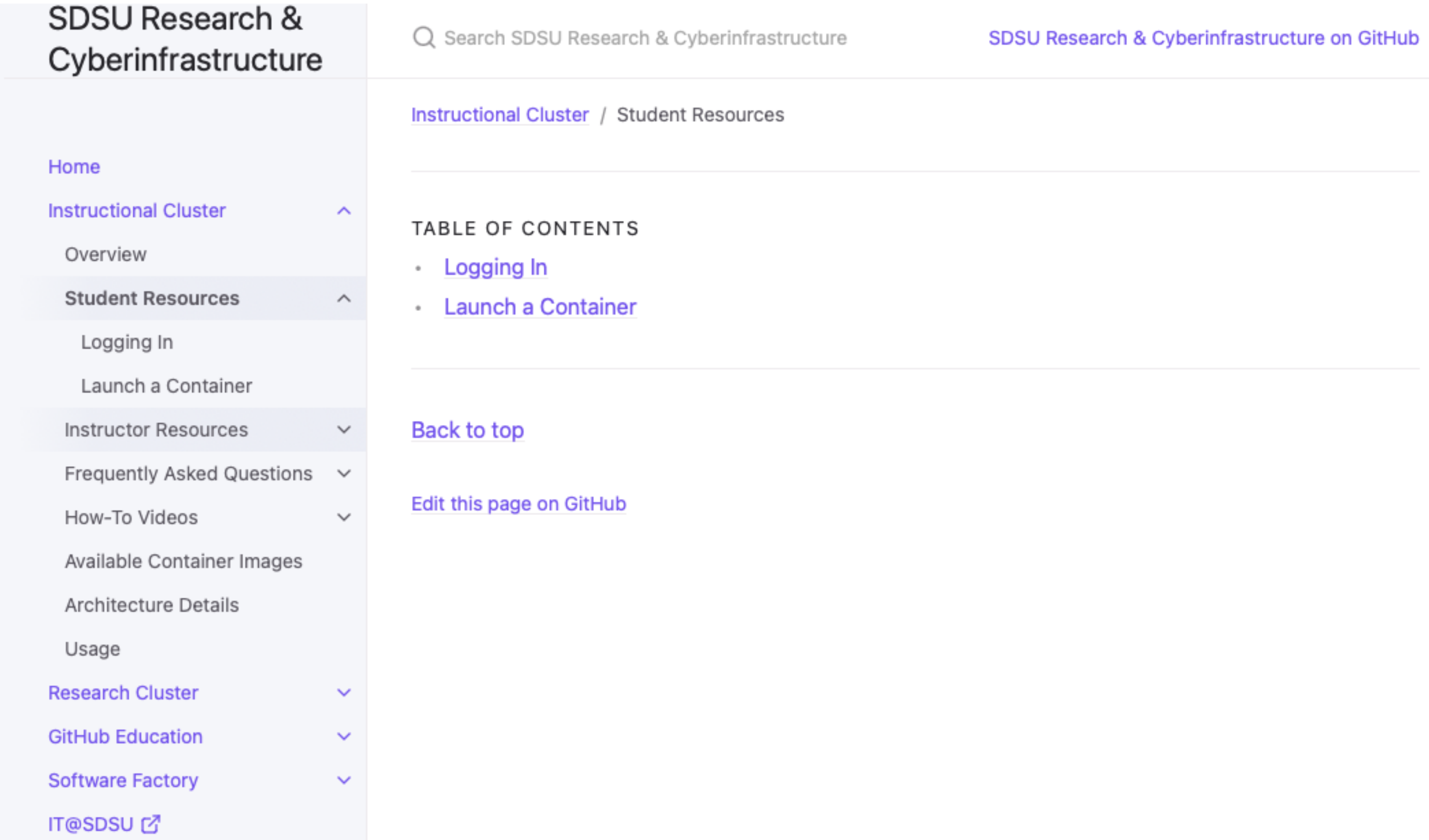
PowerEdge R750XA
(2x) Intel Xeon Gold 6338 2G CPU, 32C/64T
(4x) Nvidia A100 GPU, 80 GB RAM
512 GB System RAM

The GPUs available have 10GB of RAM
Suitable for training/fine-tuning 7-8 billion parameter models.
Students 75GB of persistent storage space

Accessing GPU Cluster

Documentation

https://sdsu-research-ci.github.io/instructionalcluster/students



Logging In

Go to:

https://jupyterhub.sdsu.edu/

Enter your SDSUid credentials

Login to CILogon

SDSUid (e.g. jsmith@sdsu.edu)

Password

Don't Remember Login

Login

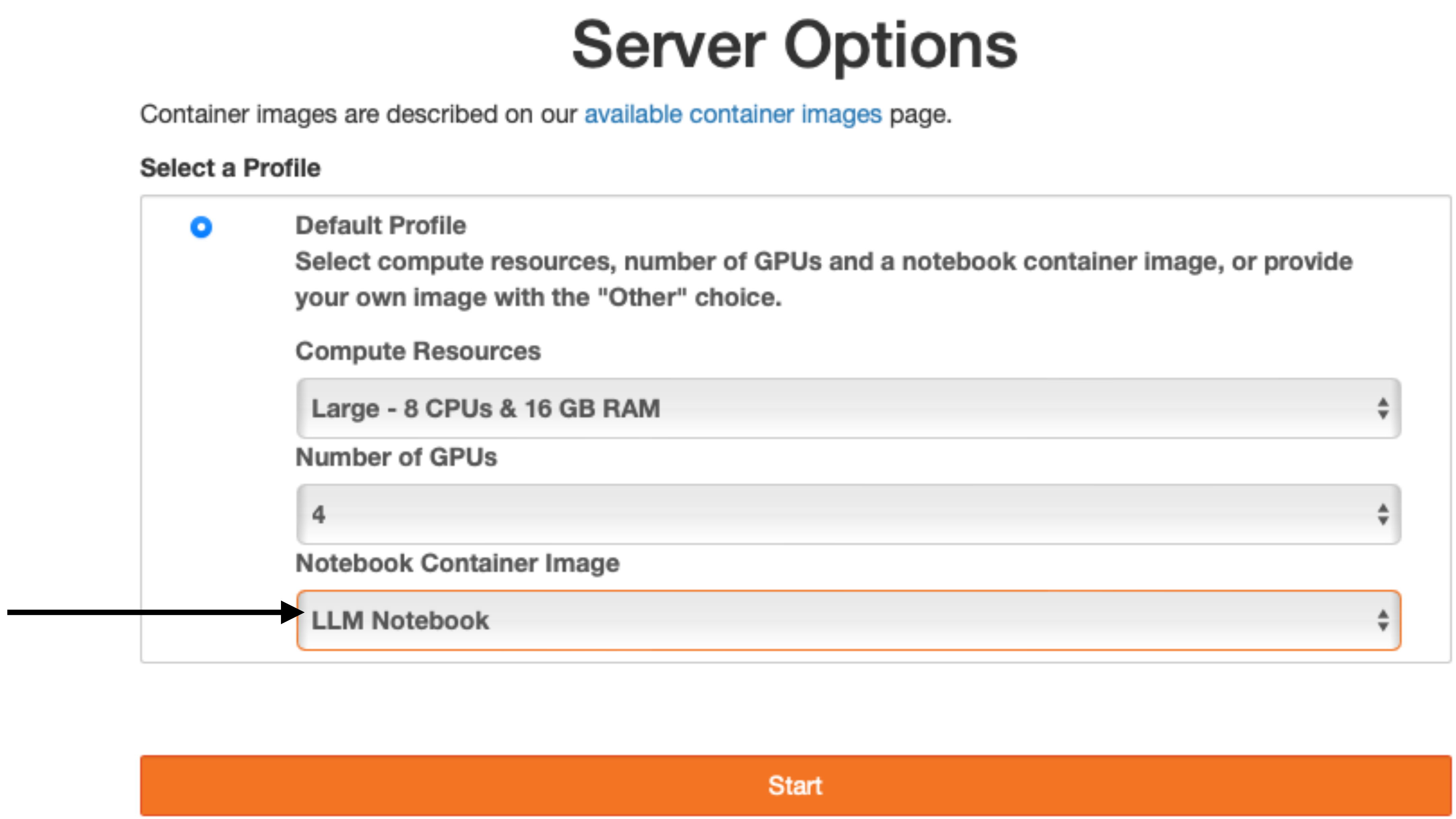
› Recover your password

› Need Help?

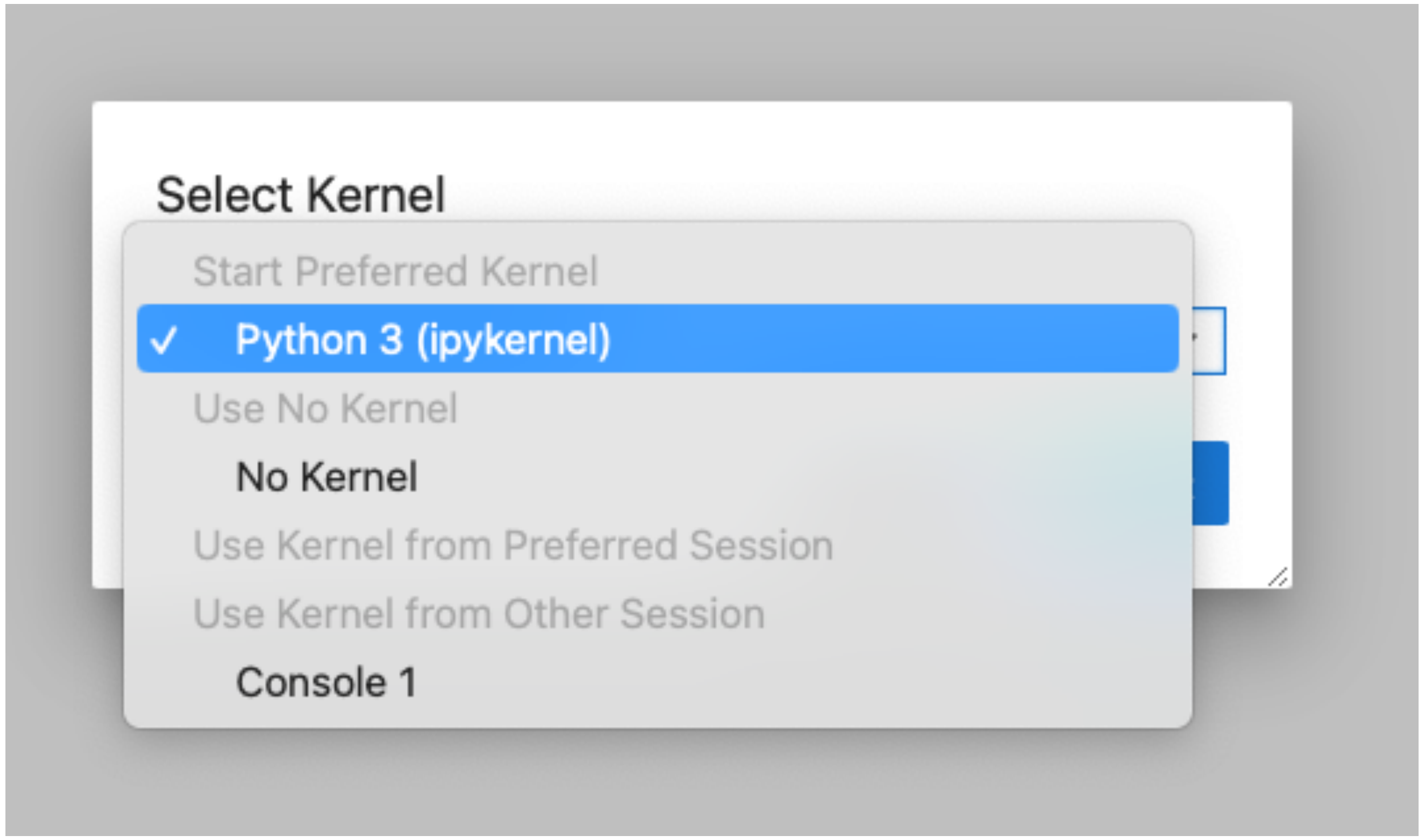
CILogon

CILogon facilitates secure access to CyberInfrastructure (CI).

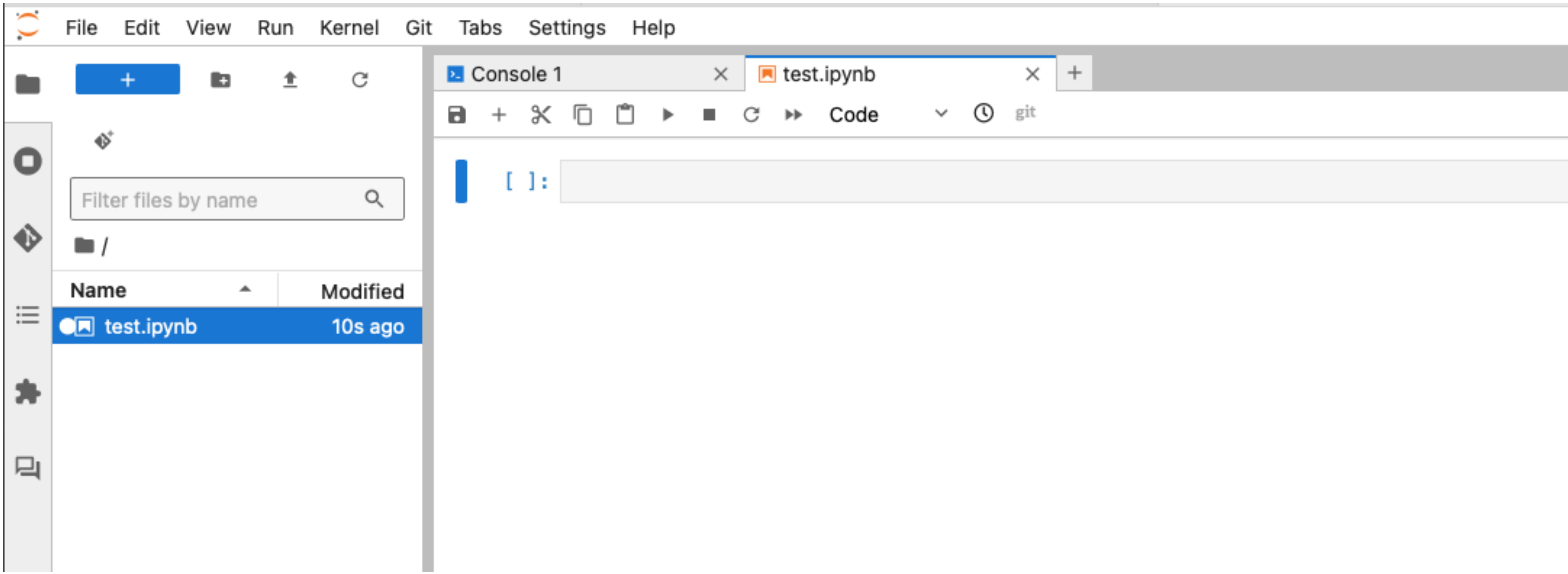
Enter Server Options



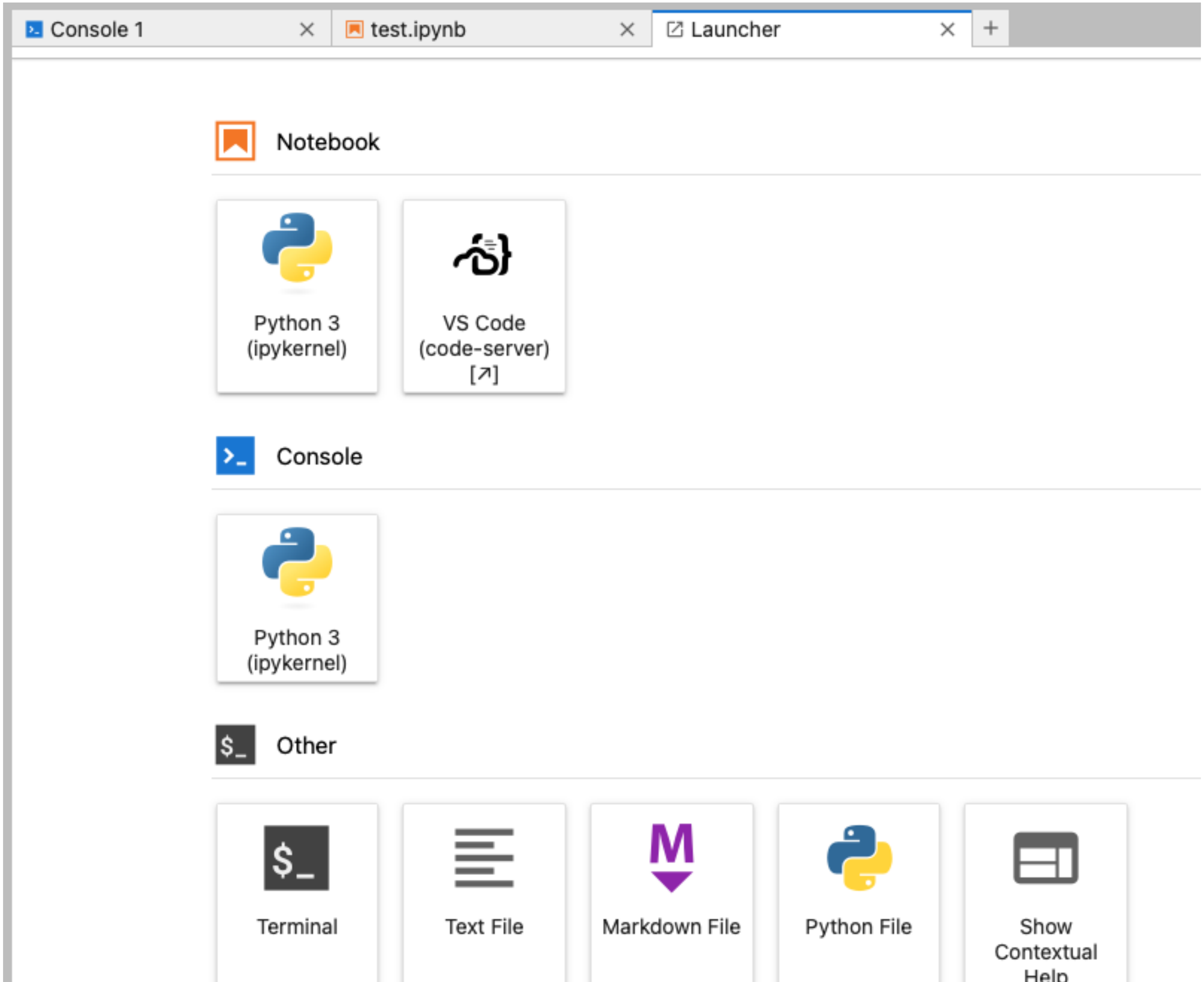
Select Kernel



Get Jupyter Interface

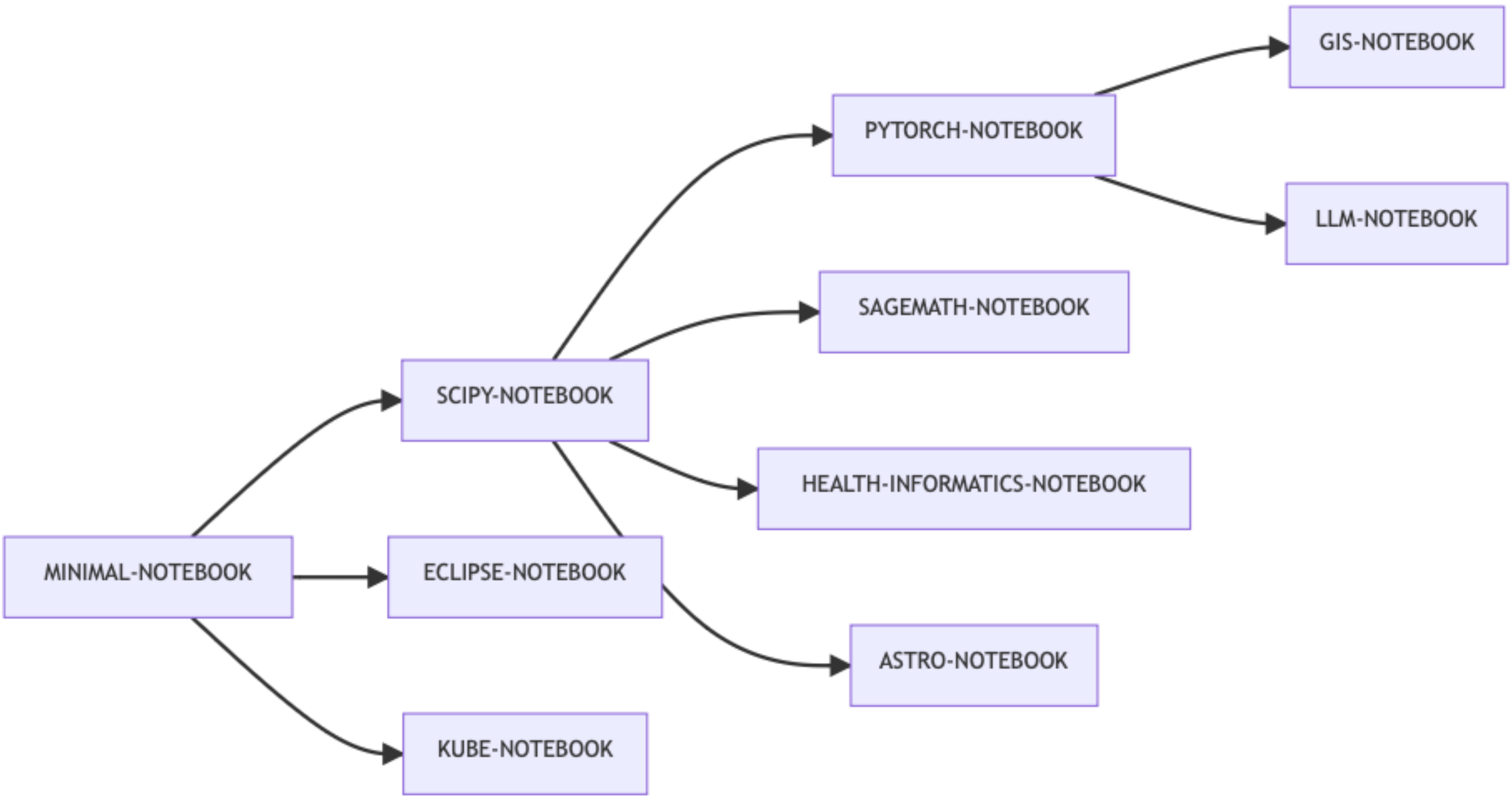


Launcher



Console 1
test.ipynb
Launcher
Notebook
Python 3 (ipykernel)
VS Code (code-server) [↗]
Console
Python 3 (ipykernel)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help

Large Language Model Notebook



minimal-notebook

- Ubuntu 22.04.4
- JupyterLab 4.2.4
- Python 3.11.9
- Git 2.34.1
- vi
- nano
- wget
- curl
- unzip
- tzdata

SciPy Notebook

altair, beautifulsoup4, bokeh, bottleneck, cloudpickle, conda-forge::blas=*=openblas, cython, dask, dill, h5py, jupyterlab-git, matplotlib-base, numba, numexpr, openpyxl, pandas, patsy, protobuf, pytables, scikit-image, scikit-learn, scipy, seaborn, sqlalchemy, statsmodel, sympy, widgetsnbextension, xlrd packages

ipympl and ipywidgets for interactive visualizations and plots in Python notebooks

Facets for visualizing machine learning datasets

PyTorch Notebook

pytorch machine learning library

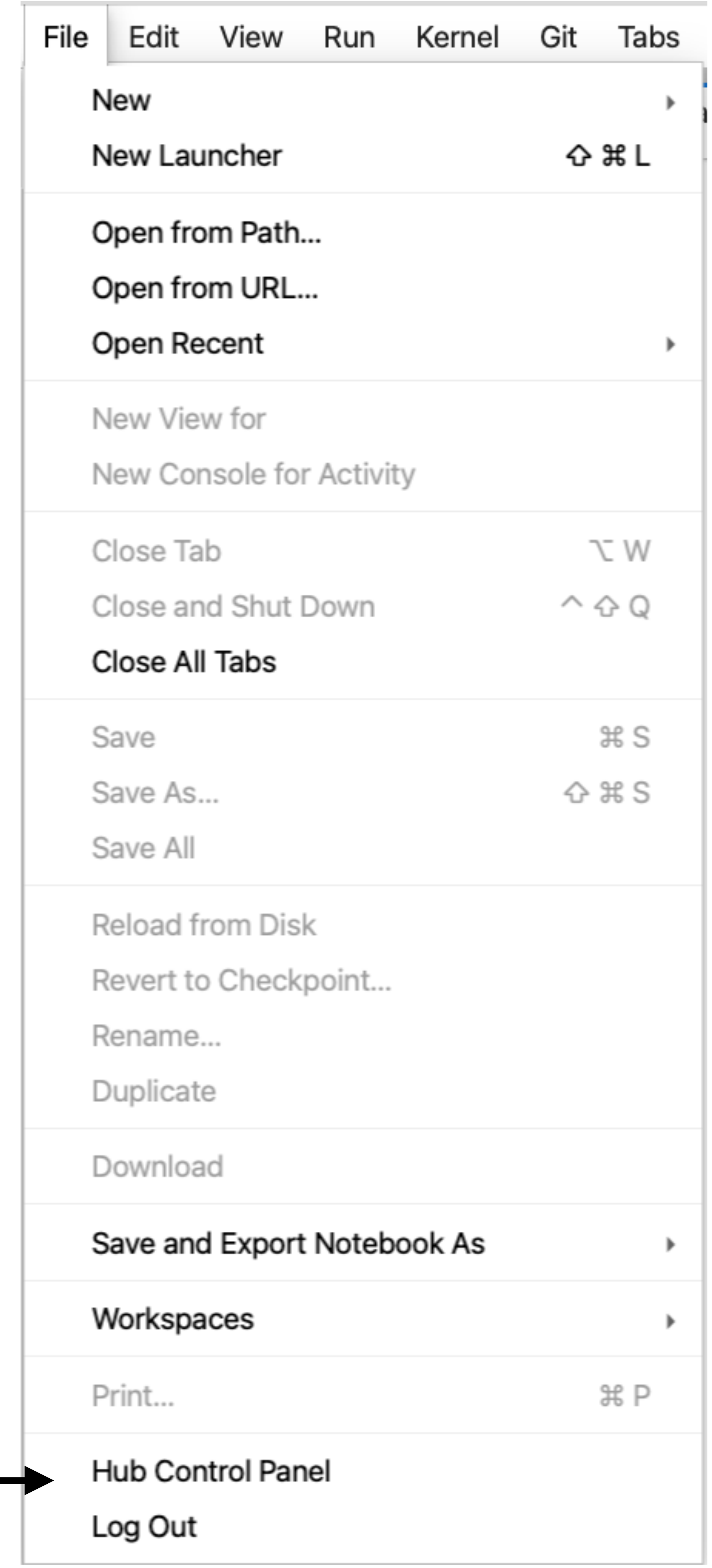
torch, torchaudio and torchvision

Large Language Model Notebook

rclone
FastChat
Ollama
VS Code Server
Jupyter AI
bitsandbytes
transformers
peft
accelerate
trl
ollama-python
openai
pyaudio
portaudio
cuda-nvcc

deepspeed
langchain
huggingface_hub
auto_gptq
autoawq
xformers
dask-kubernetes
chromadb

Manually Stopping the Server



Warning

I have not been able to run an existing model on the cluster

There is some problem with the version of the libraries

Back to Tokens

Goodbye , and thanks for all the fish !

17212, 43571, 11, 326, 11707, 395, 722, 290, 13897, 0

Tokens are just a way to represent works in numbers

They don’t capture the relationship between words

To capture the relationship between tokens, convert them to a vector

Byte Pair Encoding (BPE)

Original Algorithm (for compression)

Find the most common pair of characters
Replace with new symbol
Repeat

Example from Wikipedia

aaabdaaabac

ZabdZabac
Z=aa

ZYdZYac
Y=ab
Z=aa

XdXac
X=ZY
Y=ab
Z=aa

Byte-level BPE

Used by BERT models, GPT-2

When coming across words not in the vocabulary

Convert to UTF-8 and encode pairs of characters

No need for <|unk|>

```
import tiktoken

tokenizer = tiktoken.get_encoding("gpt2")
text = "aaabdaaabac"
integers = tokenizer.encode(text)
for i in integers:
    print(f"{i} -> {tokenizer.decode([i])}")
```

"aaabdaaabac"

7252 -> aa
397 -> ab
6814 -> da
64 -> a
397 -> ab
330 -> ac

"This is a cat"

1212 -> This
318 -> is
257 -> a
3797 -> cat

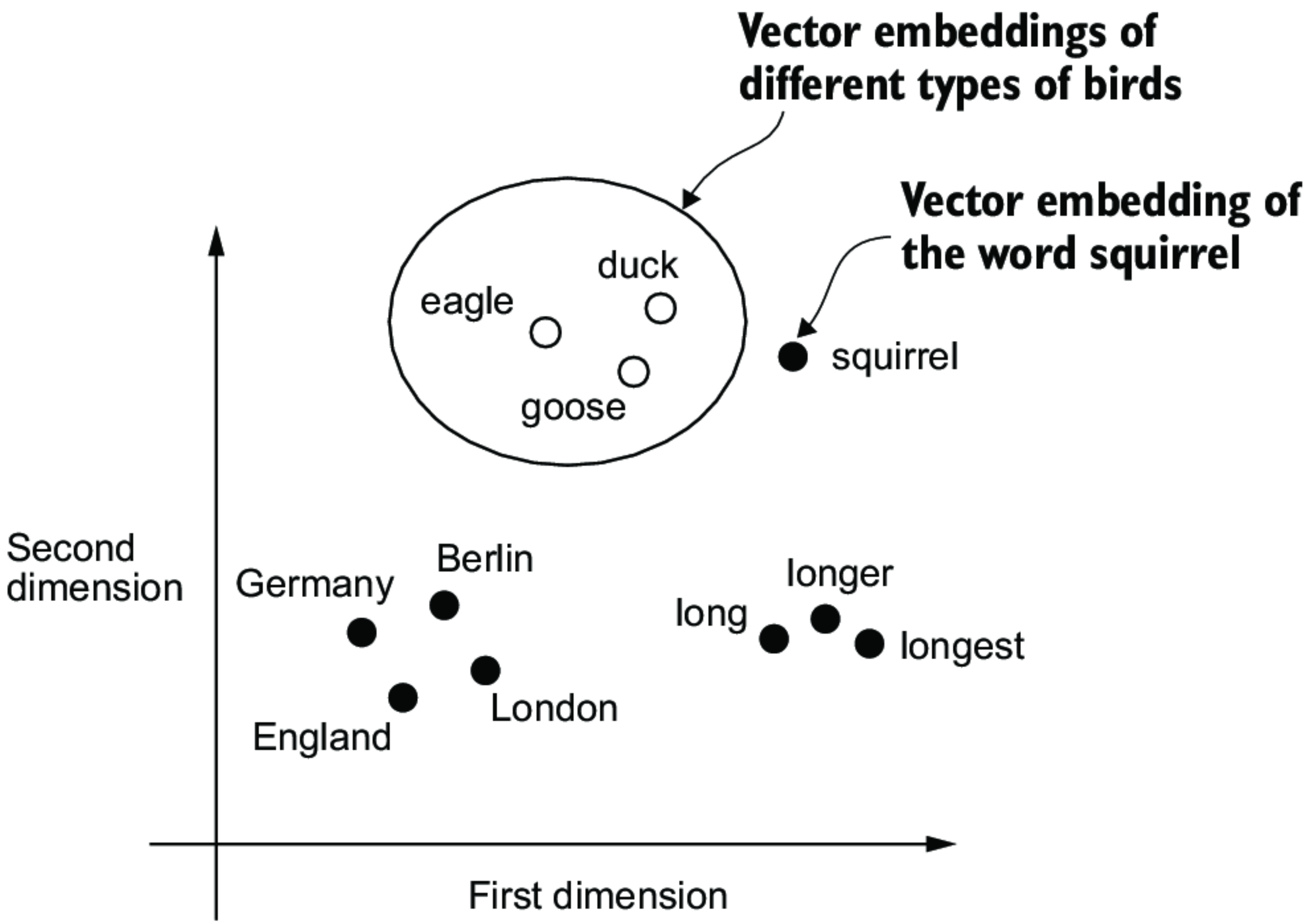
"Thisisacat"

1212 -> This
271 -> is
330 -> ac
265 -> at

Token to Embeddings

Represent a token as a vector in n-space

Related tokens should be close to each other



Type of Embeddings

Token Embeddings

Vector representation (embedding) using a lookup table

Segment Embeddings

Which sentence a token belongs to

Position Embeddings

Position of each token in the sequence

Embedding Space

Higher the dimension of the space

- Captures more information about the relationship between tokens
- Requires more computation

Embedding Size

GPT-2 Models	768 dimensions
GPT-3 (175 parameters)	12,288 dimensions
Bert-base	768 dimensions
Bert-large	1024 dimensions

Map each token to a vector in the space

How to do the Embedding

N number of tokens

D dimension of embedding space (hyperparameter)

Create a matrix (weight matrix) with N rows and D columns

Fill with random values

The K'th row is the embedding (vector) of token ID K

Use training data to modify the weight matrix

Example from the Text

Small values so can see what is going on

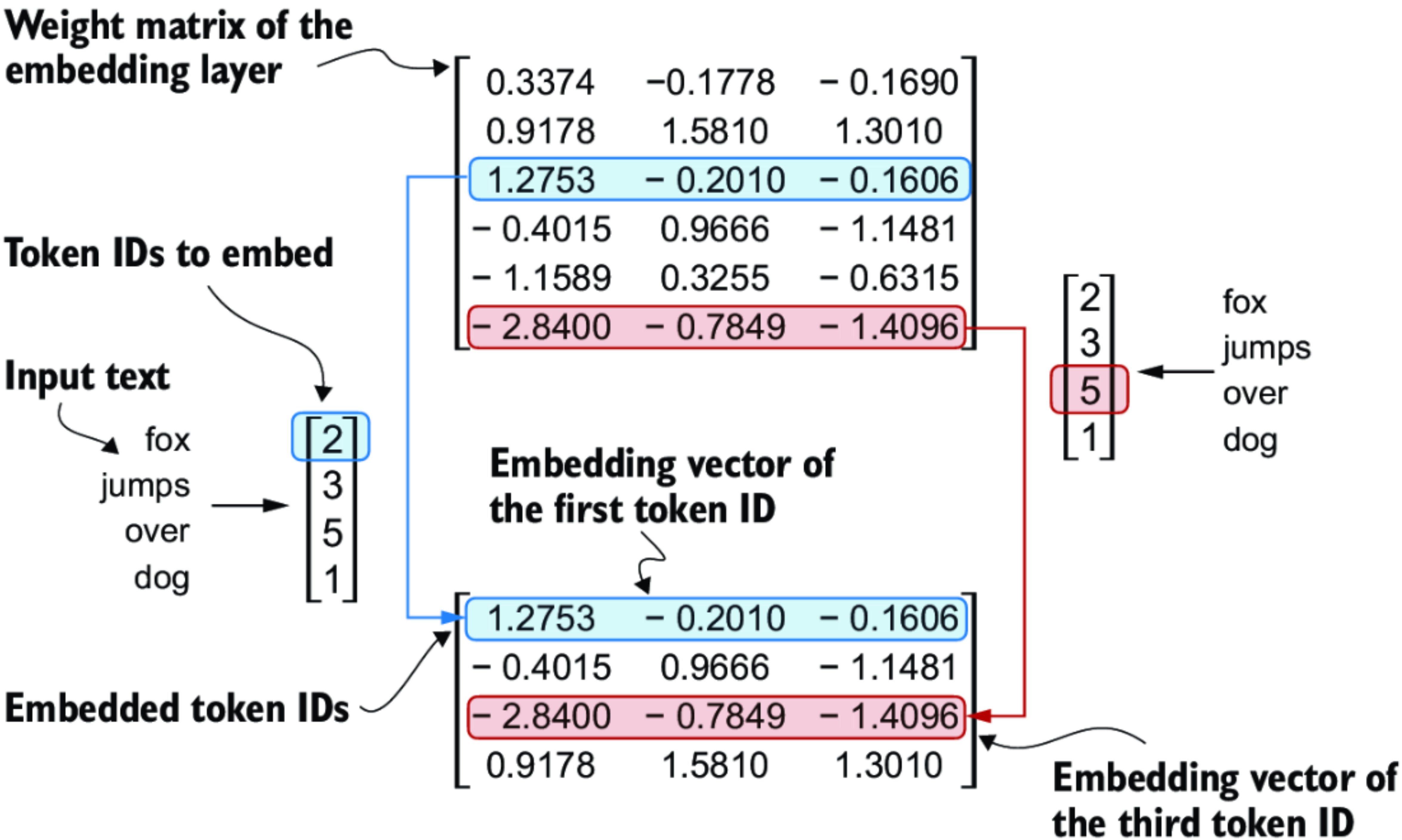
Number of tokens = 6

Dimension of embedding space = 3

Input text

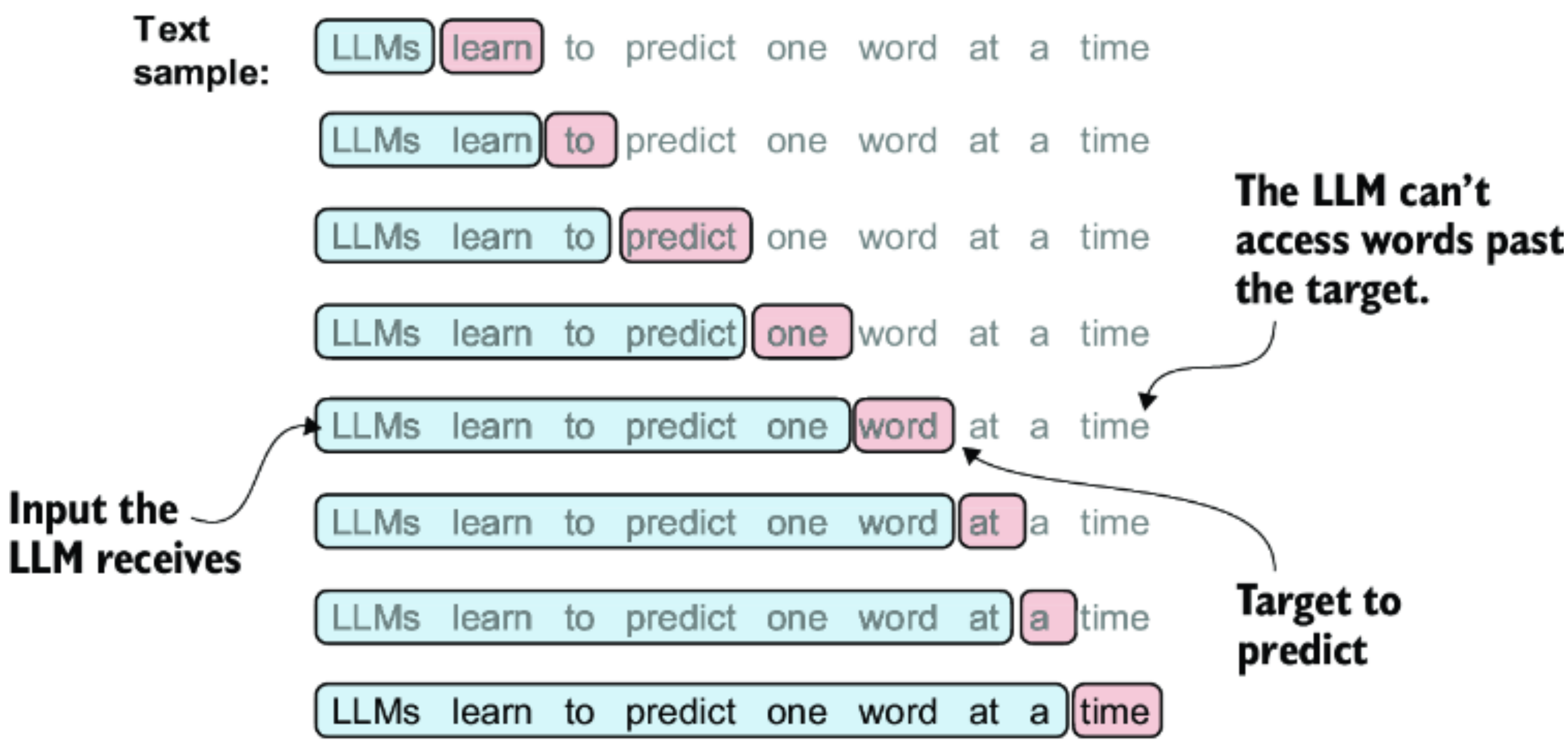
Fox jumps over dog

Example from the Text

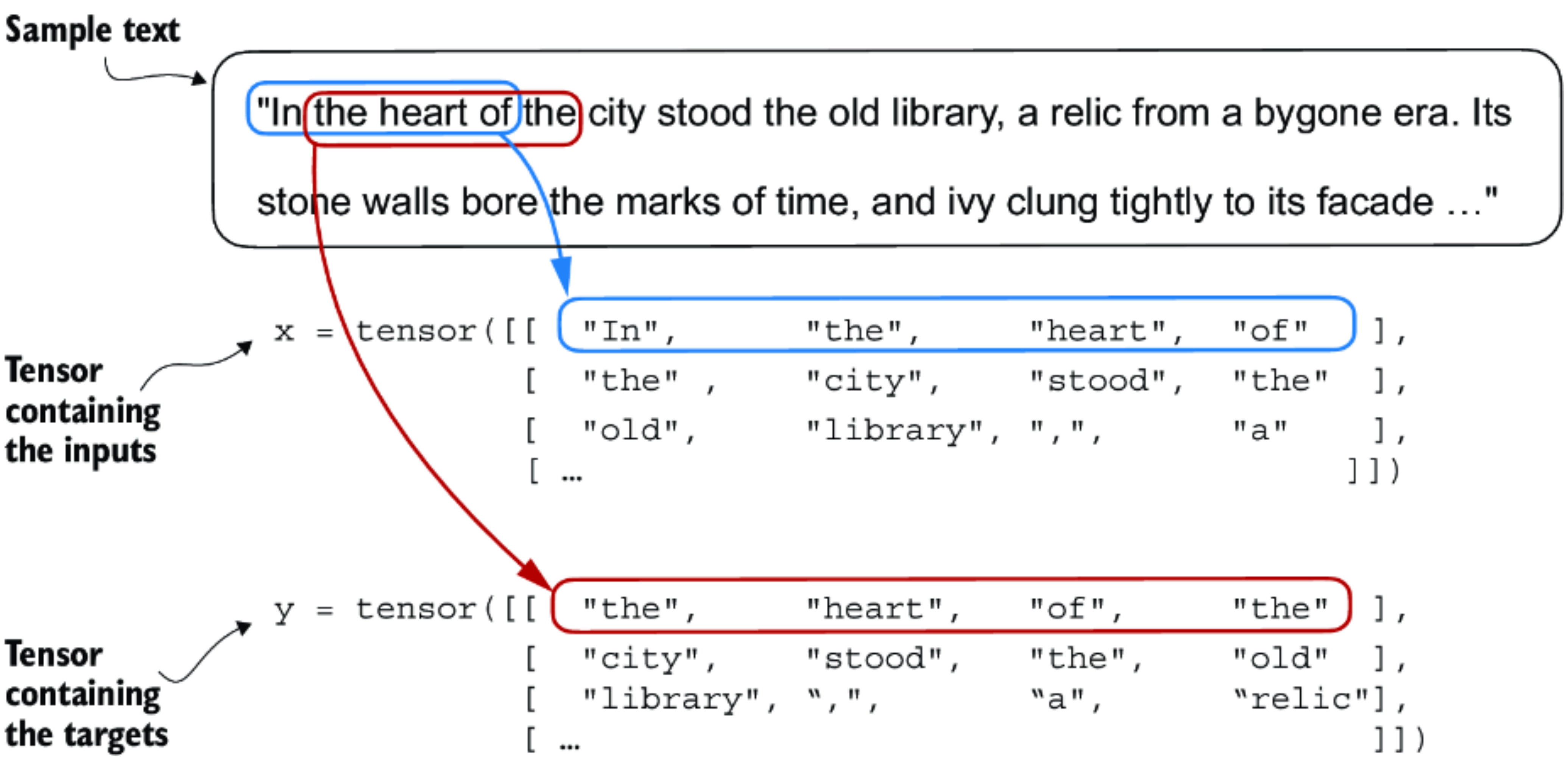


LLM Predict the next Word

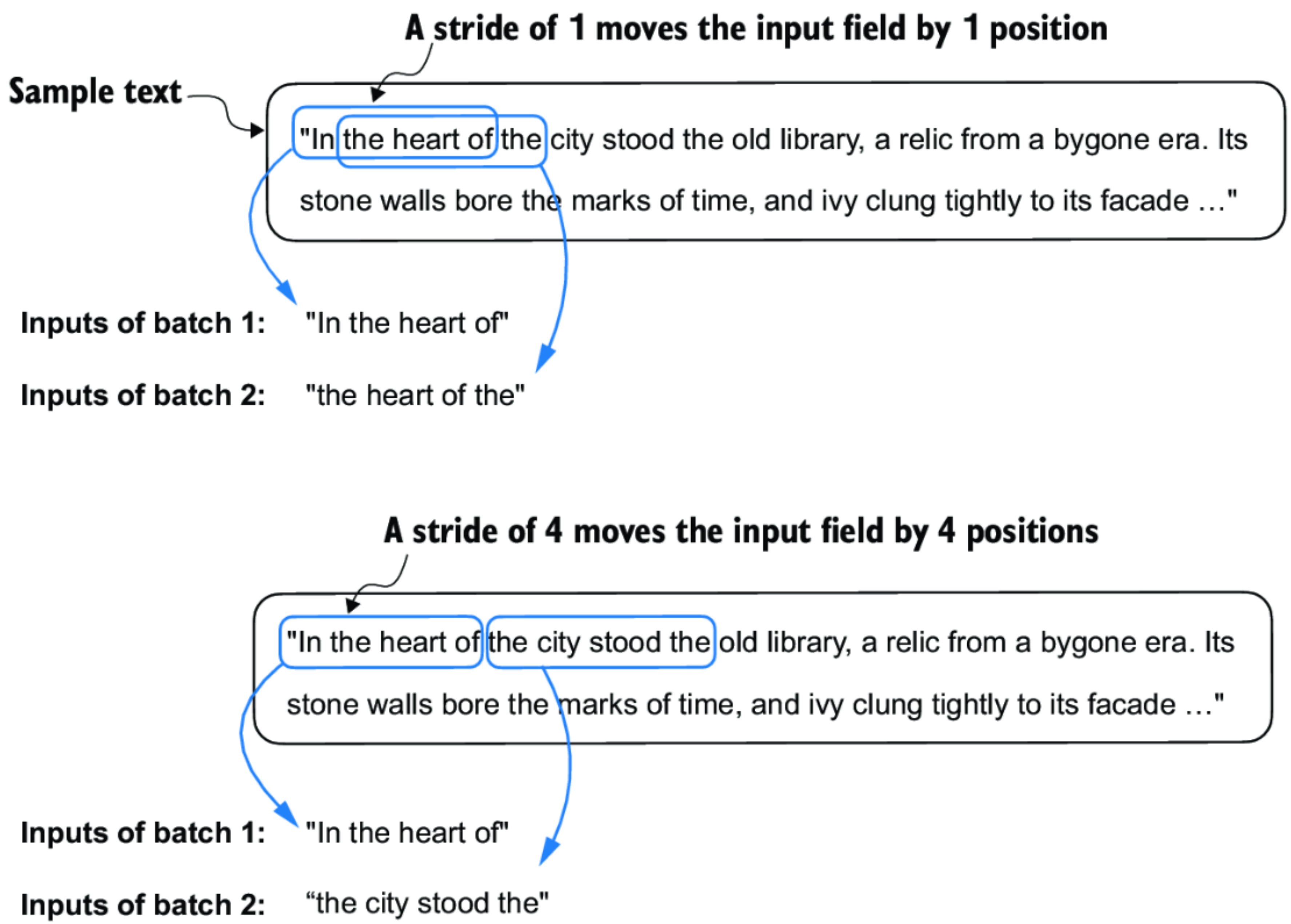
Training



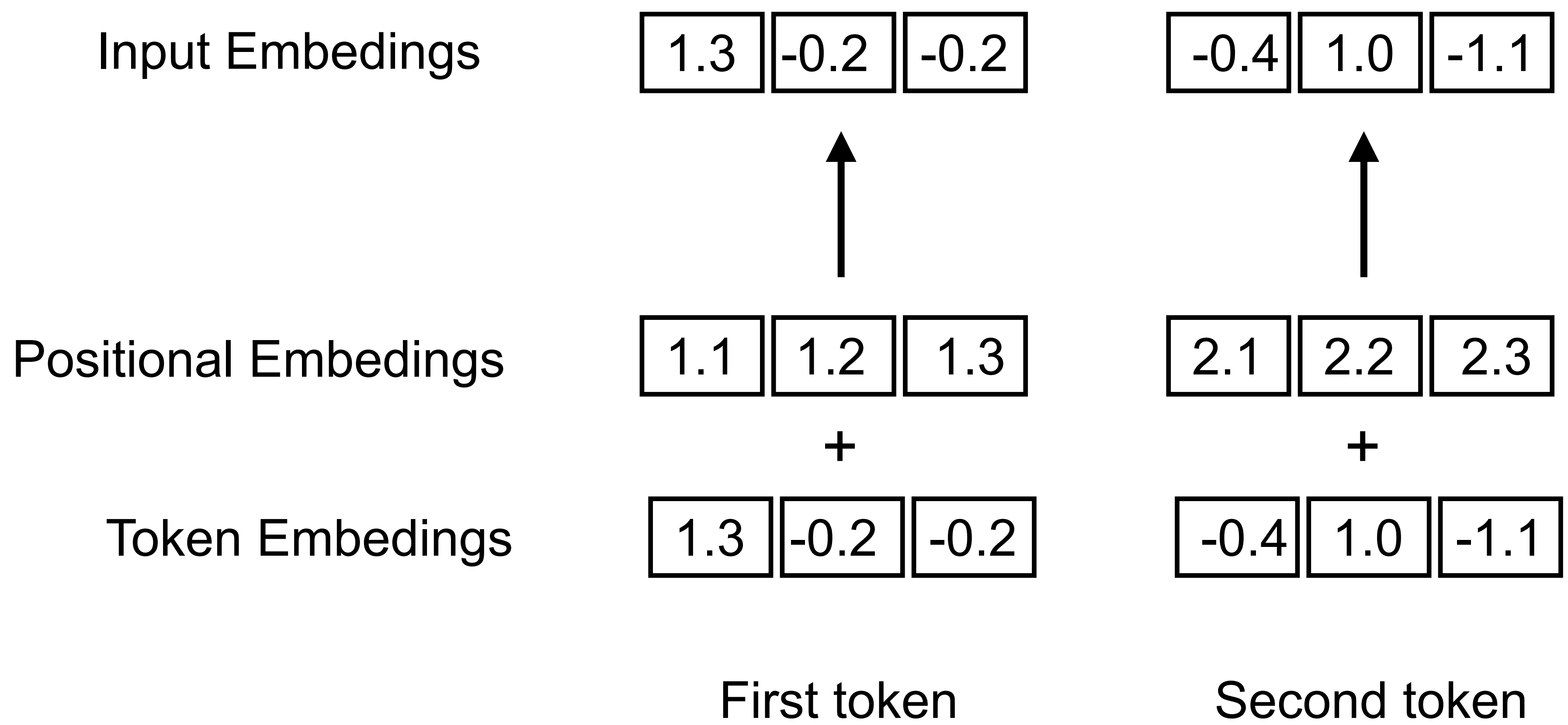
Training


Sample text
"In the heart of the city stood the old library, a relic from a bygone era. Its
stone walls bore the marks of time, and ivy clung tightly to its facade …"
x = tensor([["In", "the", "heart", "of"],
["the" , "city", "stood", "the"],
["old", "library", ",", "a"],
[…]])
Tensor containing the inputs
y = tensor([["the", "heart", "of", "the"],
["city", "stood", "the", "old"],
["library", ",", "a", "relic"],
[…]])
Tensor containing the targets

Stride, Window Size, Context



Adding Absolute Positional Embeddings



GPT uses absolute positional embeddings optimized in training

Positional Embeddings - Sinusoidal

Consider the sentence: "The cat sat on the mat."

Position 1 ("The"): [sin(1/10000^(0/5)), cos(1/10000^(0/5)), sin(1/10000^(2/5)), cos(1/10000^(2/5))]
Position 2 ("cat"): [sin(2/10000^(0/5)), cos(2/10000^(0/5)), sin(2/10000^(2/5)), cos(2/10000^(2/5))]
Position 3 ("sat"): [sin(3/10000^(0/5)), cos(3/10000^(0/5)), sin(3/10000^(2/5)), cos(3/10000^(2/5))]

Positional Embeddings - Rotary (RoPE)

Combines relative and absolute position

$$f_{\{q,k\}}\left(\boldsymbol{x}_m, m\right) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

“rotate the affine-transformed word embedding vector by the number of angle multiples of its position index”

Relative Positional Embedding

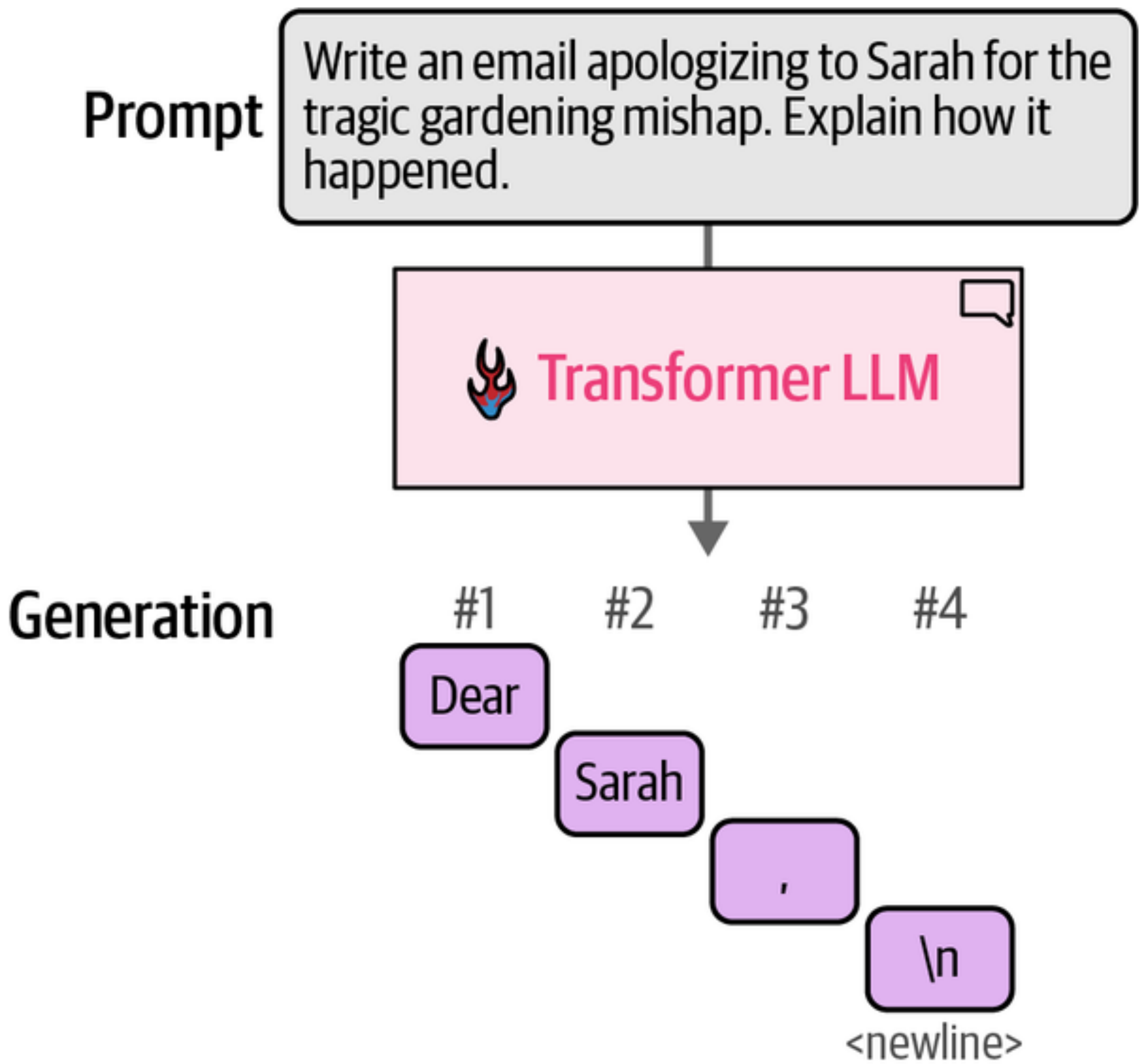
Example: The cat sat on

Distance between tokens

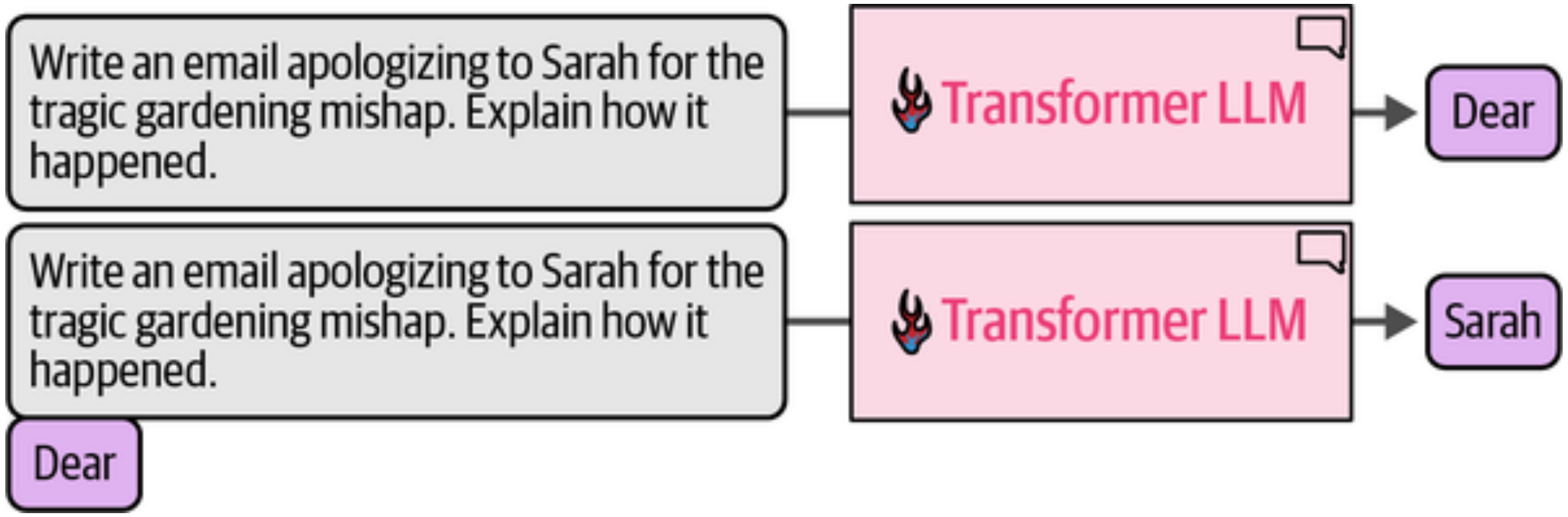
The model learns how far apart tokens are

Token Pair	Relative Distance	Embedding
The - "cat"	1	[0.2, 0.5, -0.1]
cat - "The"	-1	[-0.3, 0.1, 0.4]
cat - "sat"	1	[0.2, 0.5, -0.1]
sat - "cat"	-1	[-0.3, 0.1, 0.4]
The - "sat"	2	[0.8, -0.2, 0.3]
sat - "The"	-2	[-0.7, 0.6, -0.5]
cat - "on"	2	[0.8, -0.2, 0.3]
on - "cat"	-2	[-0.7, 0.6, -0.5]
The - "on"	+3 (clipped to +2)	[0.8, -0.2, 0.3]
on - "The"	-3 (clipped to -2)	[-0.7, 0.6, -0.5]

Big Picture - Responses One Word at a Time



Big Picture - Attach Predicted Word to input

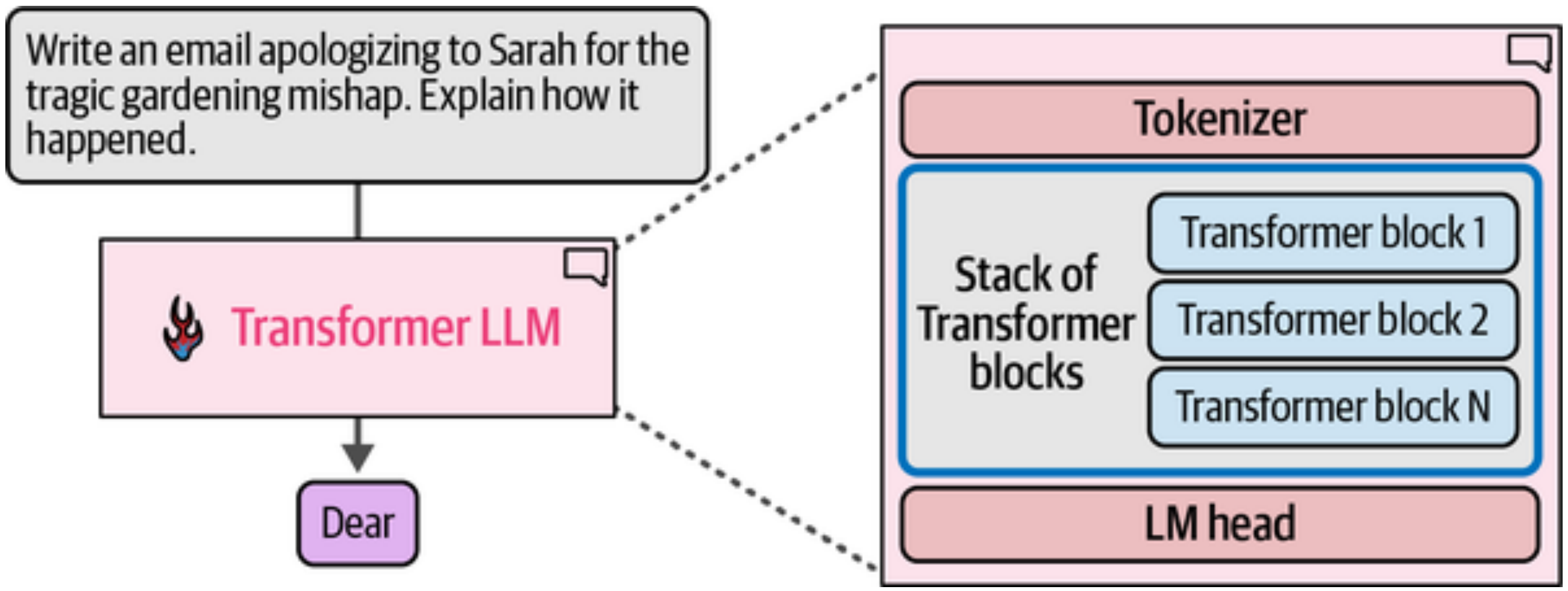


This is not how you see LLMs work

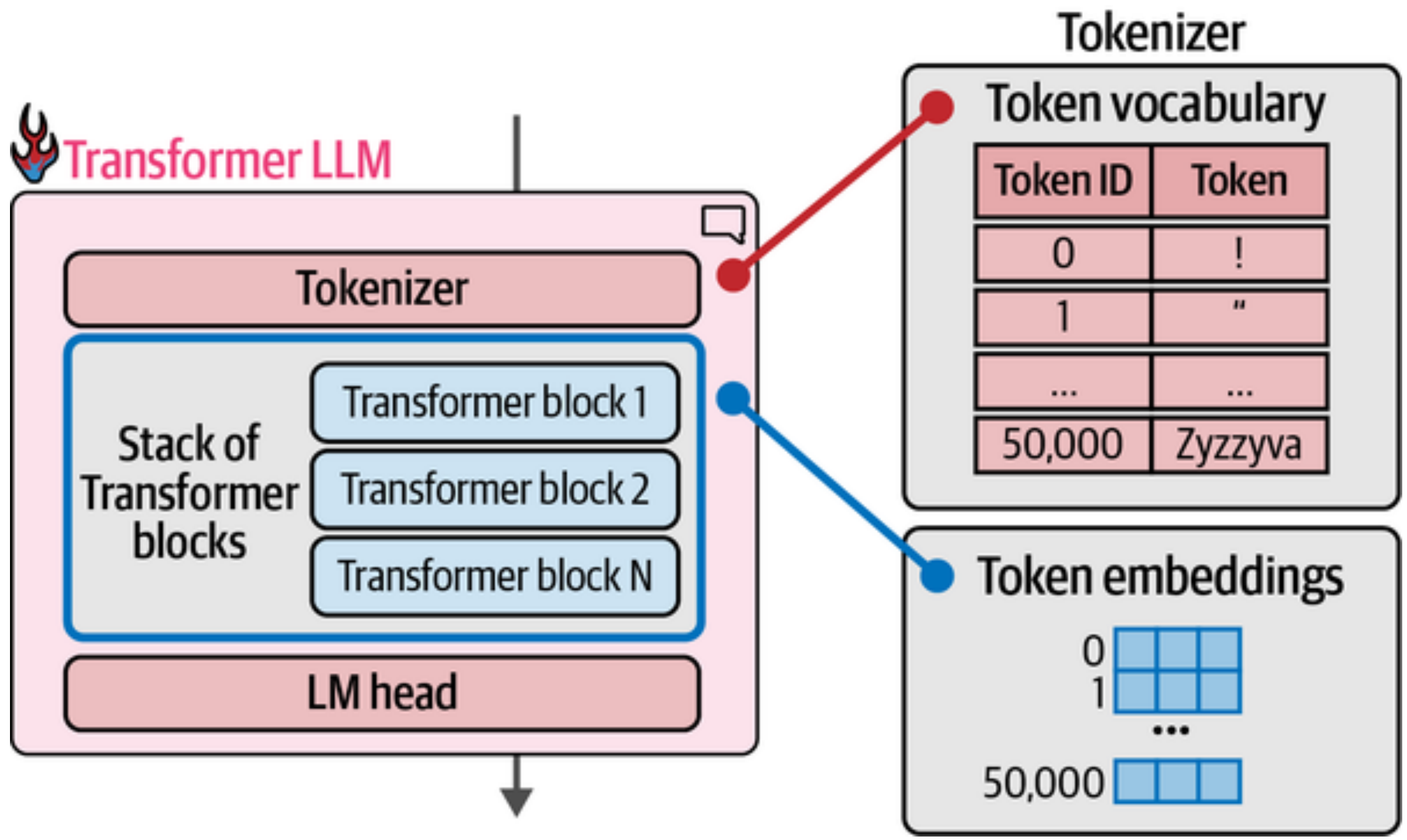
LLMs

- Trained on instruction-tuning and human preference
- To match what we want

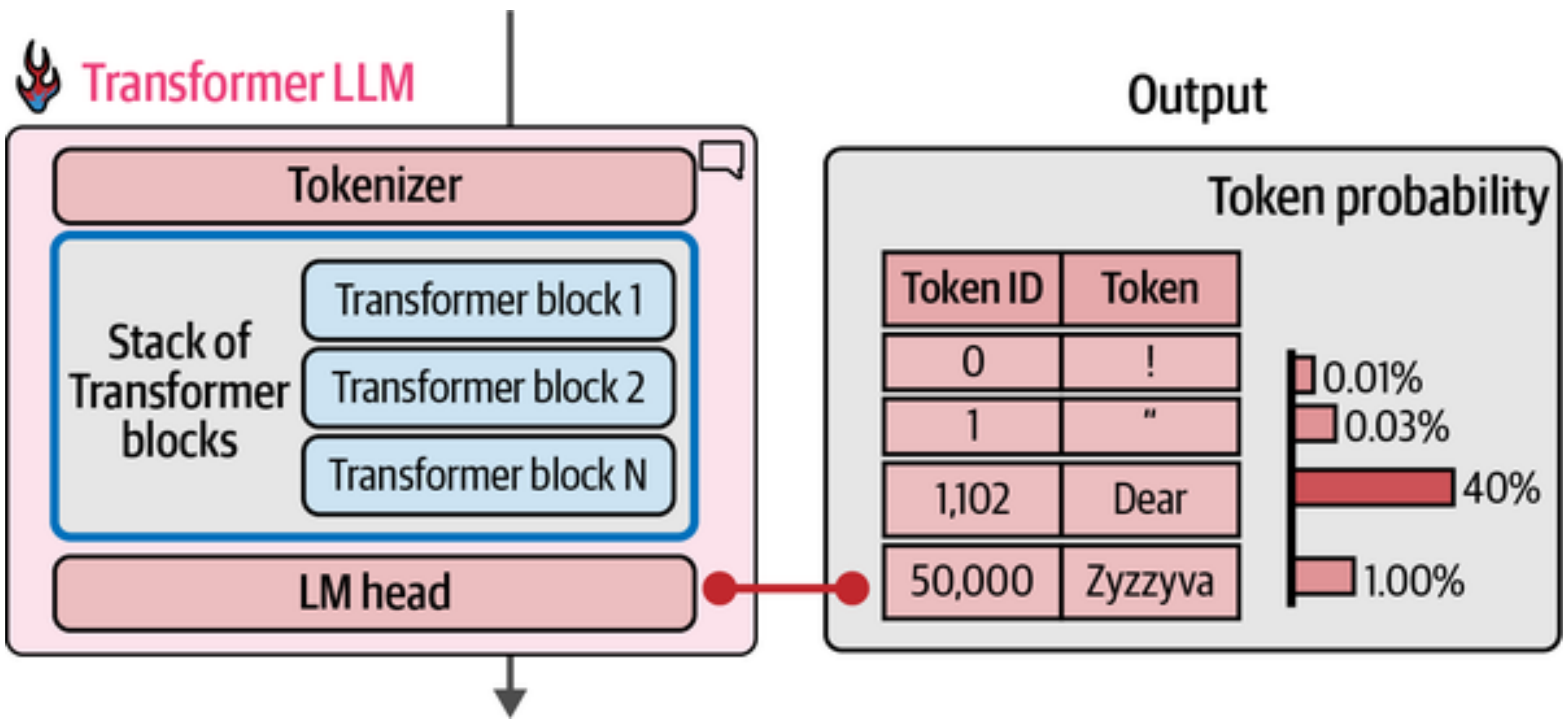
Big Picture - LM Head



Big Picture - LM Head



Big Picture - Output

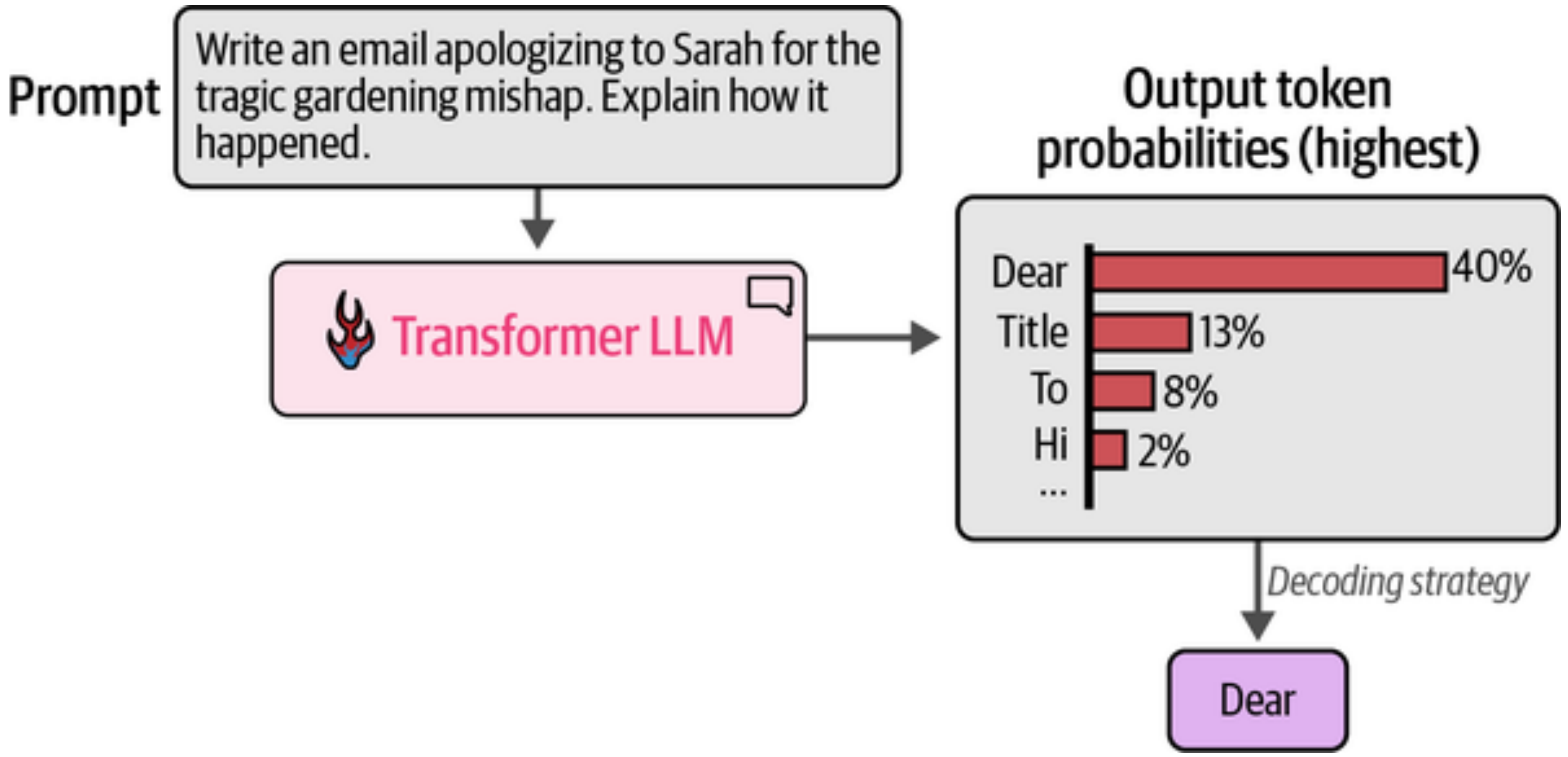


Decoding Strategy

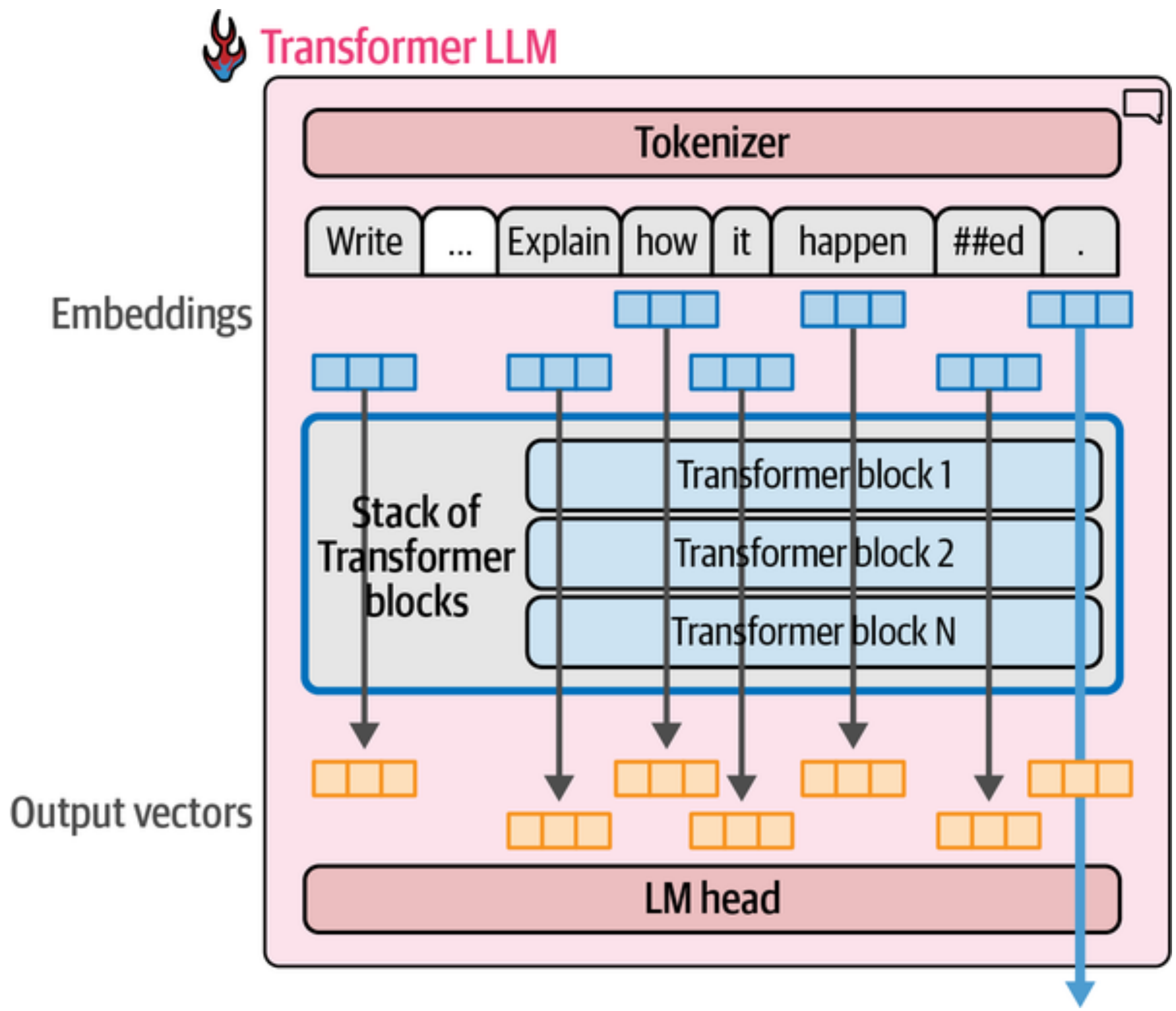
Don't just choose the token with the highest probability

Sample based on probabilities

- Choose Dear 40% of the time
- Choose Title 13% of the time



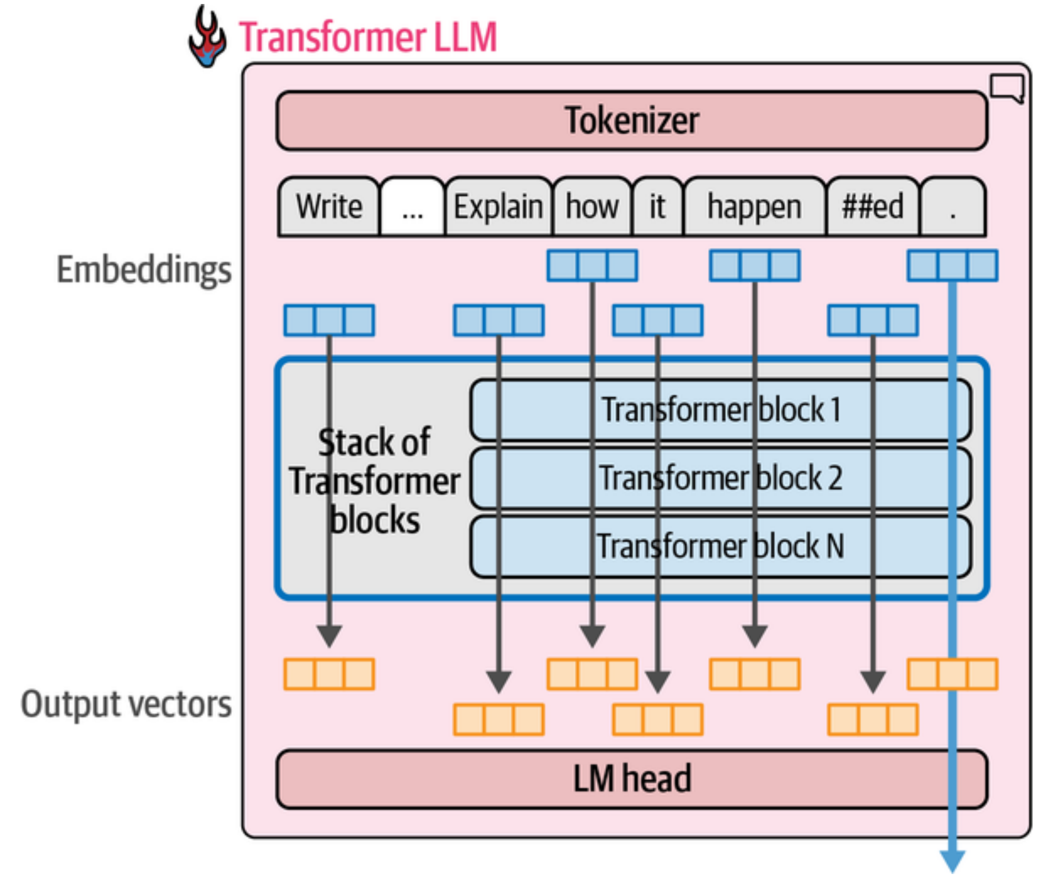
Big Picture - Processing Token in Parallel (sort of)



Big Picture - Caching Keys & Values

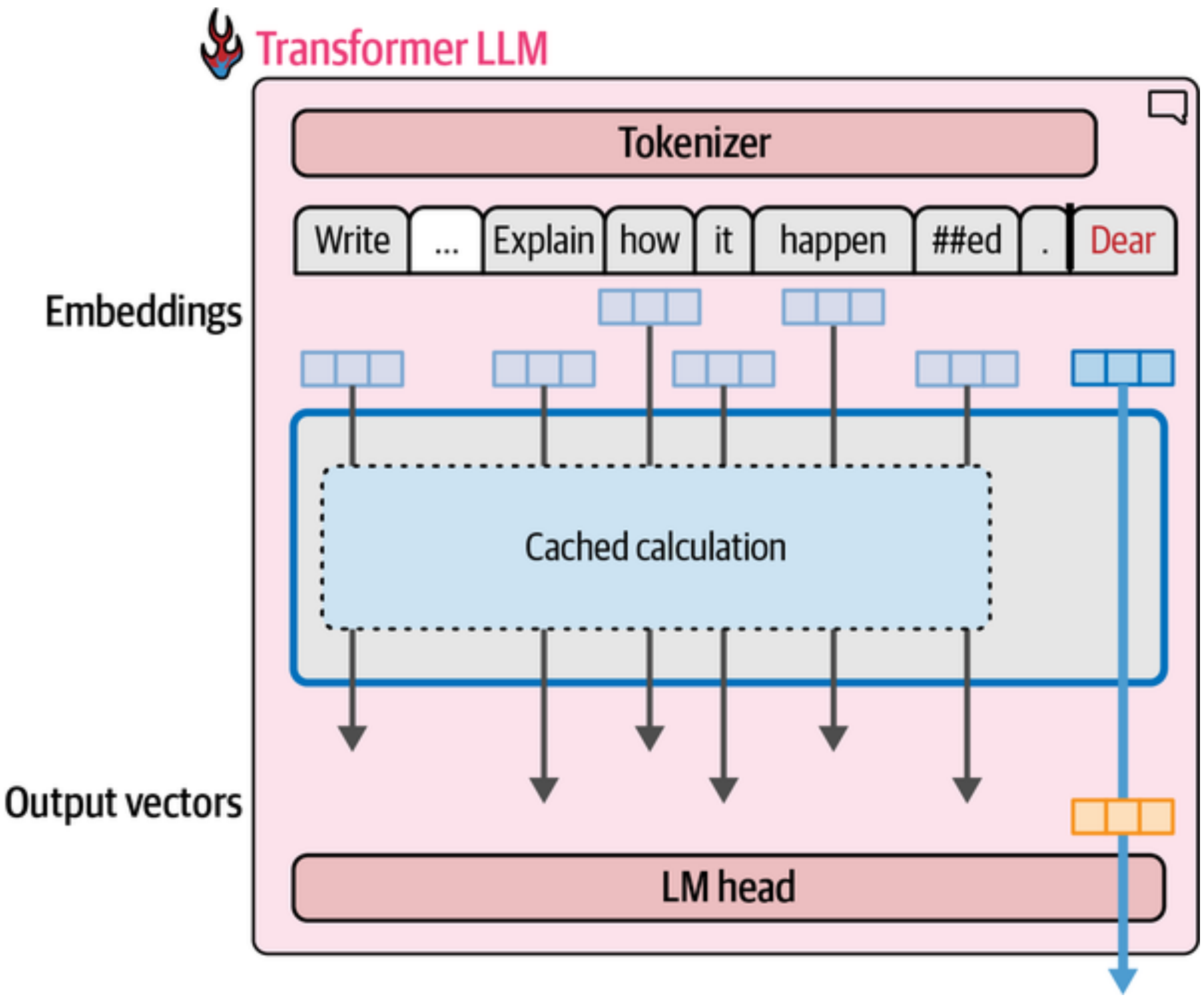
Context Length

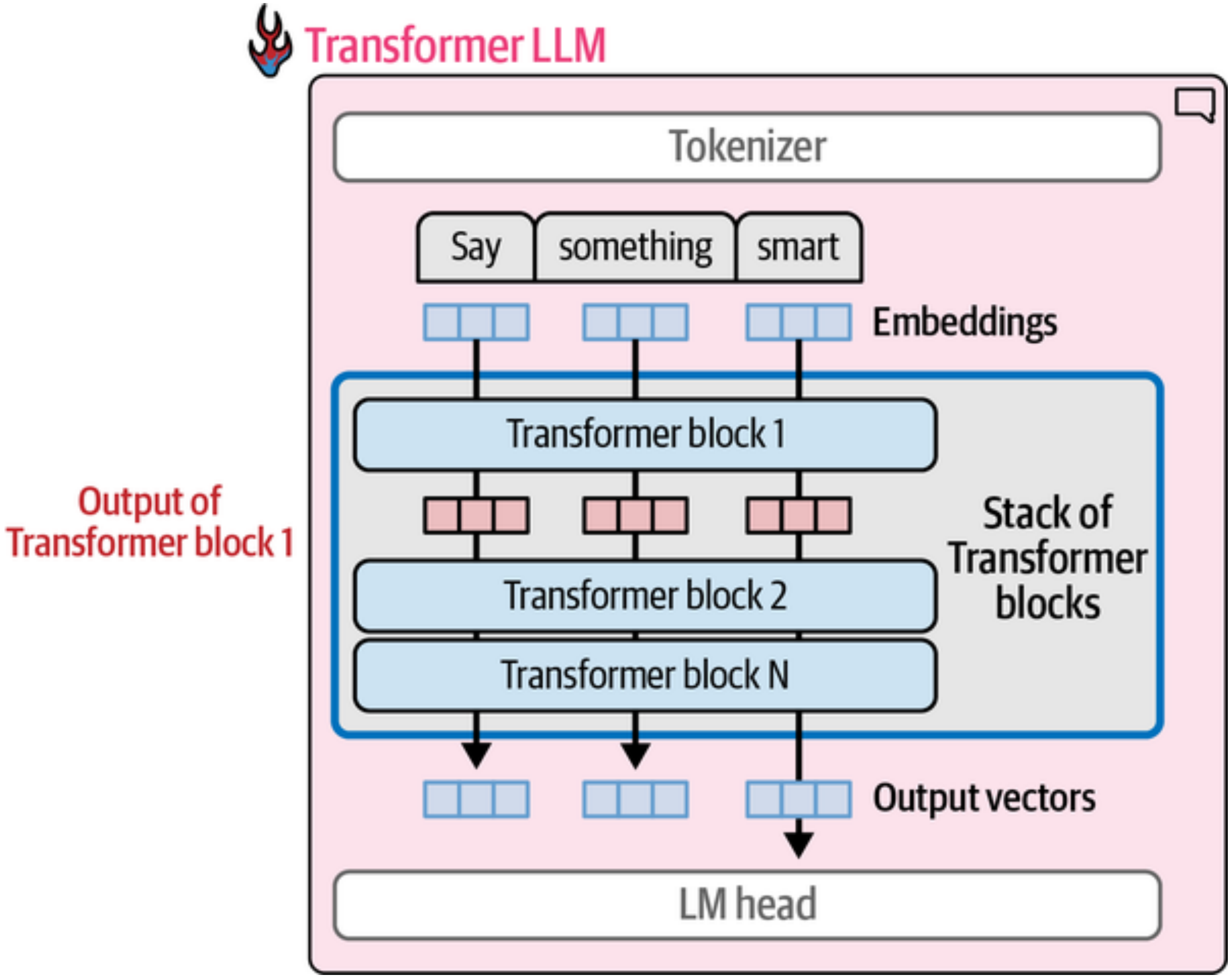
Number of tokens can be processed at once



Model	Context Length
GPT-4o	128k
GPT 3.5	4,095
GPT 4	8,192
Llama 1	2,048

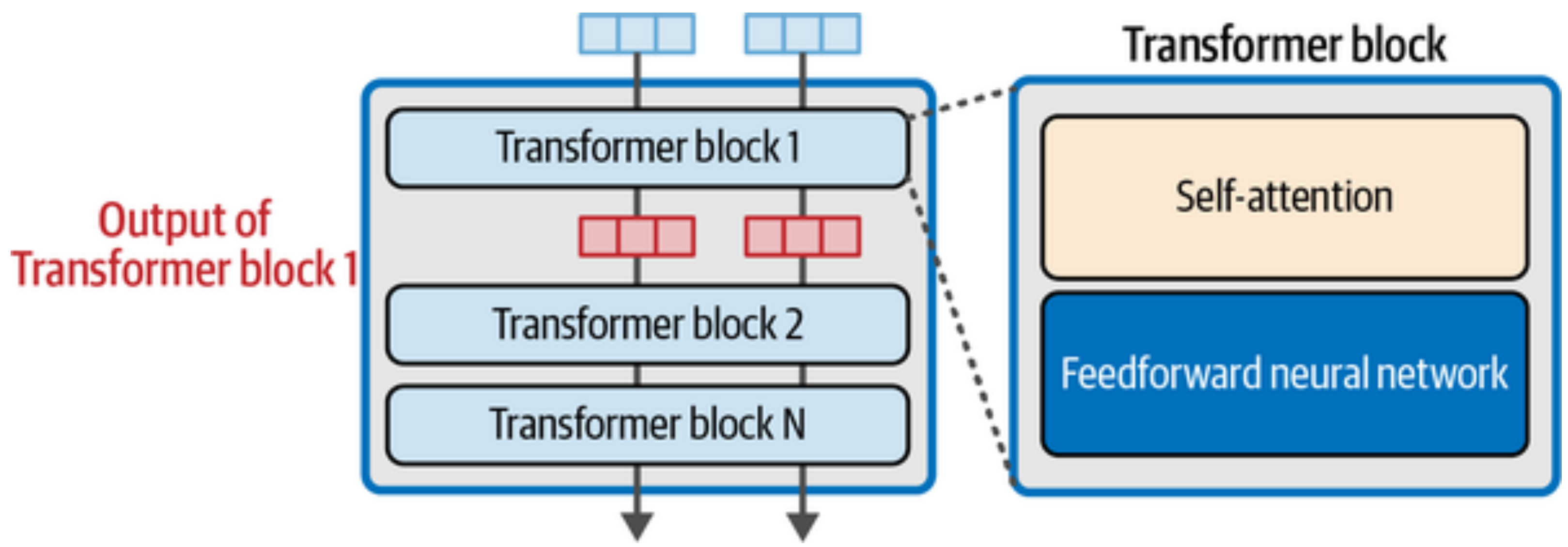
Big Picture - Processing Token in Parallel (sort of)

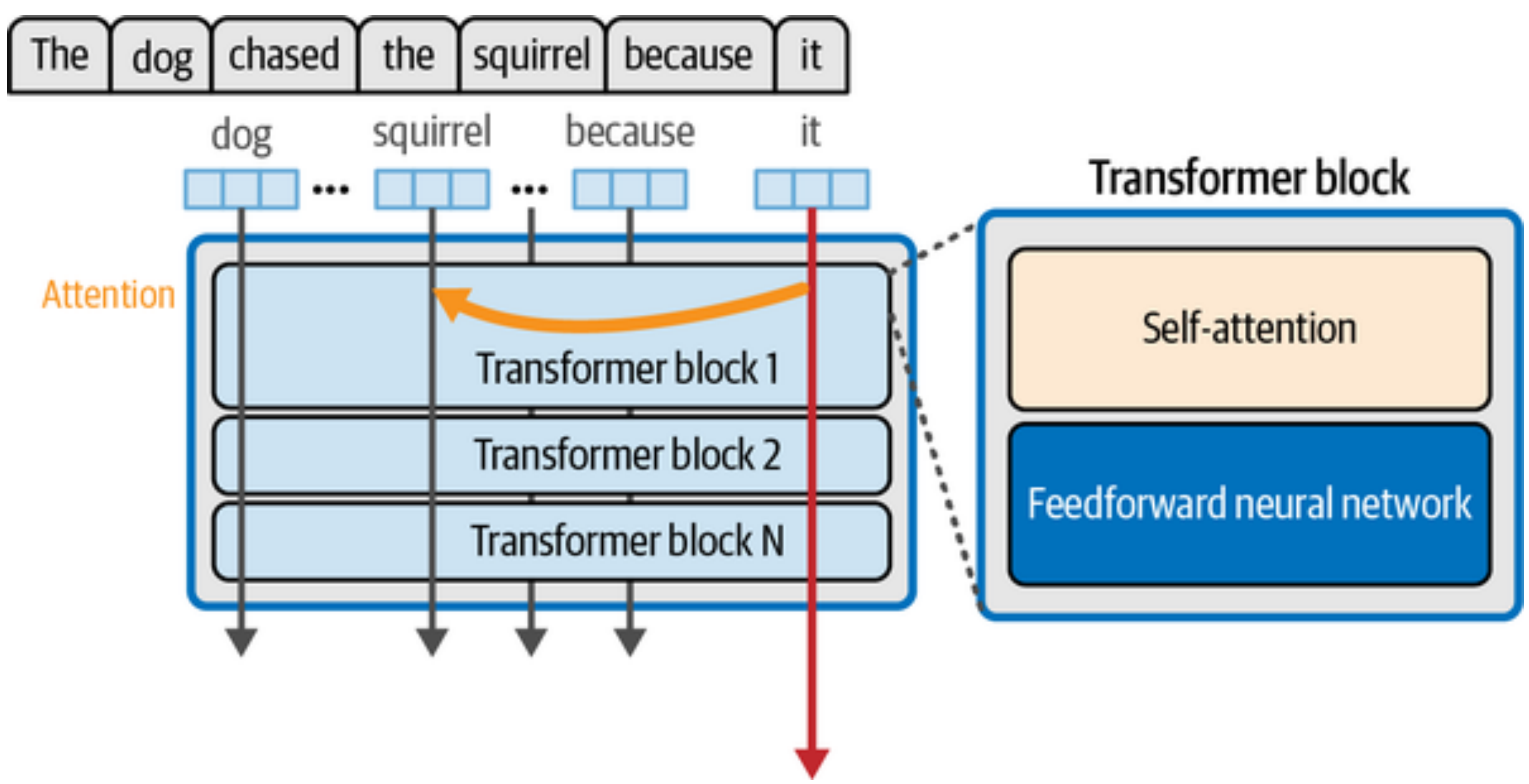




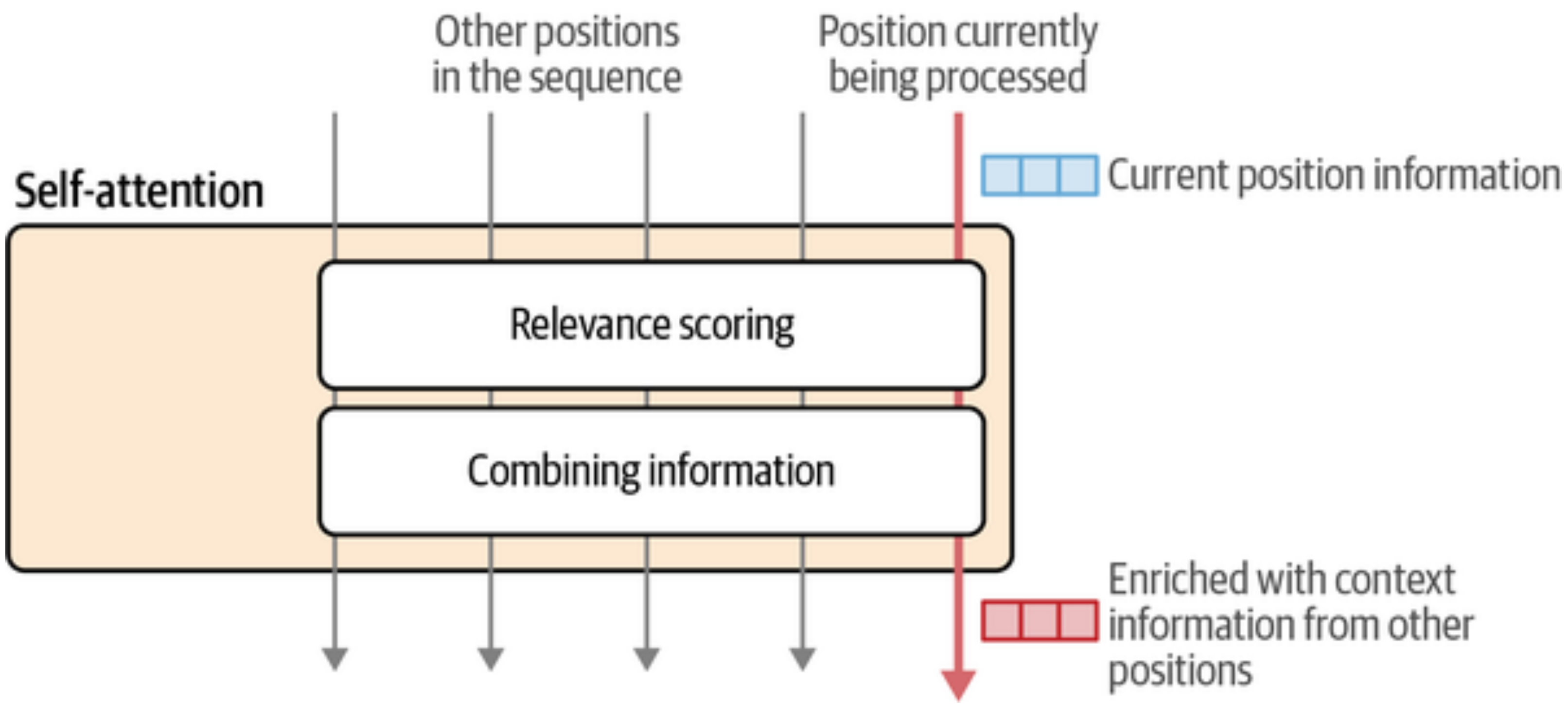
Transformer LLM
Tokenizer
Say
something
smart
Embeddings
Transformer block 1
Output of
Transformer block 1
Stack of
Transformer
blocks
Transformer block 2
Transformer block N
Output vectors
LM head

Block Contents





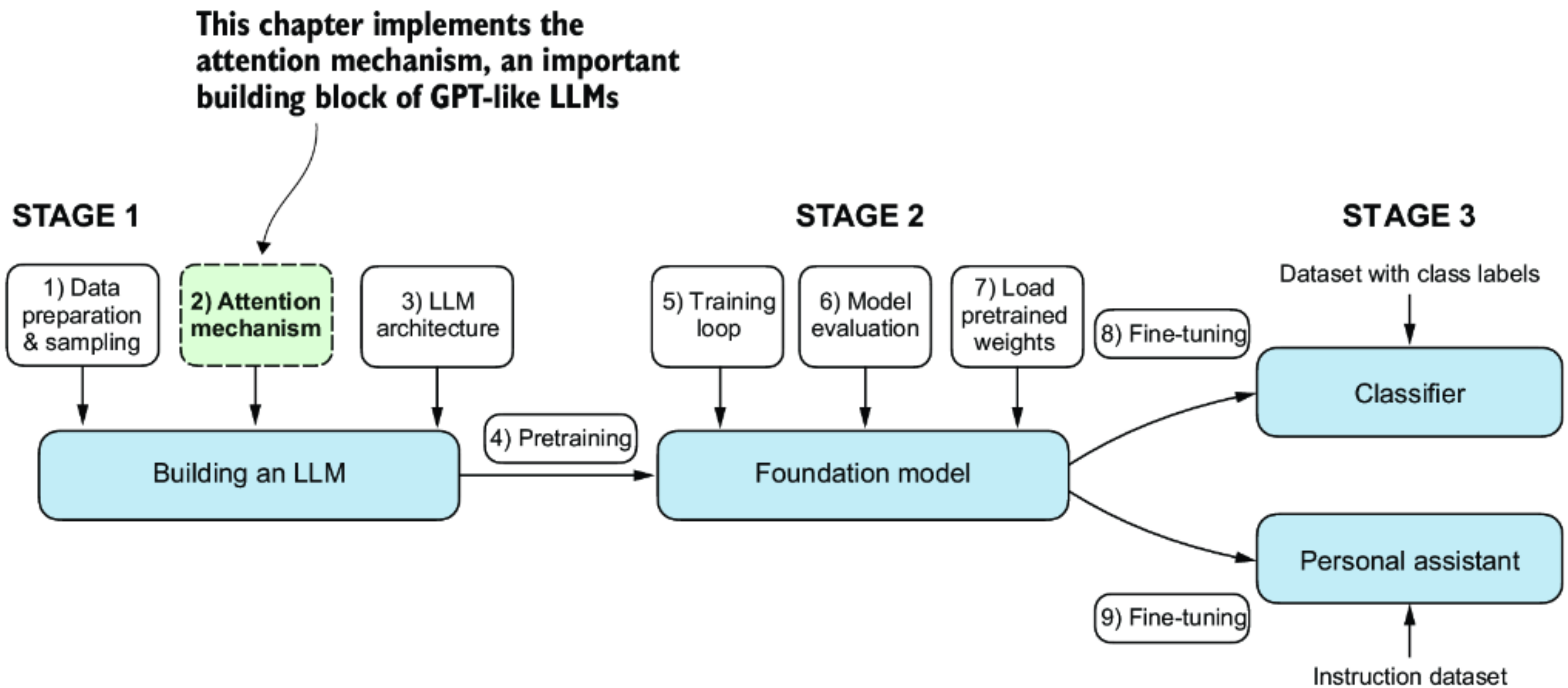
Attention

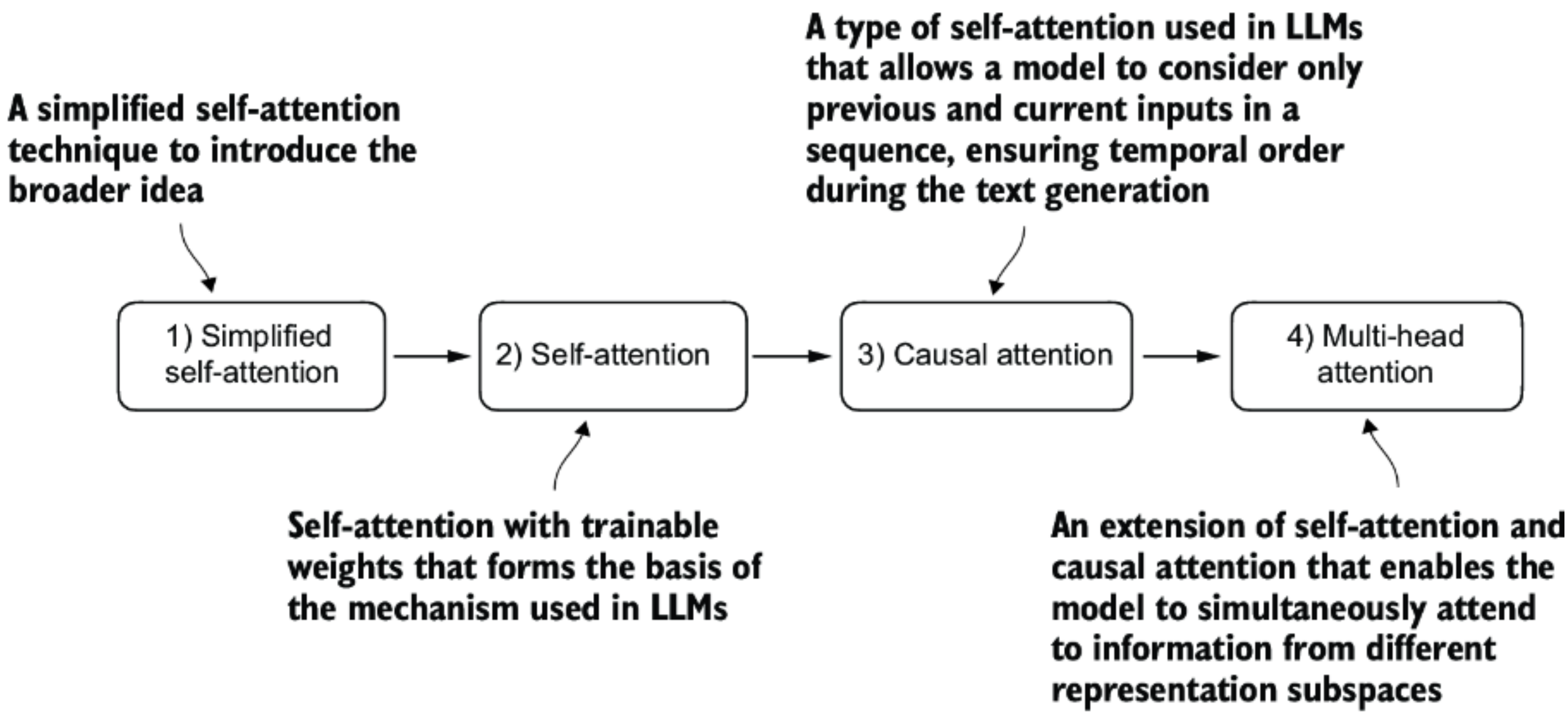


Need a way to compute how relevant each previous token is

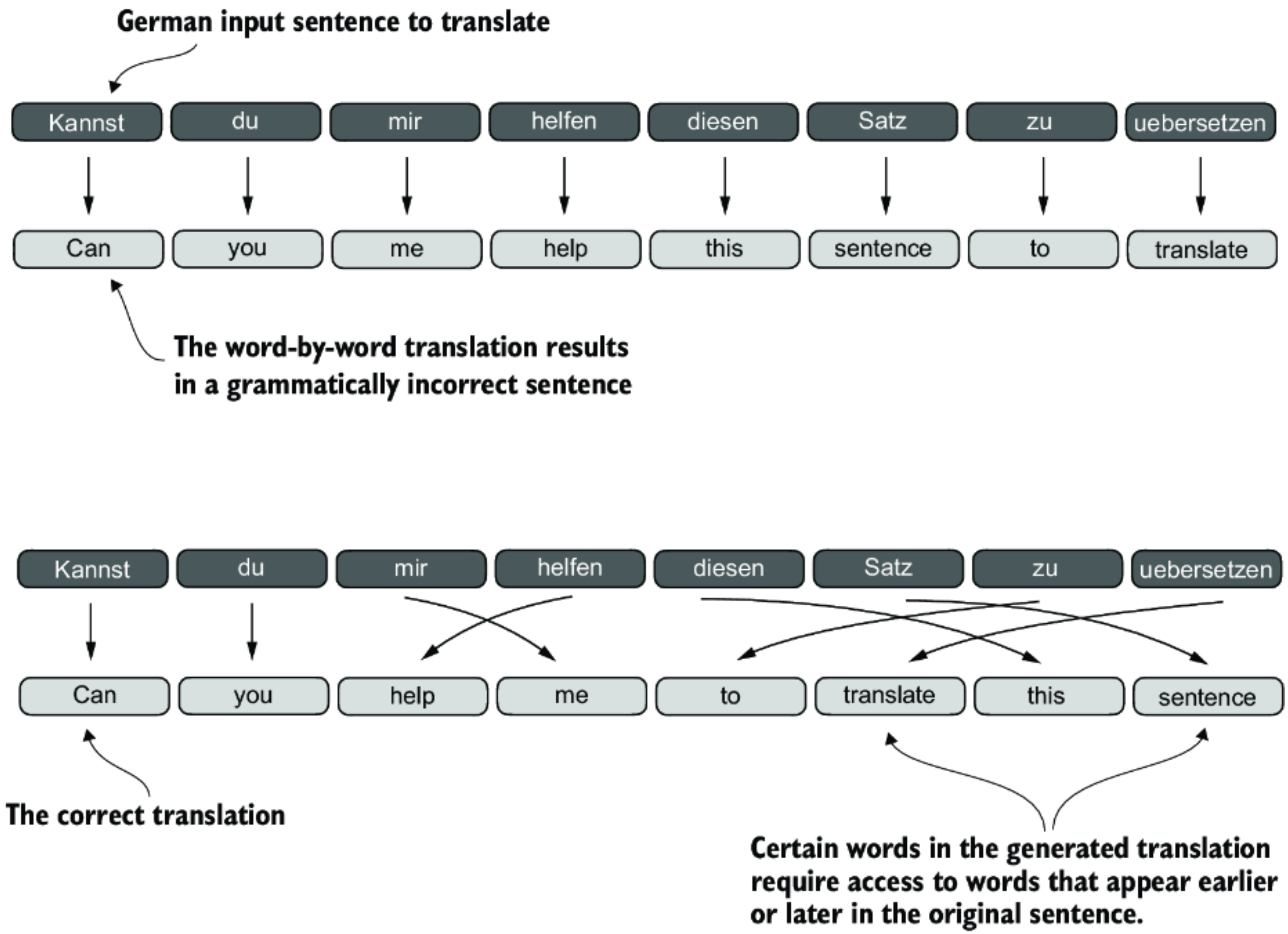
Combine those computations into output vector

Attention - Chapter 3

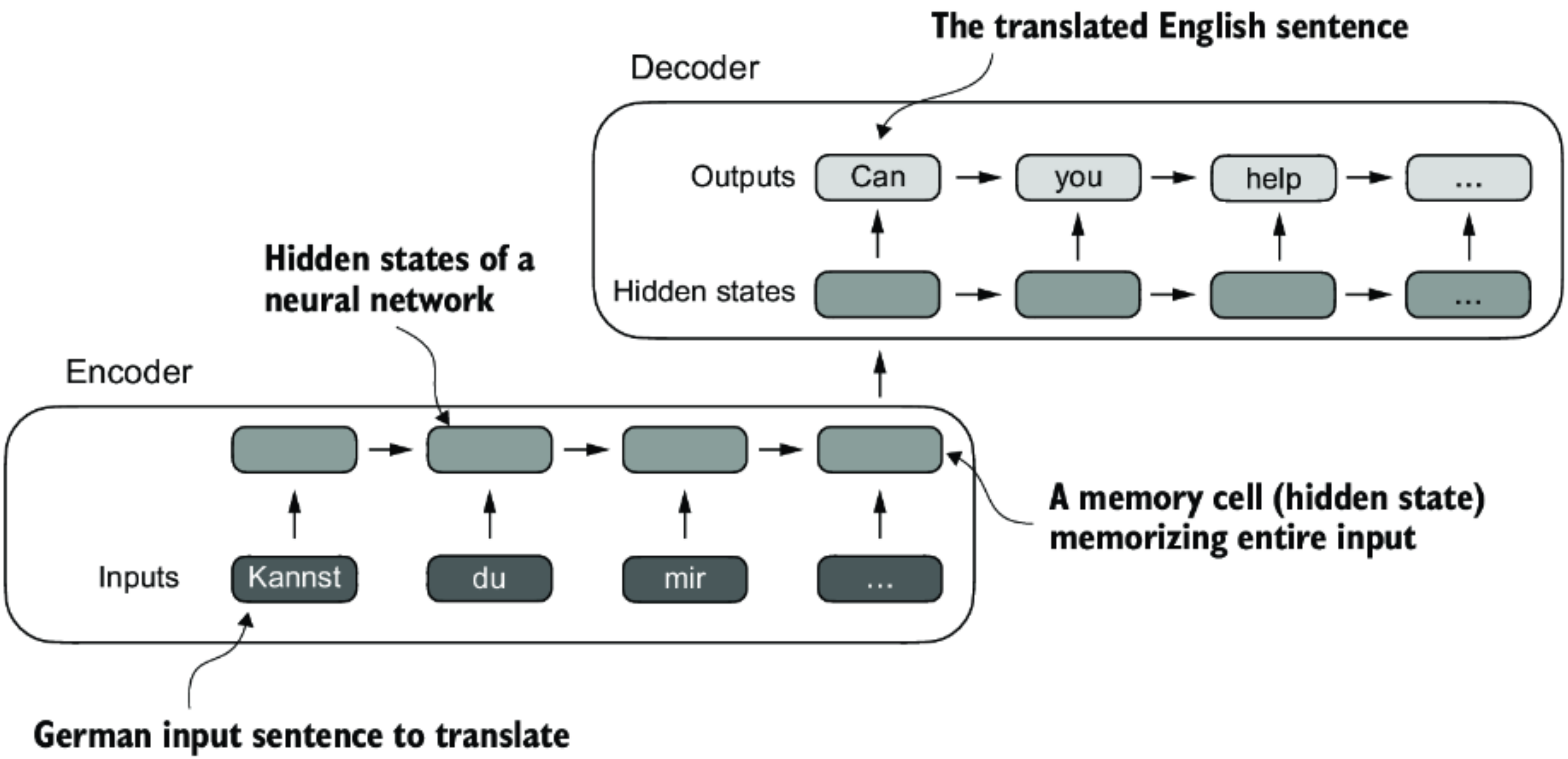



A simplified self-attention technique to introduce the broader idea
A type of self-attention used in LLMs that allows a model to consider only previous and current inputs in a sequence, ensuring temporal order during the text generation
1) Simplified self-attention
2) Self-attention
3) Causal attention
4) Multi-head attention
Self-attention with trainable weights that forms the basis of the mechanism used in LLMs
An extension of self-attention and causal attention that enables the model to simultaneously attend to information from different representation subspaces

Why We need Attention



Encoder - Decoder NN



Bahdanau Attention