

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



DeepLearning.AI



Generative AI & Large Language Models (LLMs)

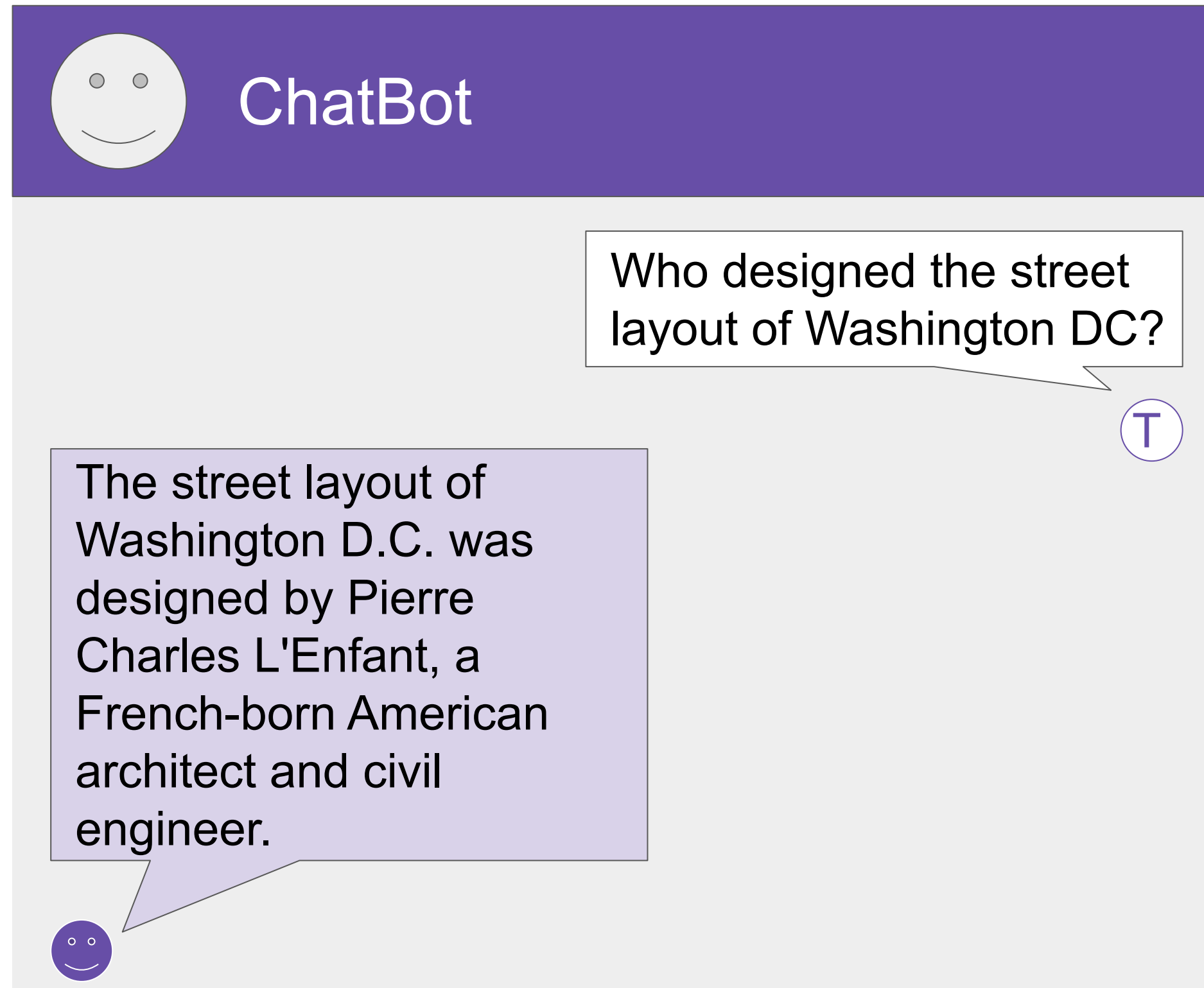
**USE CASES,
PROJECT LIFECYCLE, AND
MODEL PRE-TRAINING**

Generative AI & Large Language Model Use Cases & Model Lifecycle



Generative AI & Large Language Models

Generative AI



The image shows a chatbot interface with a purple header. On the left is a white smiley face icon and the text "ChatBot". The main area is light gray. A white speech bubble on the right contains the question "Who designed the street layout of Washington DC?". Below it is a purple circular icon with a white "T". A larger purple speech bubble on the left contains the answer: "The street layout of Washington D.C. was designed by Pierre Charles L'Enfant, a French-born American architect and civil engineer." At the bottom left of this bubble is a purple smiley face icon.

ChatBot

Who designed the street layout of Washington DC?

The street layout of Washington D.C. was designed by Pierre Charles L'Enfant, a French-born American architect and civil engineer.

Generative AI

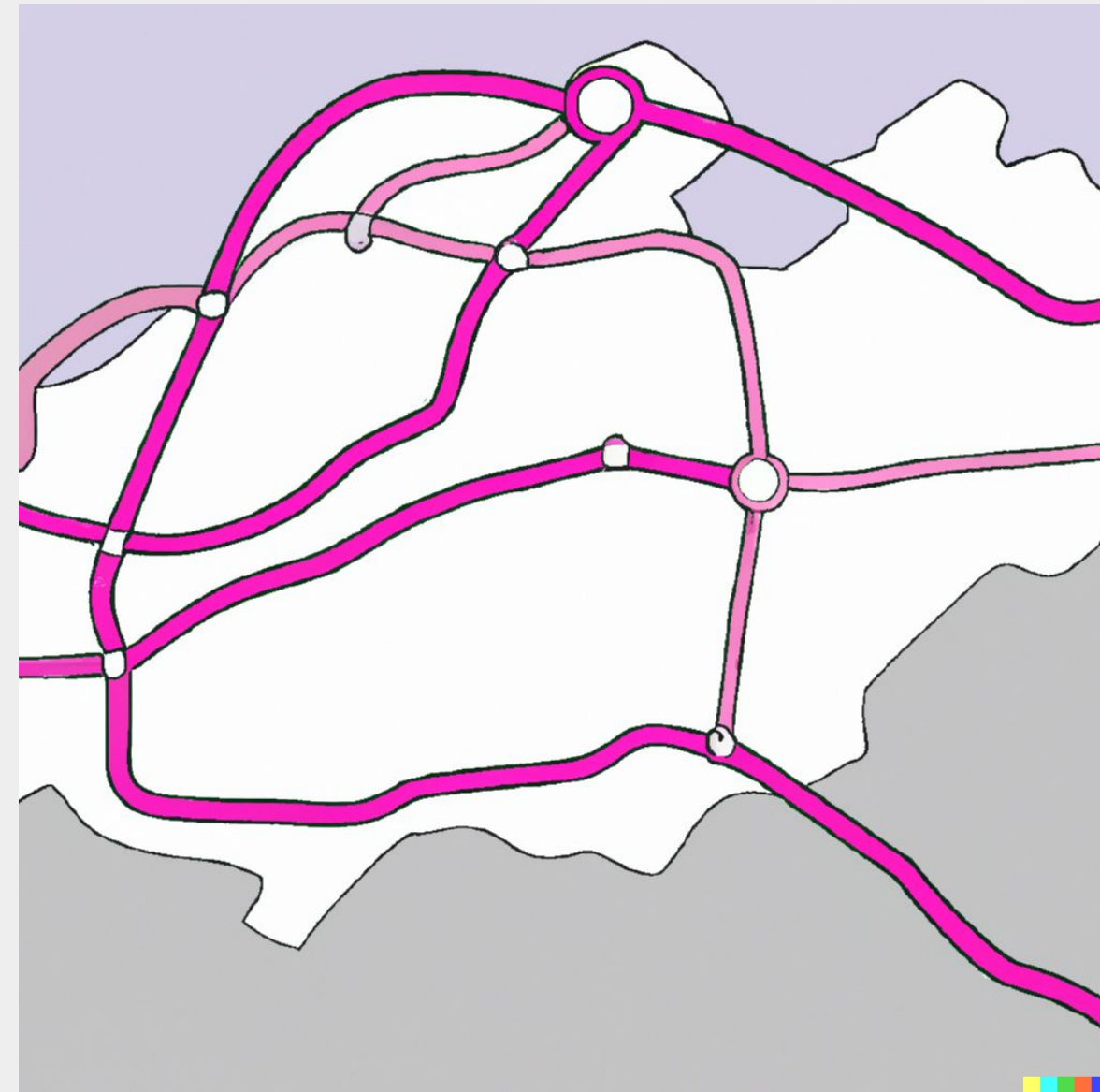
pAIntBox

What do you want to create?

**An imaginary subway map
in a coastal city.**

Image dimensions: by (Max 2048)

Generate




Generative AI

CodeAid

```
1 def binary_search(arr, x, l, r):  
2     if r >= l:  
3         mid = l + (r - l) // 2  
4         if arr[mid] == x:  
5             return mid  
6         elif arr[mid] > x:  
7             return binary_search(arr, x, l, mid - 1)  
8         else:  
9             return binary_search(arr, x, mid + 1, r)  
     else:  
         return -1
```

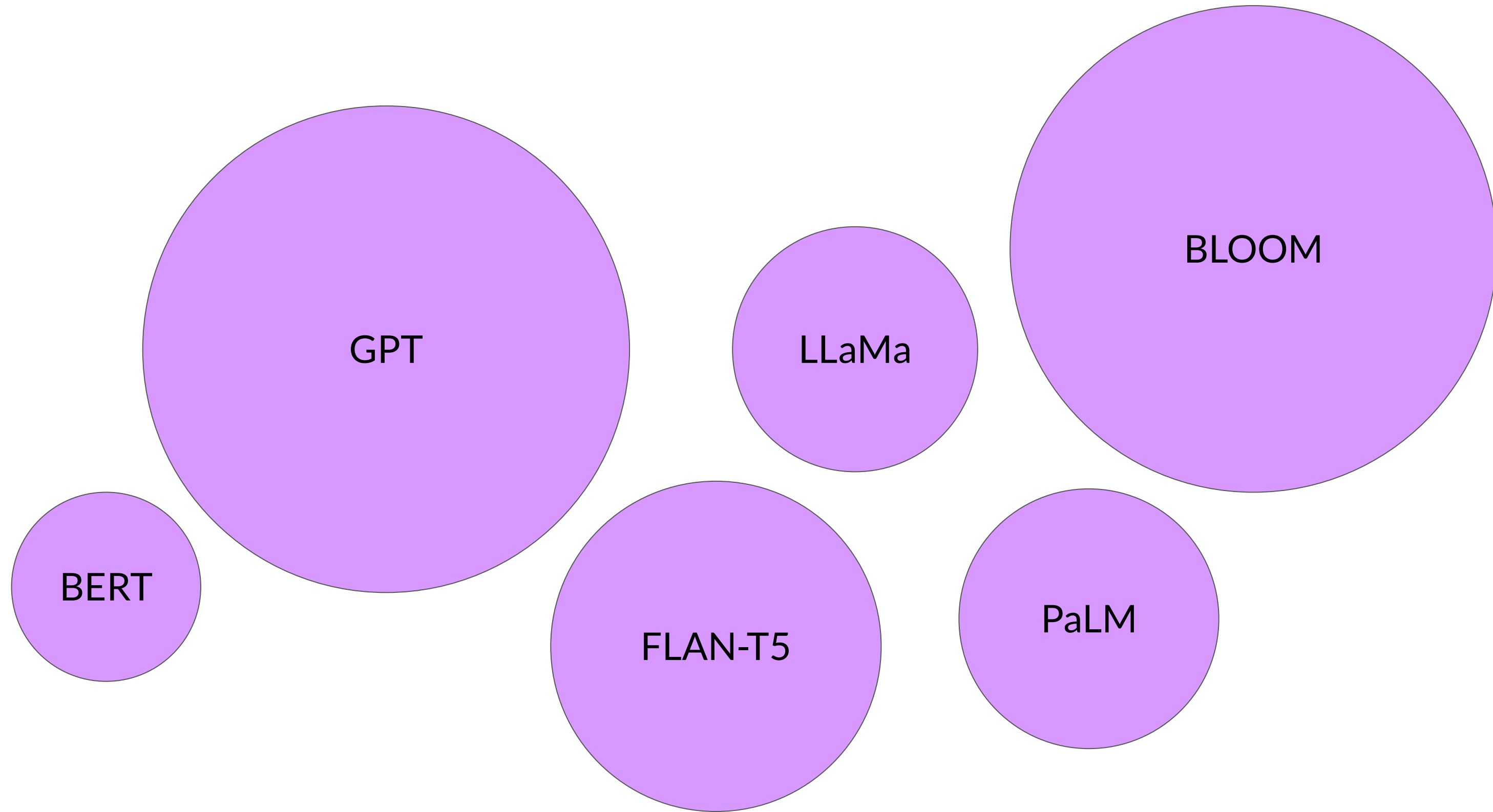
< 1/2 > Accept

Tab

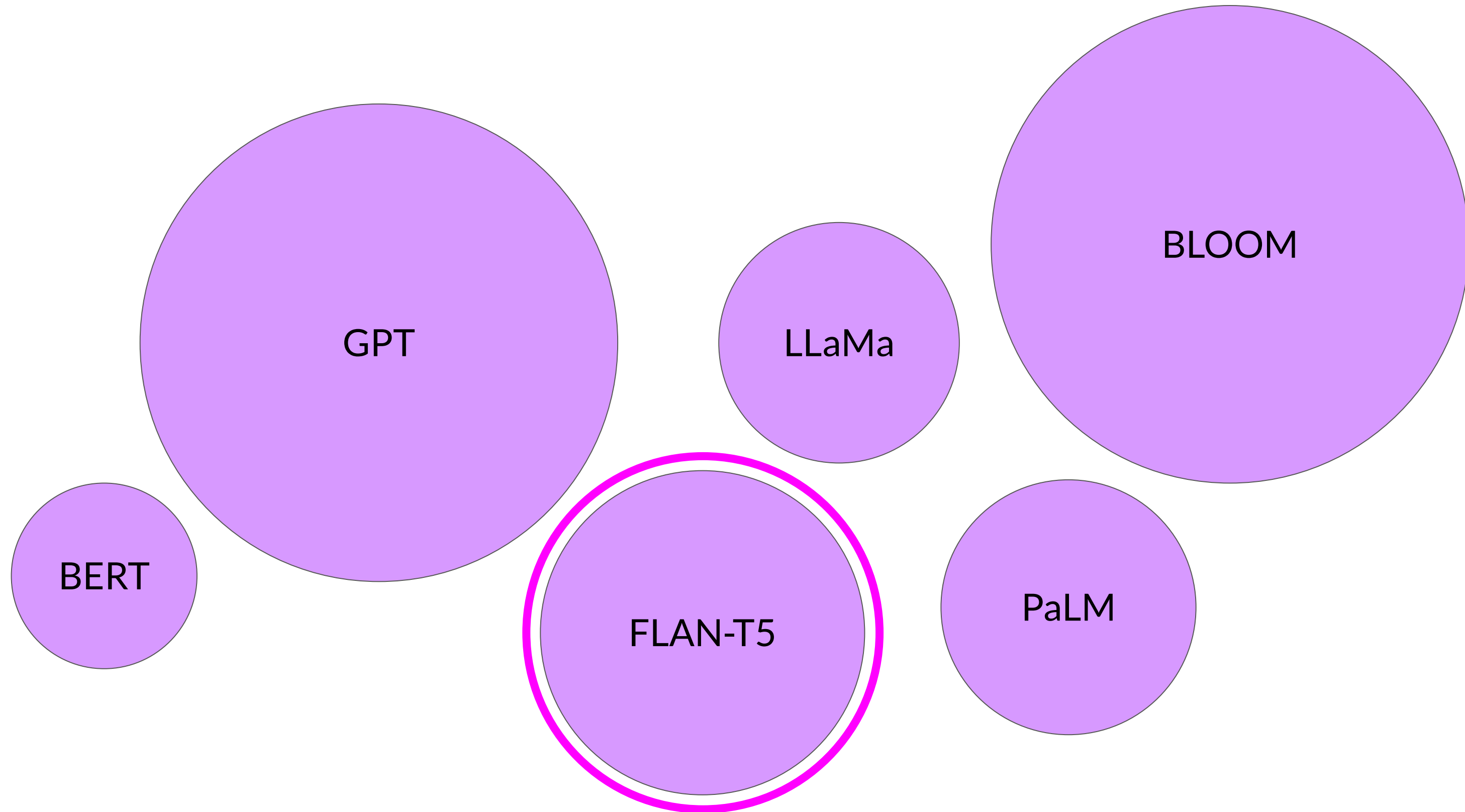
AI Connected 

Run security scan

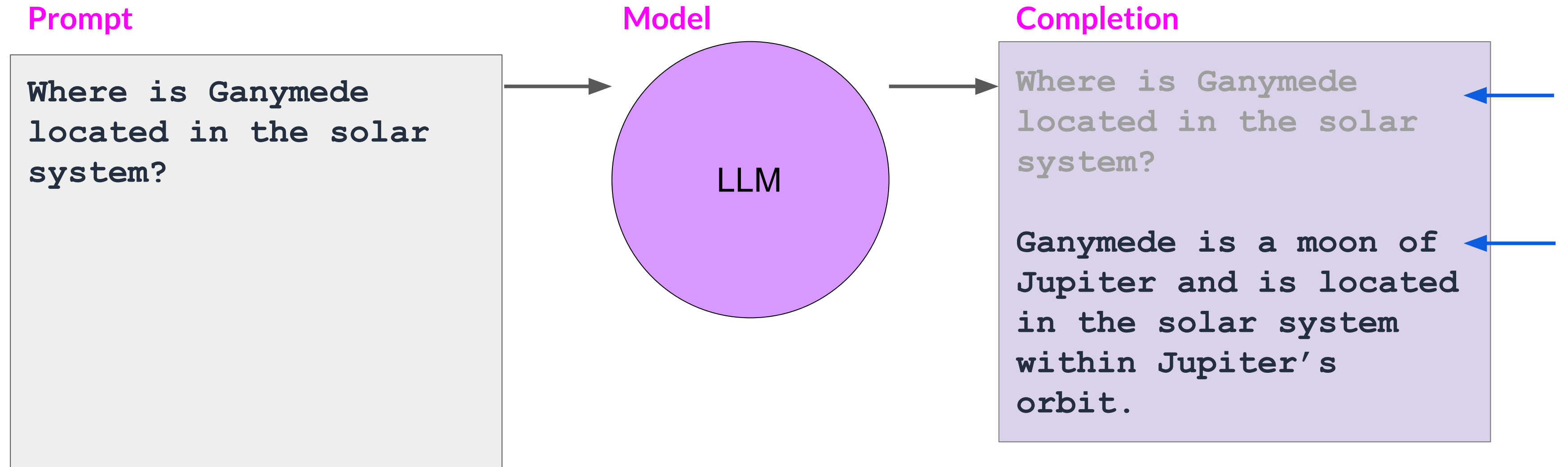
Large Language Models



Large Language Models



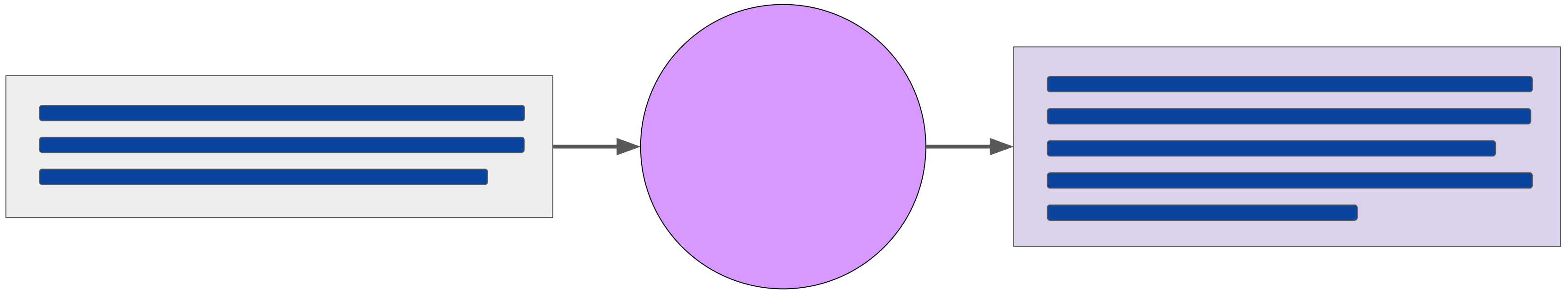
Prompts and completions



Context window

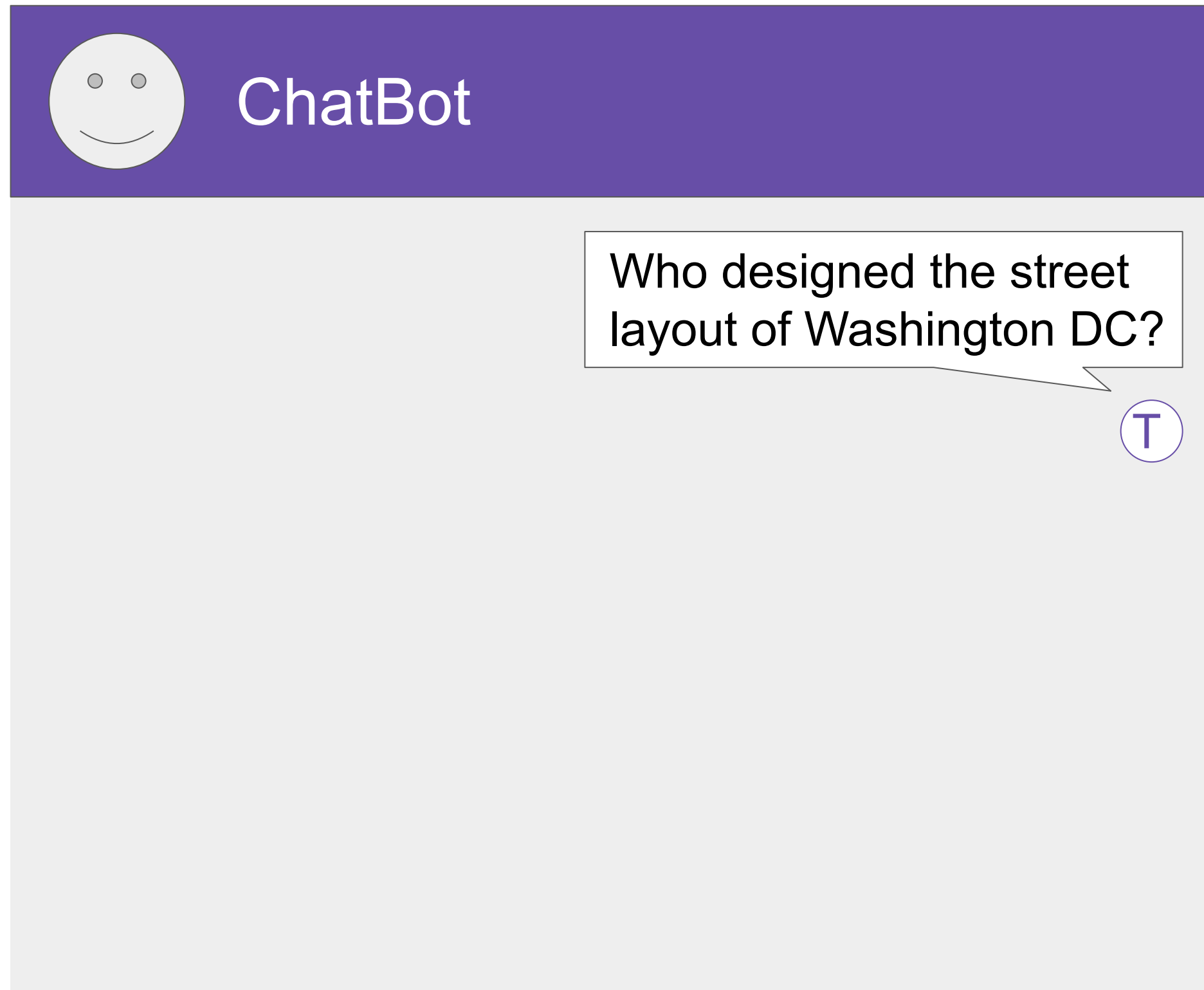
- typically a few 1000 words.

Prompts and completions

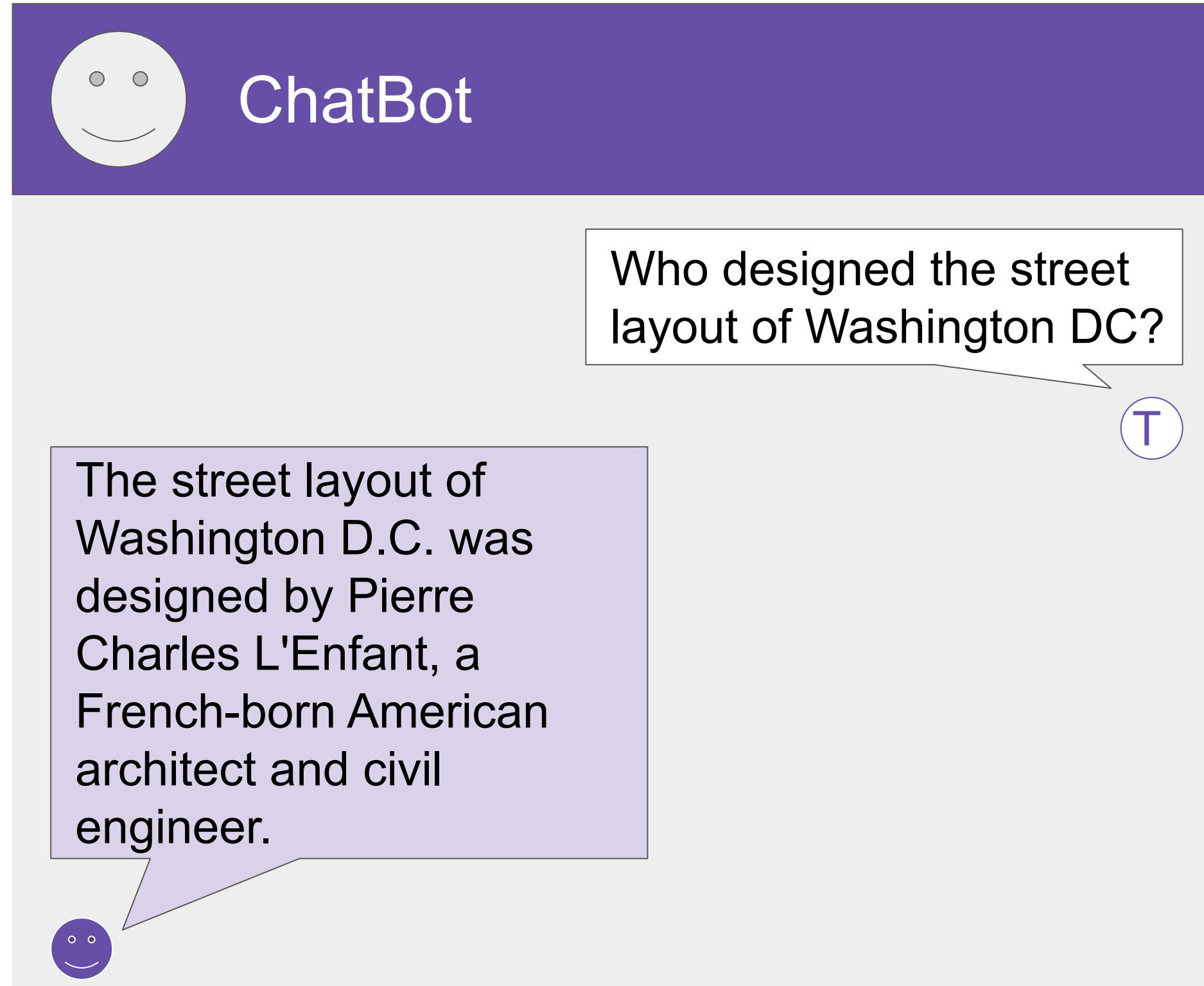


Use cases & tasks

LLM chatbot



LLM chatbot



The image shows a chatbot interface with a purple header bar. On the left of the header is a white smiley face icon, and to its right is the text "ChatBot". Below the header, the chat area has a light gray background. A white speech bubble on the right contains the question "Who designed the street layout of Washington DC?". Below this bubble is a small purple circle with a white letter "T". A larger purple speech bubble on the left contains the answer: "The street layout of Washington D.C. was designed by Pierre Charles L'Enfant, a French-born American architect and civil engineer." At the bottom left of this answer bubble is a small purple smiley face icon.

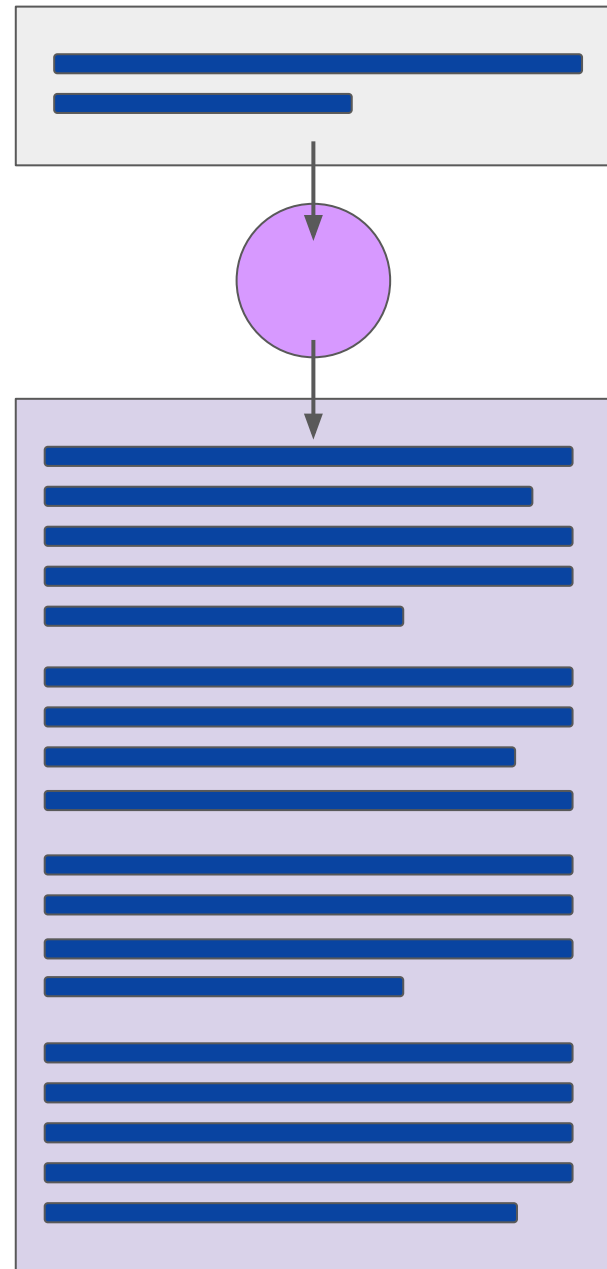
ChatBot

Who designed the street layout of Washington DC?

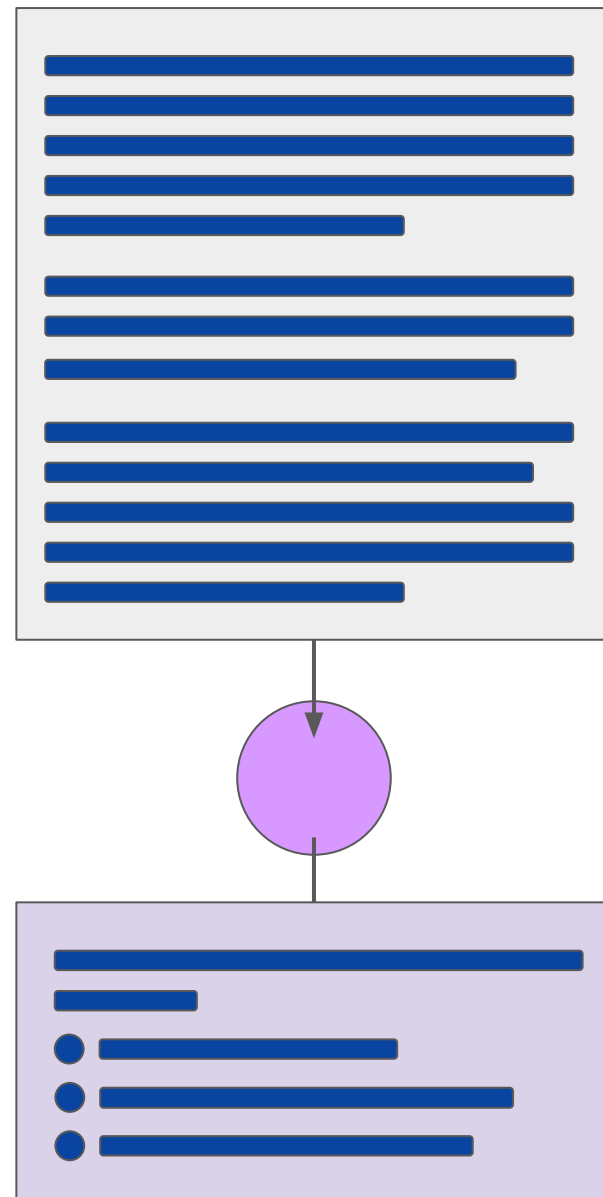
The street layout of Washington D.C. was designed by Pierre Charles L'Enfant, a French-born American architect and civil engineer.

LLM use cases & tasks

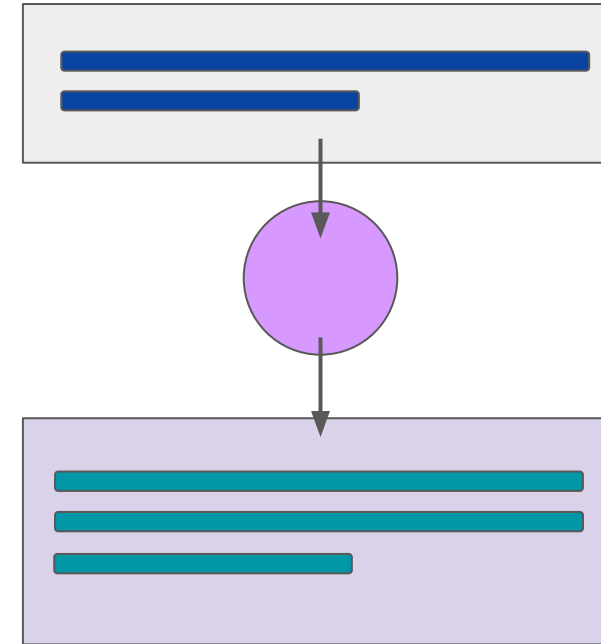
Essay Writing



Summarization

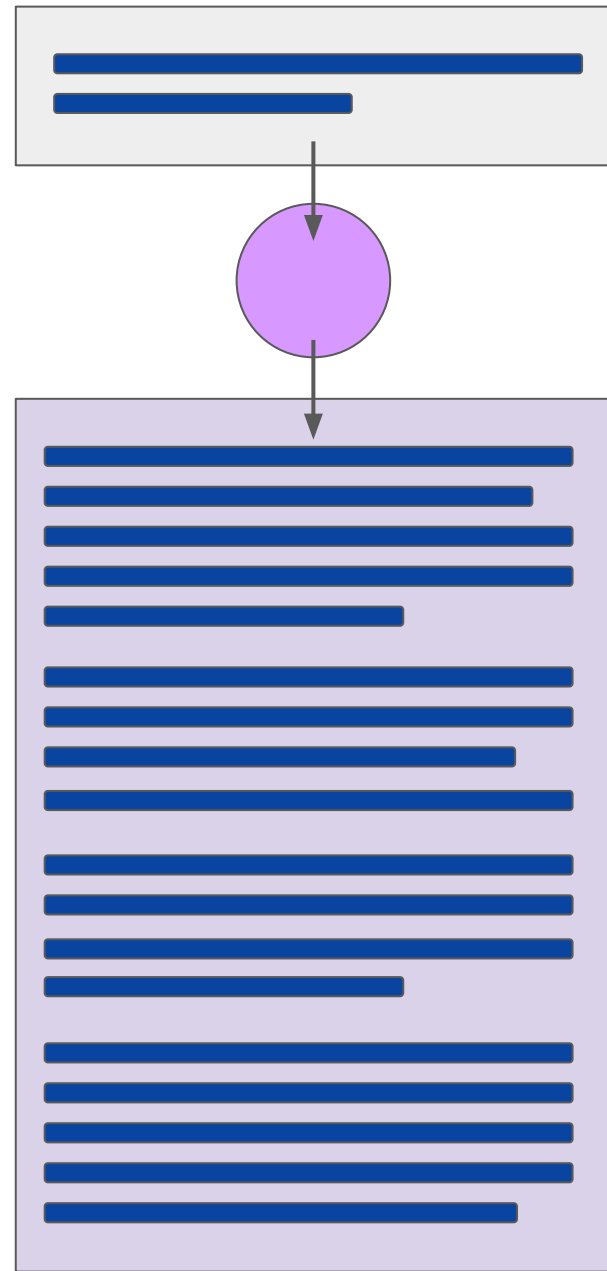


Translation

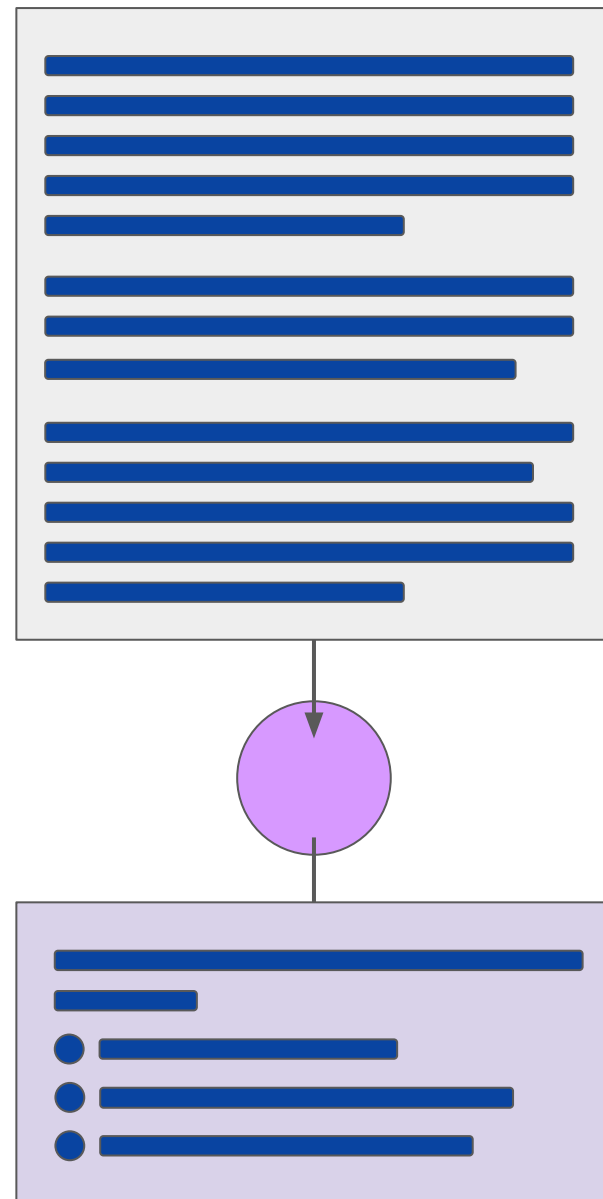


LLM use cases & tasks

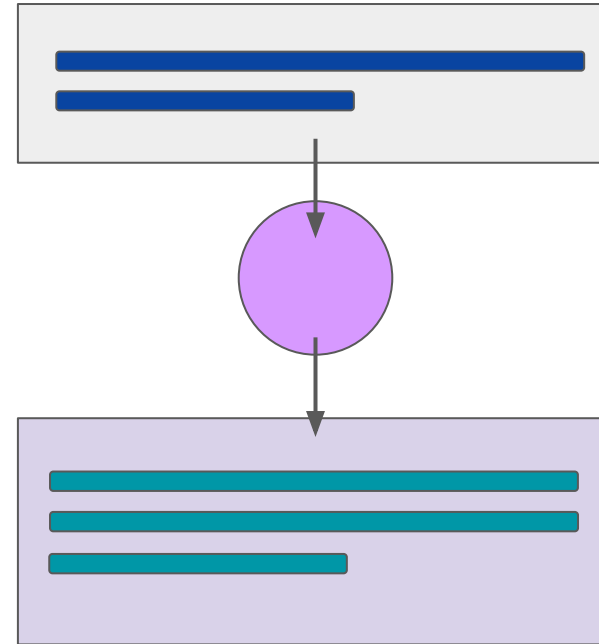
Essay Writing



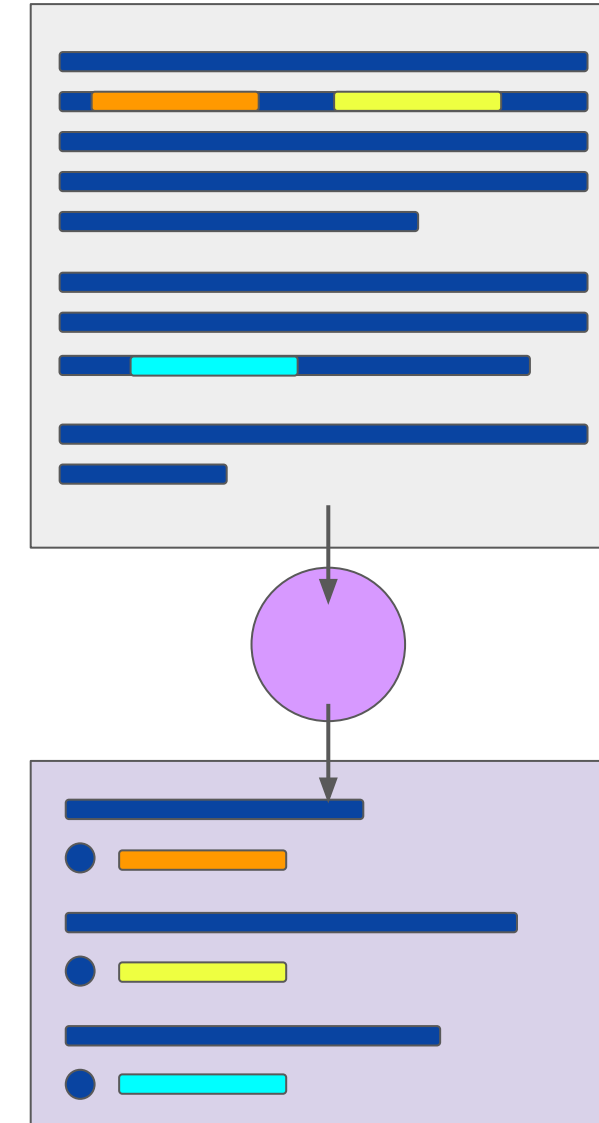
Summarization



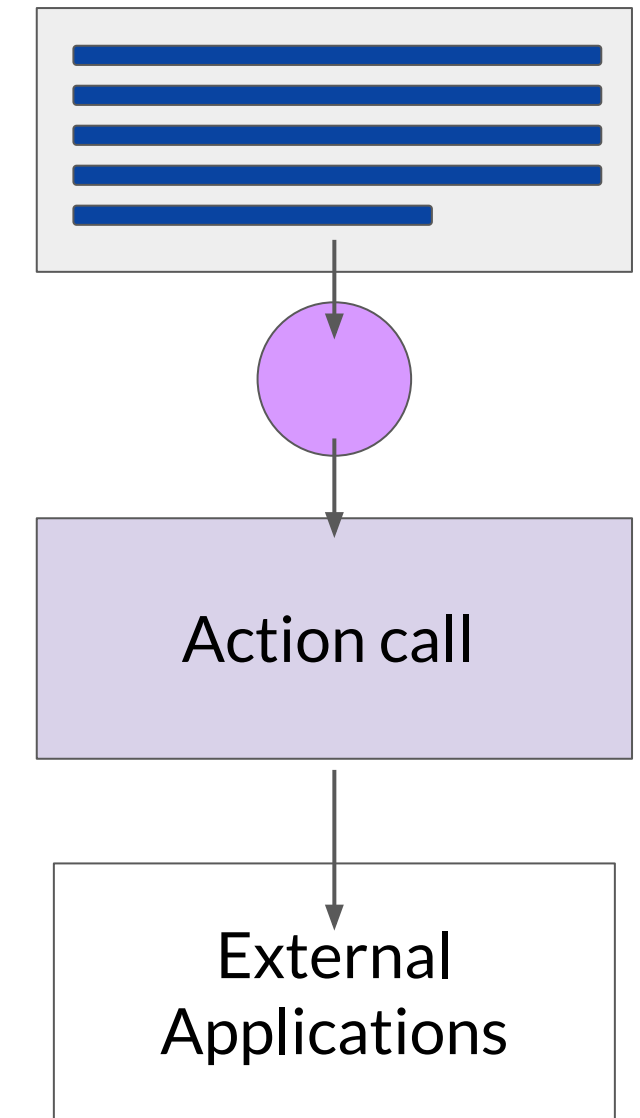
Translation



Information retrieval

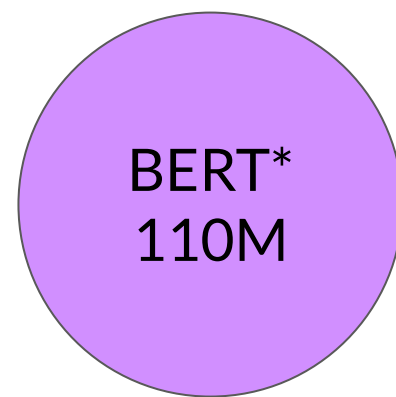


Invoke APIs and actions



The significance of scale: language understanding

BERT*
110M



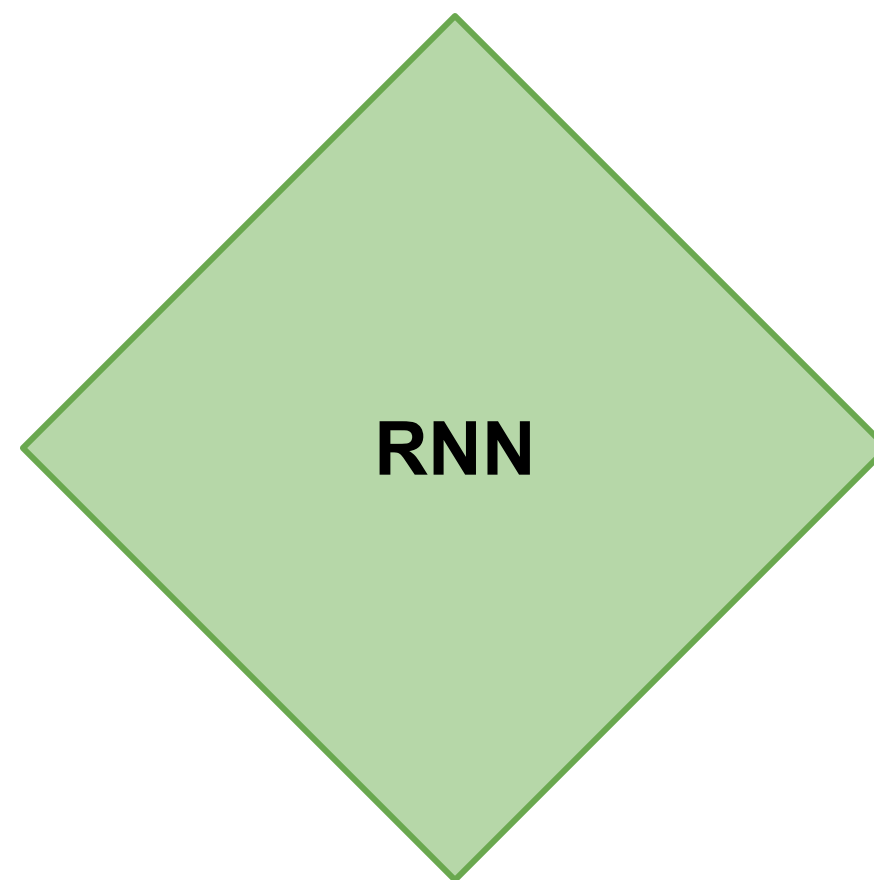
BLOOM
176B



*Bert-base

How LLMs work - Transformers architecture

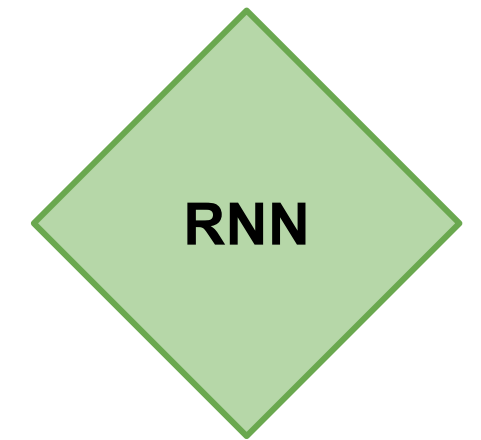
Generating text with RNNs



Generating text with RNNs

?

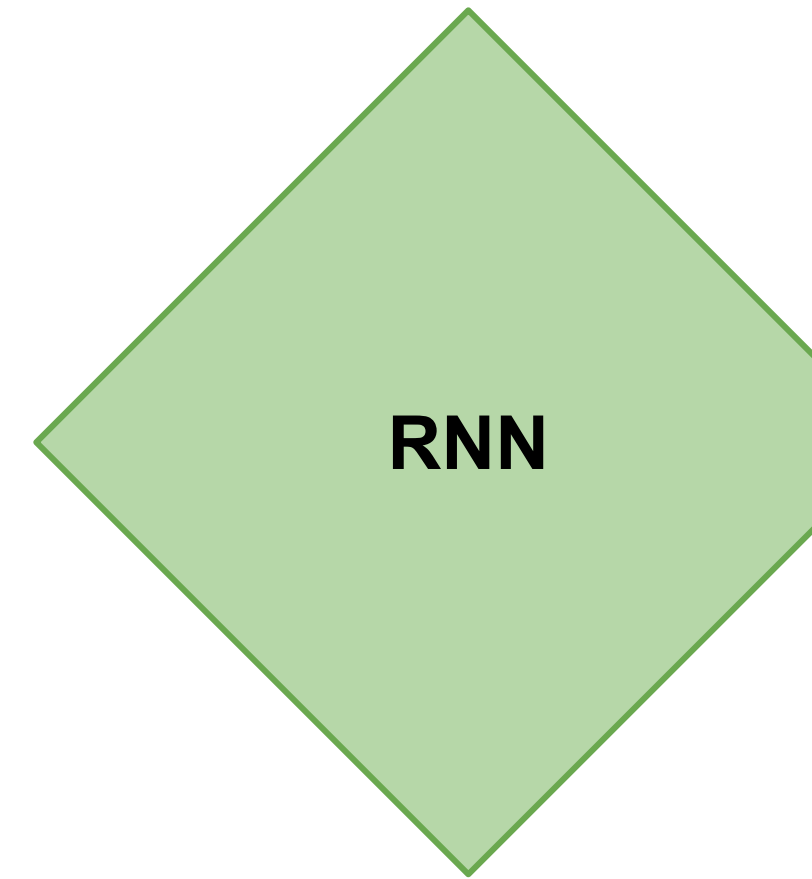
tastes ...



Generating text with RNNs



tea tastes ...



Generating text with RNNs

?

, my tea tastes ...

RNN

Generating text with RNNs

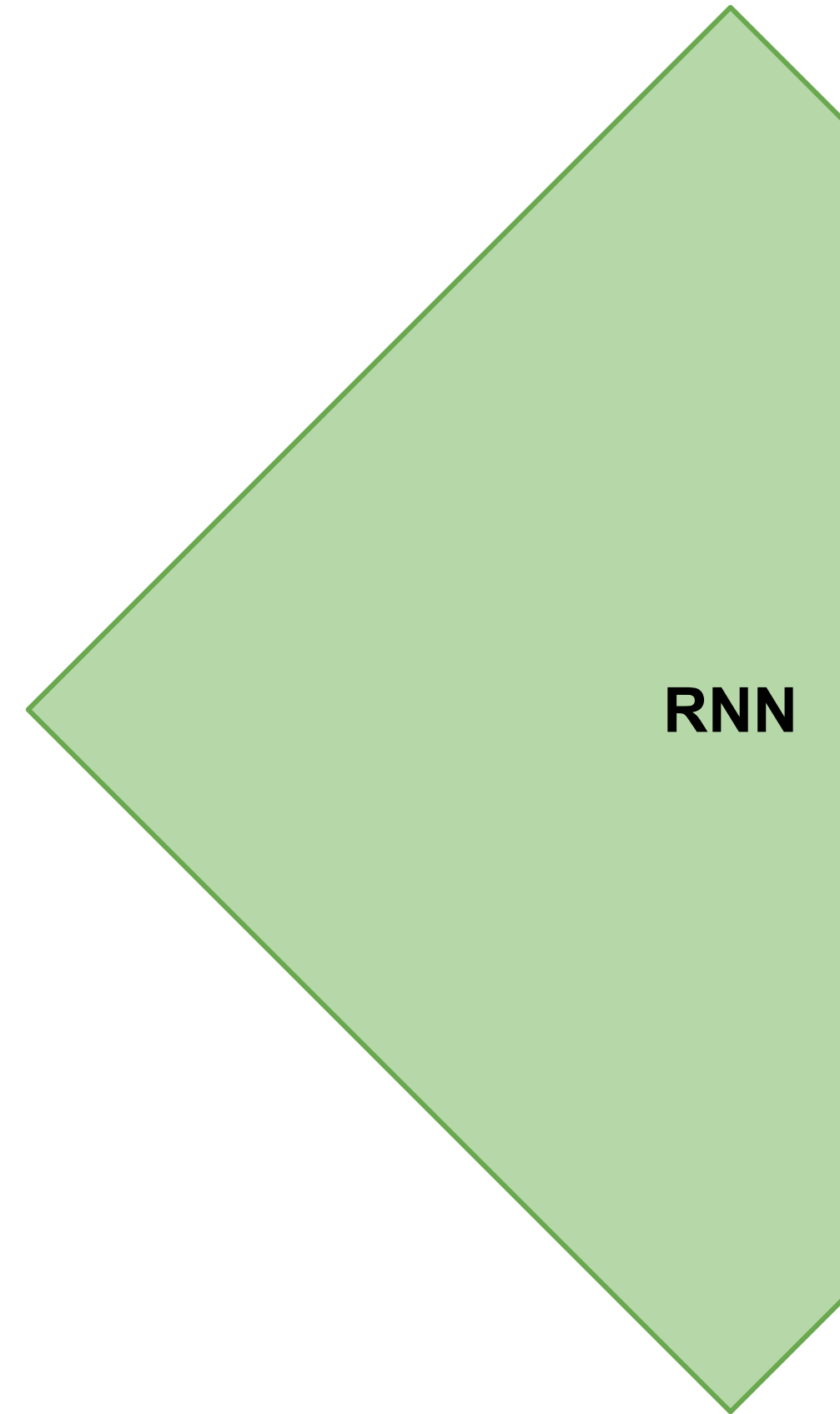
?

, my tea tastes great.

RNN

Generating text with RNNs

The milk is bad, my tea tastes ~~great~~.



Understanding language can be challenging

I took my money to the bank.

River bank?



Understanding language can be challenging

The teacher's book?

The teacher taught the student with the book.

The student's book?

Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

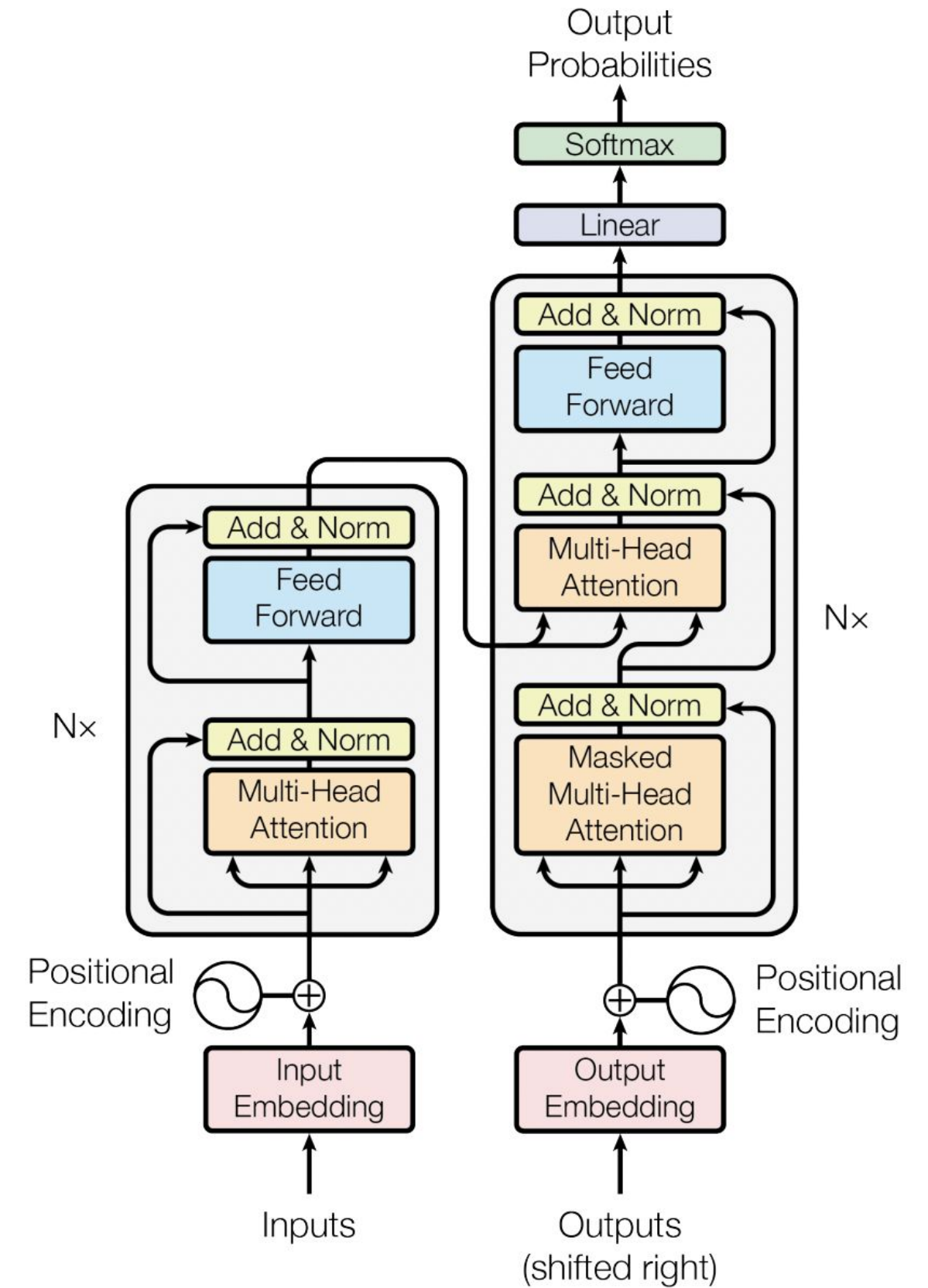
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to



Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

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

Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

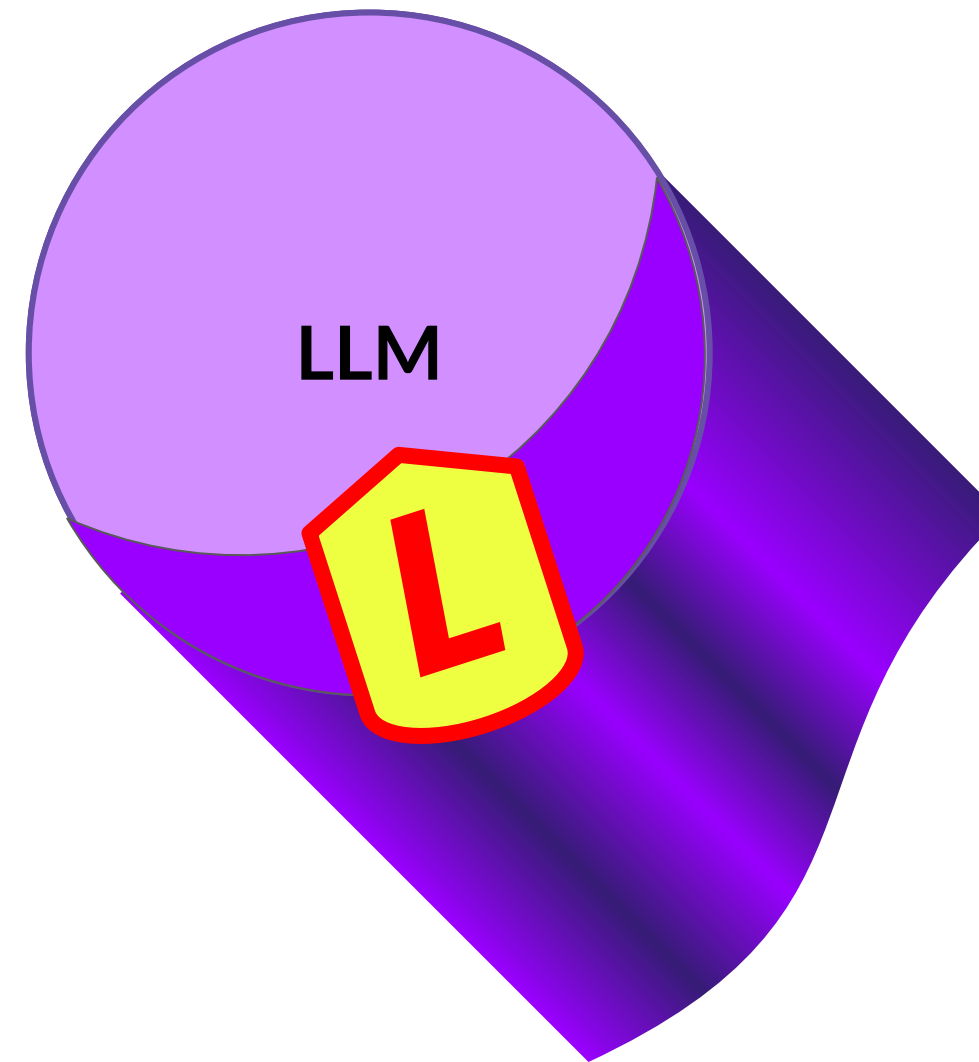
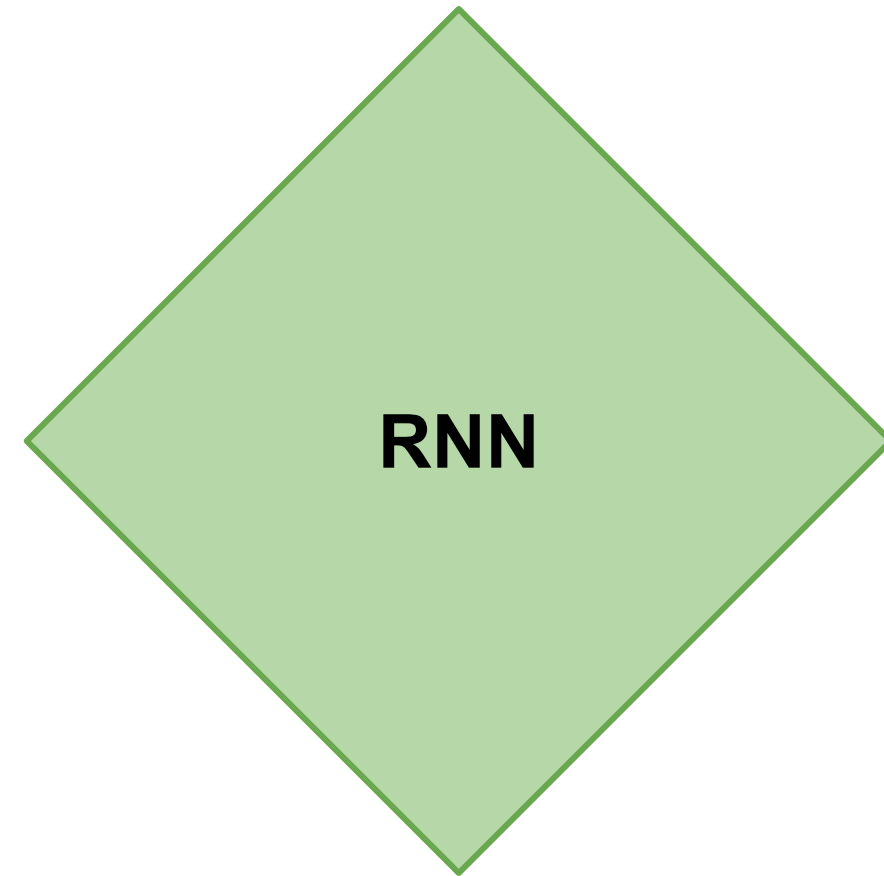
Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

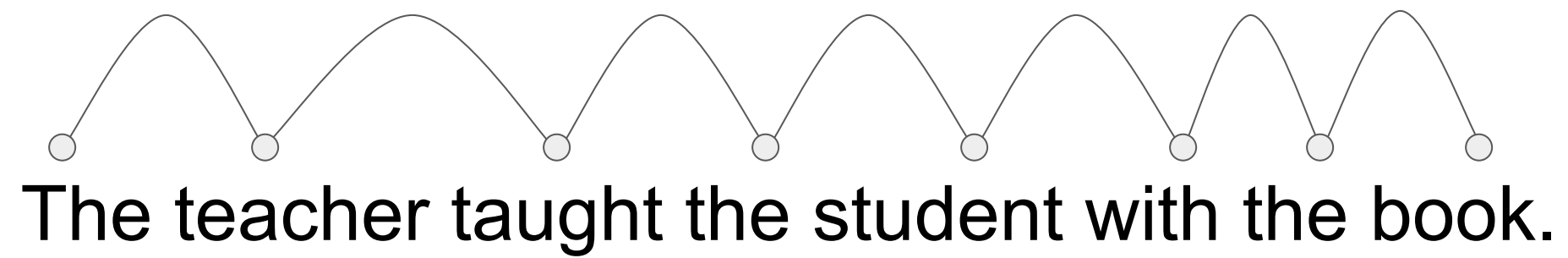
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to

- Scale efficiently
- Parallel process
- Attention to input meaning

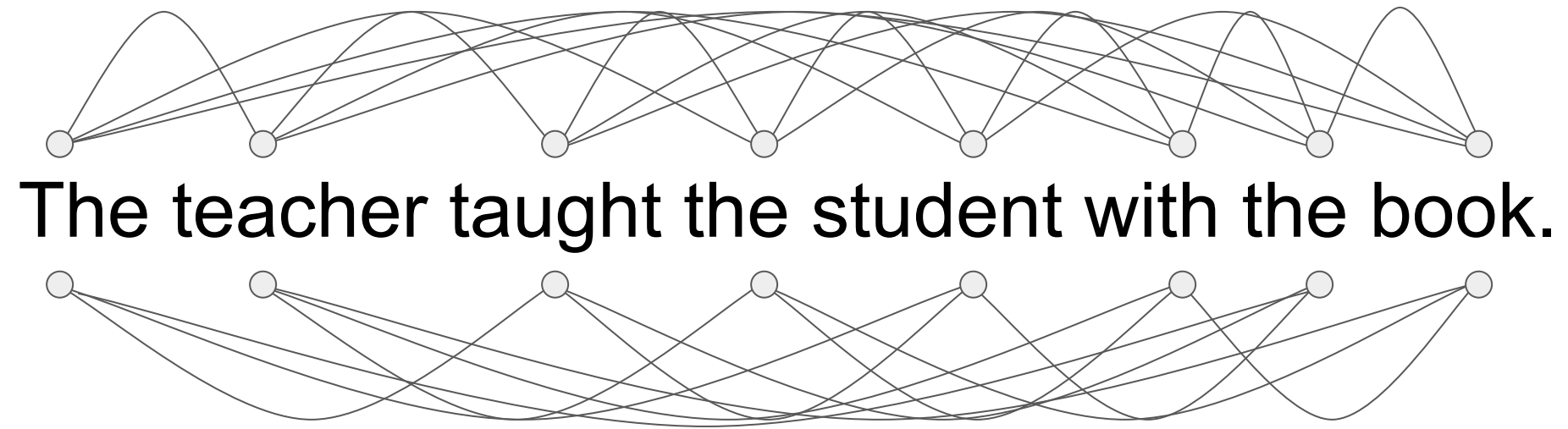
Transformers



Transformers

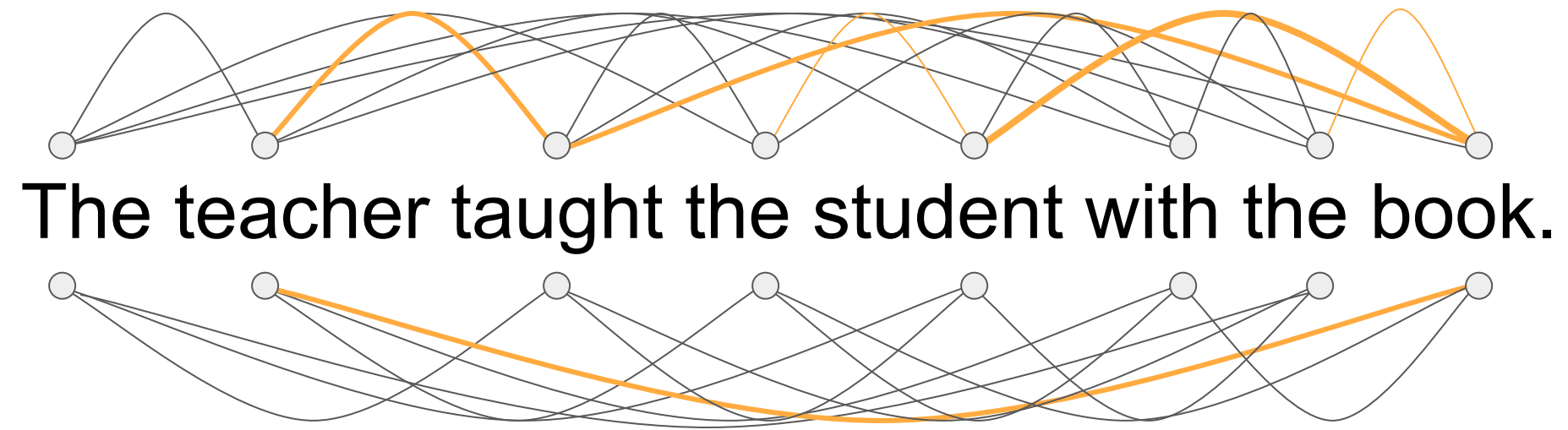


Transformers

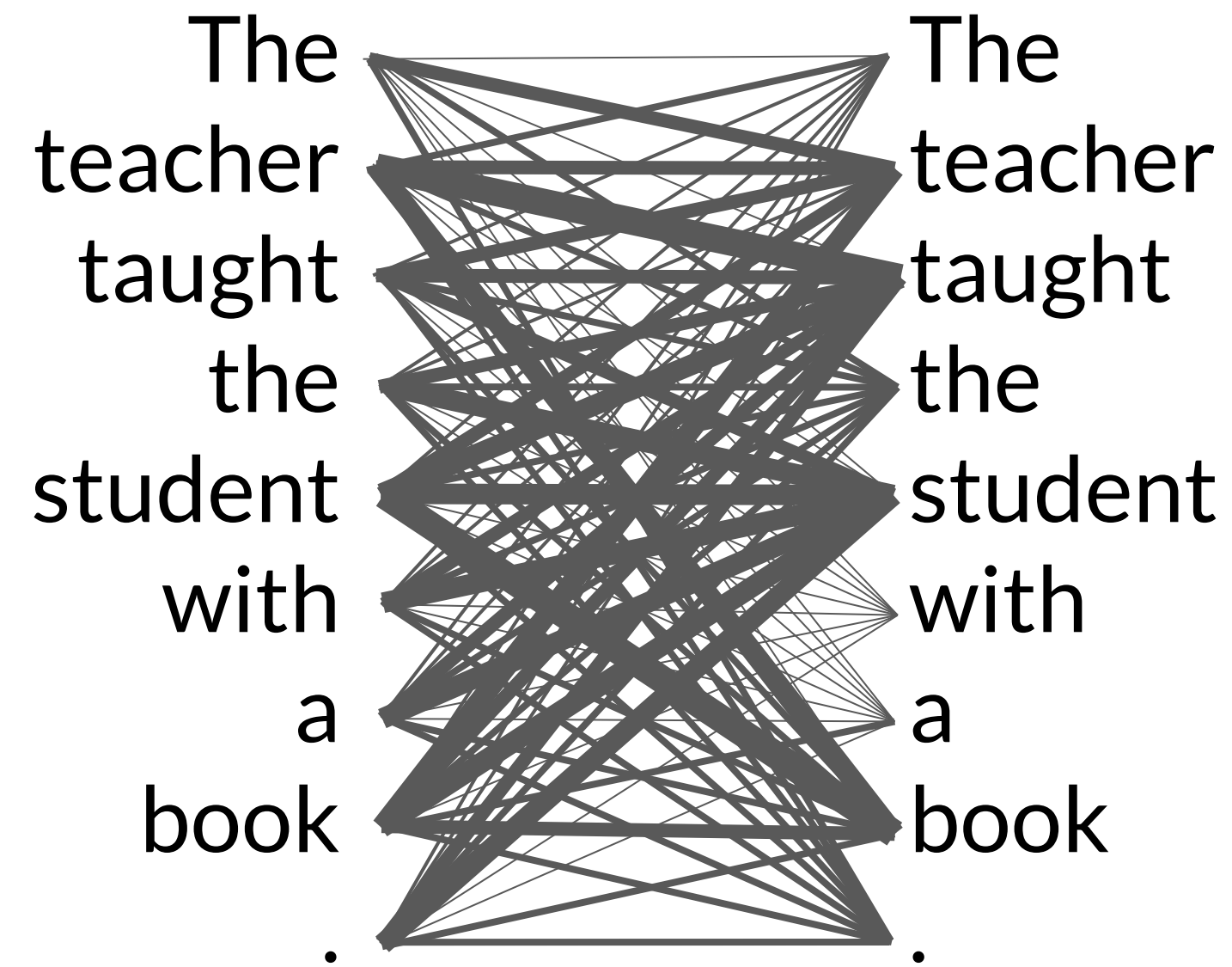


The teacher taught the student with the book.

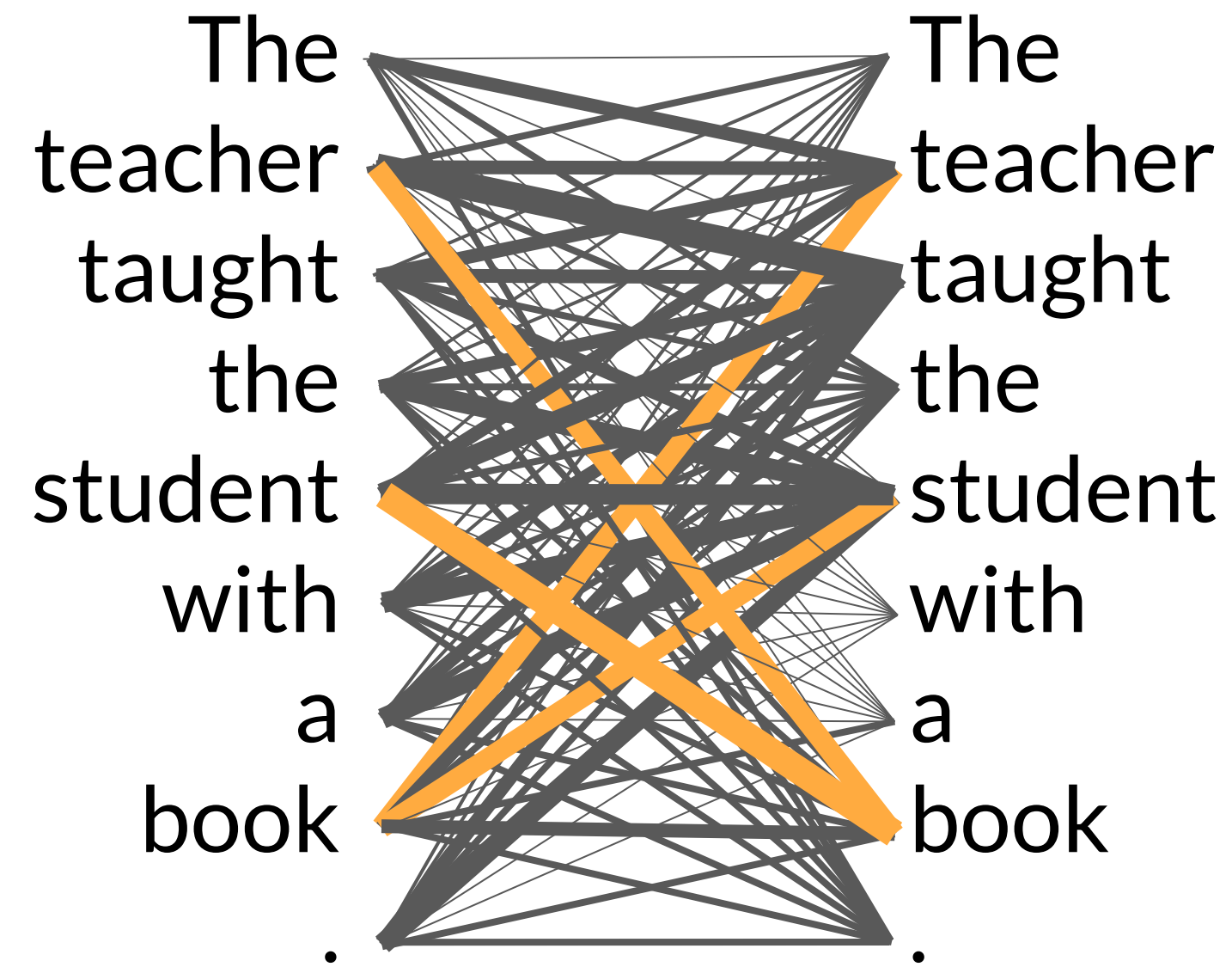
Transformers



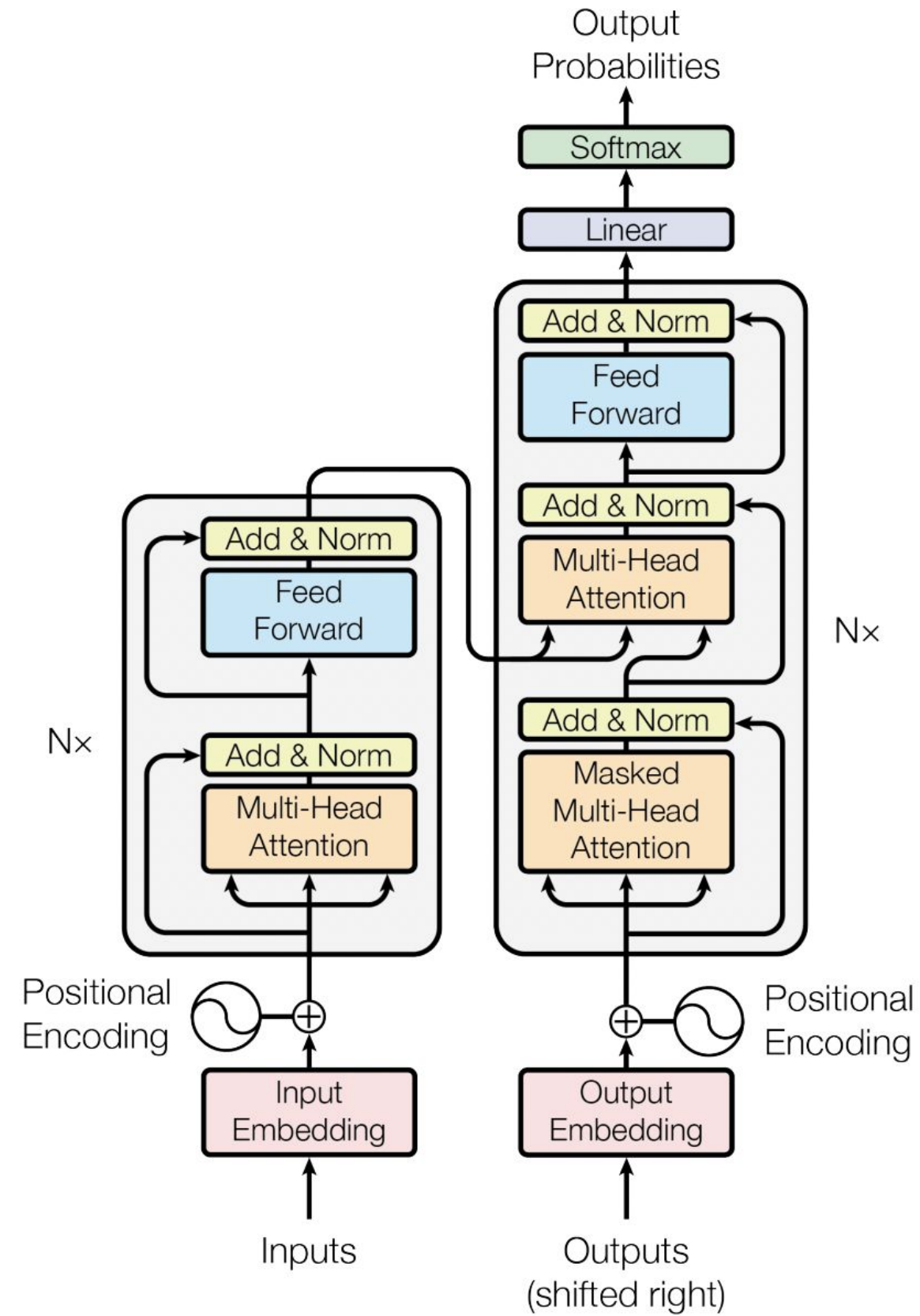
Self-attention



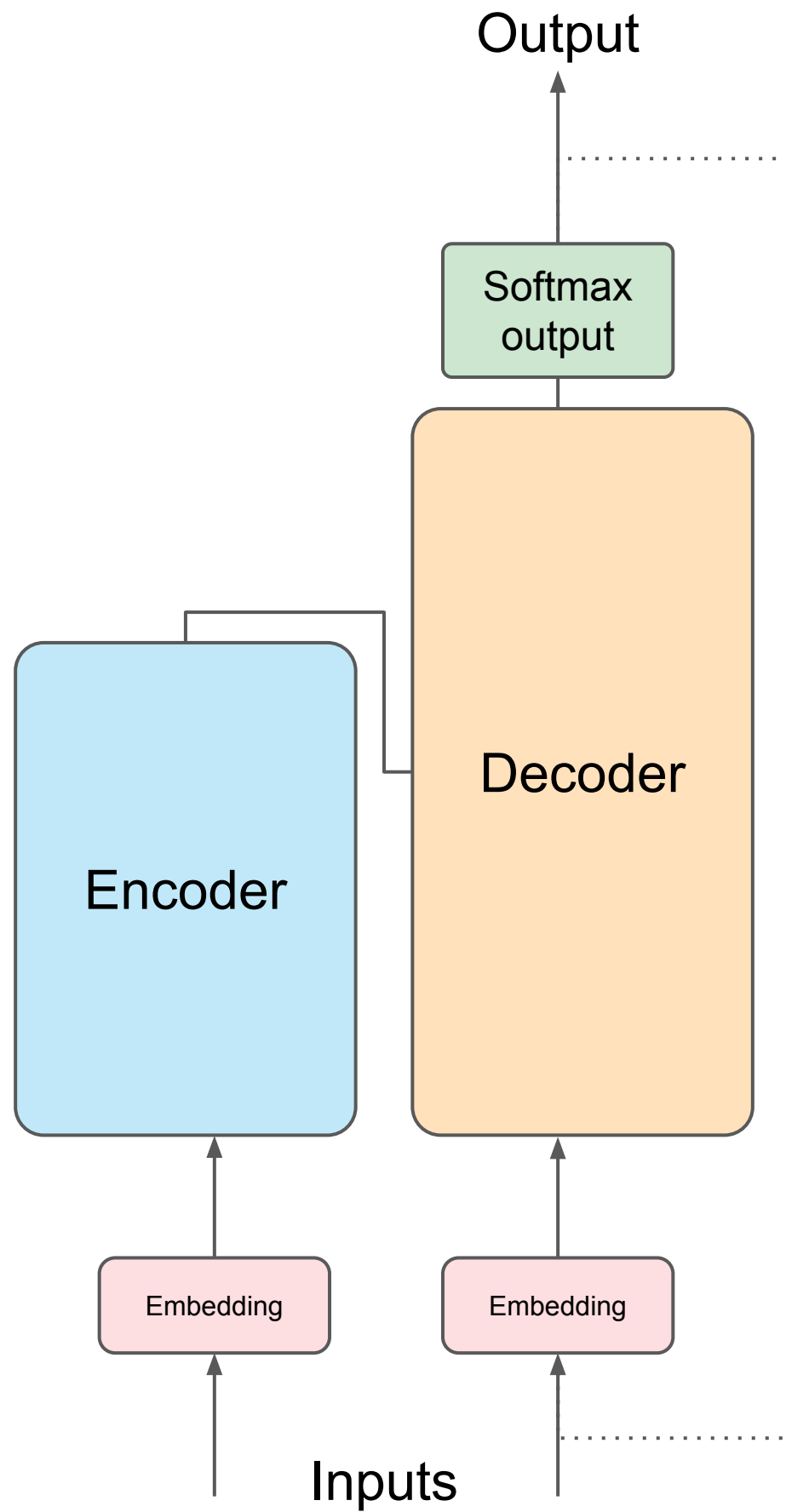
Self-attention



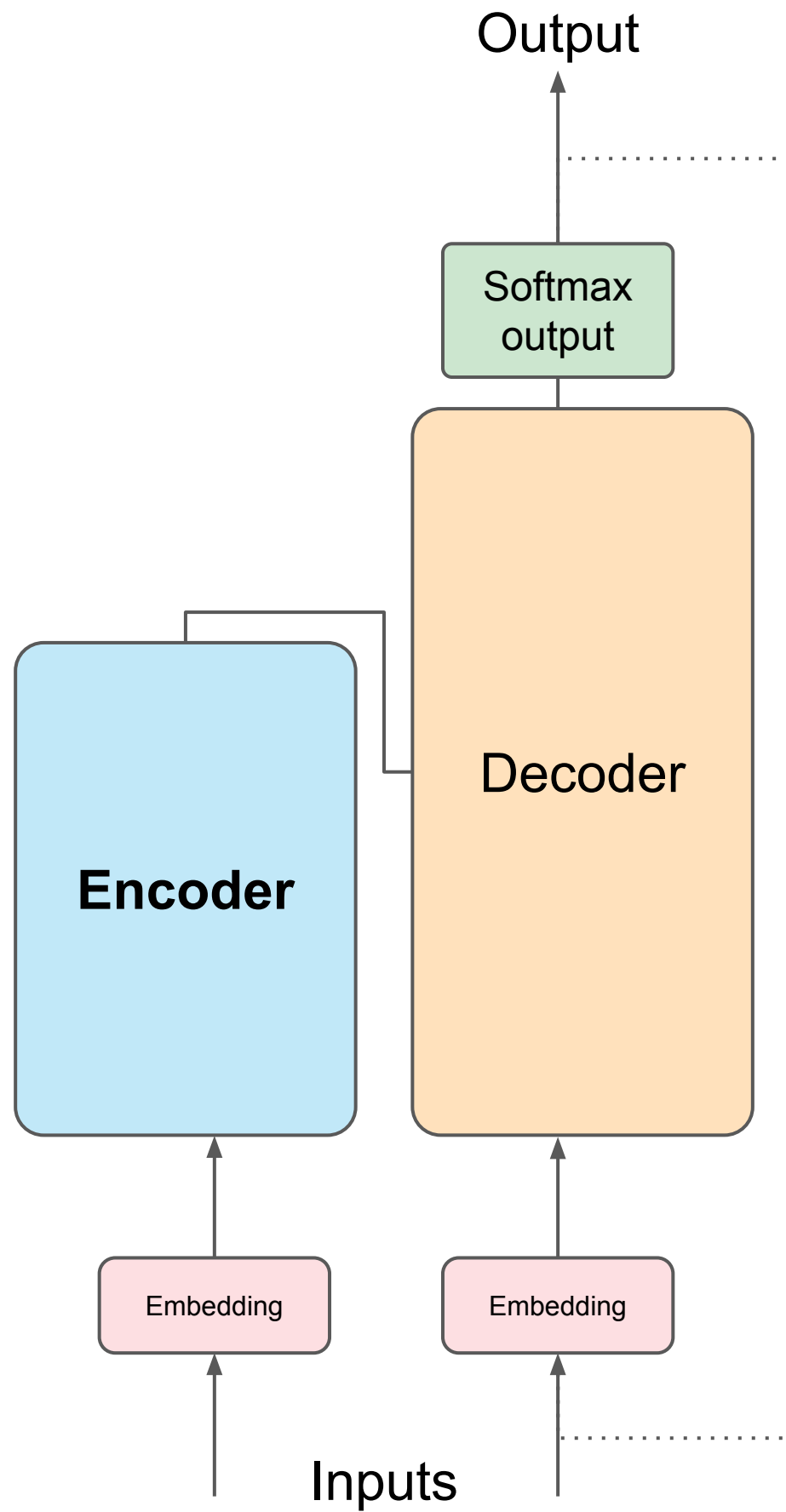
Transformers



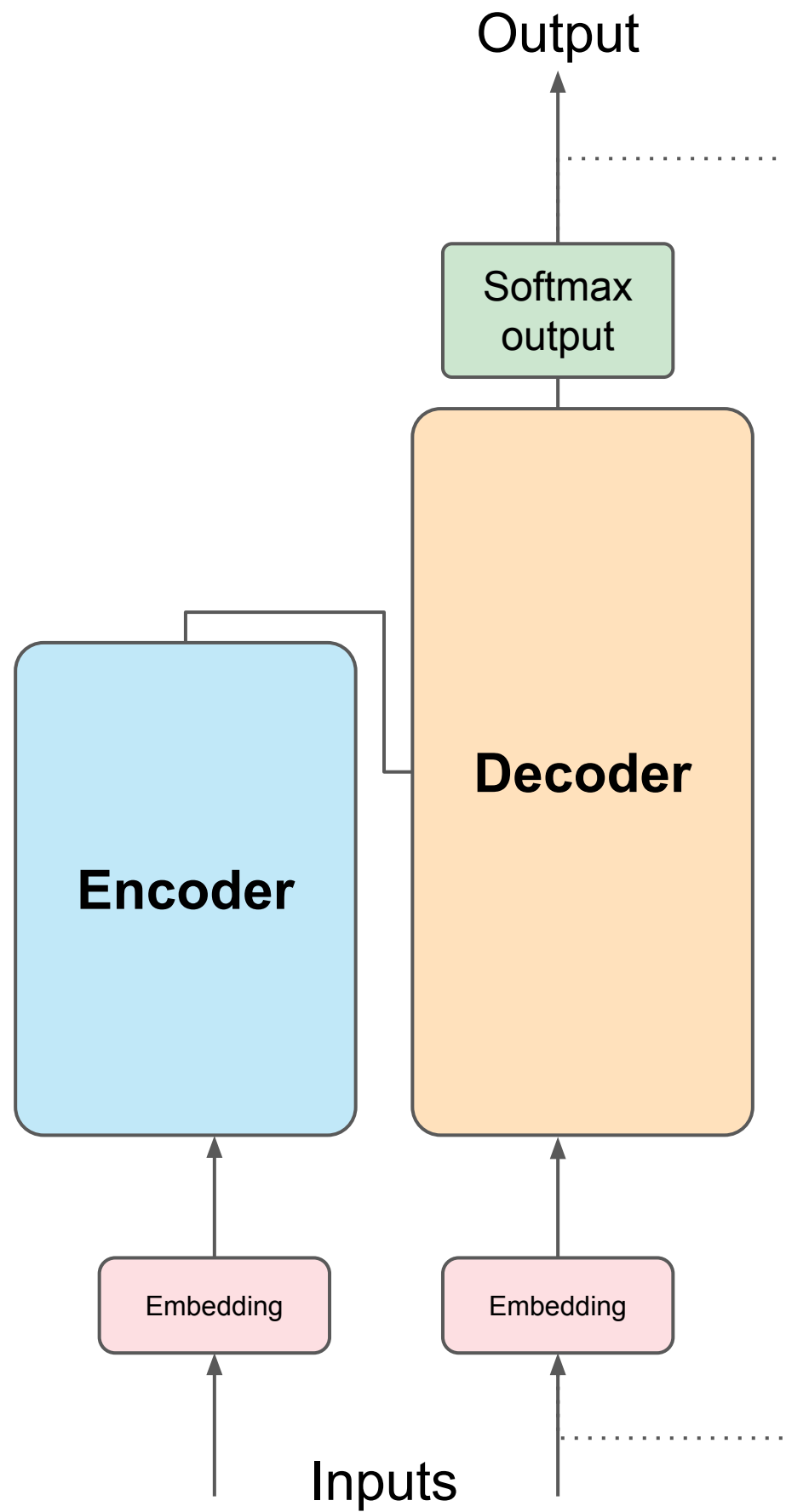
Transformers



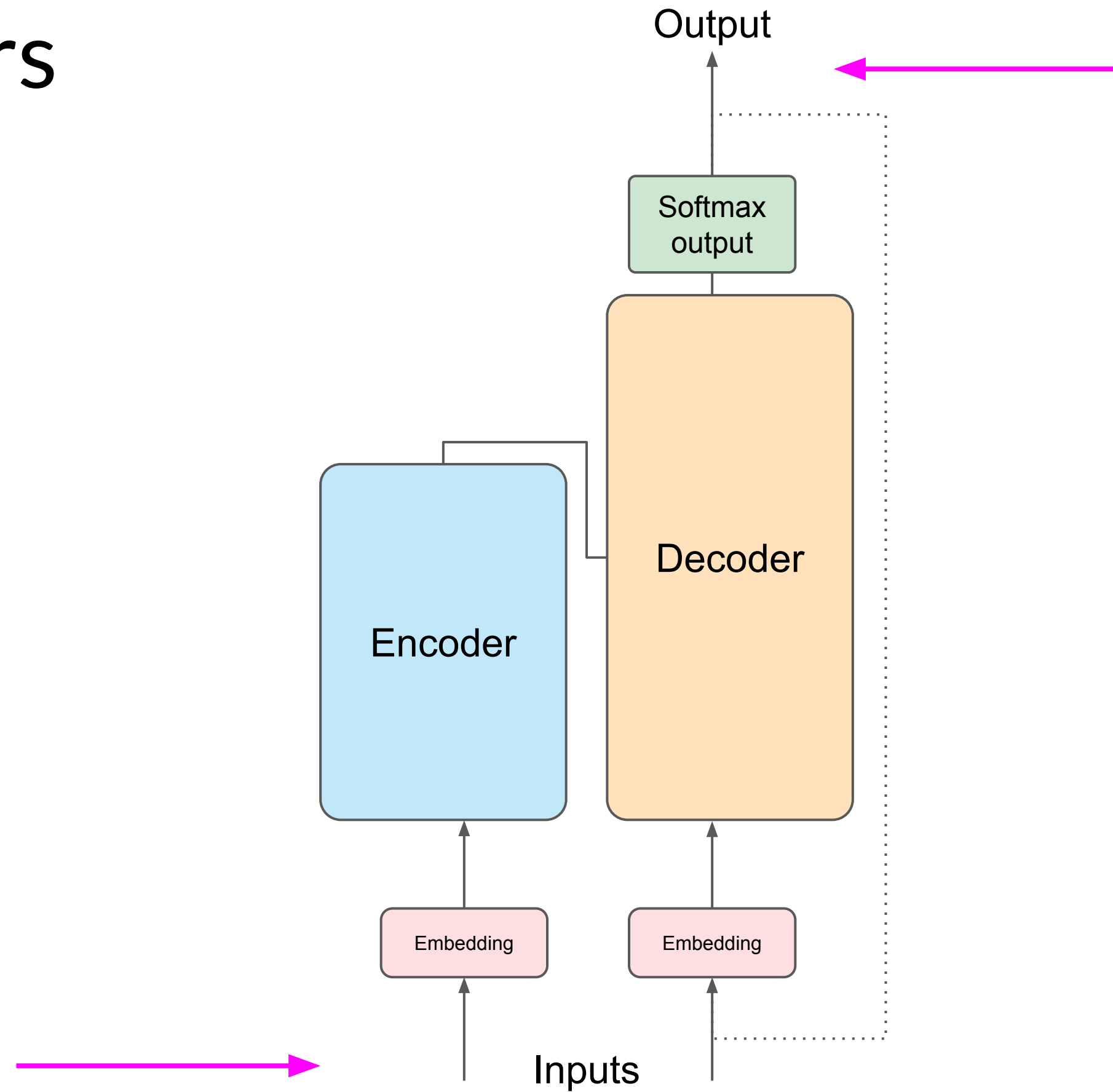
Transformers



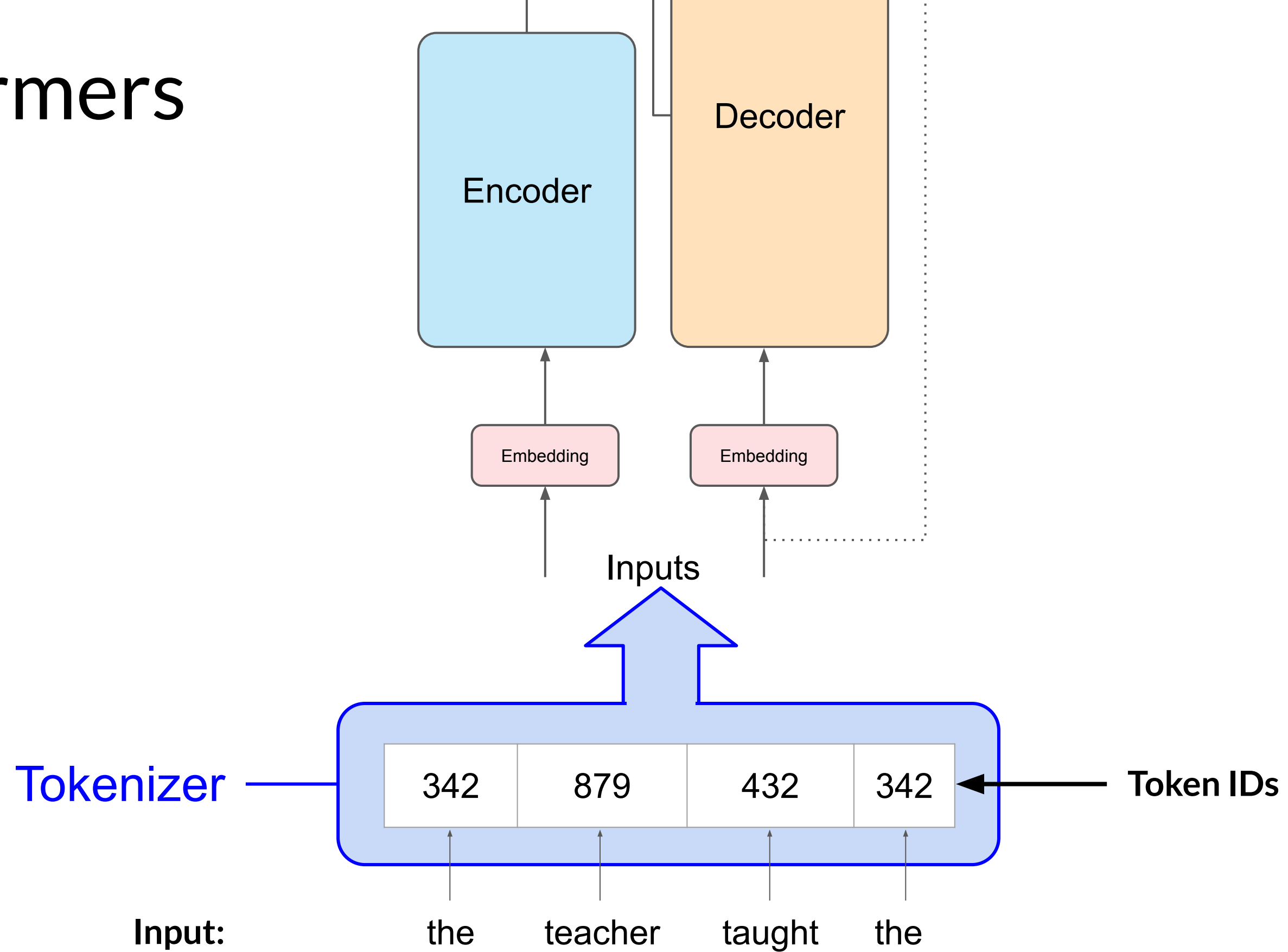
Transformers



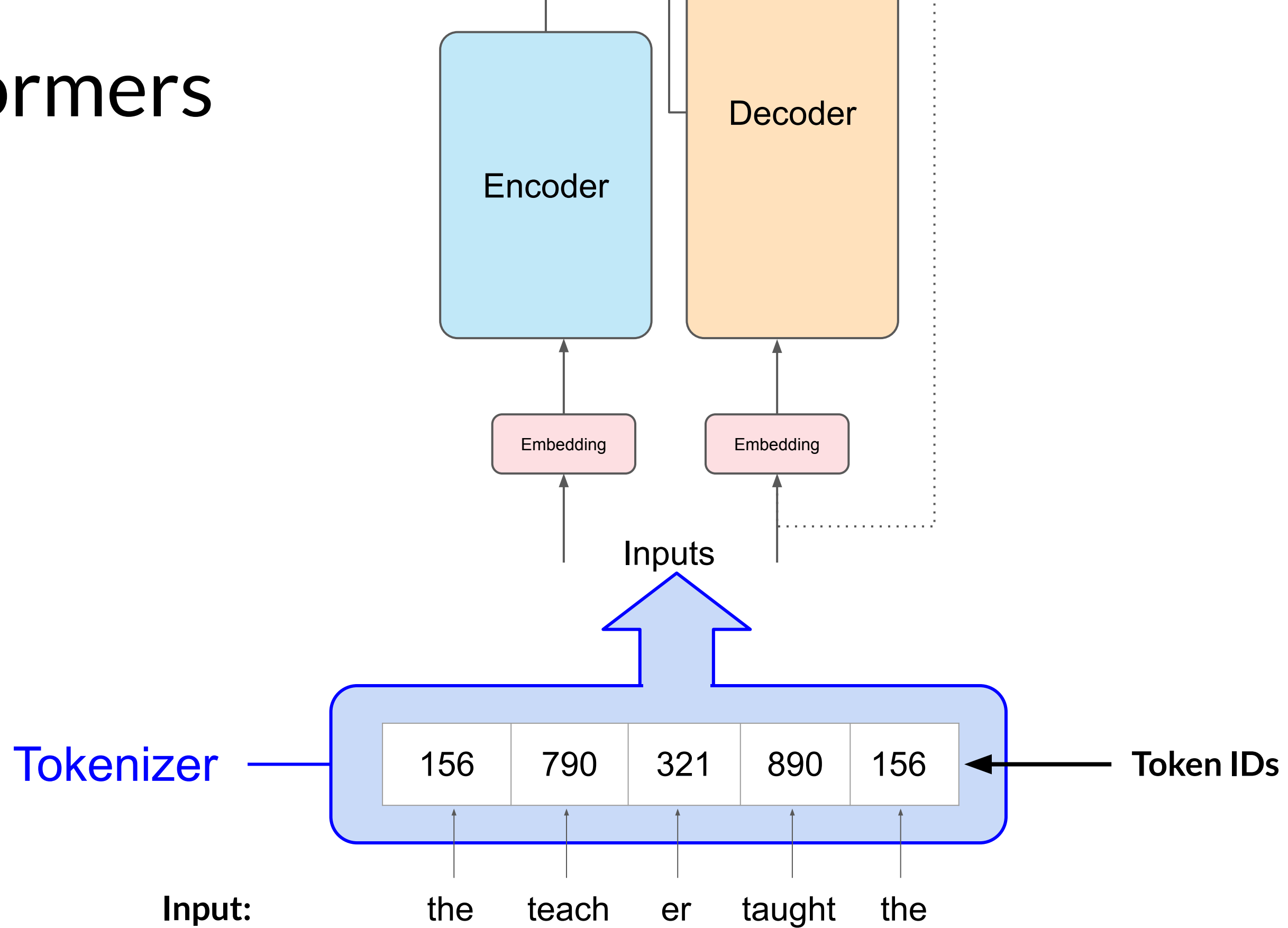
Transformers



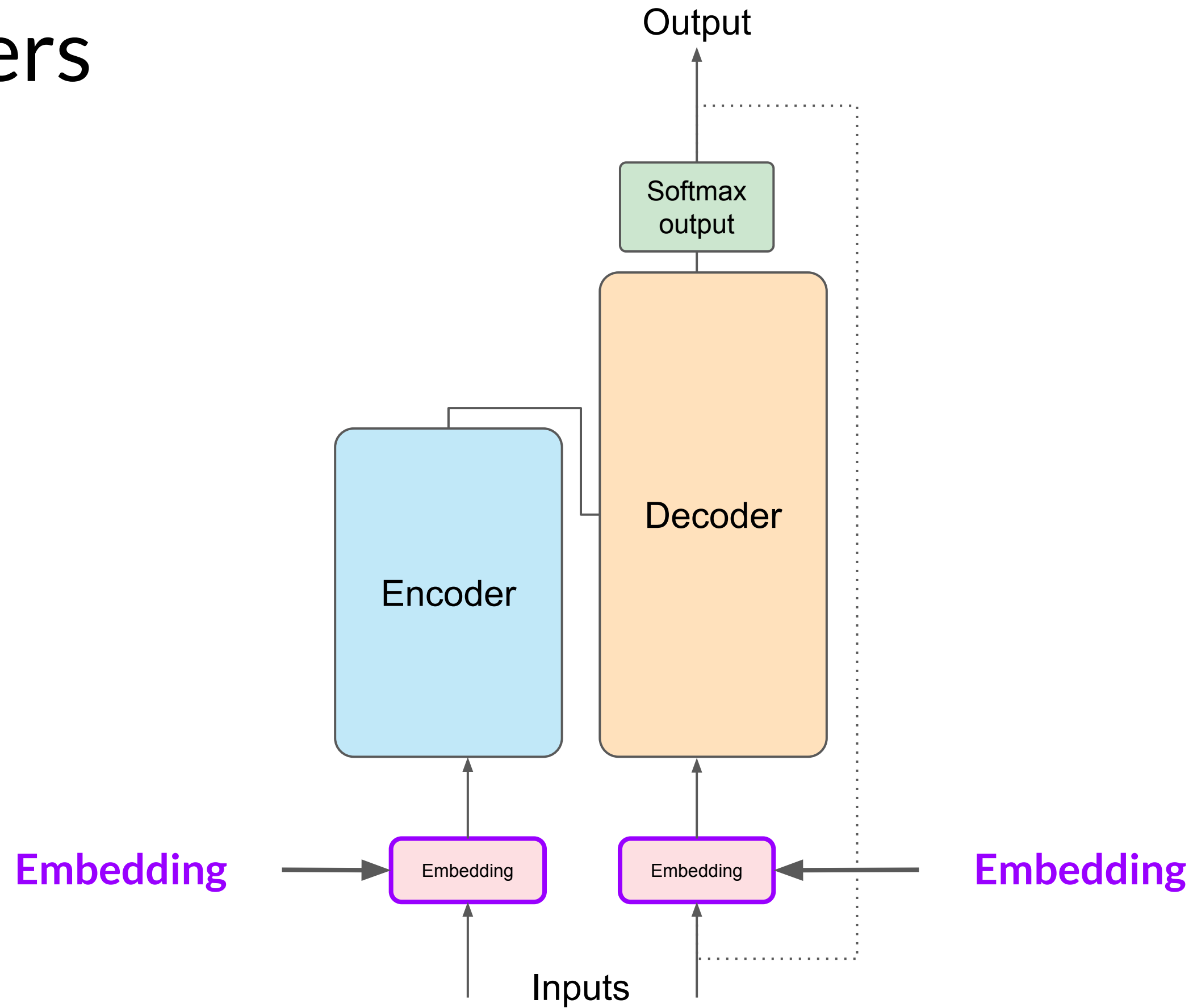
Transformers



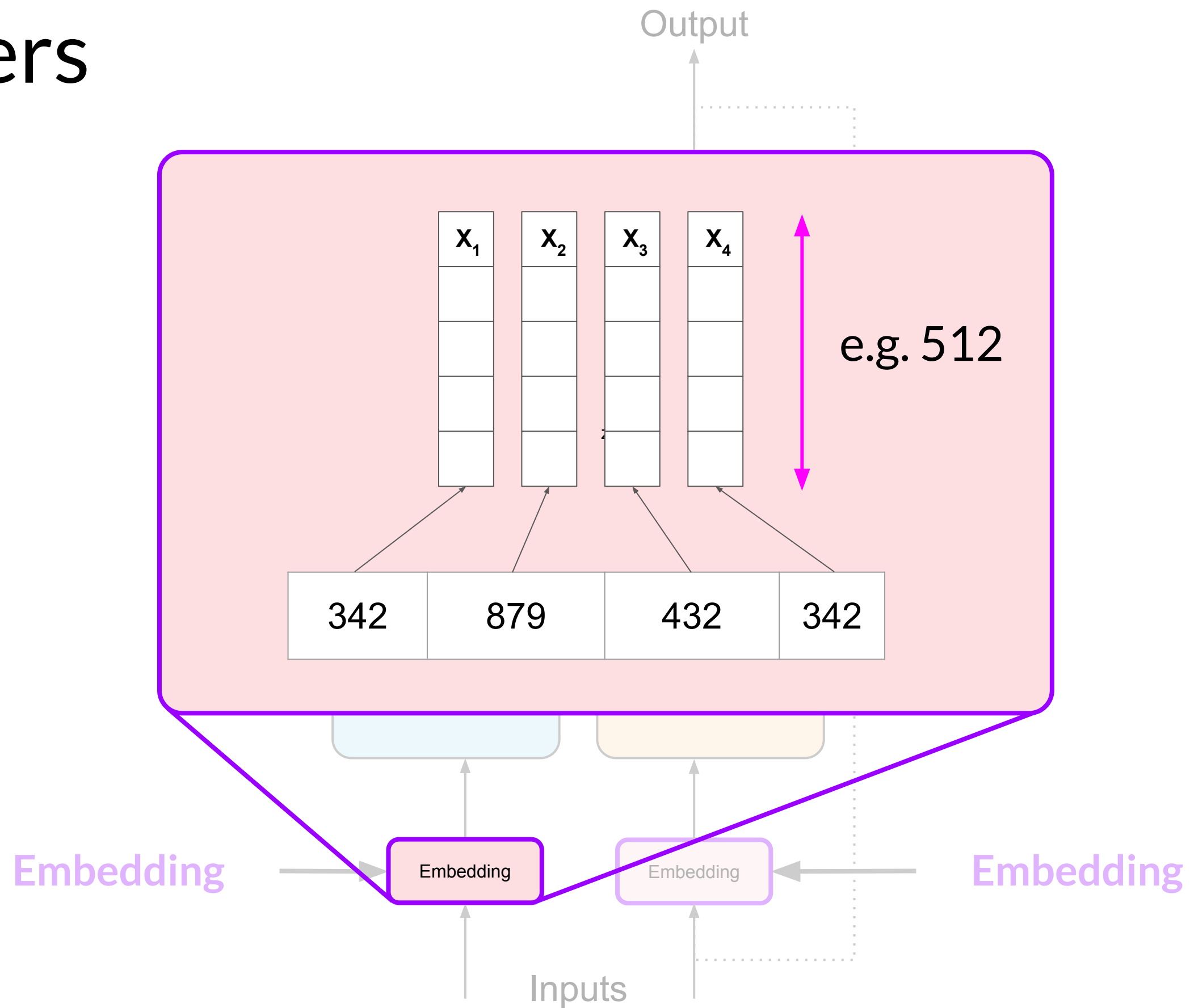
Transformers



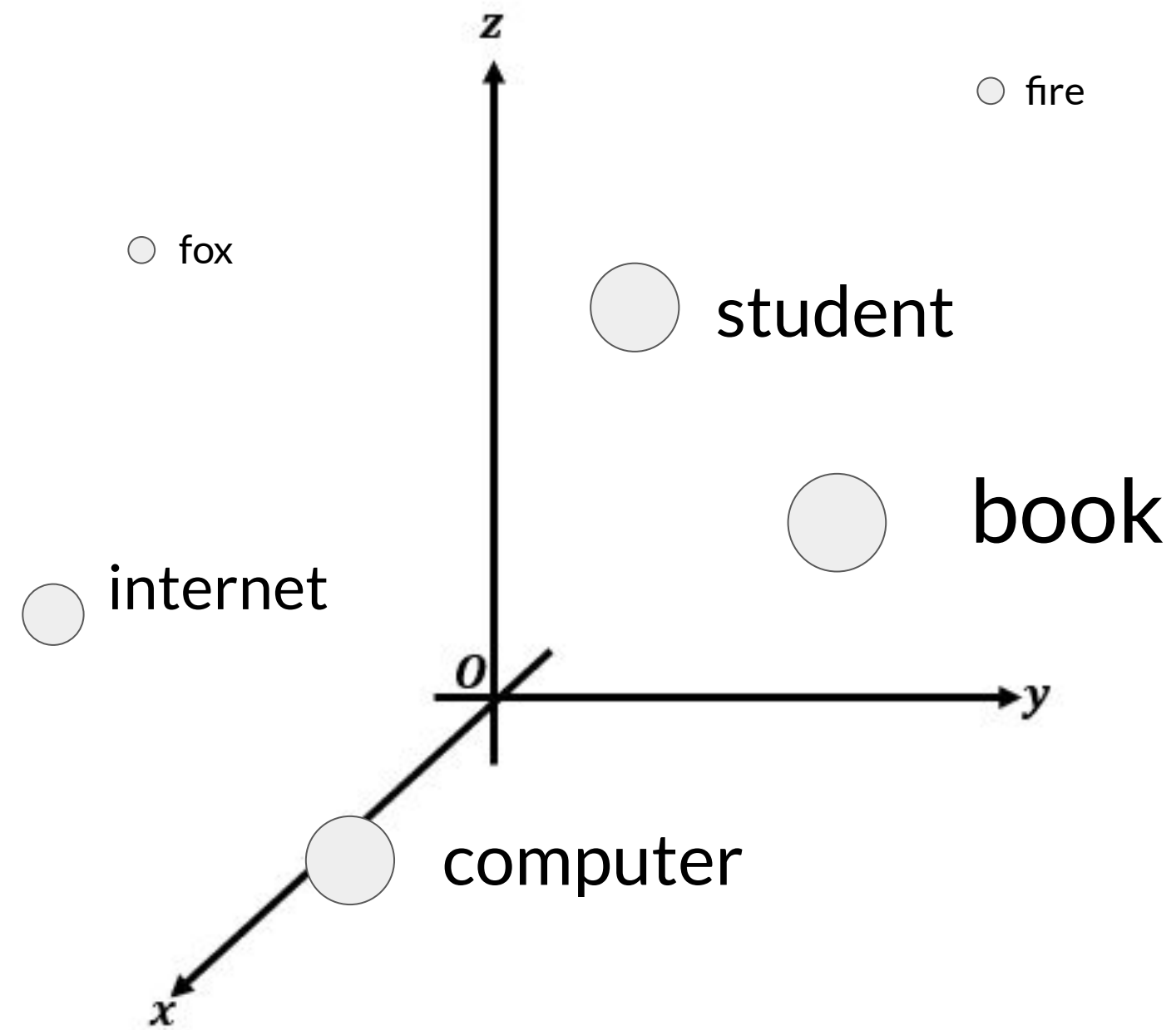
Transformers



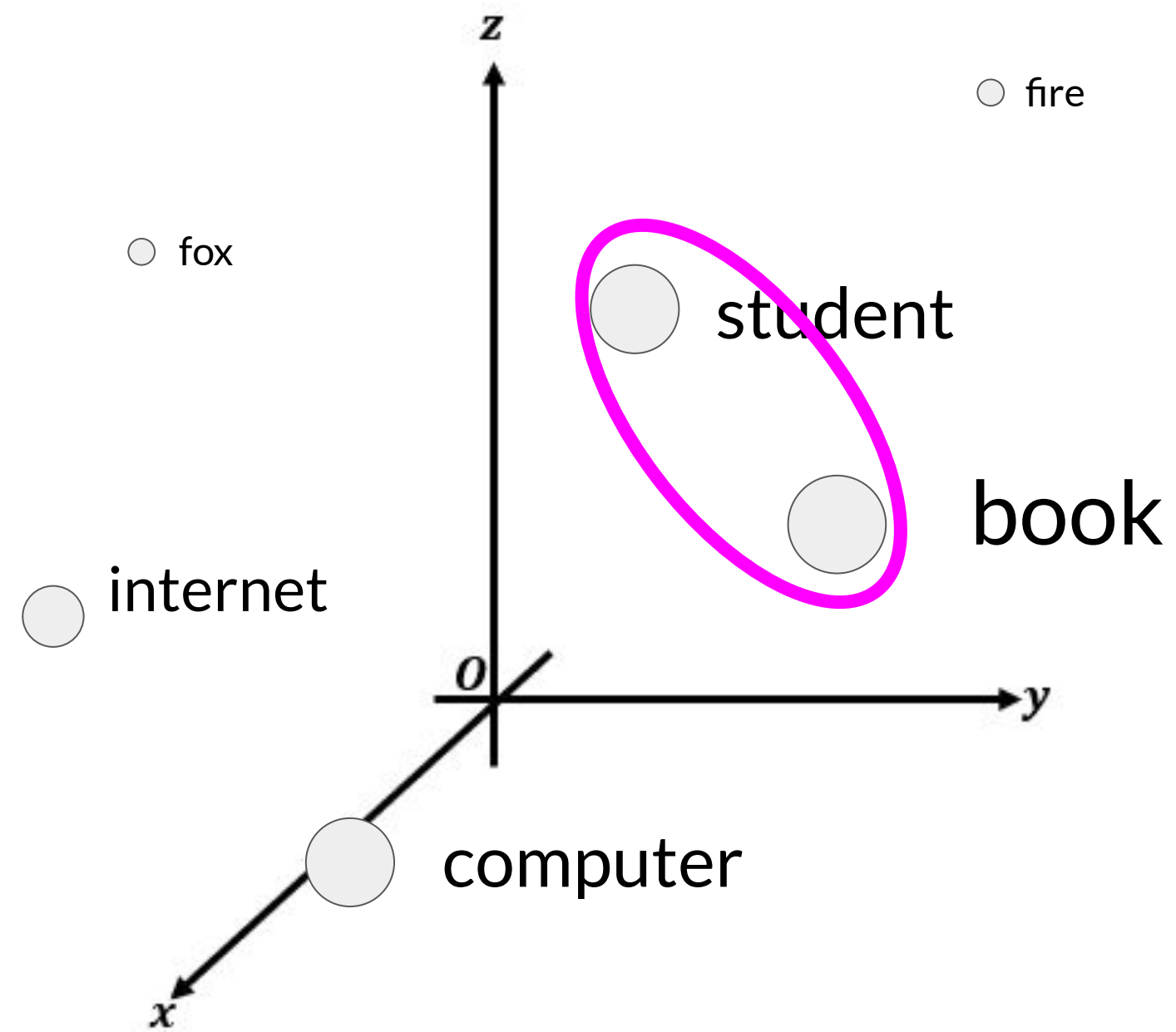
Transformers



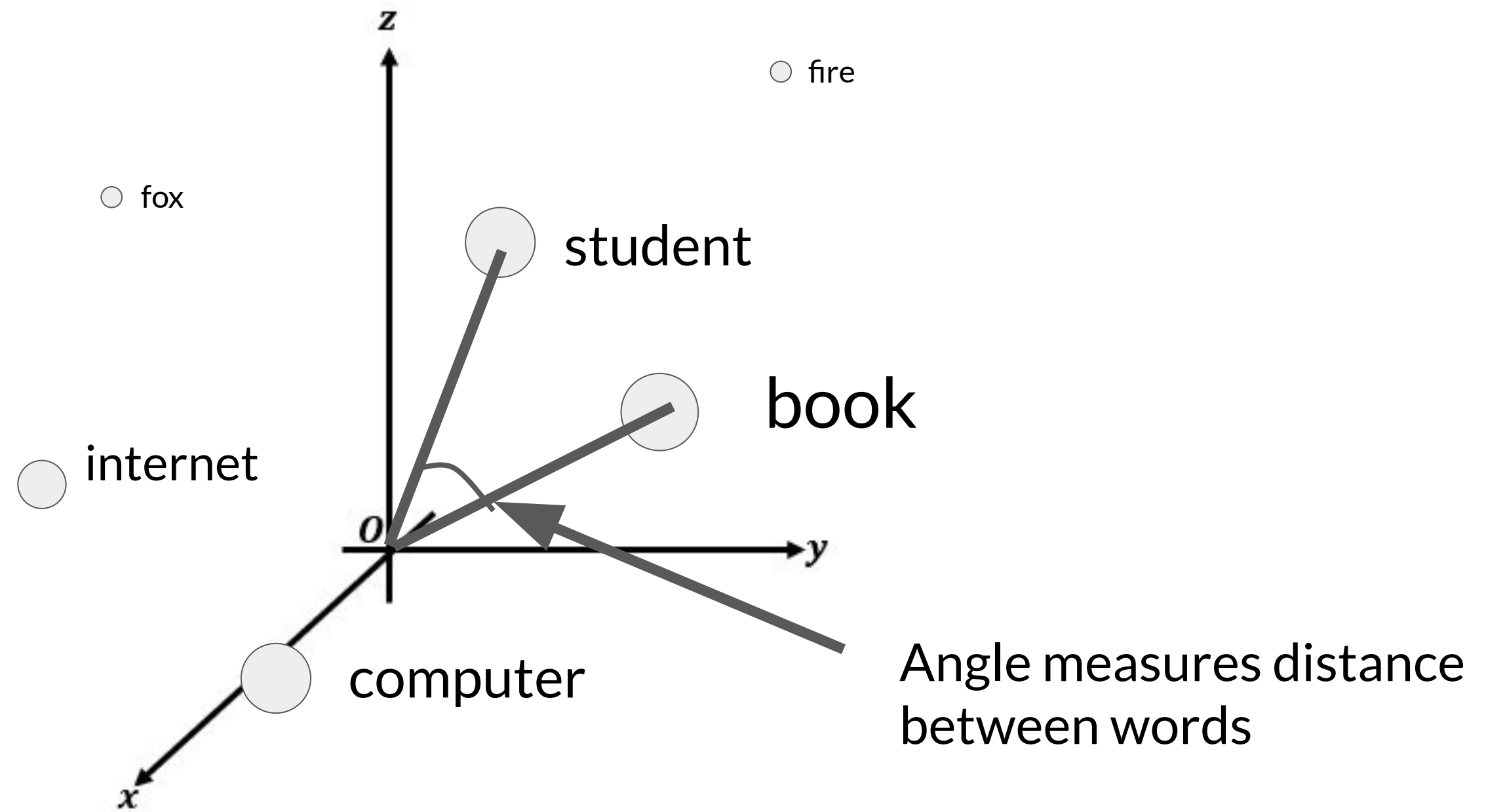
Transformers



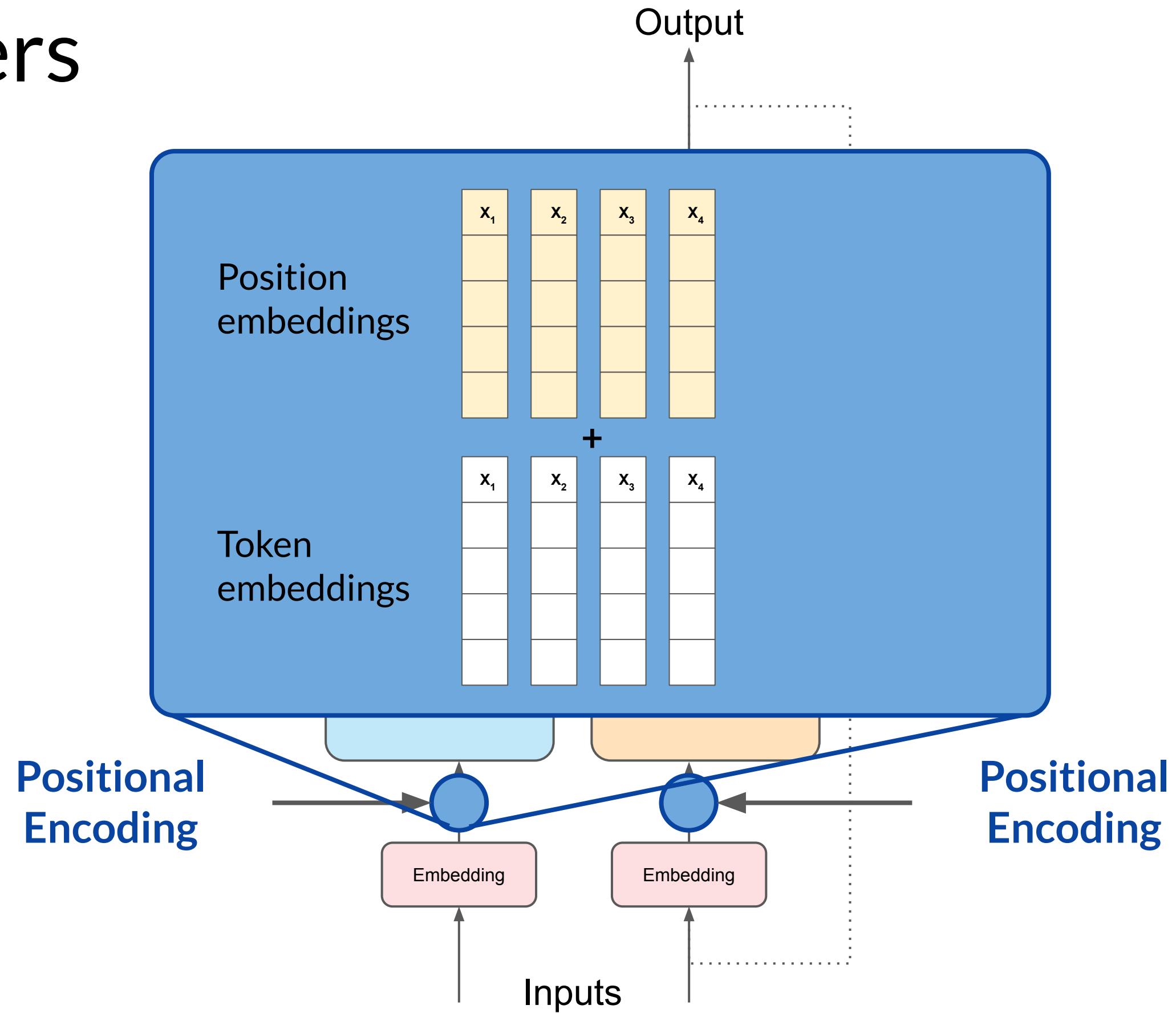
Transformers



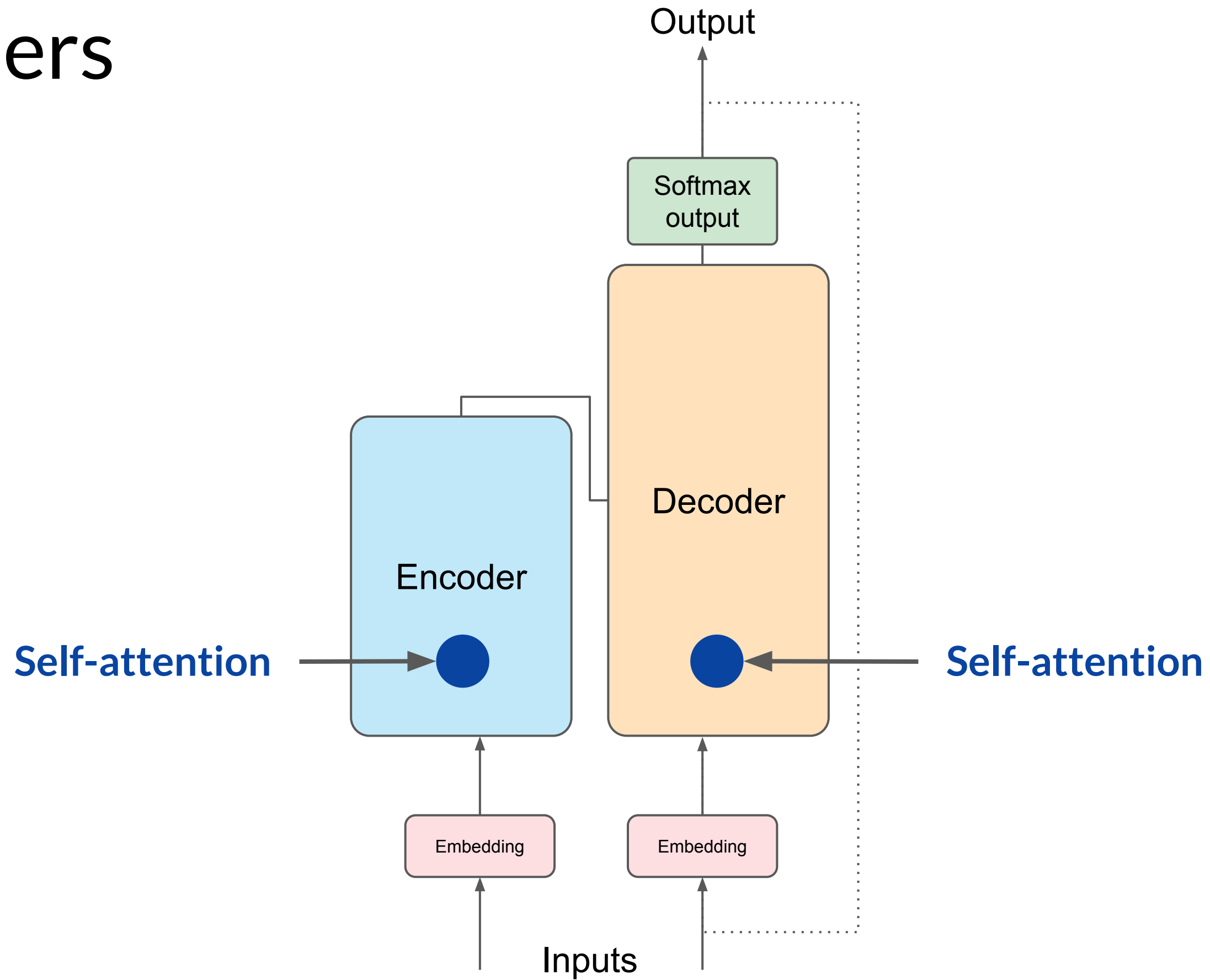
Transformers



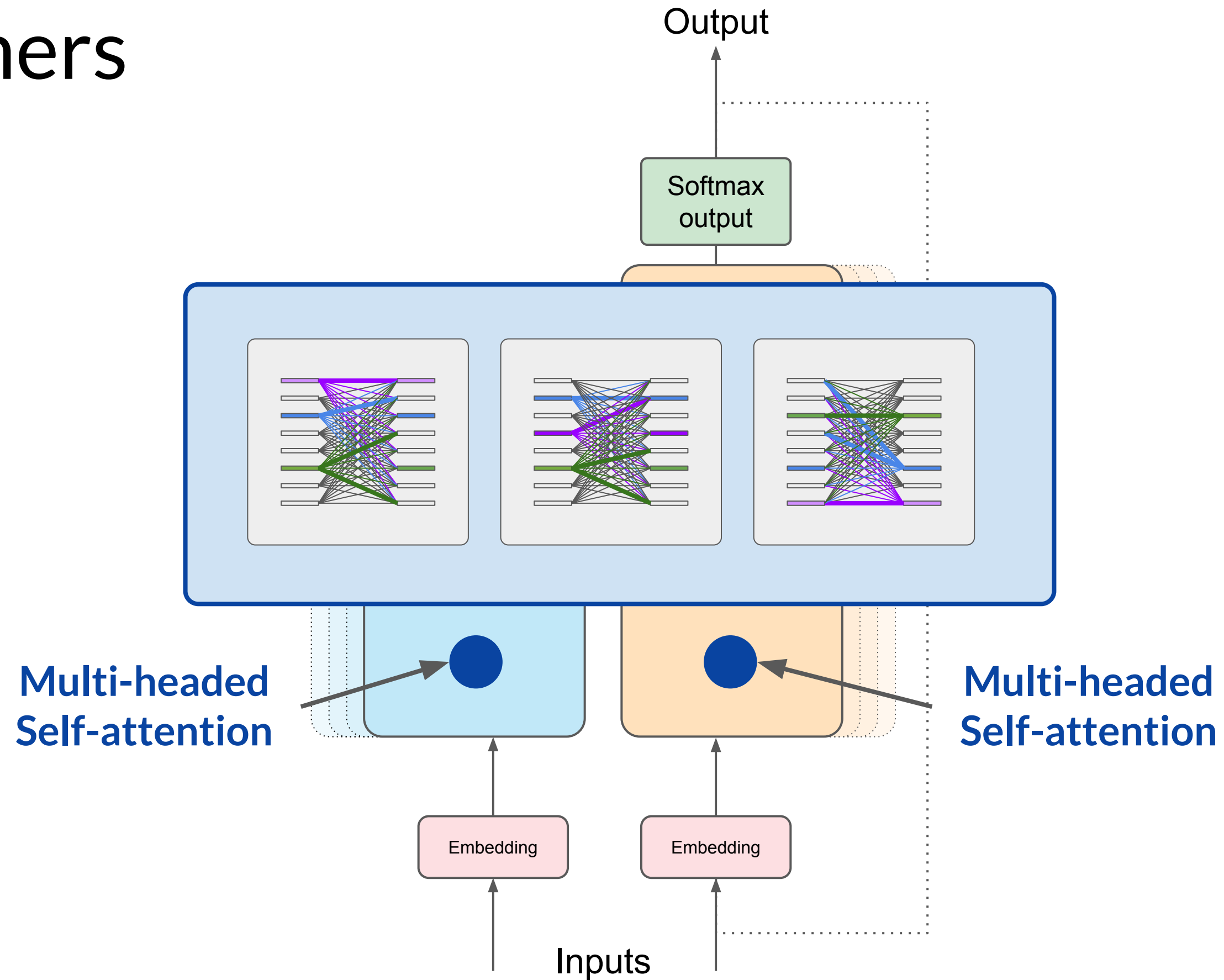
Transformers



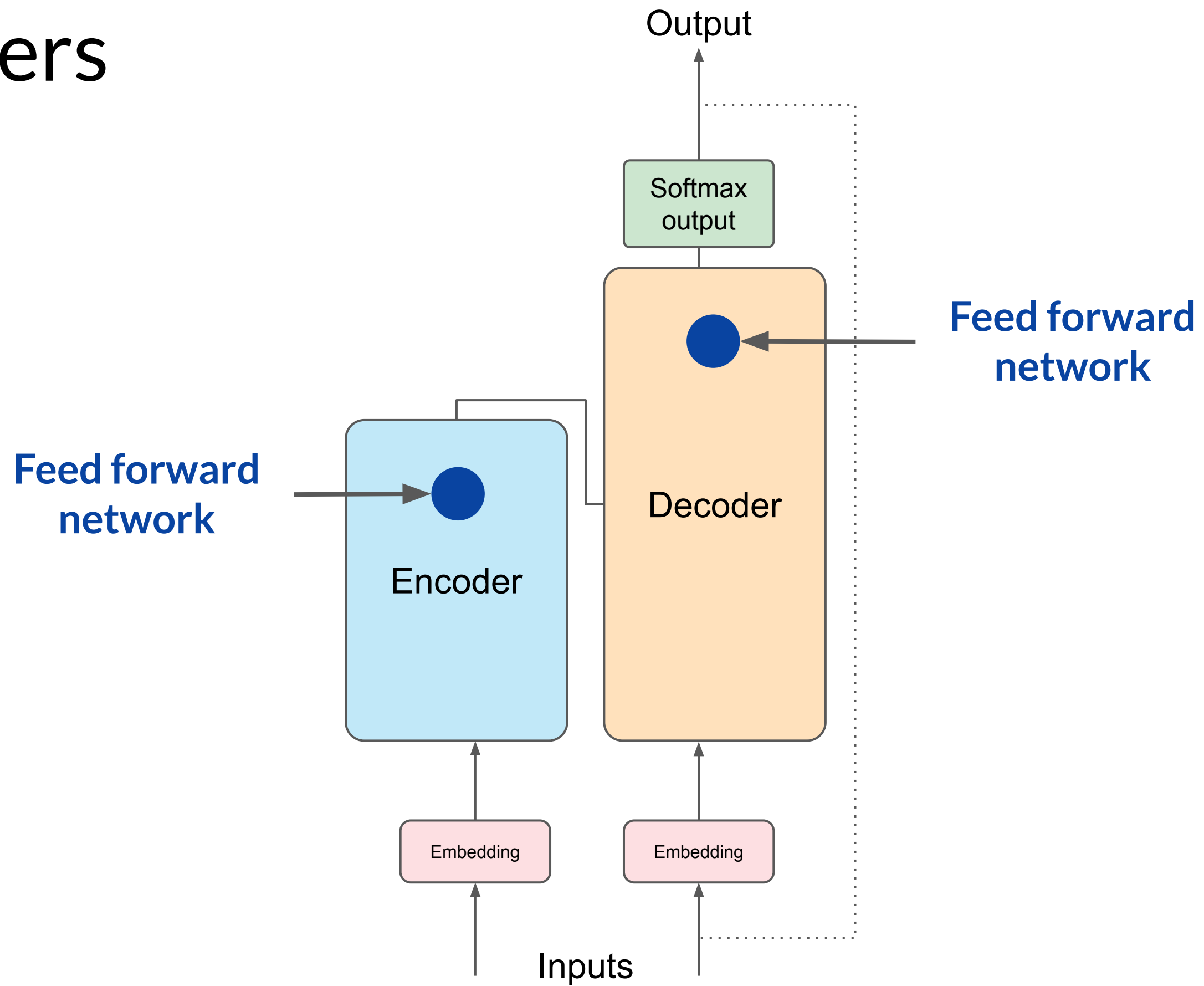
Transformers



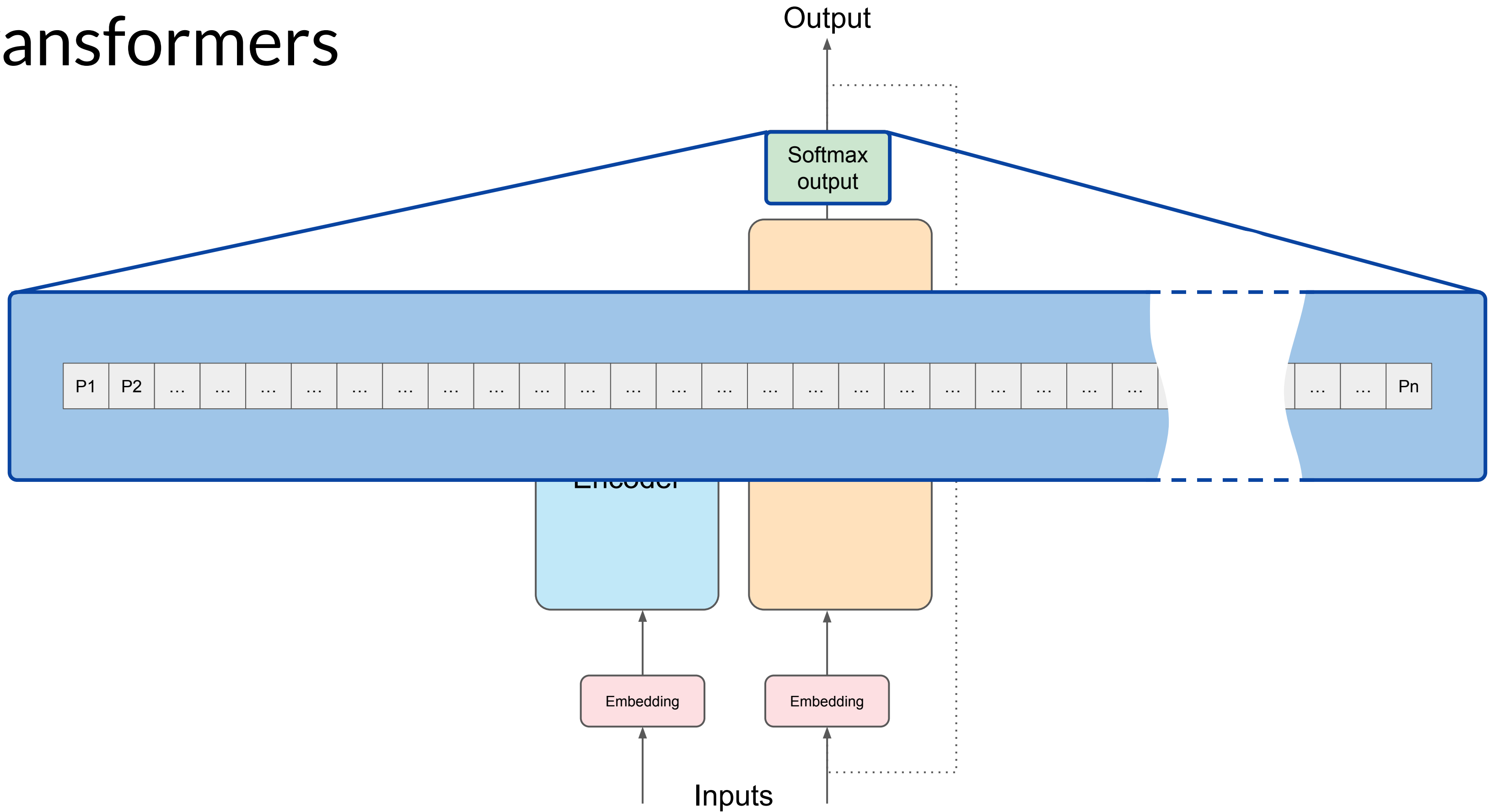
Transformers



Transformers



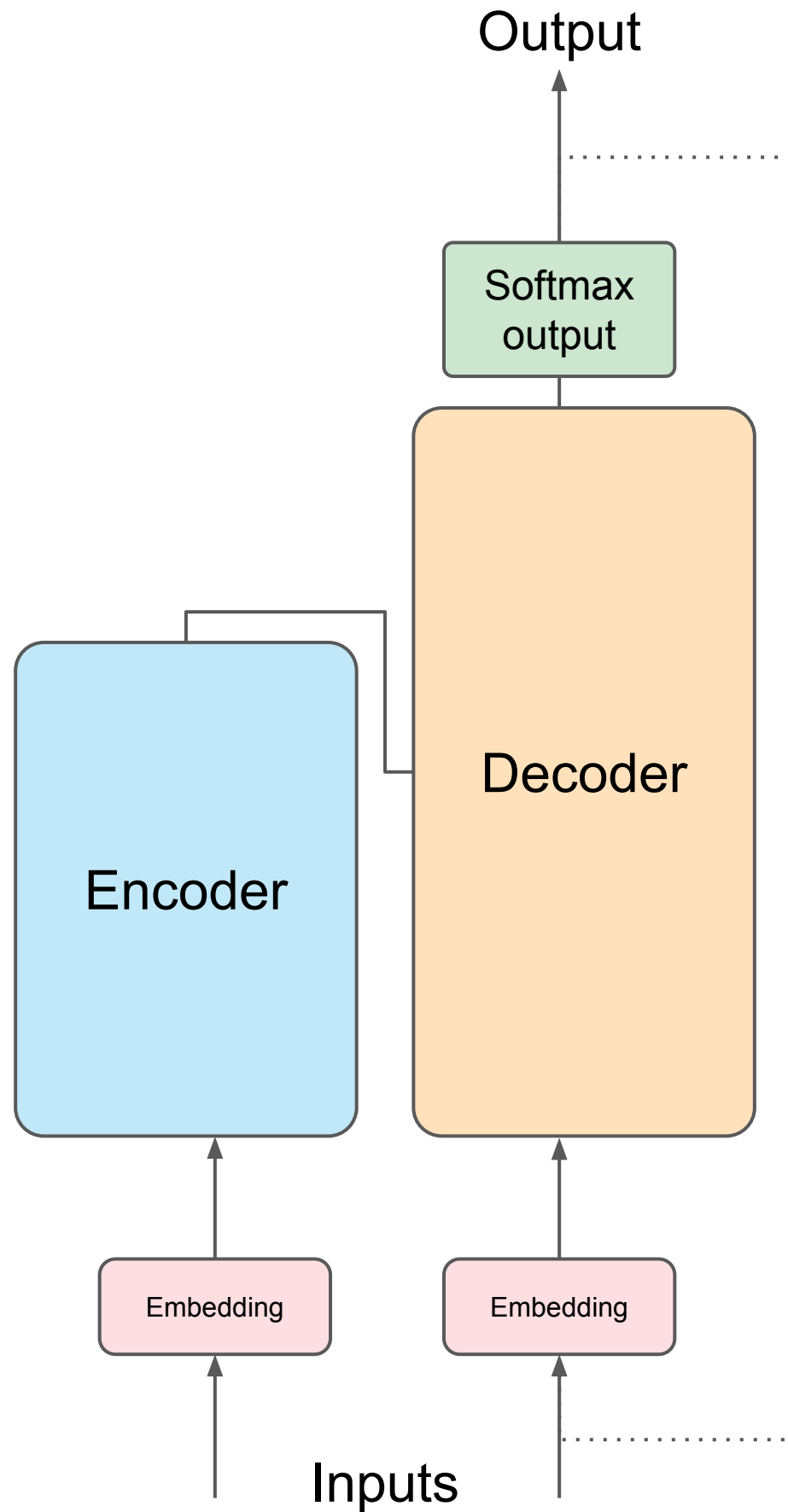
Transformers



Transformers

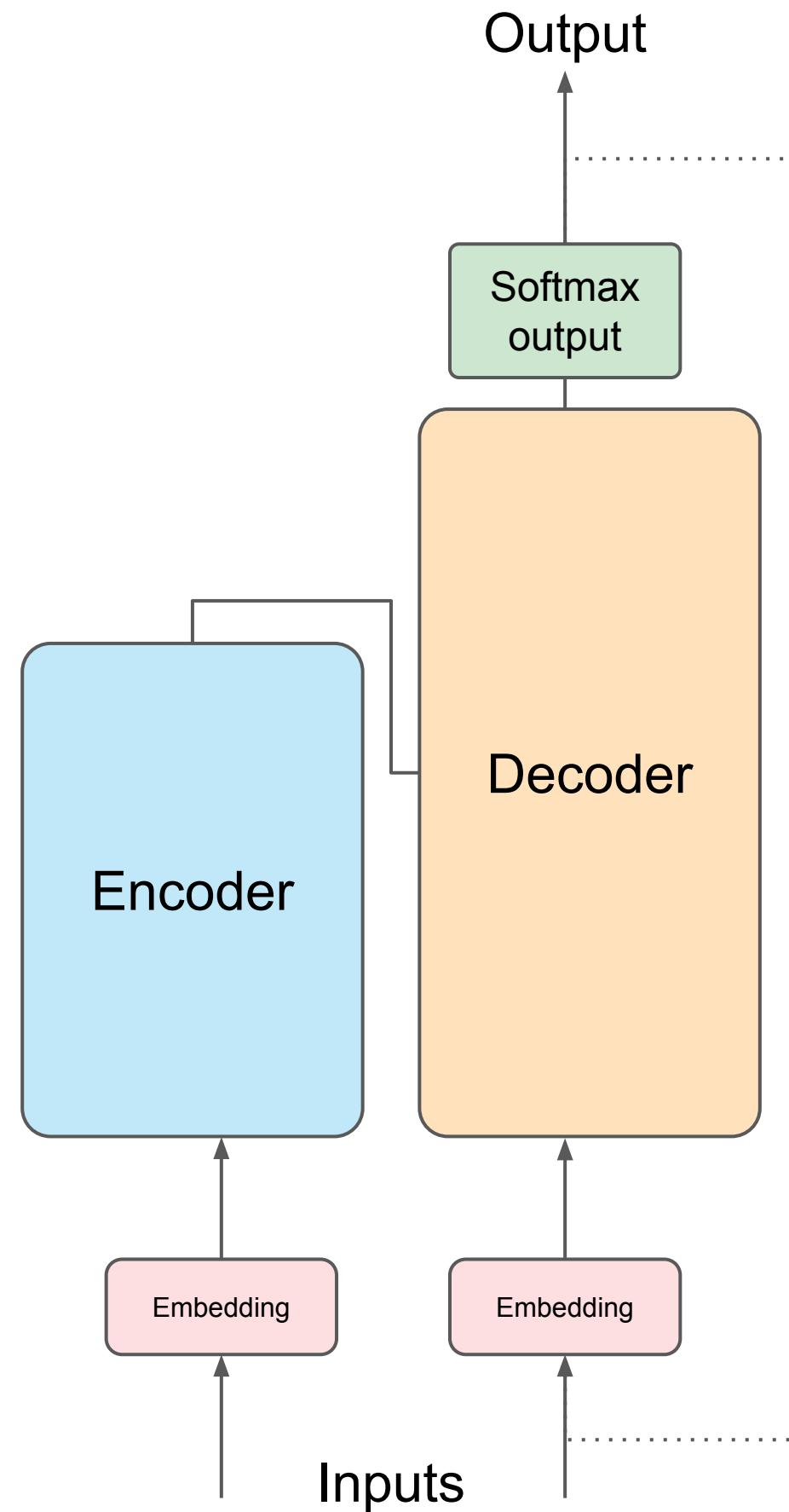
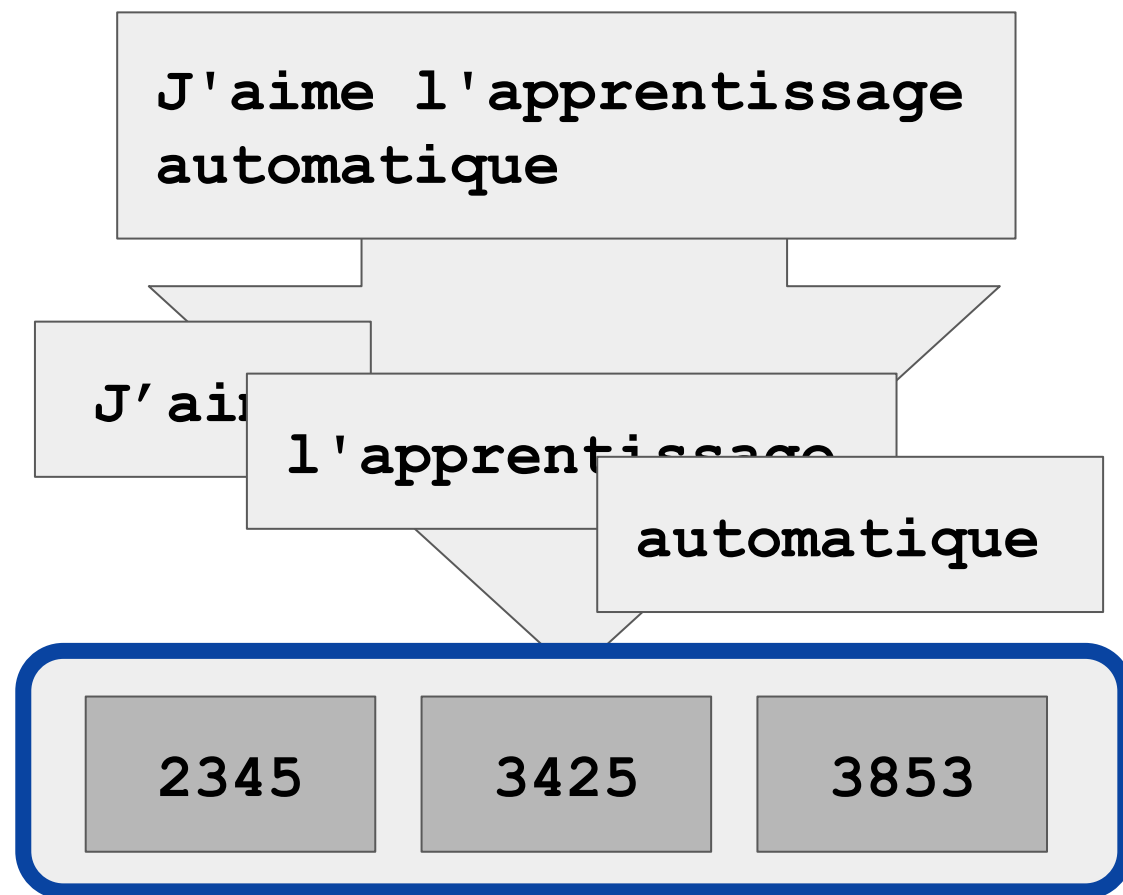
Translation:
sequence-to-sequence task

J'aime l'apprentissage automatique



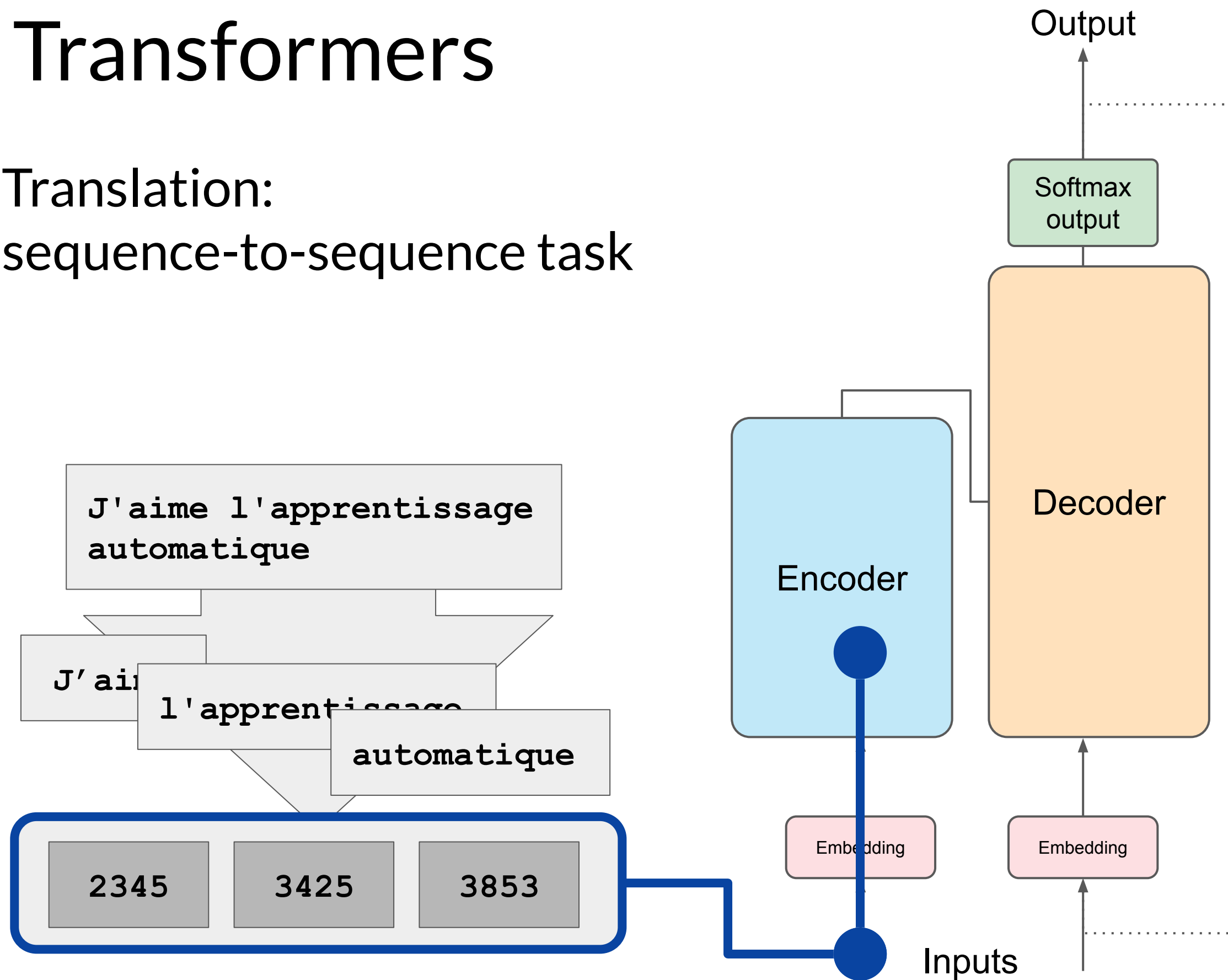
Transformers

Translation:
sequence-to-sequence task



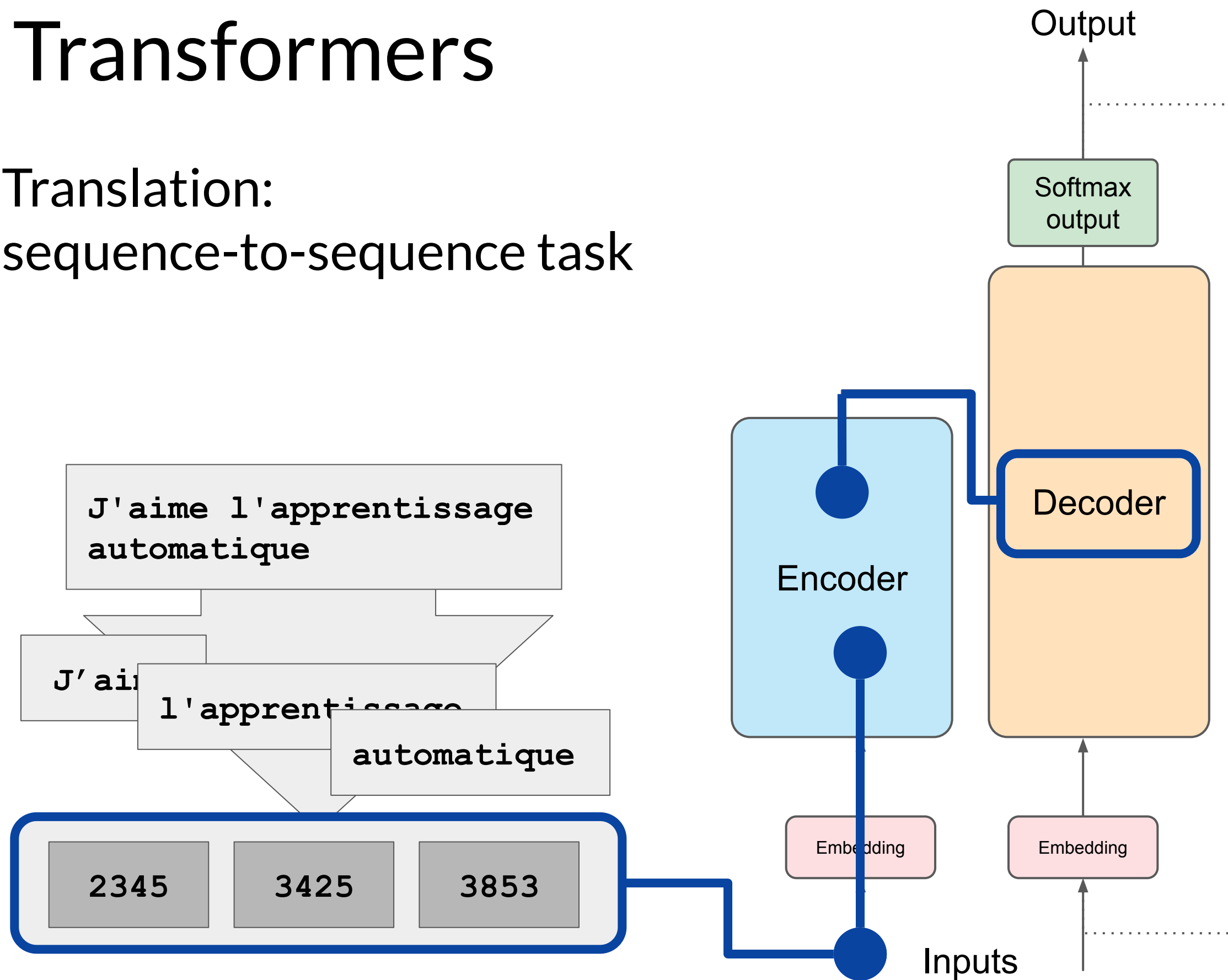
Transformers

Translation:
sequence-to-sequence task



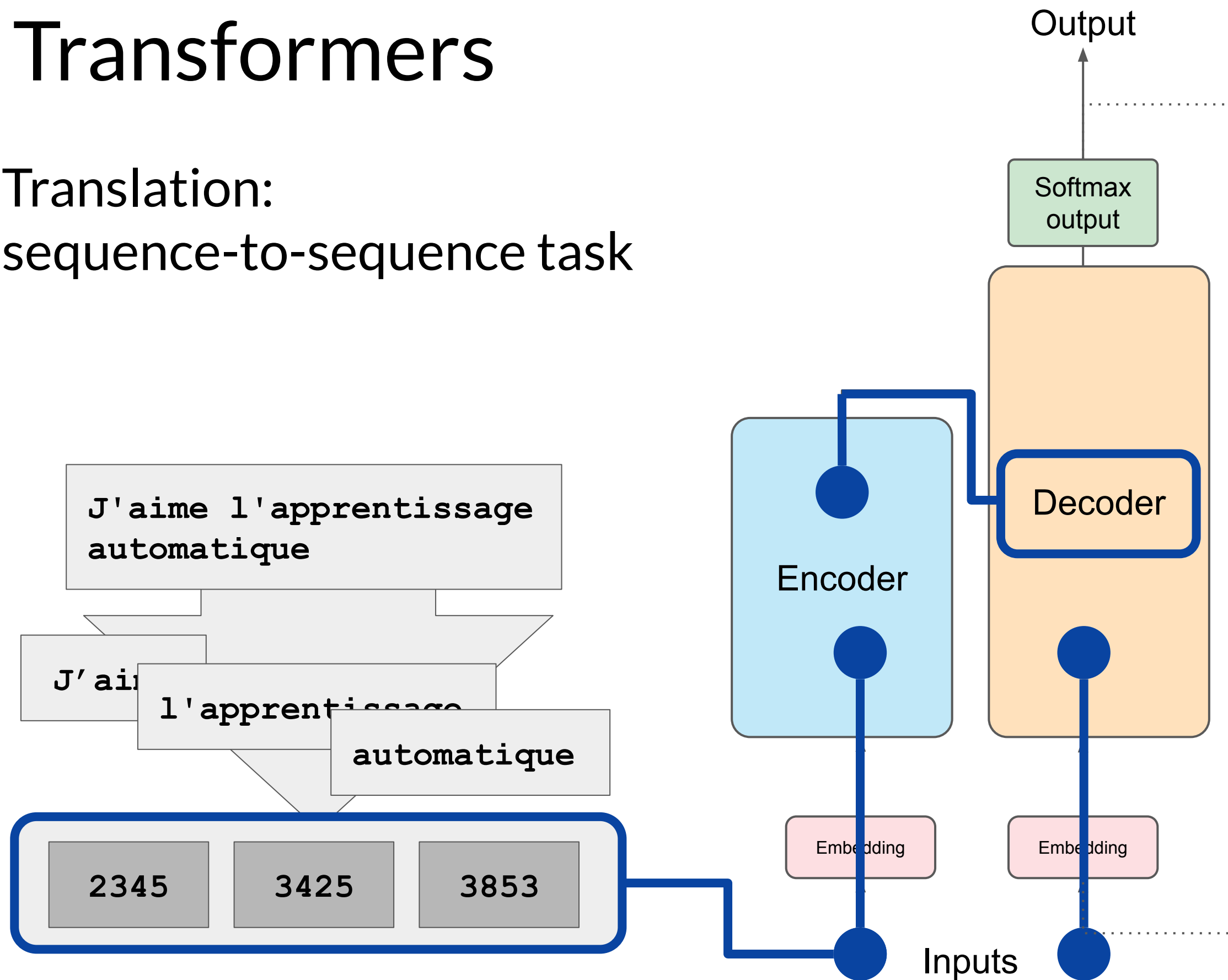
Transformers

Translation:
sequence-to-sequence task



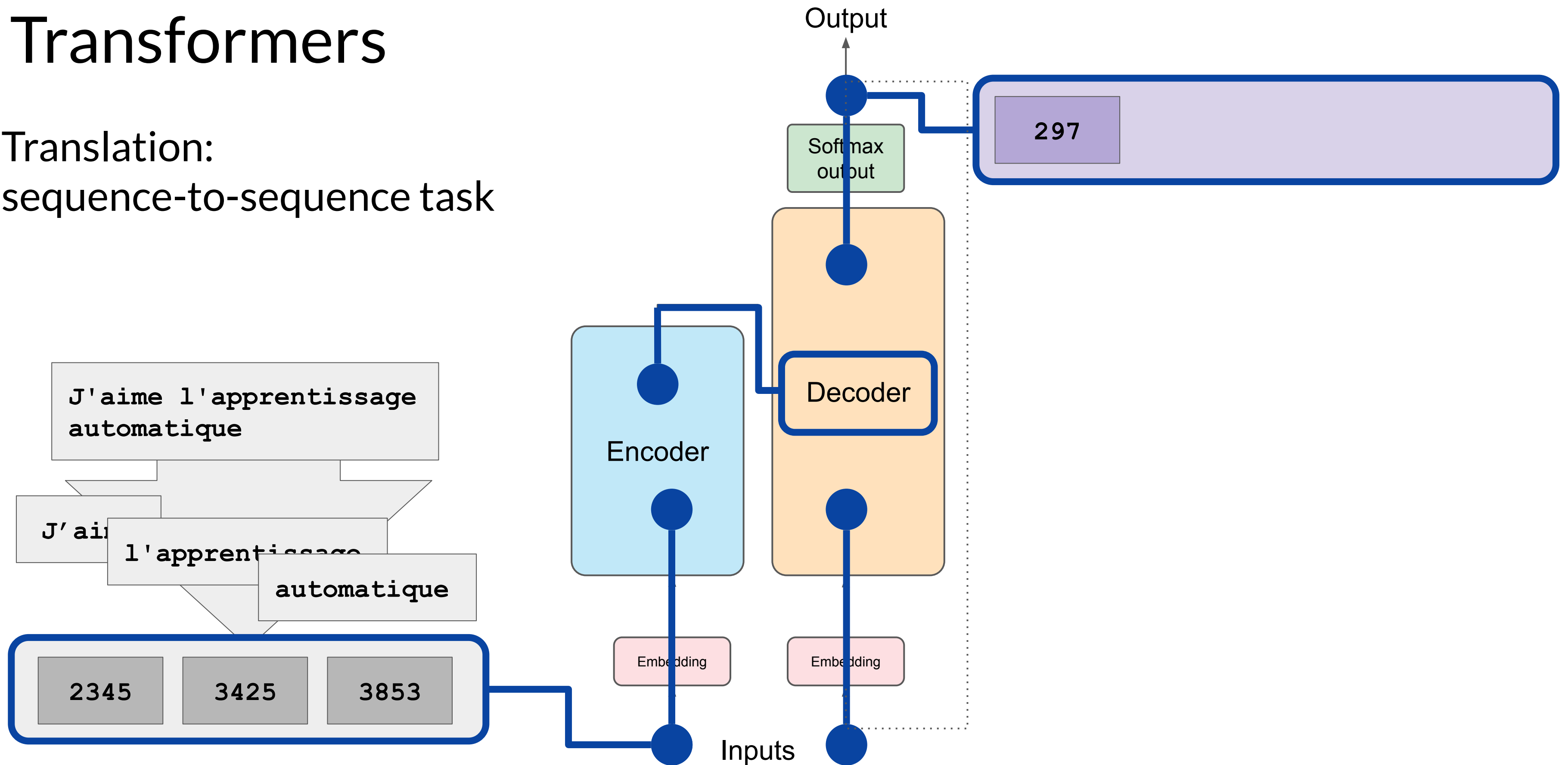
Transformers

Translation:
sequence-to-sequence task



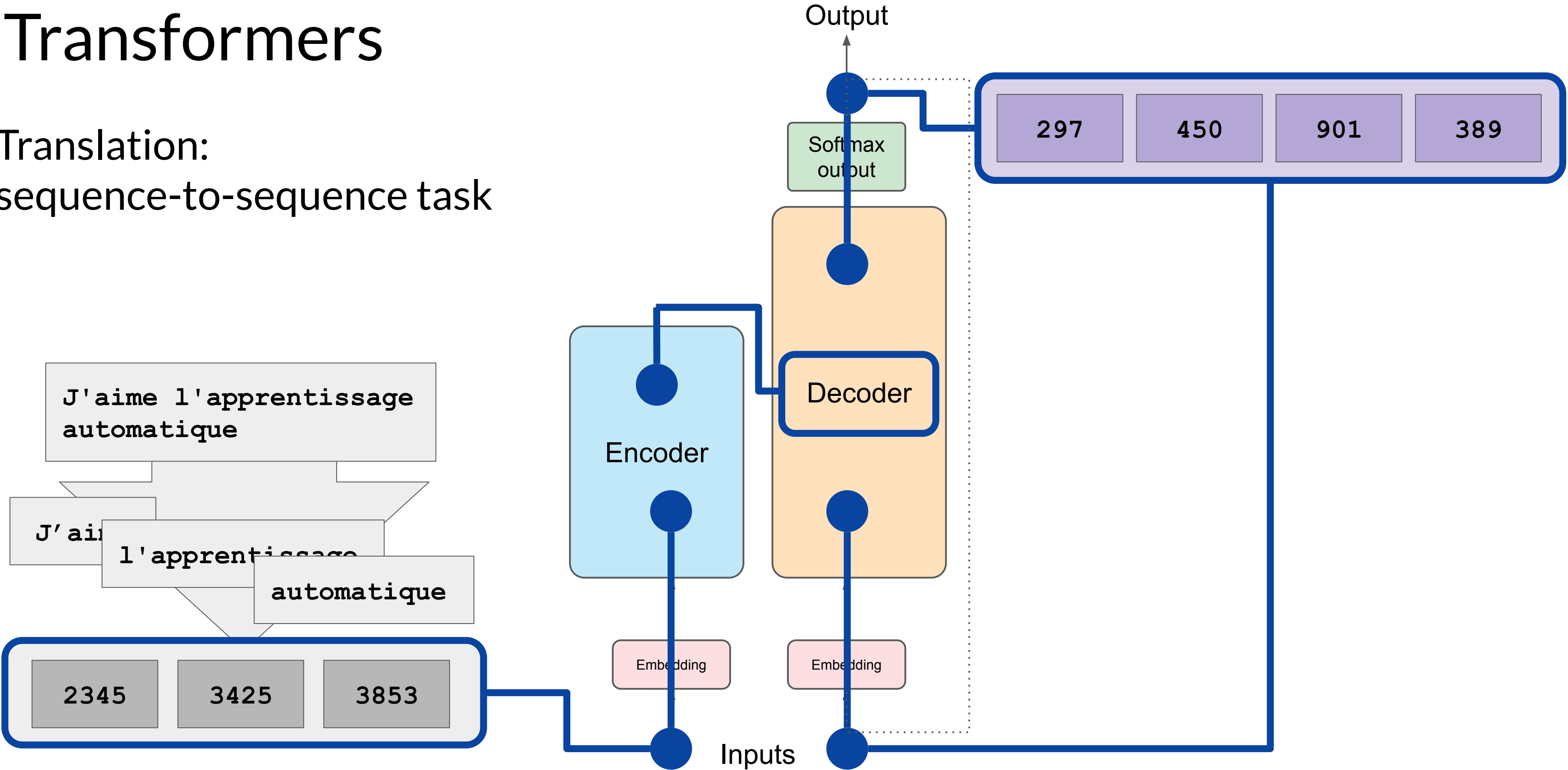
Transformers

Translation:
sequence-to-sequence task



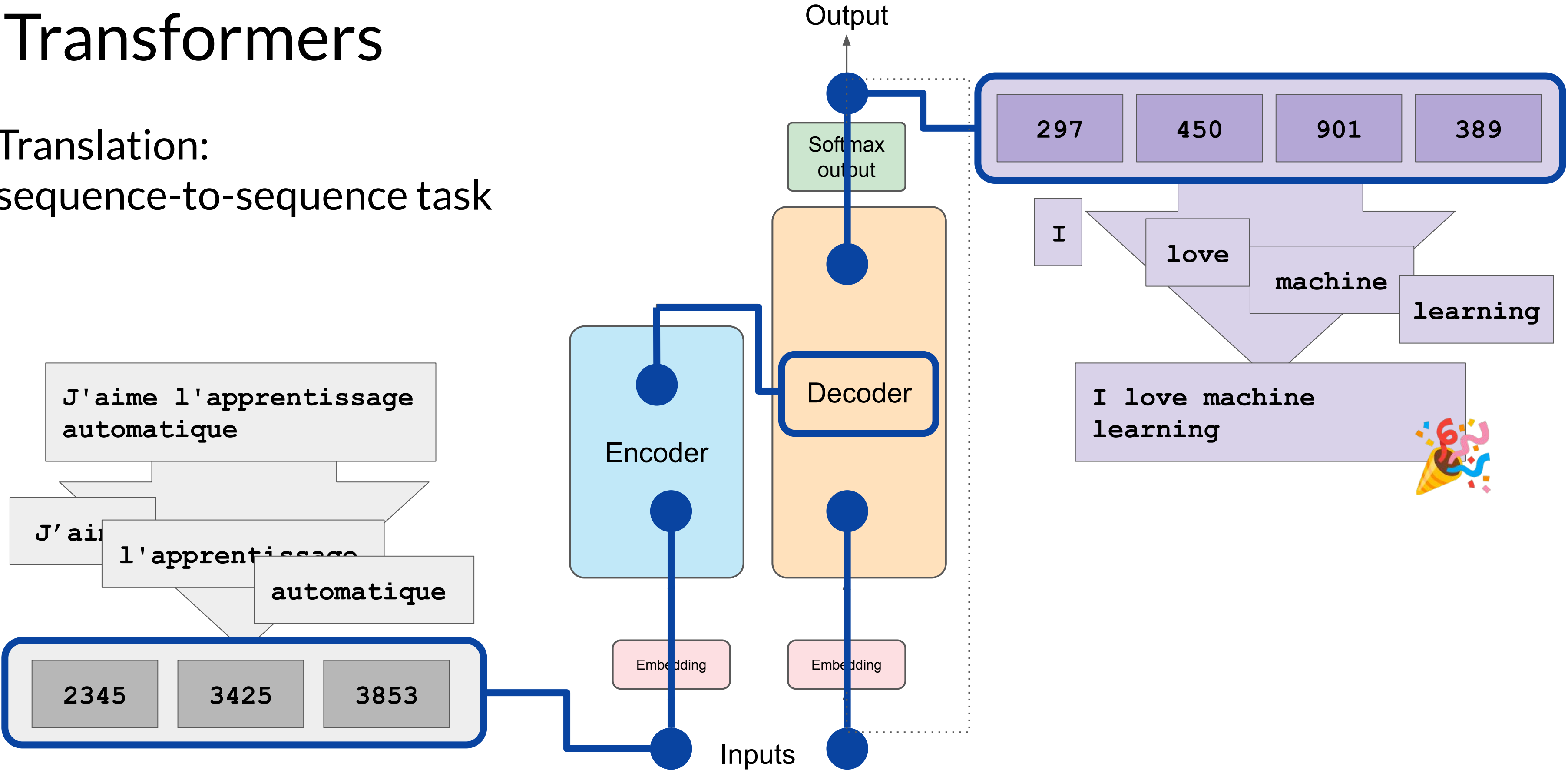
Transformers

Translation:
sequence-to-sequence task

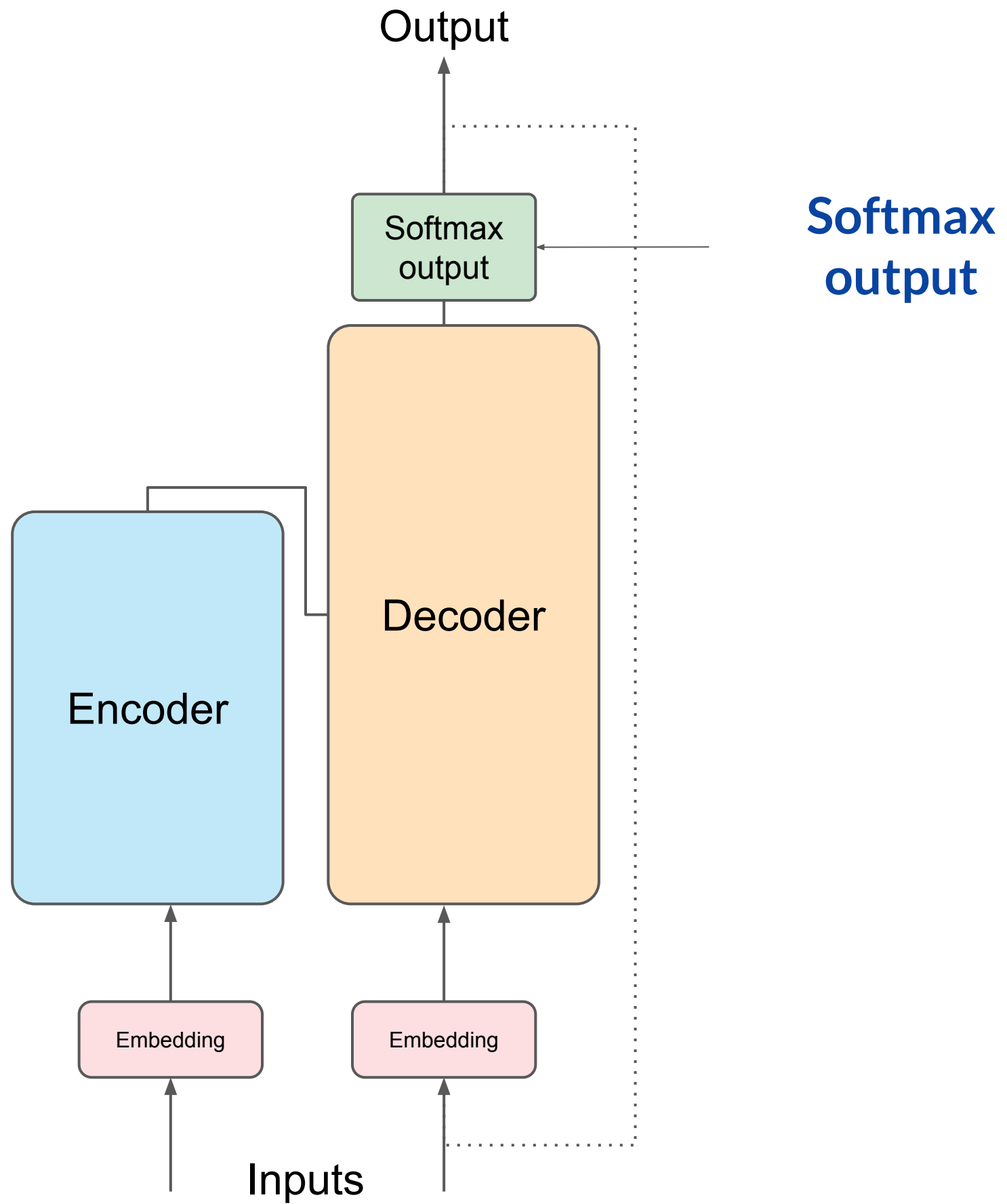


Transformers

Translation:
sequence-to-sequence task



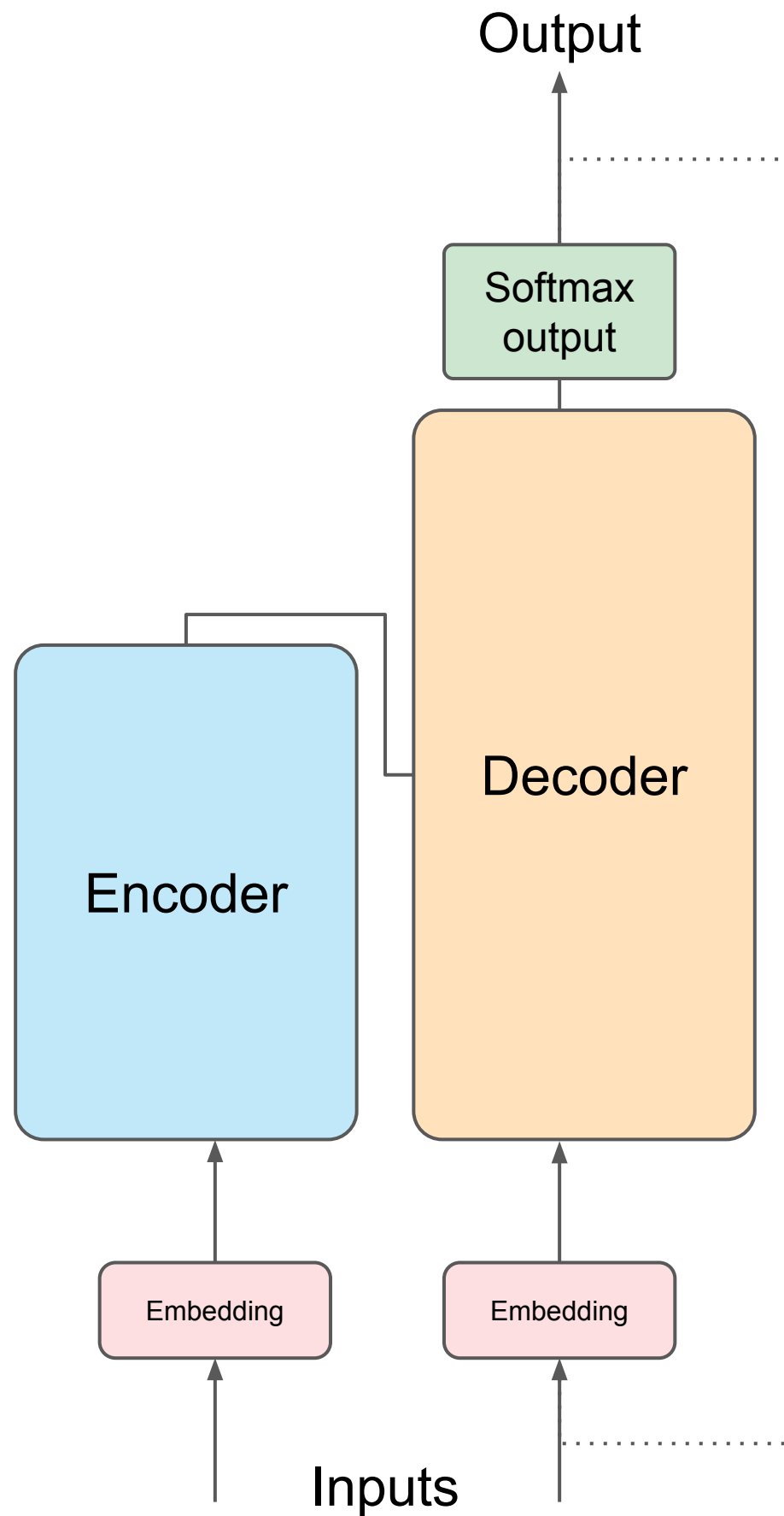
Transformers



Transformers

Encoder

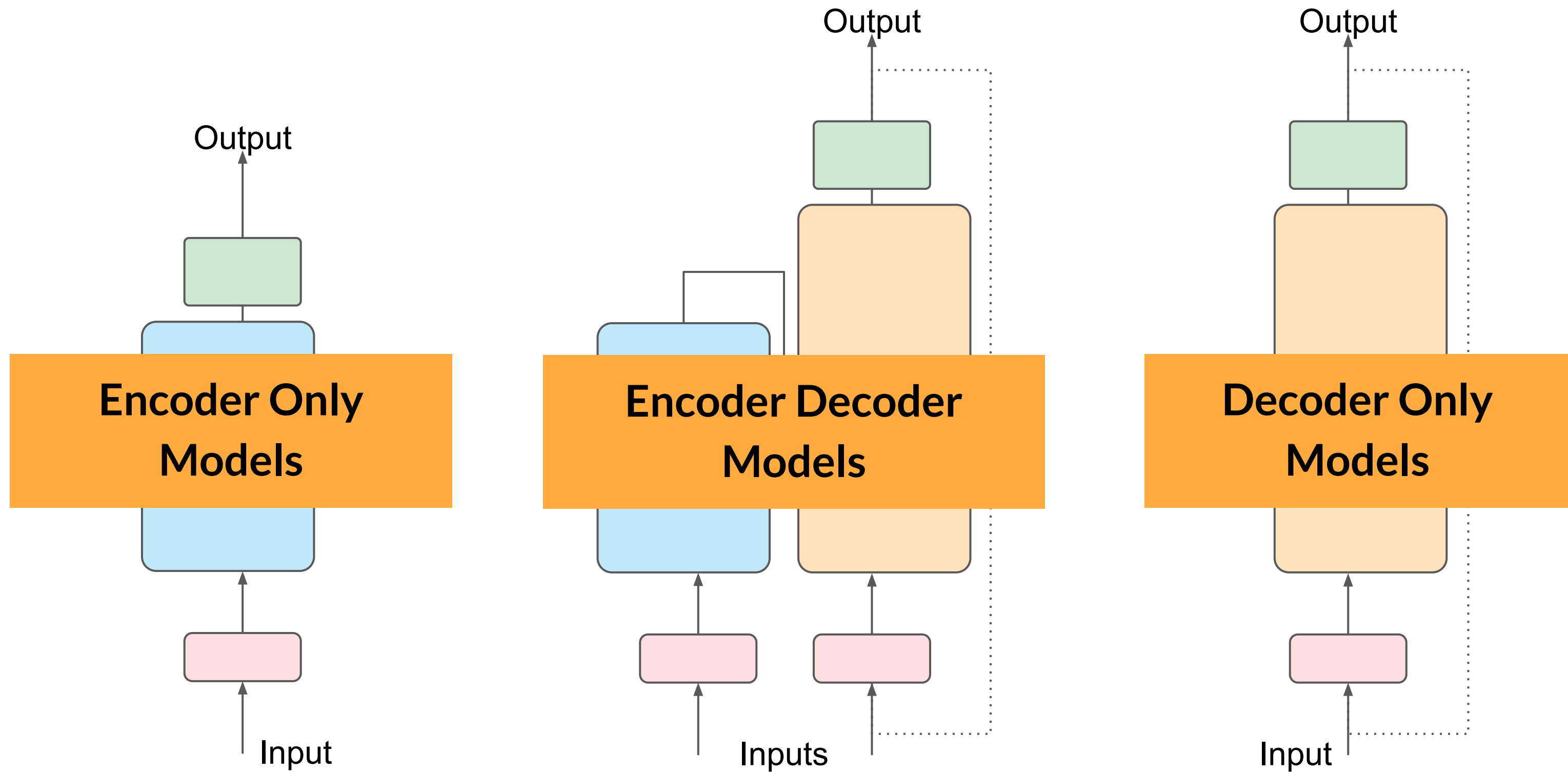
Encodes inputs (“prompts”) with contextual understanding and produces one vector per input token.



Decoder

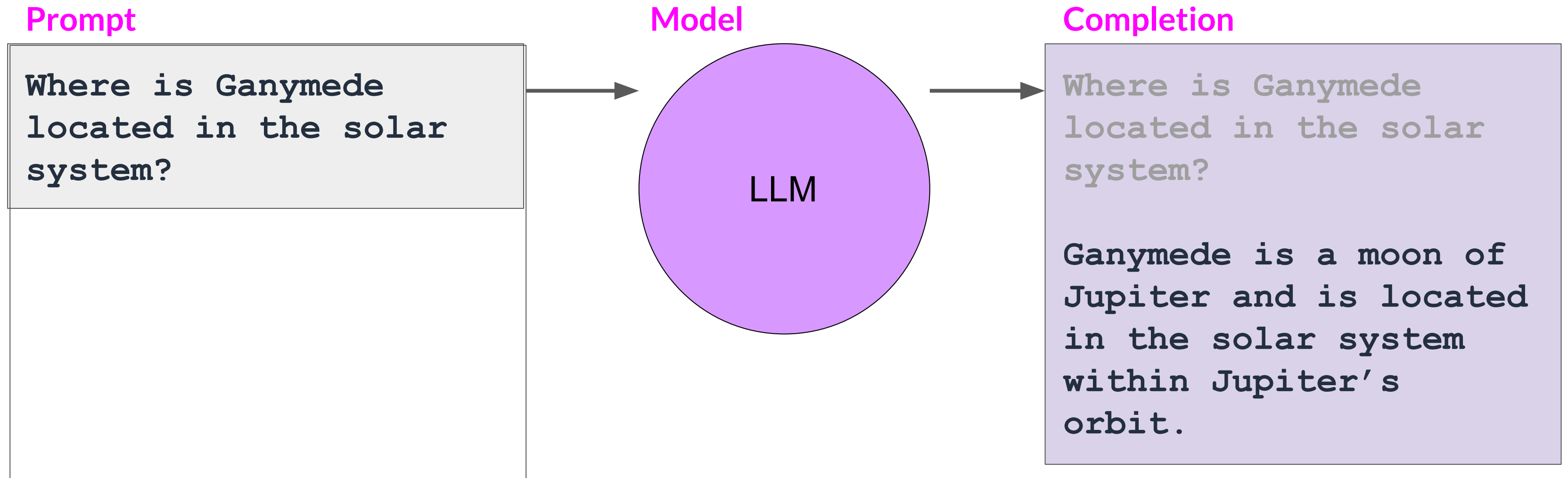
Accepts input tokens and generates new tokens.

Transformers



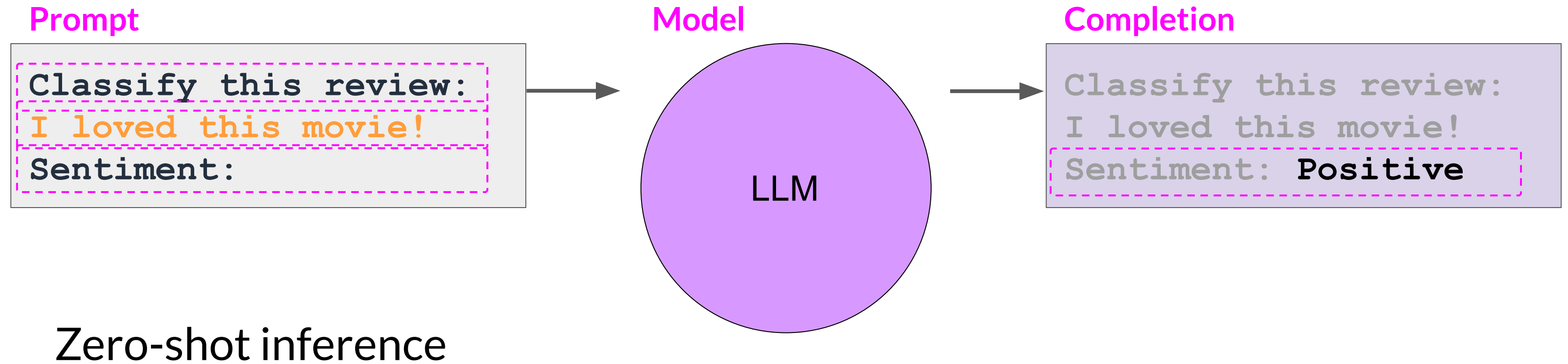
Prompting and prompt engineering

Prompting and prompt engineering

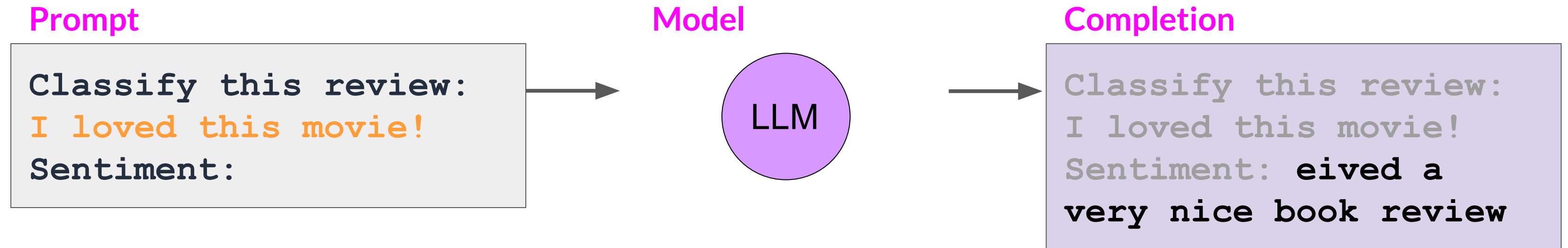


Context window: typically a few thousand words

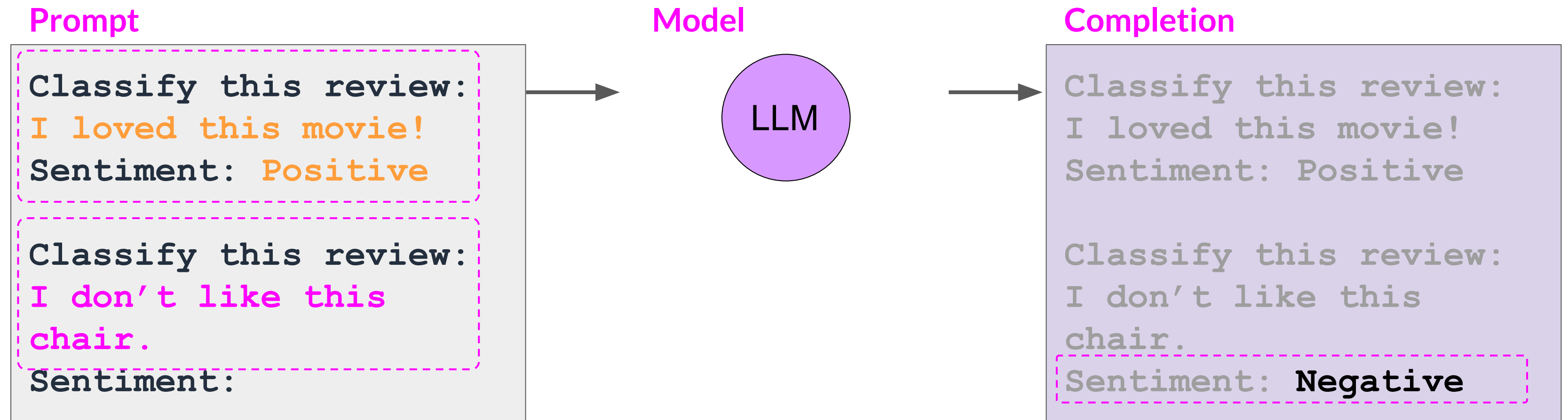
In-context learning (ICL) - zero shot inference



In-context learning (ICL) - zero shot inference

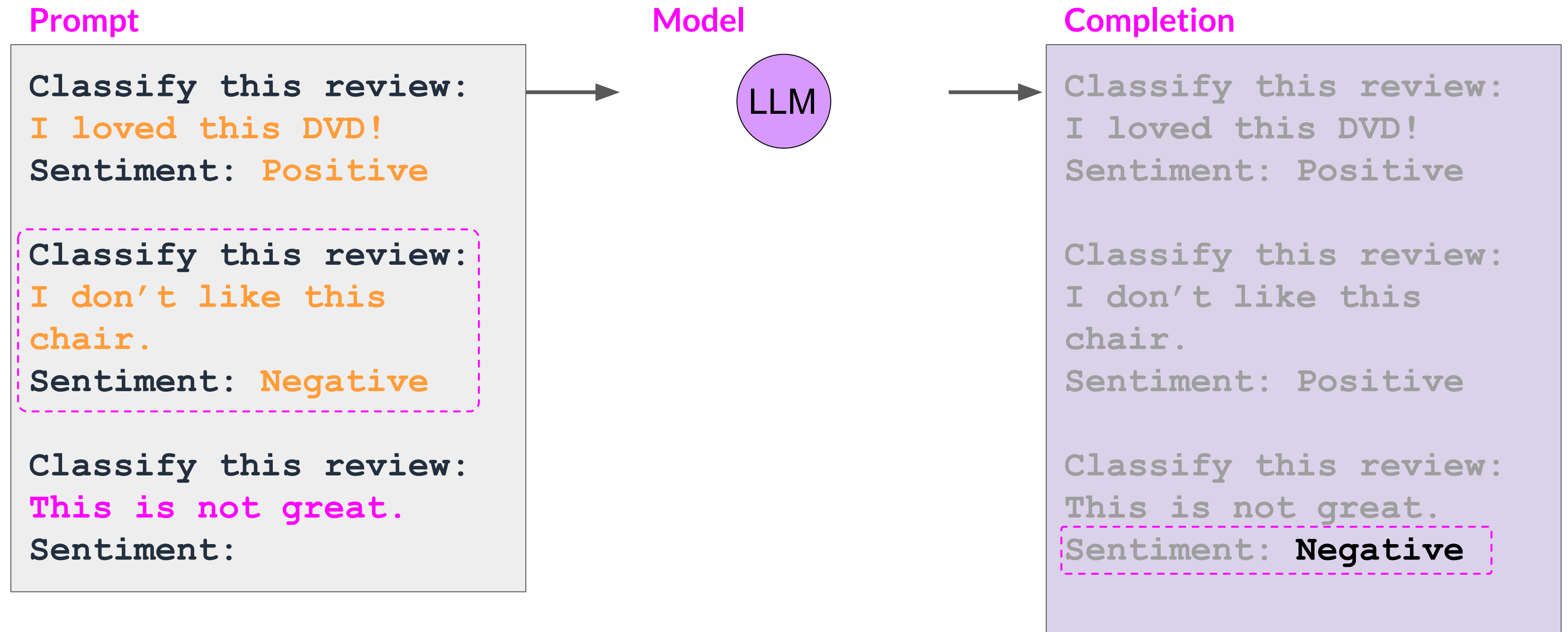


In-context learning (ICL) - one shot inference



One-shot inference

In-context learning (ICL) - few shot inference



Summary of in-context learning (ICL)

Prompt // Zero Shot

Classify this review:
I loved this movie!
Sentiment:

Prompt // One Shot

Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment:

Prompt // Few Shot >5 or 6 examples

Classify this review:
I loved this movie!
Sentiment: Positive

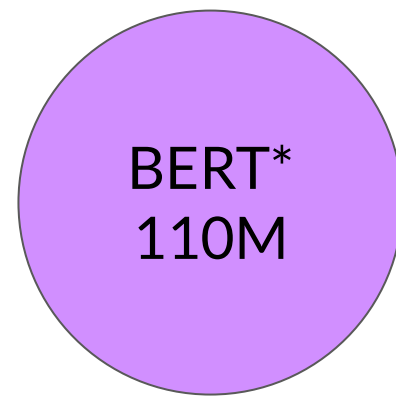
Classify this review:
I don't like this
chair.
Sentiment: Negative

Classify this review:
Who would use this
product?
Sentiment:

Context Window
(few thousand words)

The significance of scale: task ability

BERT*
110M



BLOOM
176B →

*Bert-base

Generative configuration parameters for inference

Generative configuration - inference parameters

The image shows a user interface for configuring generative AI inference parameters. On the left is a large text input area with the placeholder text "Enter your prompt here...". To the right is a configuration panel with four sliders and a "Submit" button. The sliders are: "Max new tokens" (set to 200), "Sample top K" (set to 25), "Sample top P" (set to 1), and "Temperature" (set to 0.8). A dashed pink box highlights the four sliders, and a bracket on the right labels this section as "Inference configuration parameters".

Parameter	Value
Max new tokens	200
Sample top K	25
Sample top P	1
Temperature	0.8

Generative configuration - max new tokens

Enter your prompt here...

Max new tokens

Sample top K

Sample top P

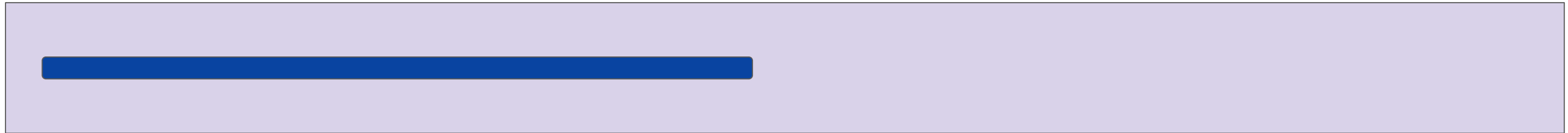
Temperature

Submit

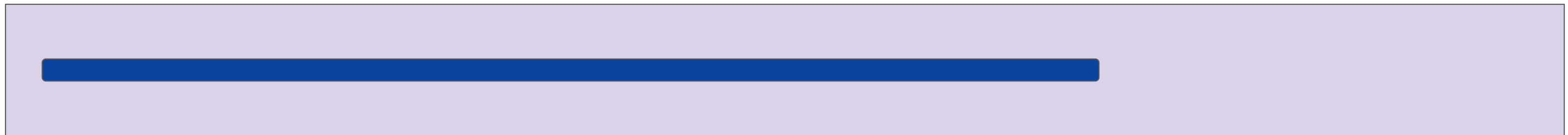
Max new tokens

Generative config - max new tokens

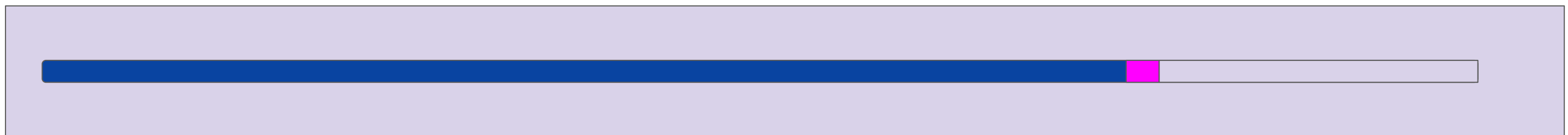
max_new_tokens = 100



max_new_tokens = 150



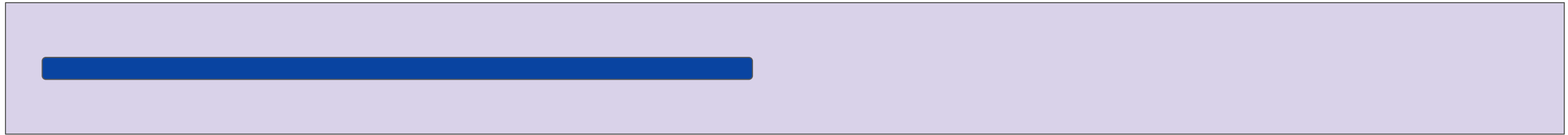
max_new_tokens = 200



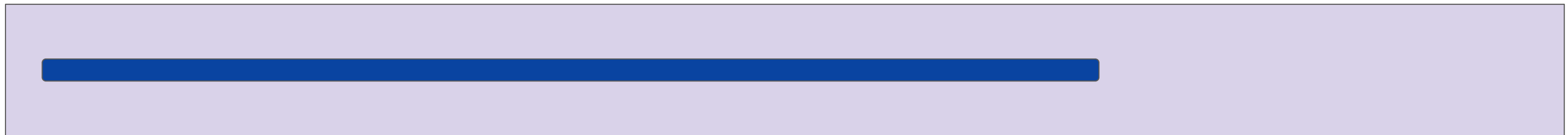
Softmax
output

Generative config - max new tokens

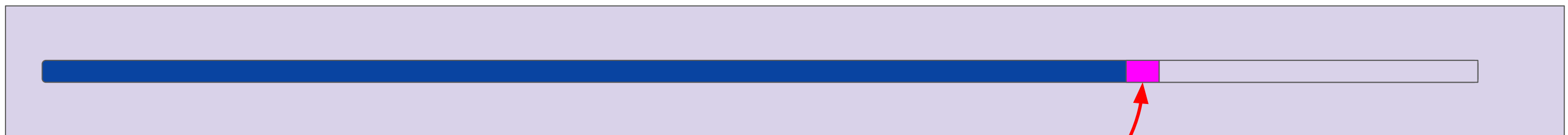
max_new_tokens = 100



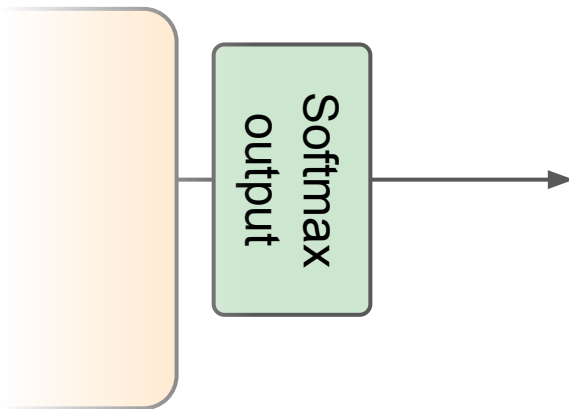
max_new_tokens = 150



max_new_tokens = 200



Stop token



Generative config - greedy vs. random sampling

Token probability

Softmax output

0.20	cake
0.10	donut
0.02	banana
0.01	apple
...	...

prob word

greedy: The word/token with the highest probability is selected.

Softmax output

0.20	cake
0.10	donut
0.02	banana
0.01	apple
...	...

prob word

random(-weighted) sampling: select a token using a random-weighted strategy across the probabilities of all tokens.

Here, there is a 20% chance that 'cake' will be selected, but 'banana' was actually selected.

Generative configuration - top-k and top-p

Enter your prompt here...

Max new tokens

Sample top K

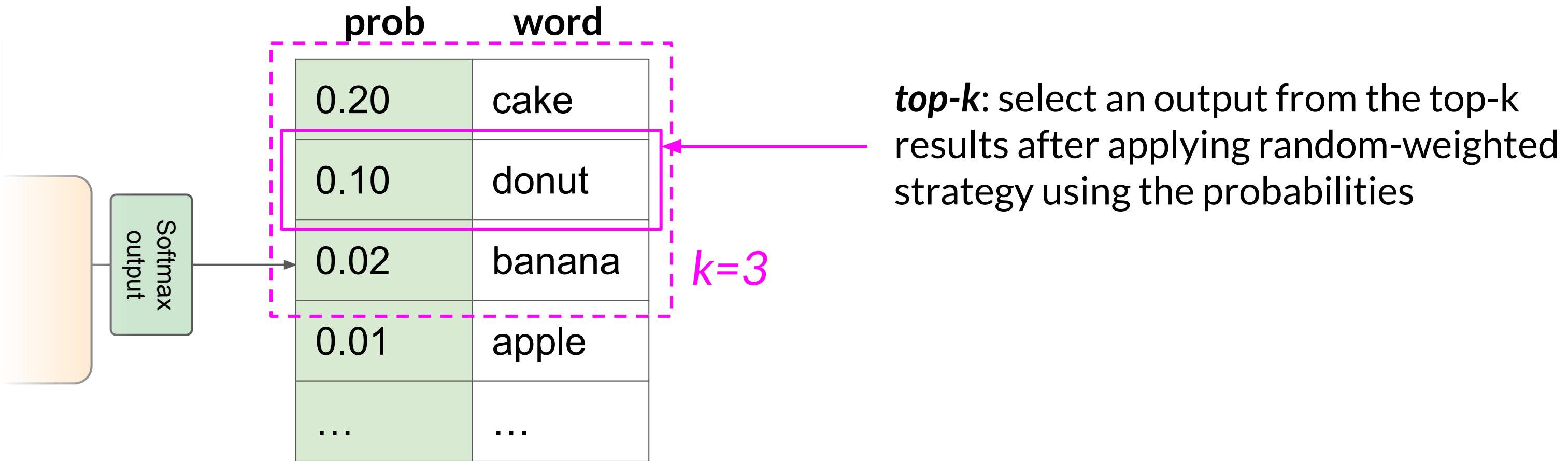
Sample top P

Temperature

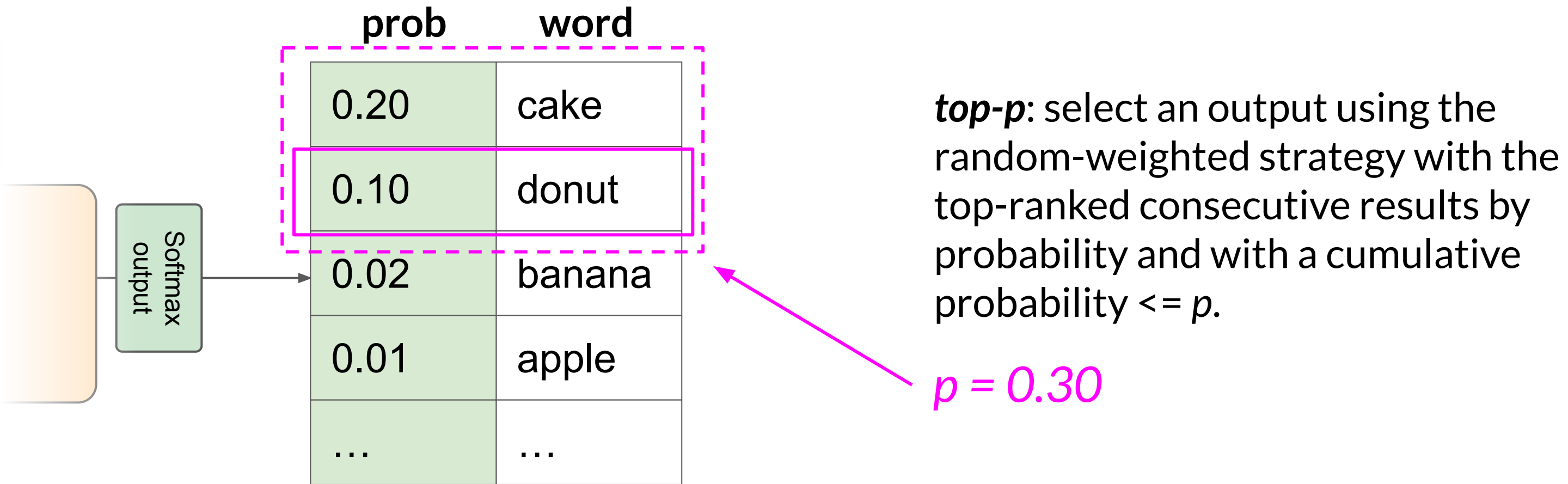
Submit

Top-k and top-p sampling

Generative config - top-k sampling



Generative config - top-p sampling



Generative configuration - temperature

Enter your prompt here...

Max new tokens

Sample top K

Sample top P

Temperature

Submit

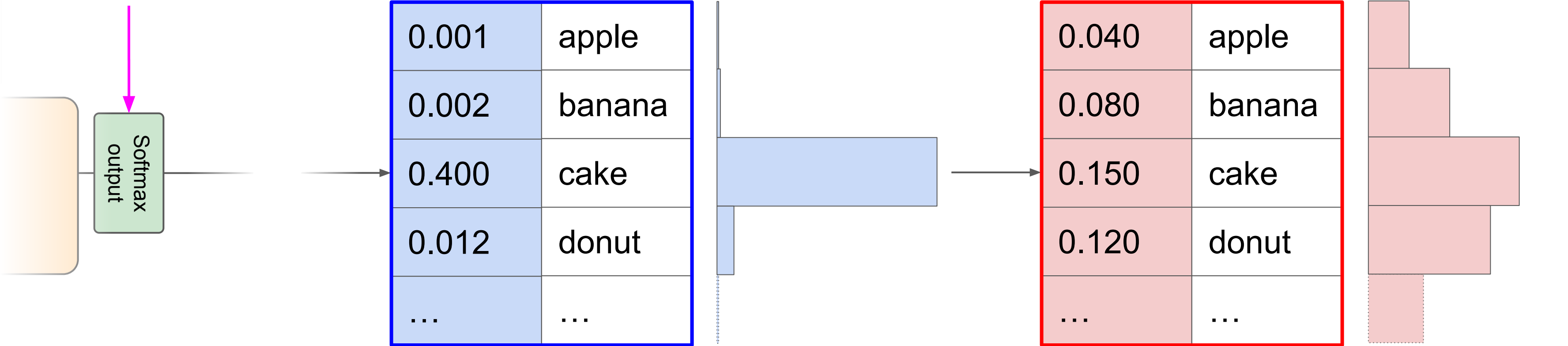
Temperature

Generative config - temperature

Temperature
setting

Cooler temperature (e.g <1)

Higher temperature (>1)

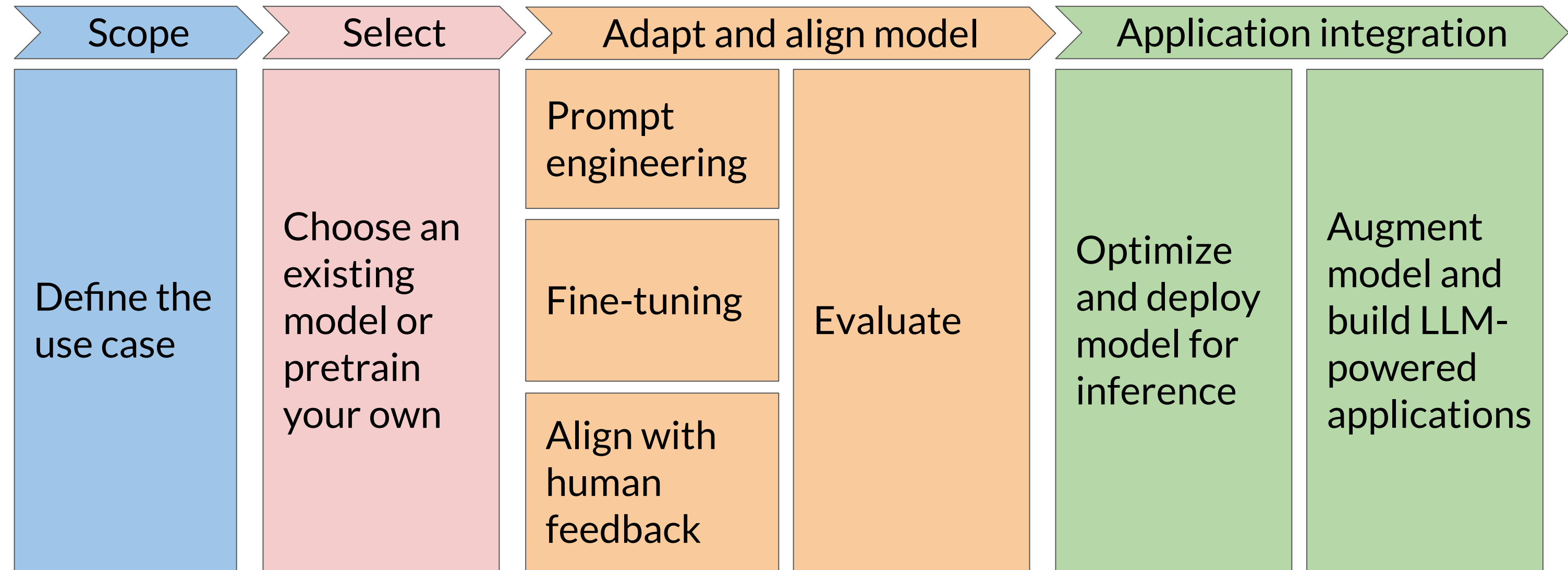


Strongly peaked
probability
distribution

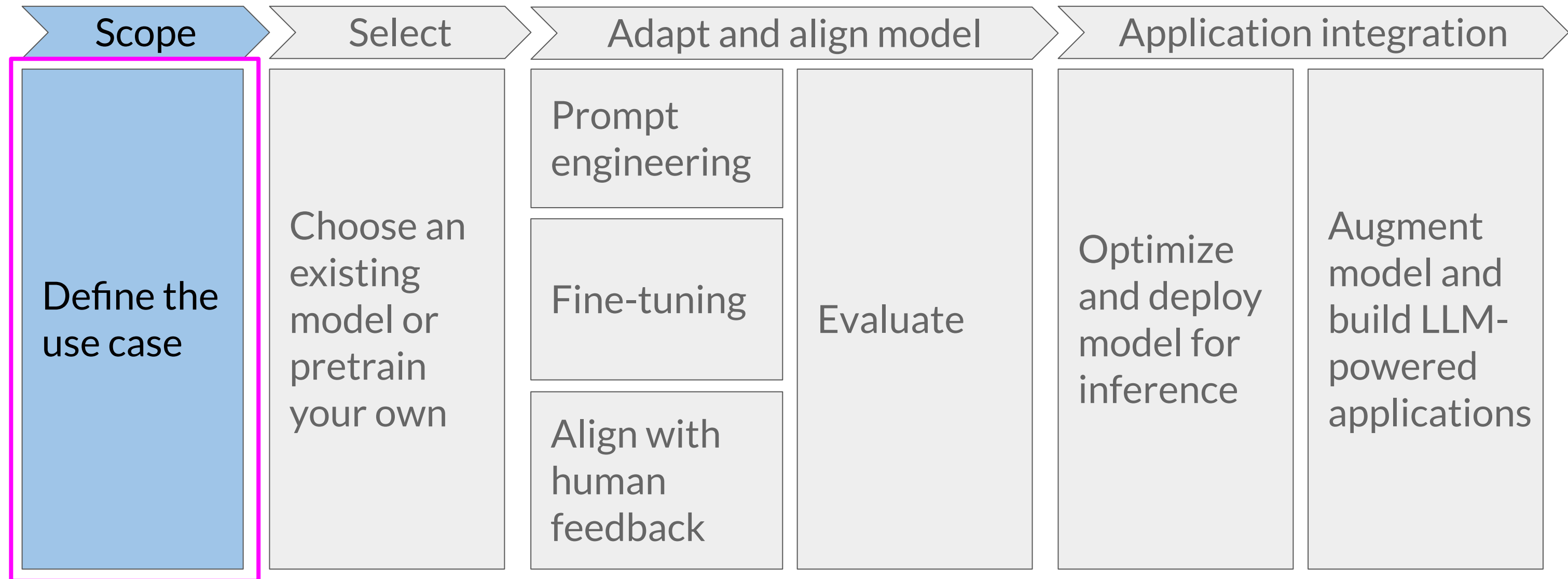
Broader, flatter
probability
distribution

Generative AI project lifecycle

Generative AI project lifecycle

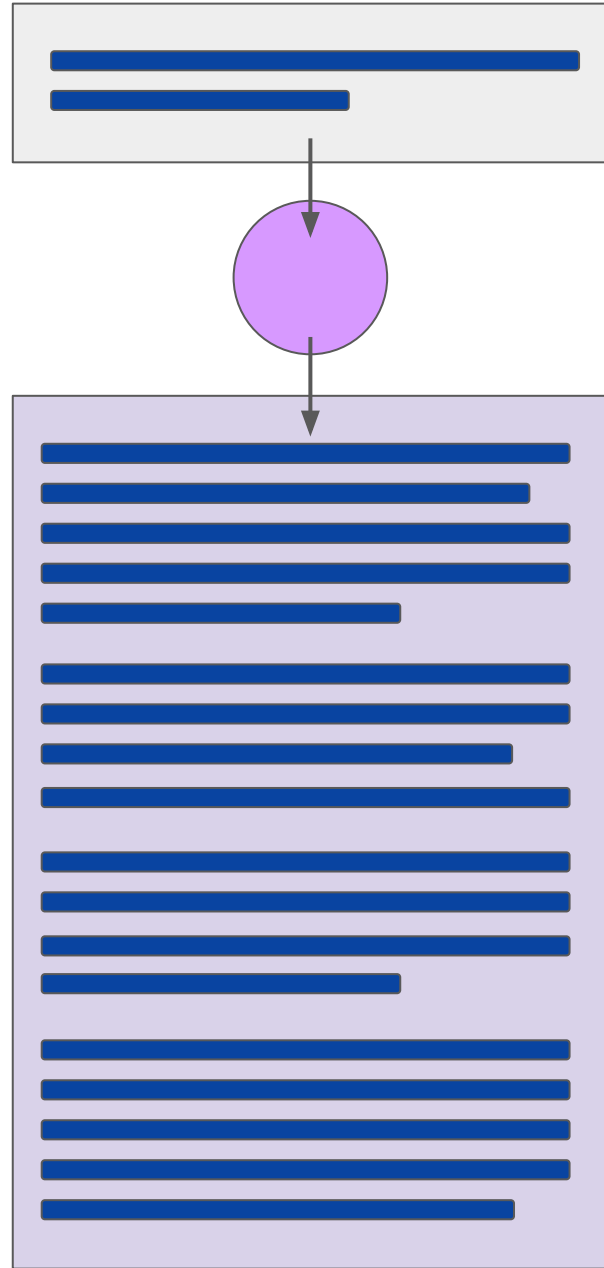


Generative AI project lifecycle

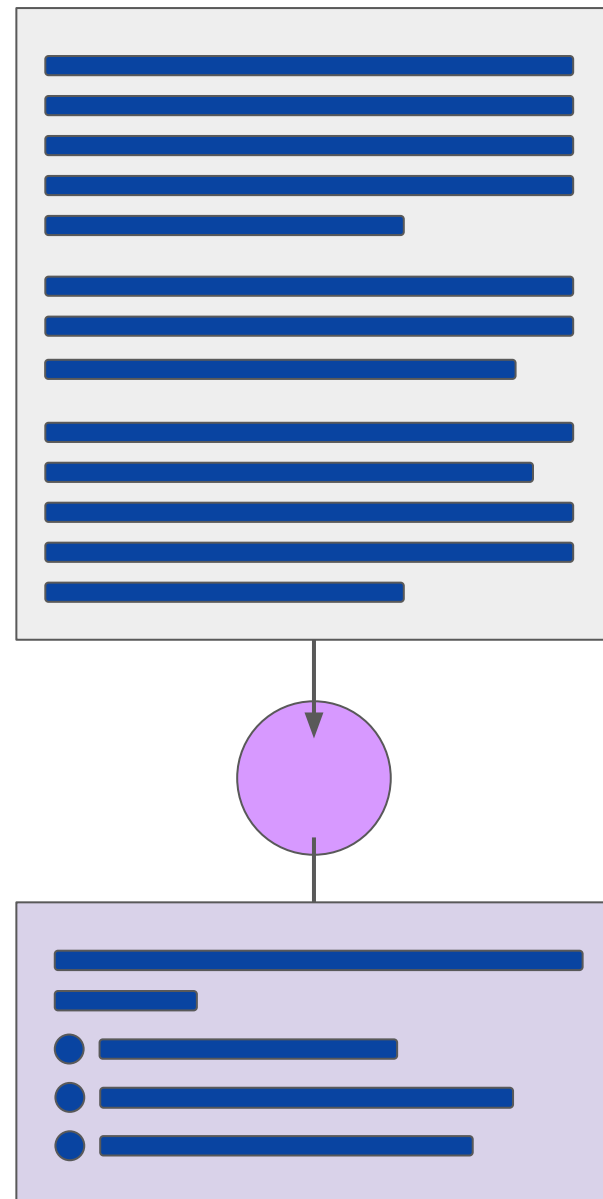


Good at many tasks?

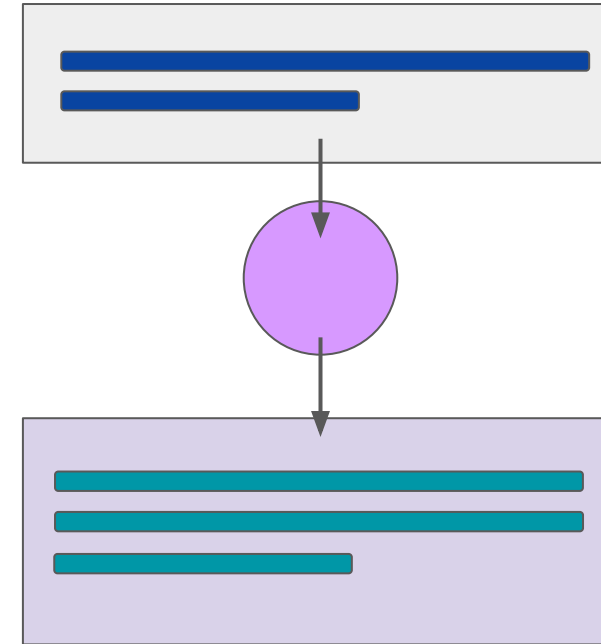
Essay Writing



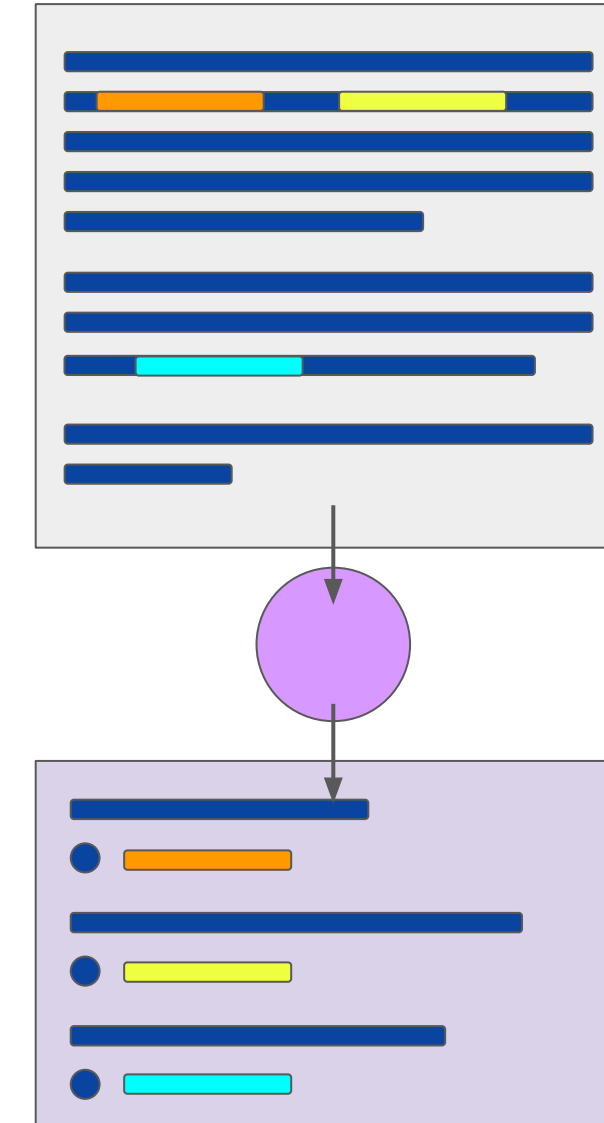
Summarization



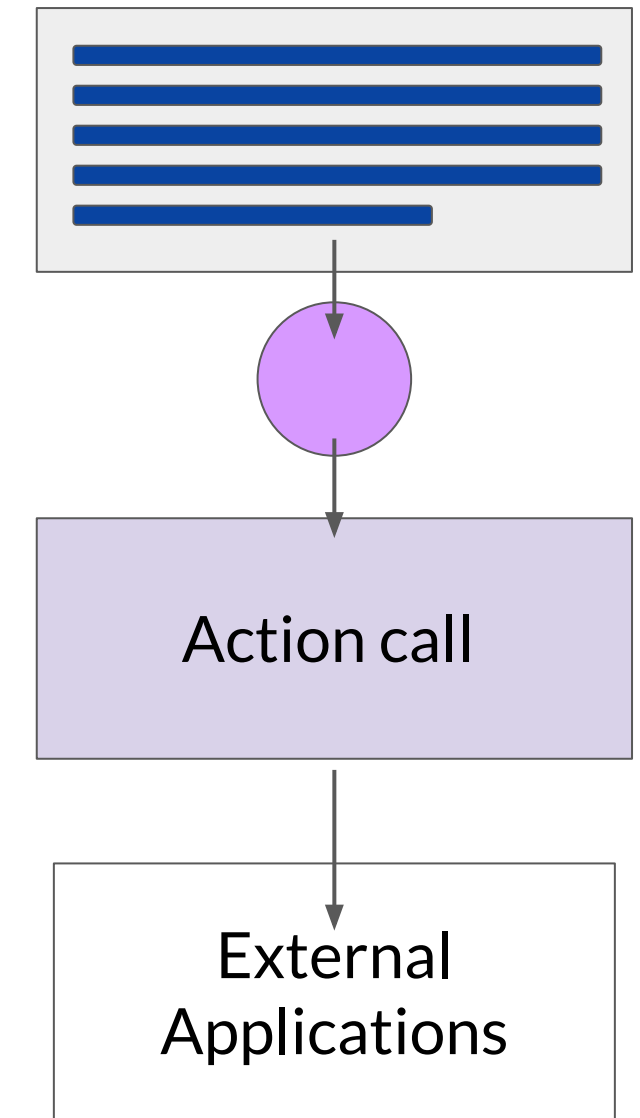
Translation



Information retrieval

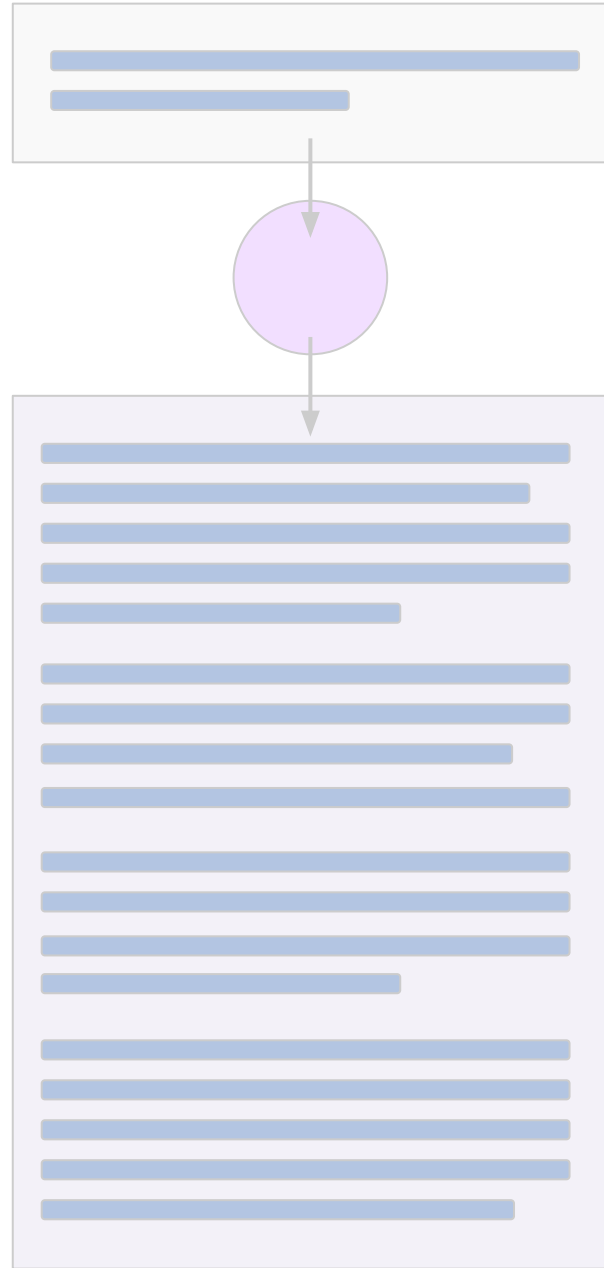


Invoke APIs and actions

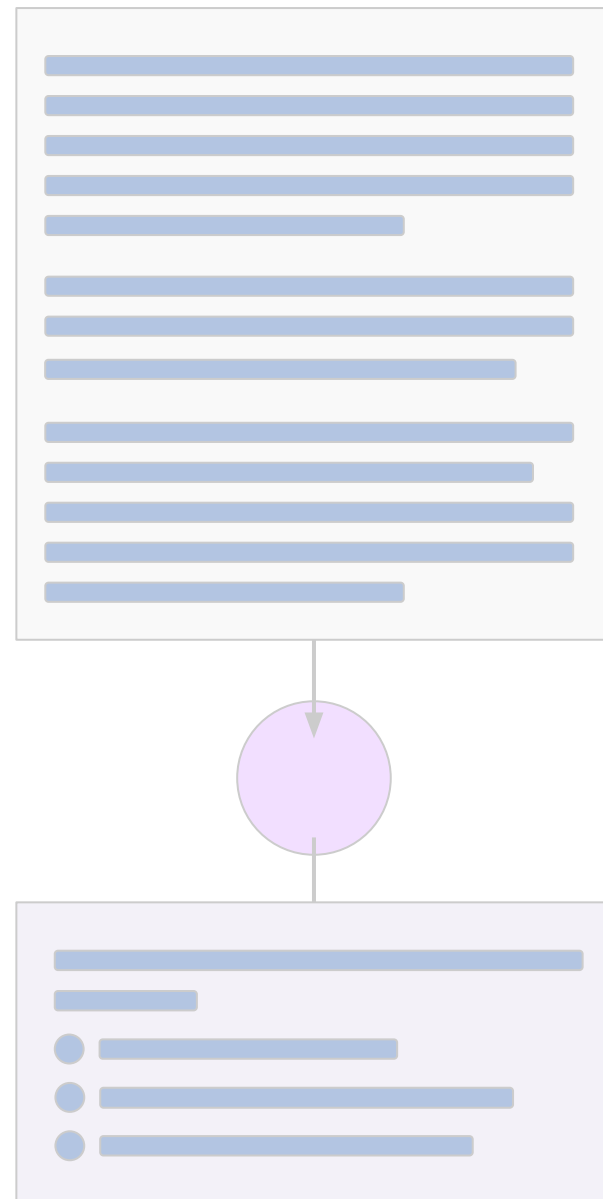


Or good at a single task?

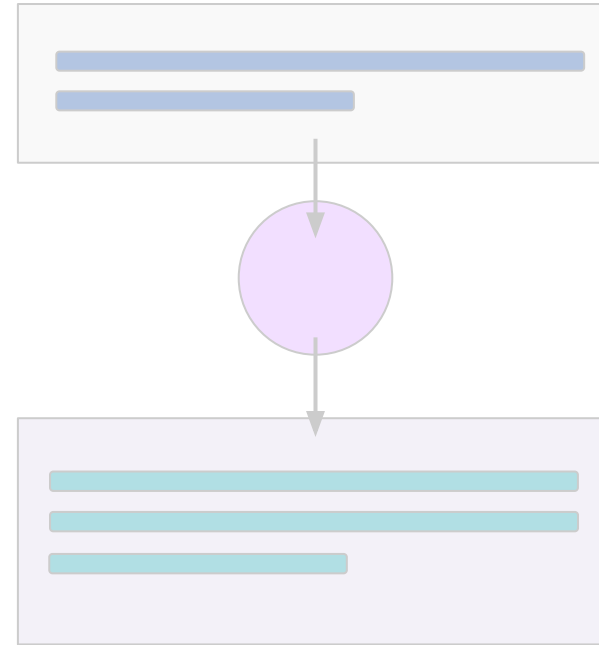
Essay Writing



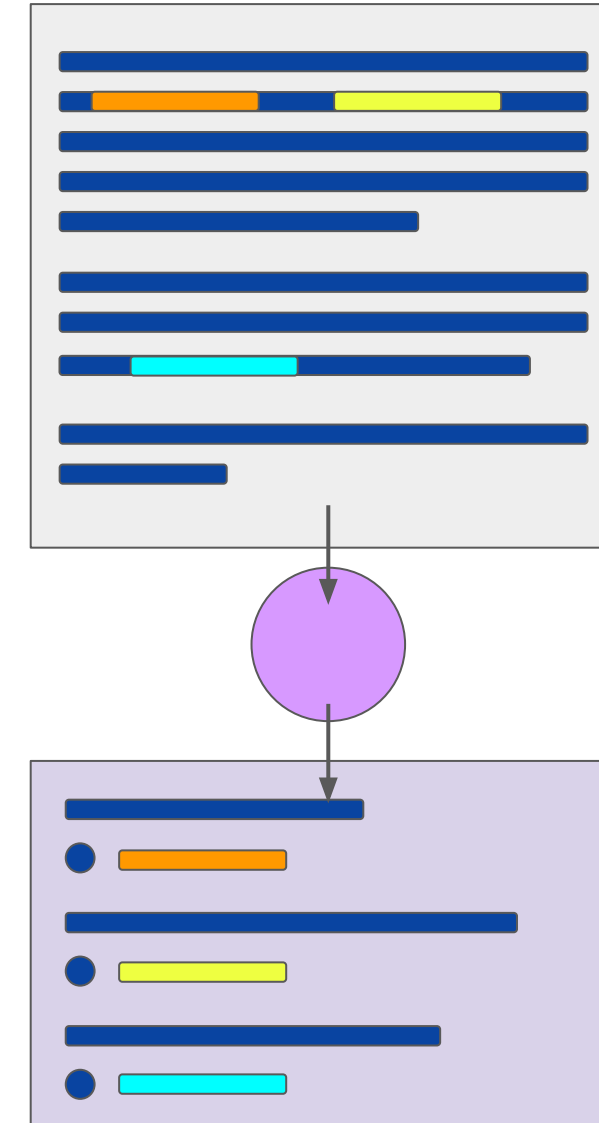
Summarization



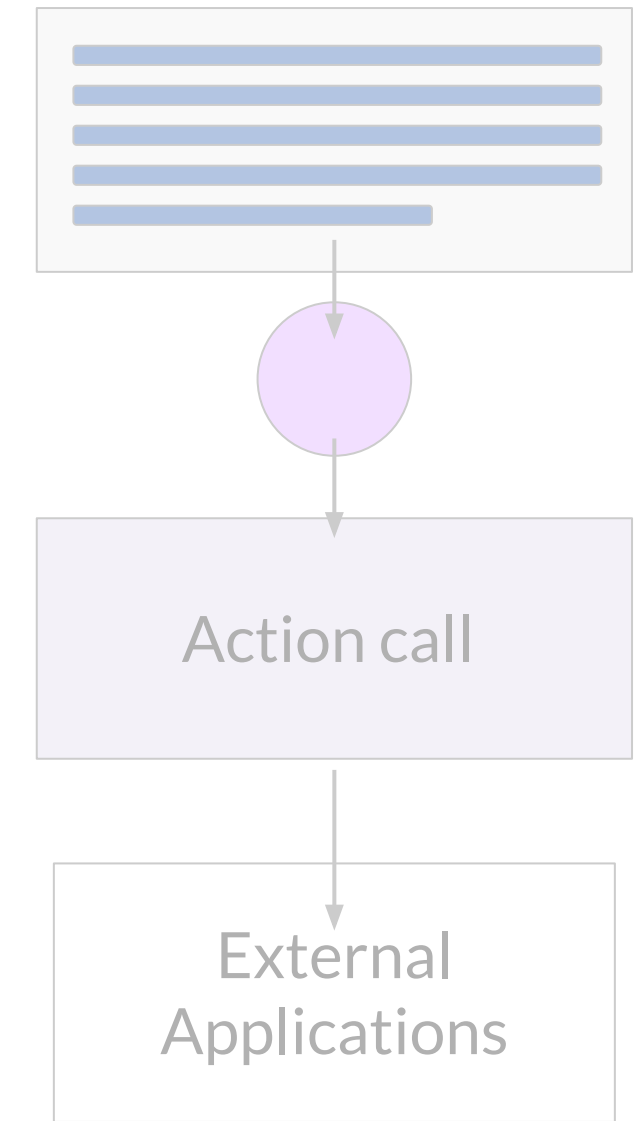
Translation



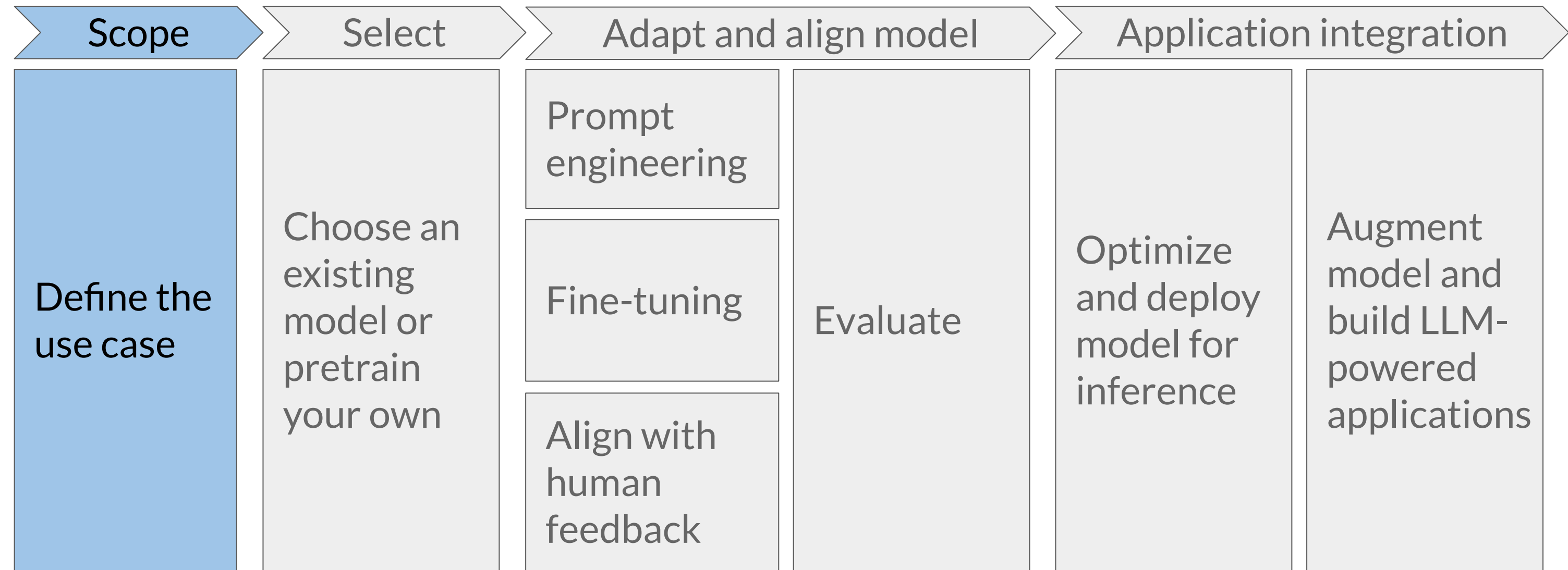
Information retrieval



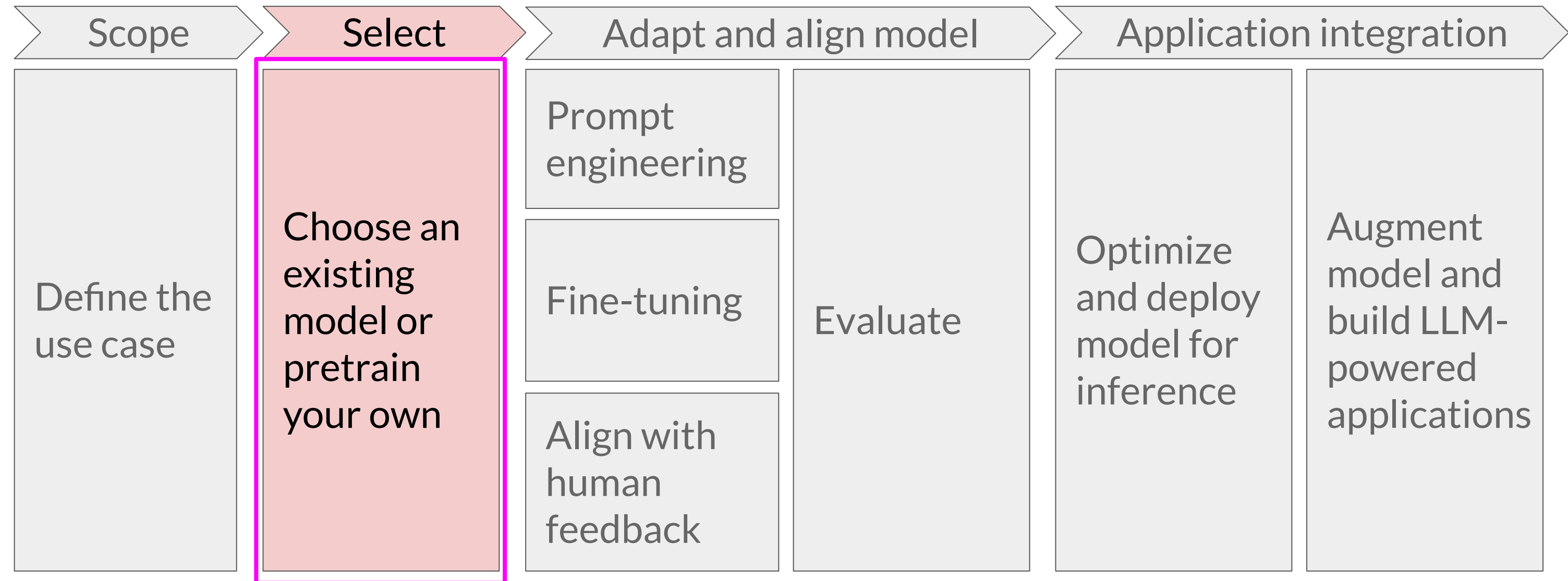
Invoke APIs and actions



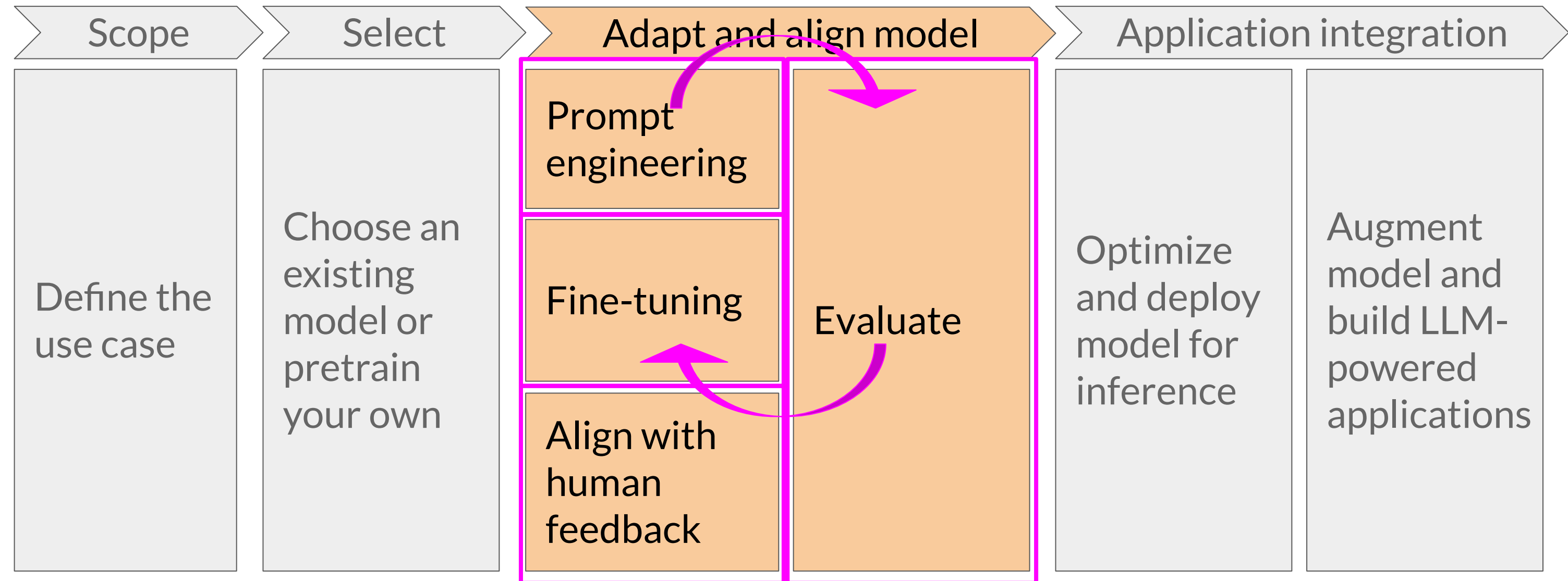
Generative AI project lifecycle



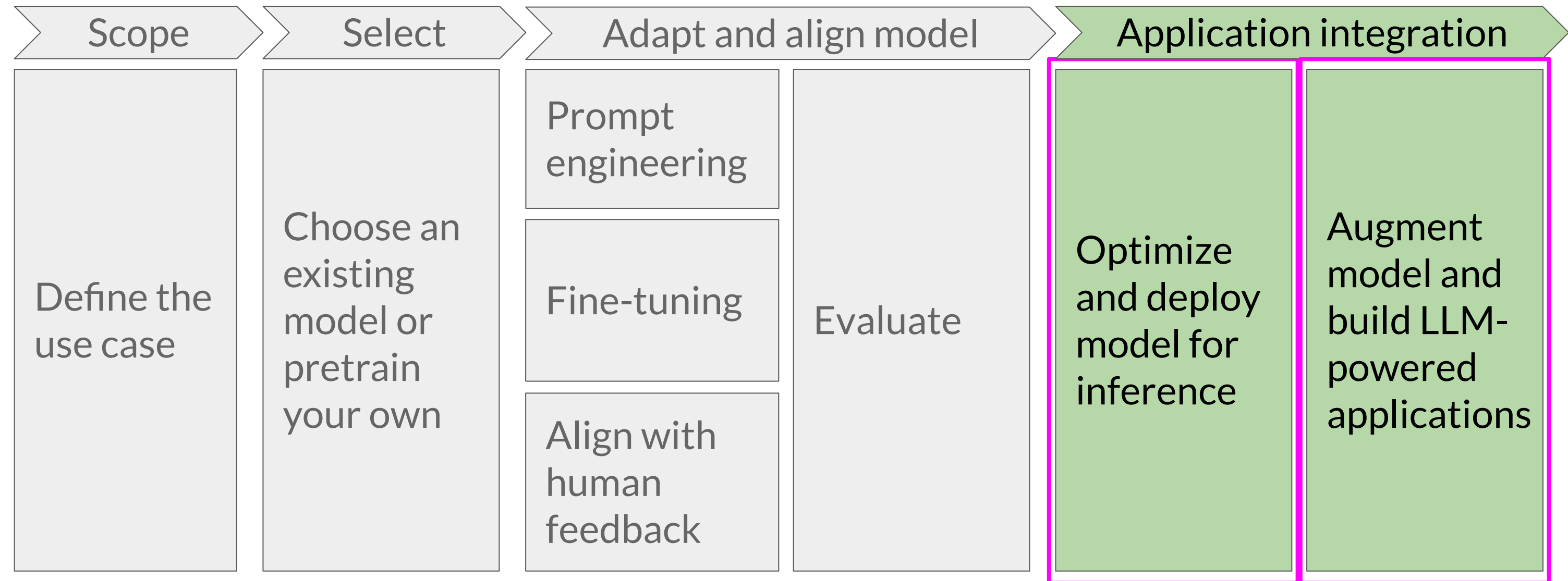
Generative AI project lifecycle



Generative AI project lifecycle



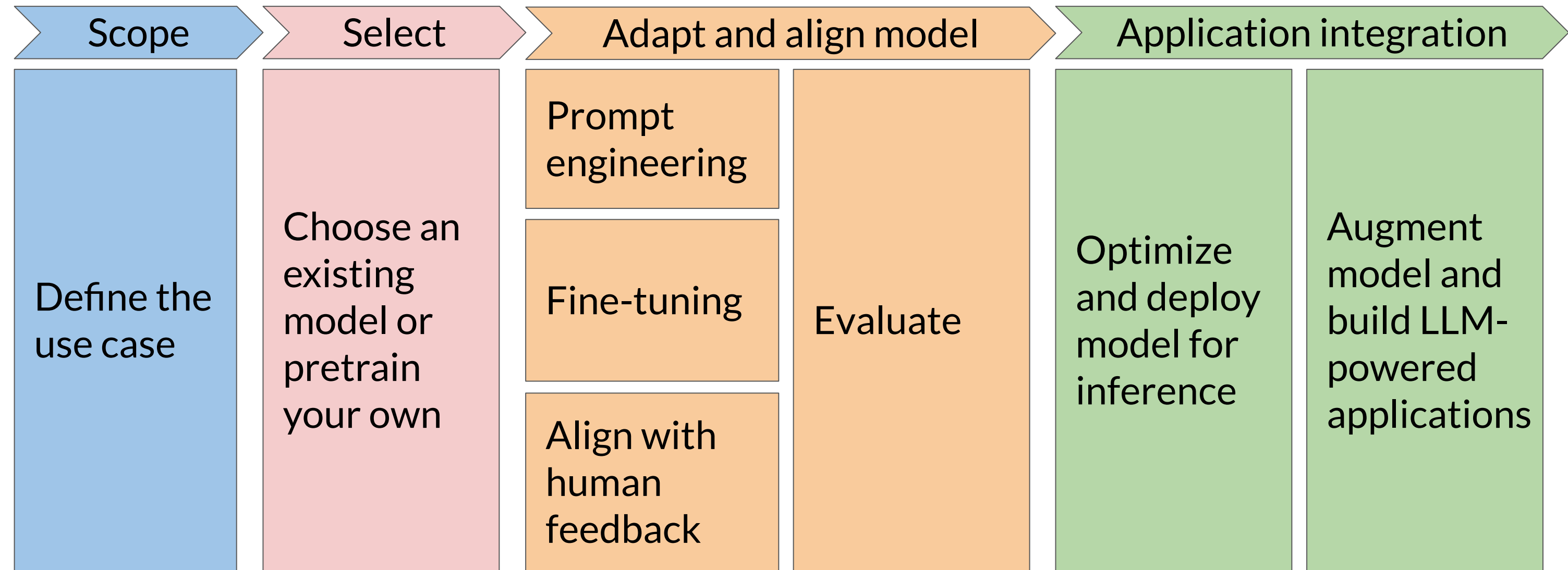
Generative AI project lifecycle



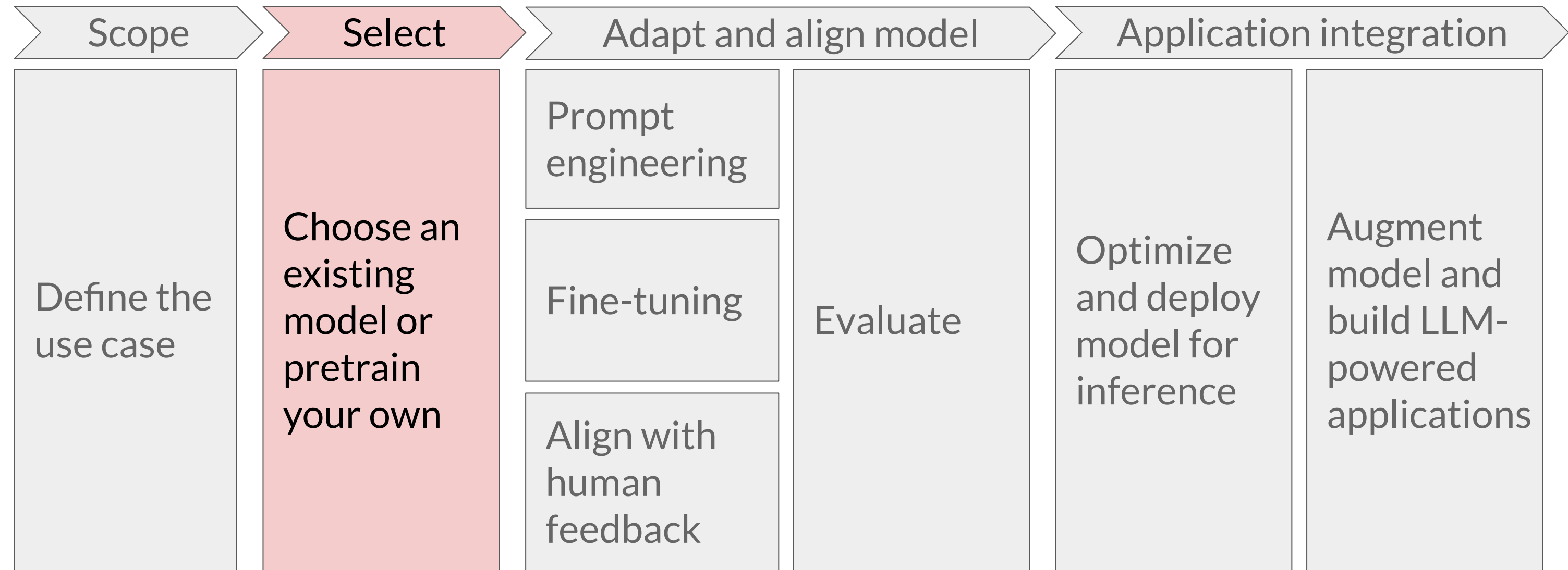
Pre-training and scaling laws



Generative AI project lifecycle

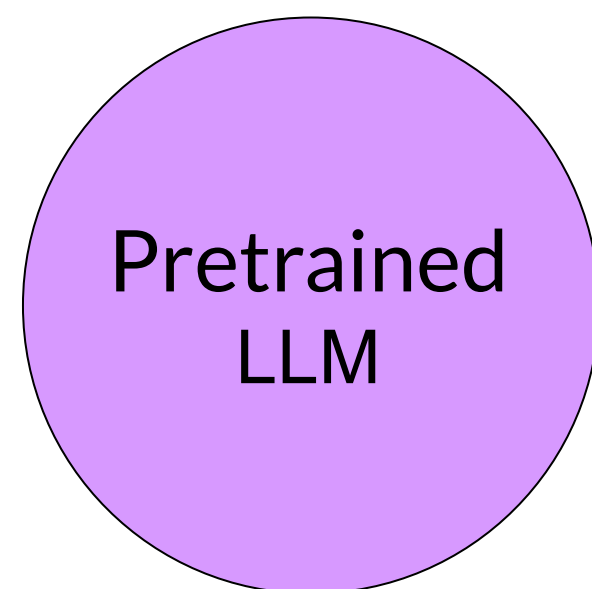


Generative AI project lifecycle

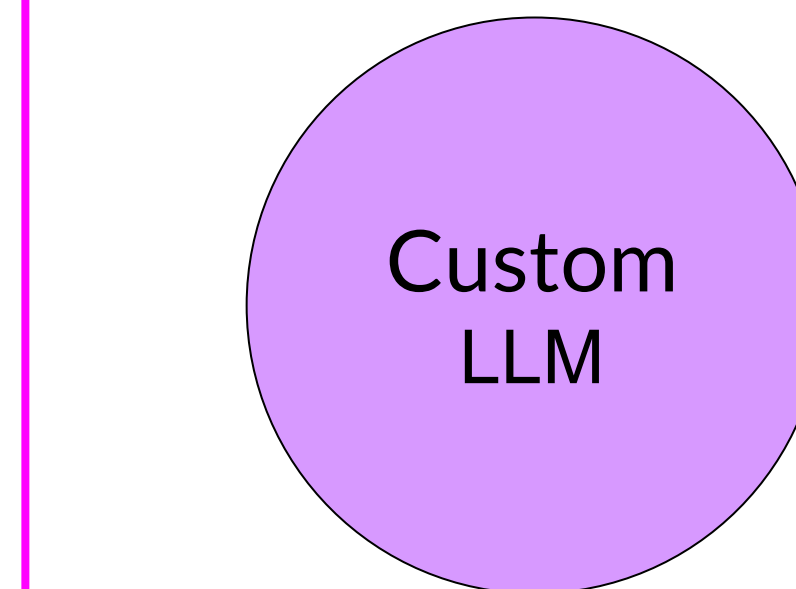


Considerations for choosing a model

Foundation model



Train your own model



Considerations for choosing a model

Foundation model

Pretrained
LLM

Train your own model

Custom
LLM

Model hubs

Model Card for T5 Large

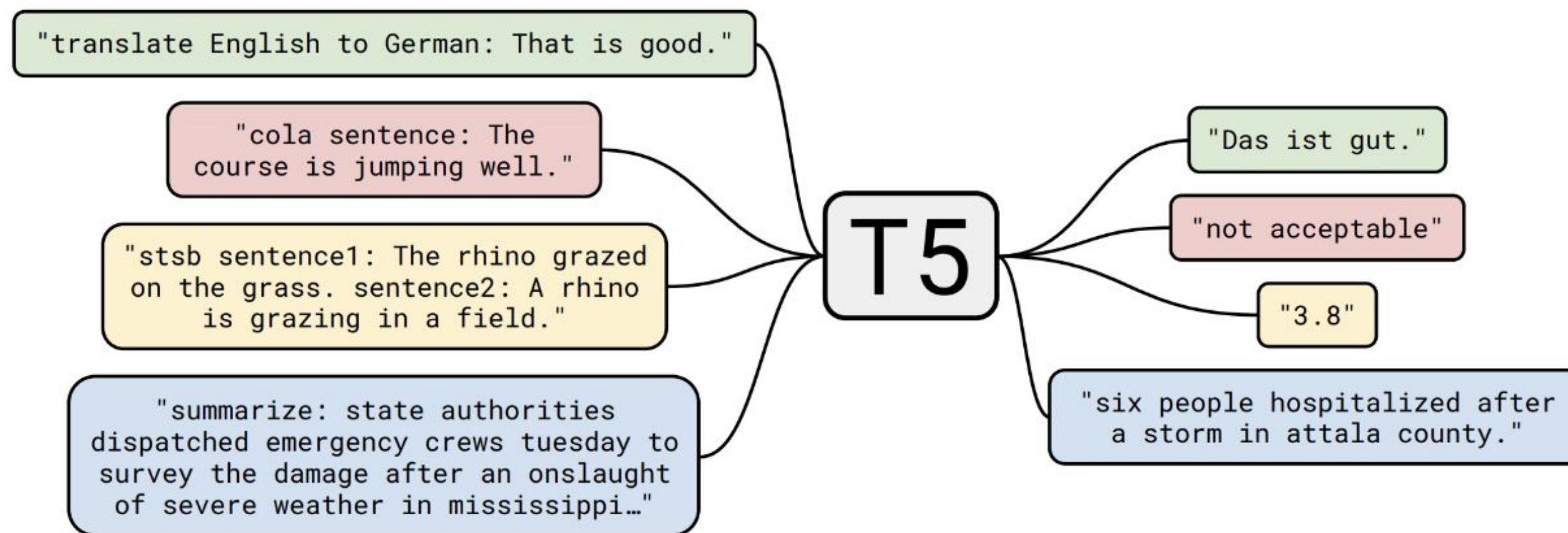
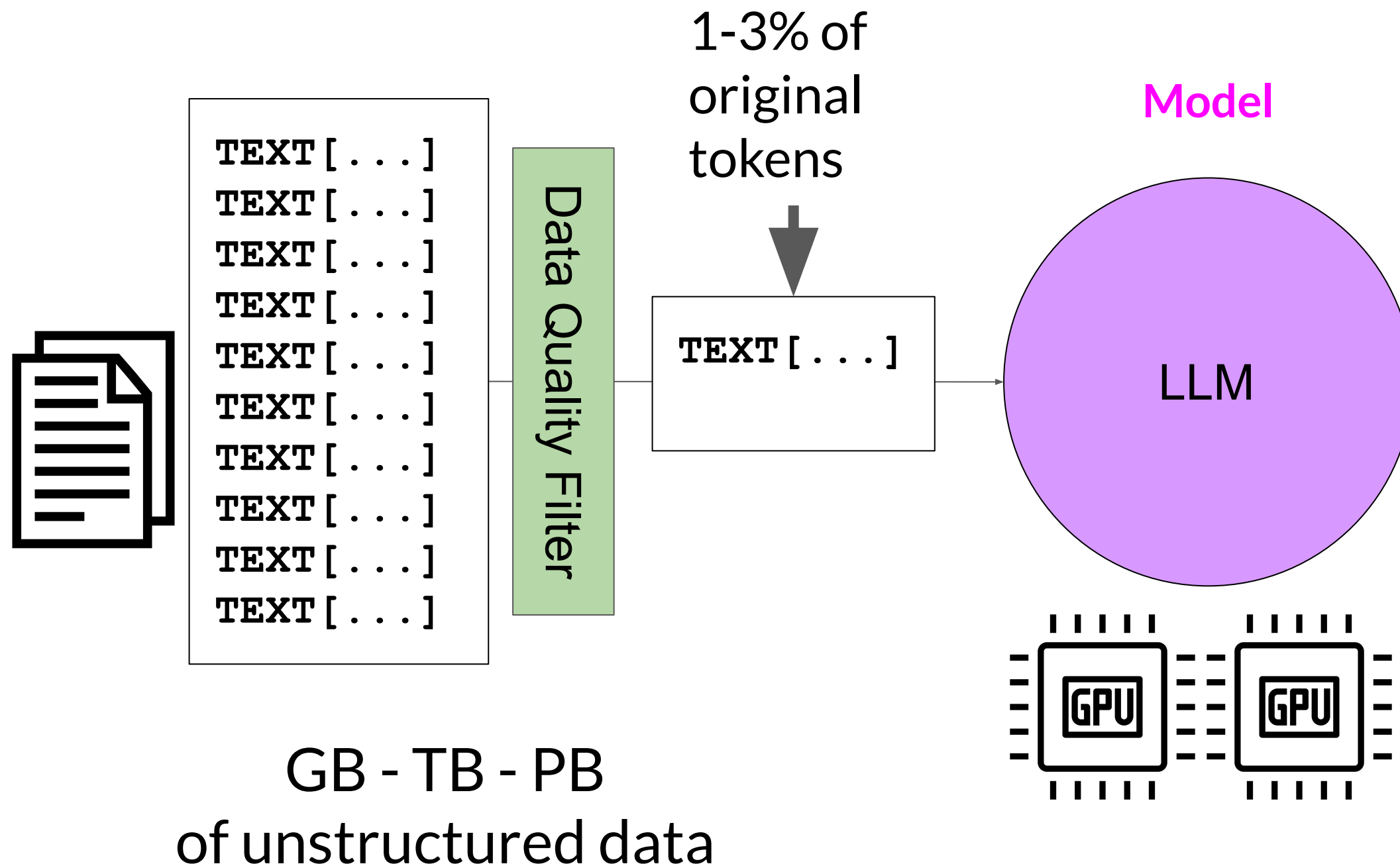


Table of Contents

1. [Model Details](#)
2. [Uses](#)
3. [Bias, Risks, and Limitations](#)
4. [Training Details](#)
5. [Evaluation](#)

Model architectures and pre-training objectives

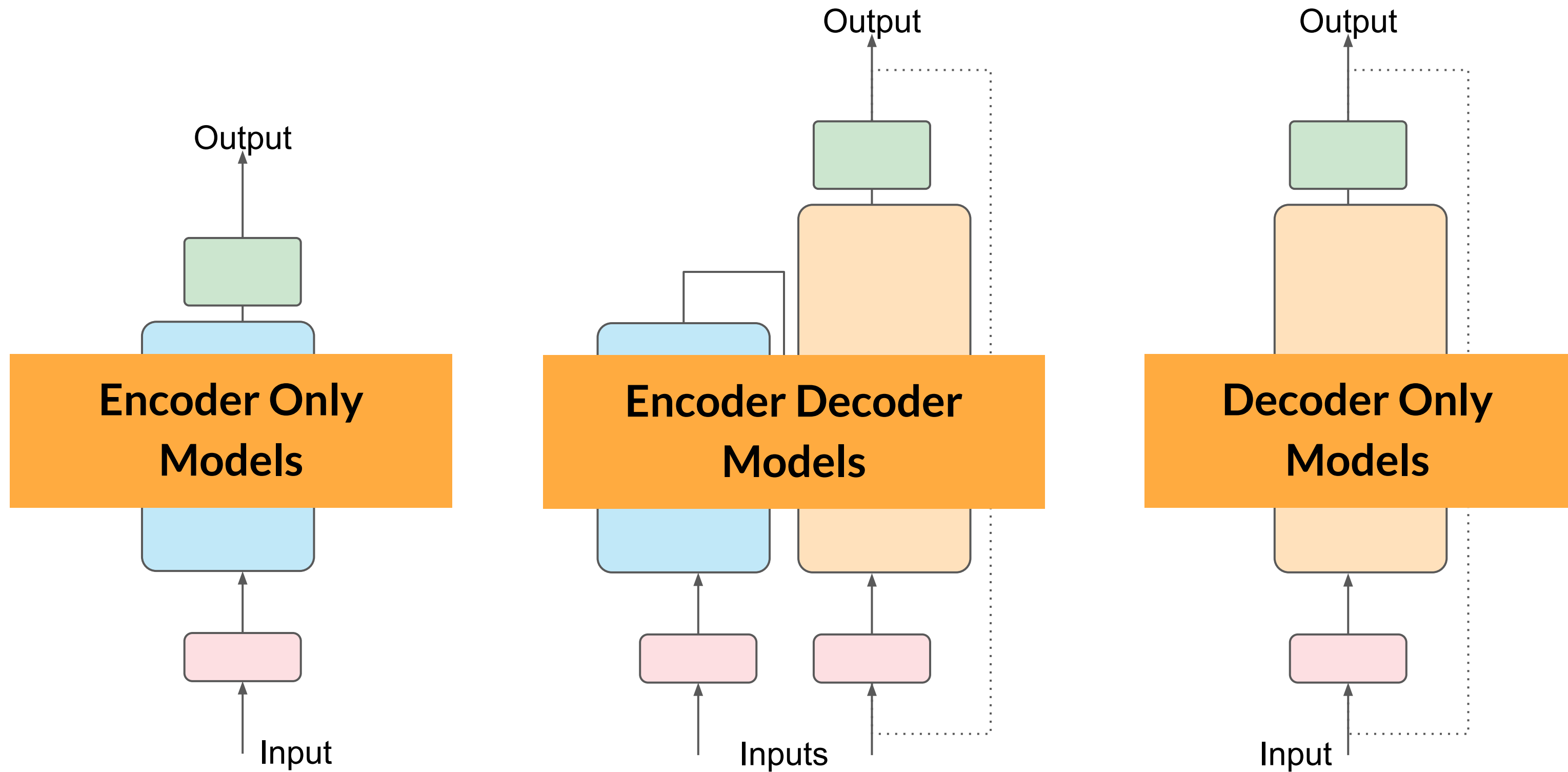
LLM pre-training at a high level



Token String	Token ID	Embedding / Vector Representation
'_The'	37	[-0.0513, -0.0584, 0.0230, ...]
'_teacher'	3145	[-0.0335, 0.0167, 0.0484, ...]
'_teaches'	11749	[-0.0151, -0.0516, 0.0309, ...]
'_the'	8	[-0.0498, -0.0428, 0.0275, ...]
'_student'	1236	[-0.0460, 0.0031, 0.0545, ...]
...

Vocabulary

Transformers



Autoencoding models

Masked Language Modeling (MLM)

The	teacher	<MASK>	the	student
-----	---------	--------	-----	---------

Original text



The teacher
teaches the
student.

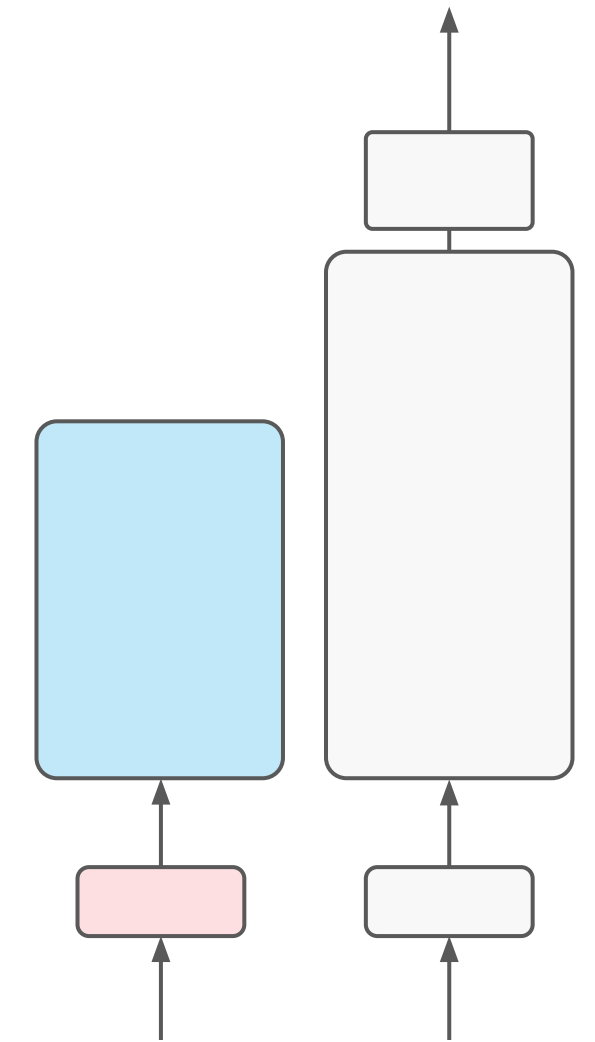
[...]

Encoder-only
LLM

Objective: Reconstruct text ("denoising")

The	teacher	teaches	the	student
-----	---------	---------	-----	---------

Bidirectional context



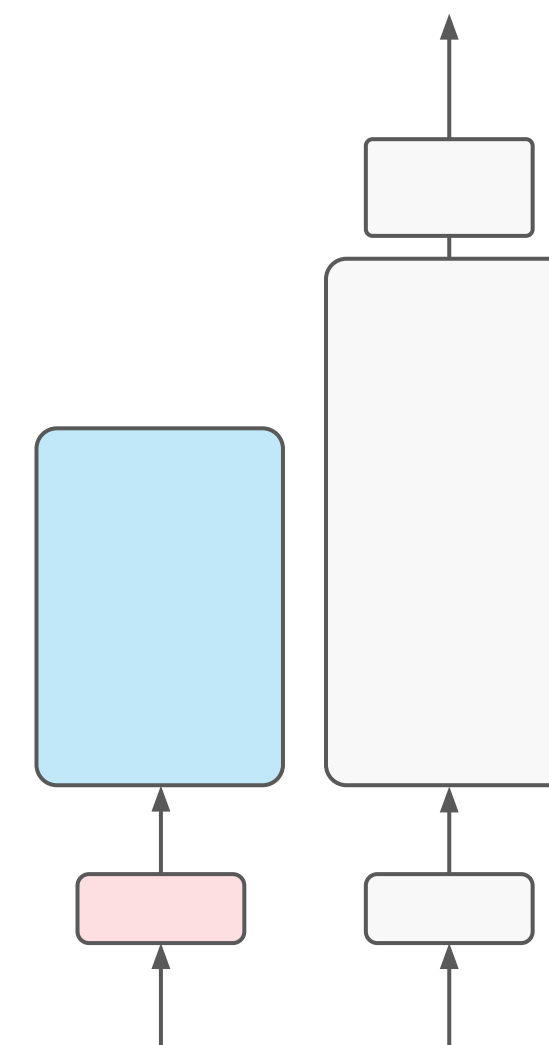
Autoencoding models

Good use cases:

- Sentiment analysis
- Named entity recognition
- Word classification

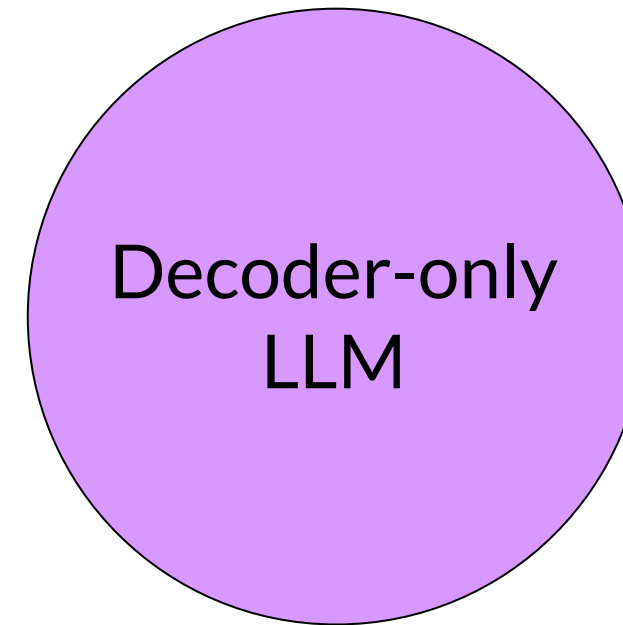
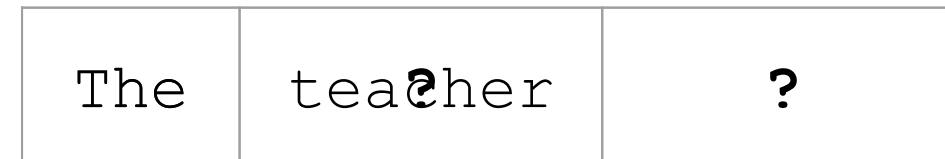
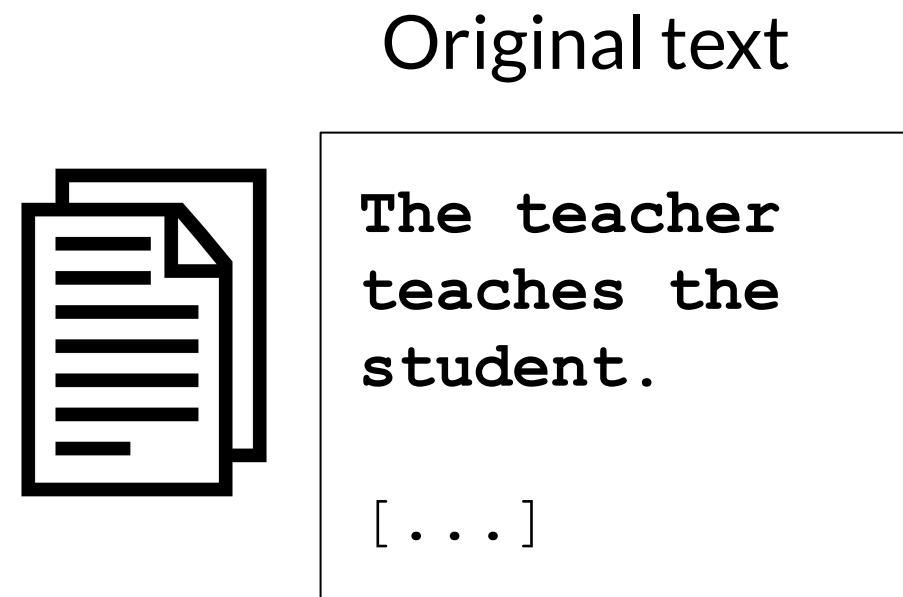
Example models:

- BERT
- ROBERTA



Autoregressive models

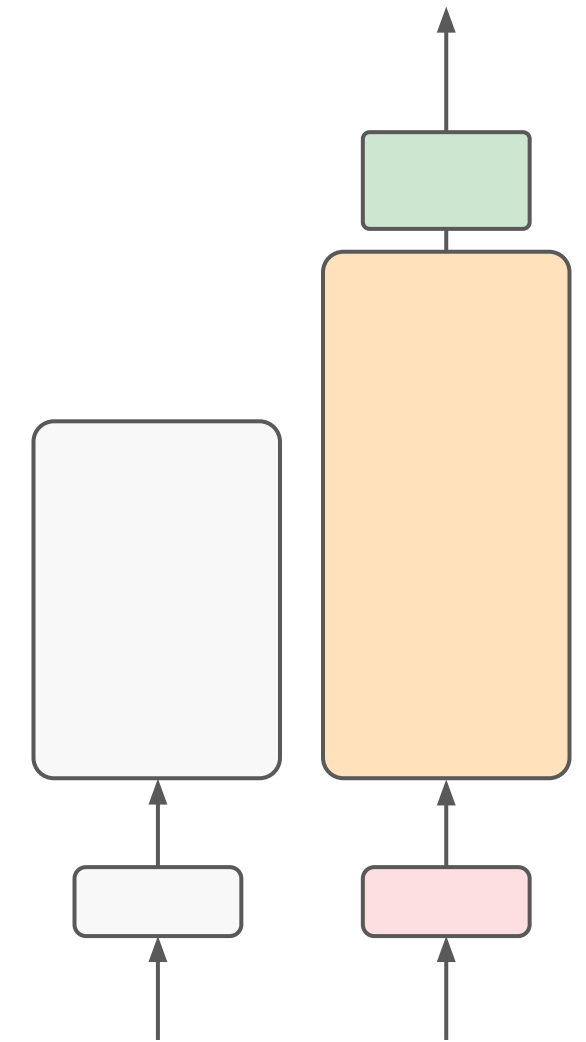
Causal Language Modeling (CLM)



Objective: Predict next token



Unidirectional context



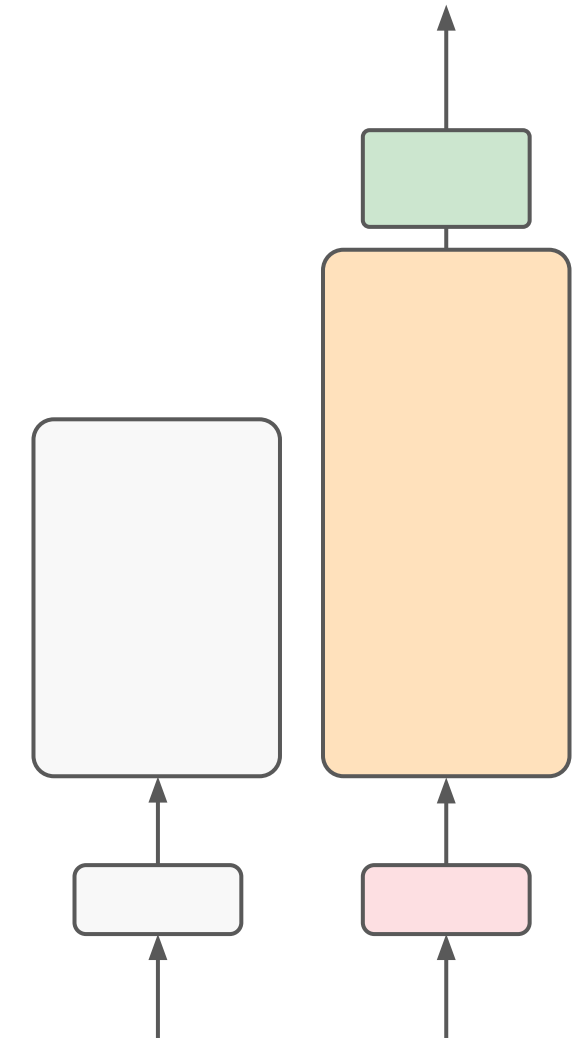
Autoregressive models

Good use cases:

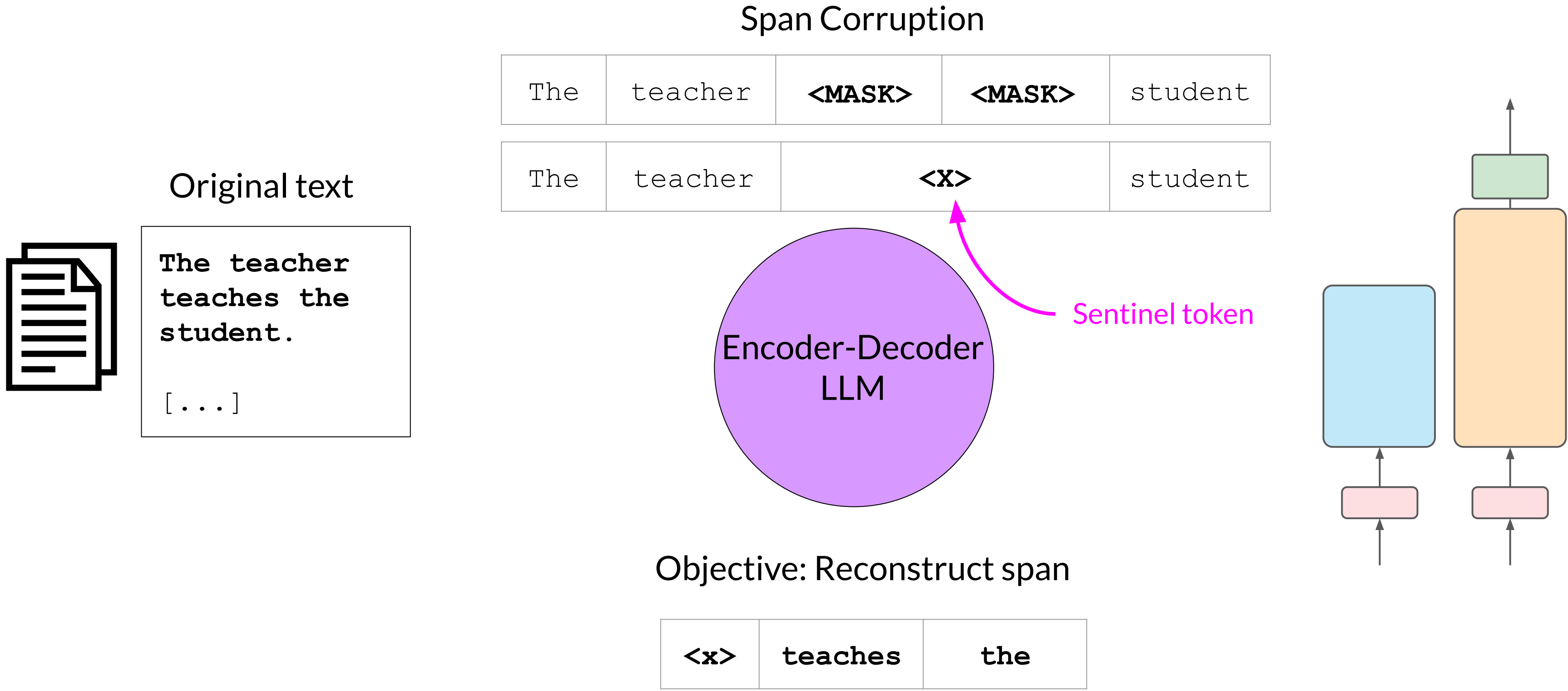
- Text generation
- Other emergent behavior
 - Depends on model size

Example models:

- GPT
- BLOOM



Sequence-to-sequence models



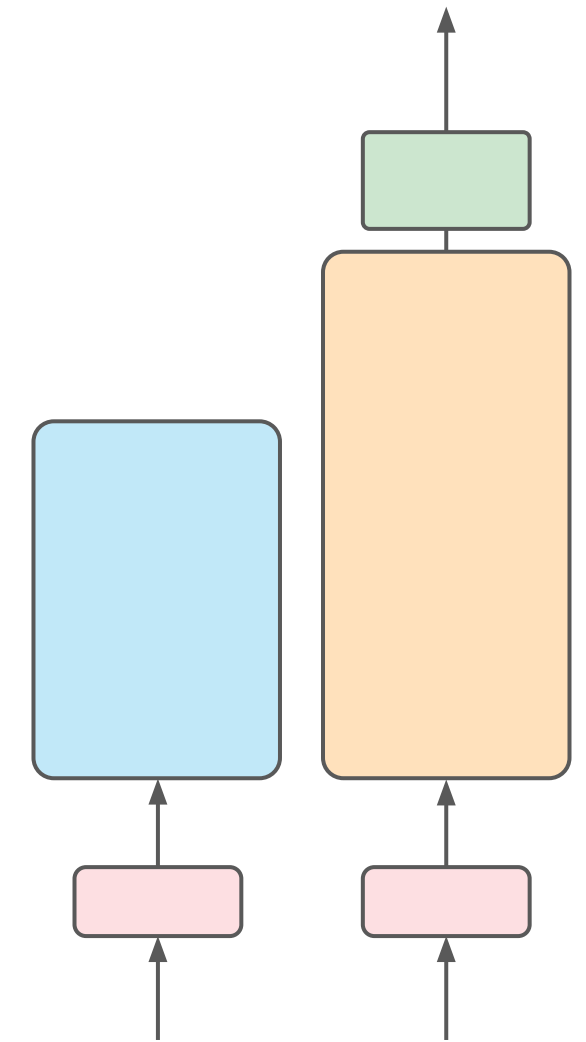
Sequence-to-sequence models

Good use cases:

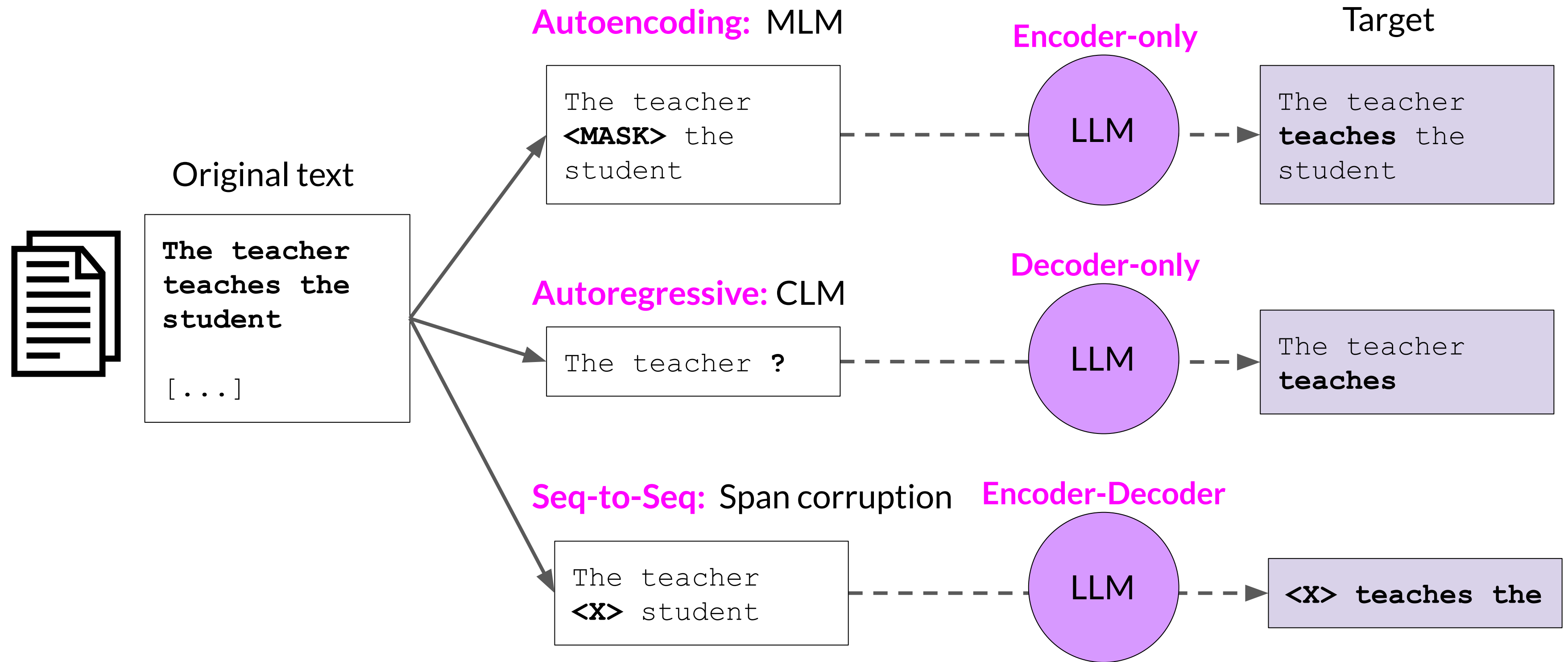
- Translation
- Text summarization
- Question answering

Example models:

- T5
- BART

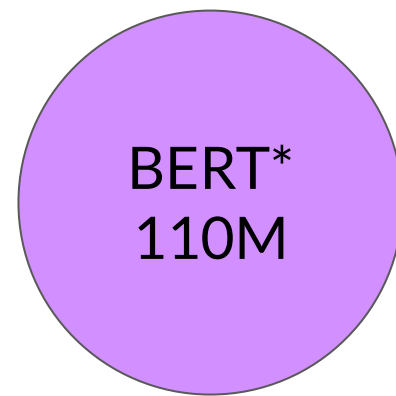


Model architectures and pre-training objectives



The significance of scale: task ability

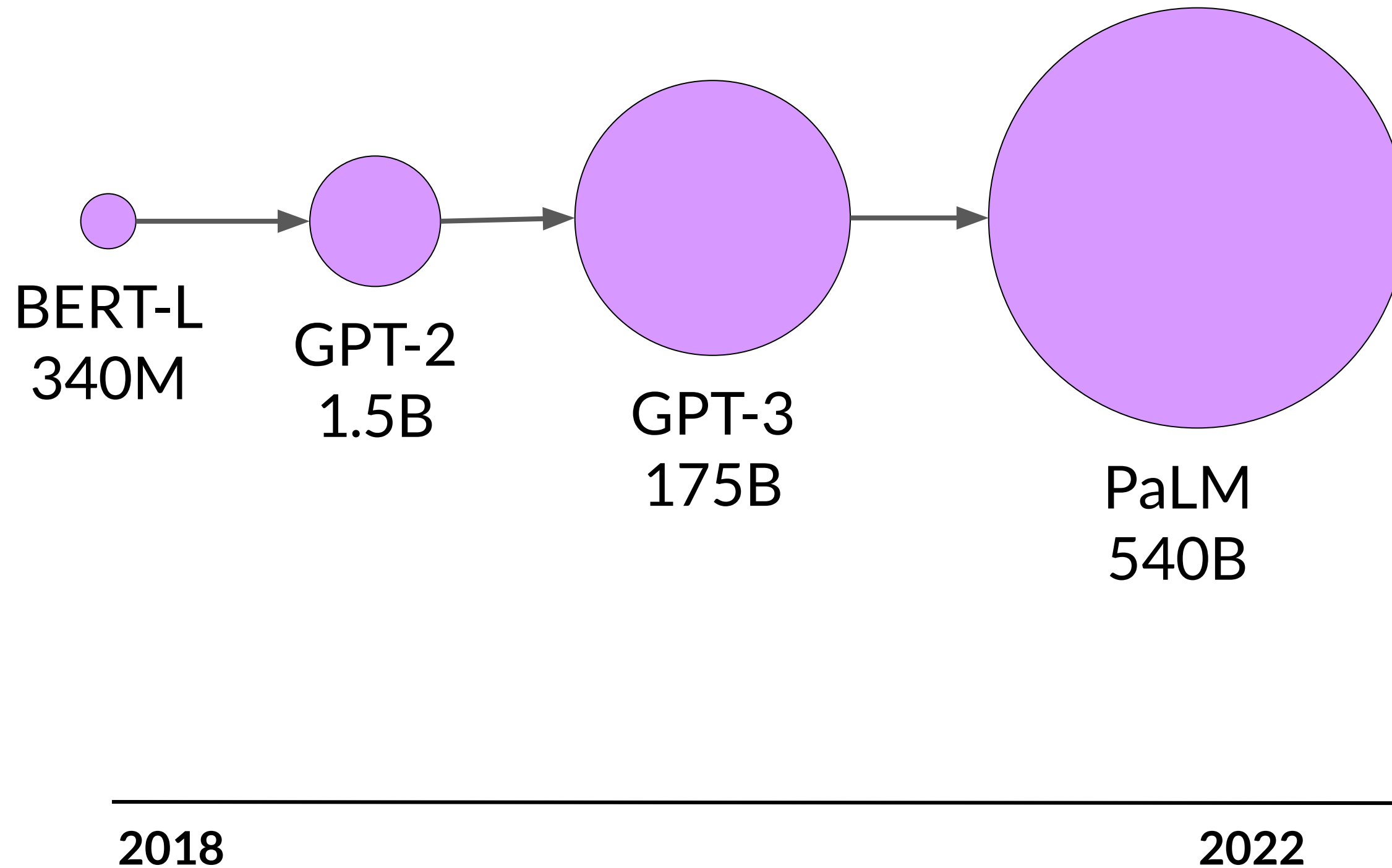
BERT*
110M



BLOOM
176B →

*Bert-base

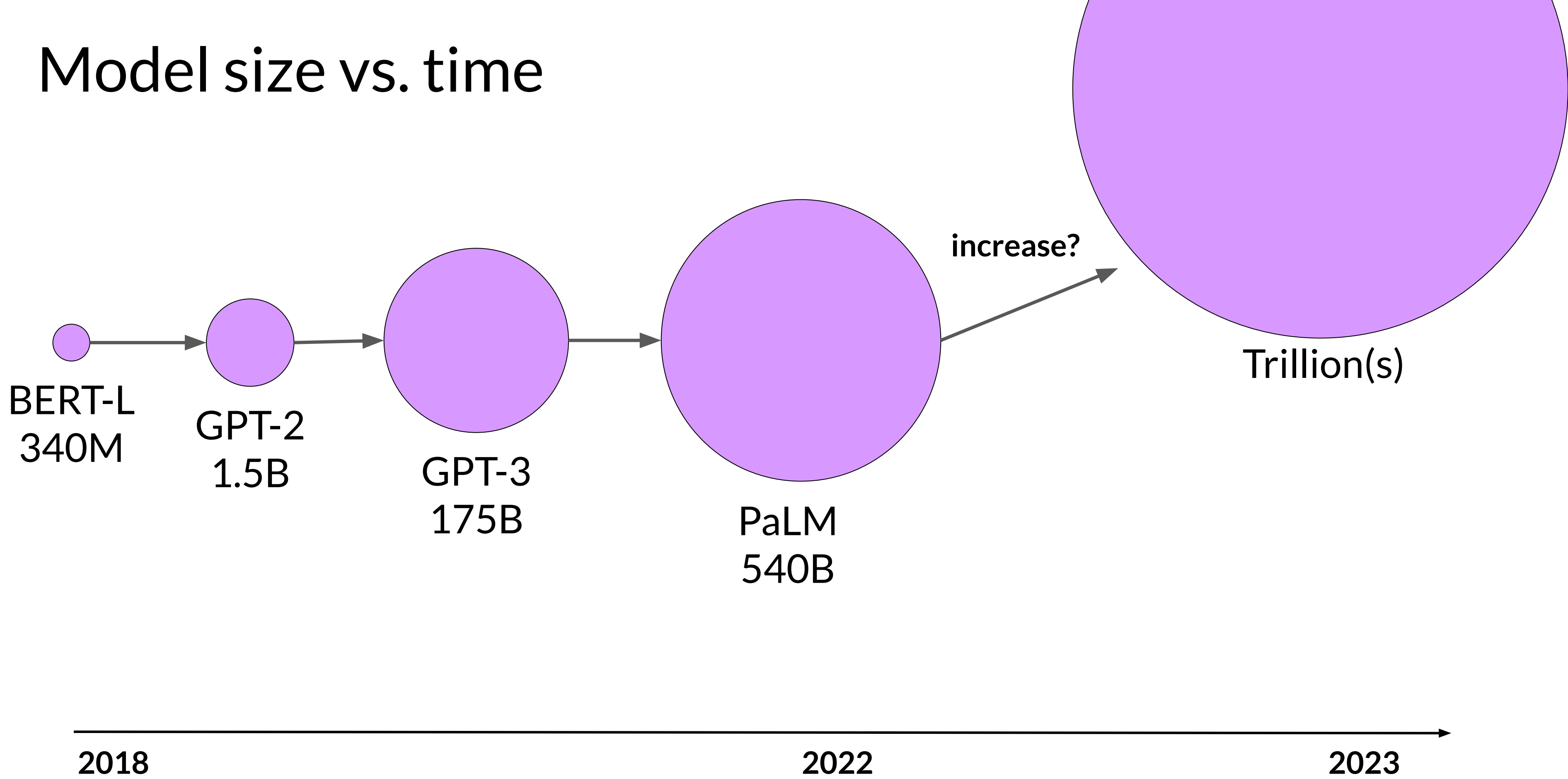
Model size vs. time



Growth powered by:

- Introduction of transformer
- Access to massive datasets
- More powerful compute resources

Model size vs. time




Computational challenges

```
OutOfMemoryError: CUDA out of memory.
```



Approximate GPU RAM needed to store 1B parameters

1 parameter = 4 bytes (32-bit float)
1B parameters = 4×10^9 bytes = 4GB



4GB @ 32-bit
full precision

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, <https://github.com/facebookresearch/bitsandbytes>

Additional GPU RAM needed to train 1B parameters

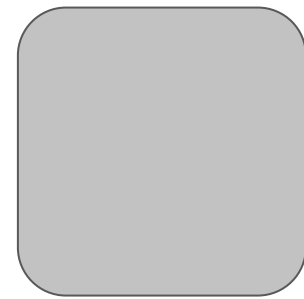
	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter

~20 extra bytes
per parameter

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, <https://github.com/facebookresearch/bitsandbytes>

Approximate GPU RAM needed to train 1B-params

Memory needed to store model



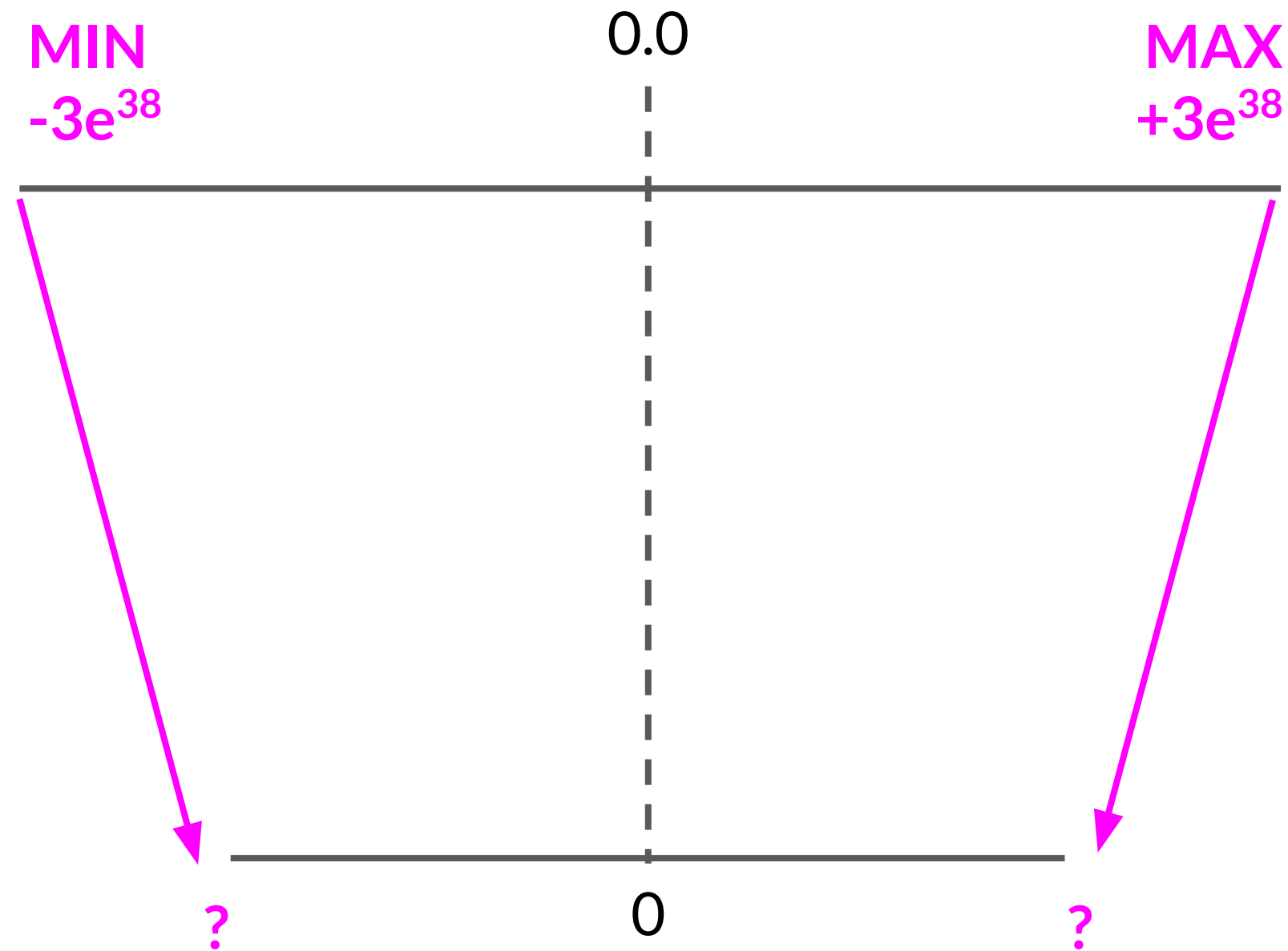
**4GB @ 32-bit
full precision**

Memory needed to train model



**24GB @ 32-bit
full precision**

Quantization



FP32

32-bit floating point

Range:

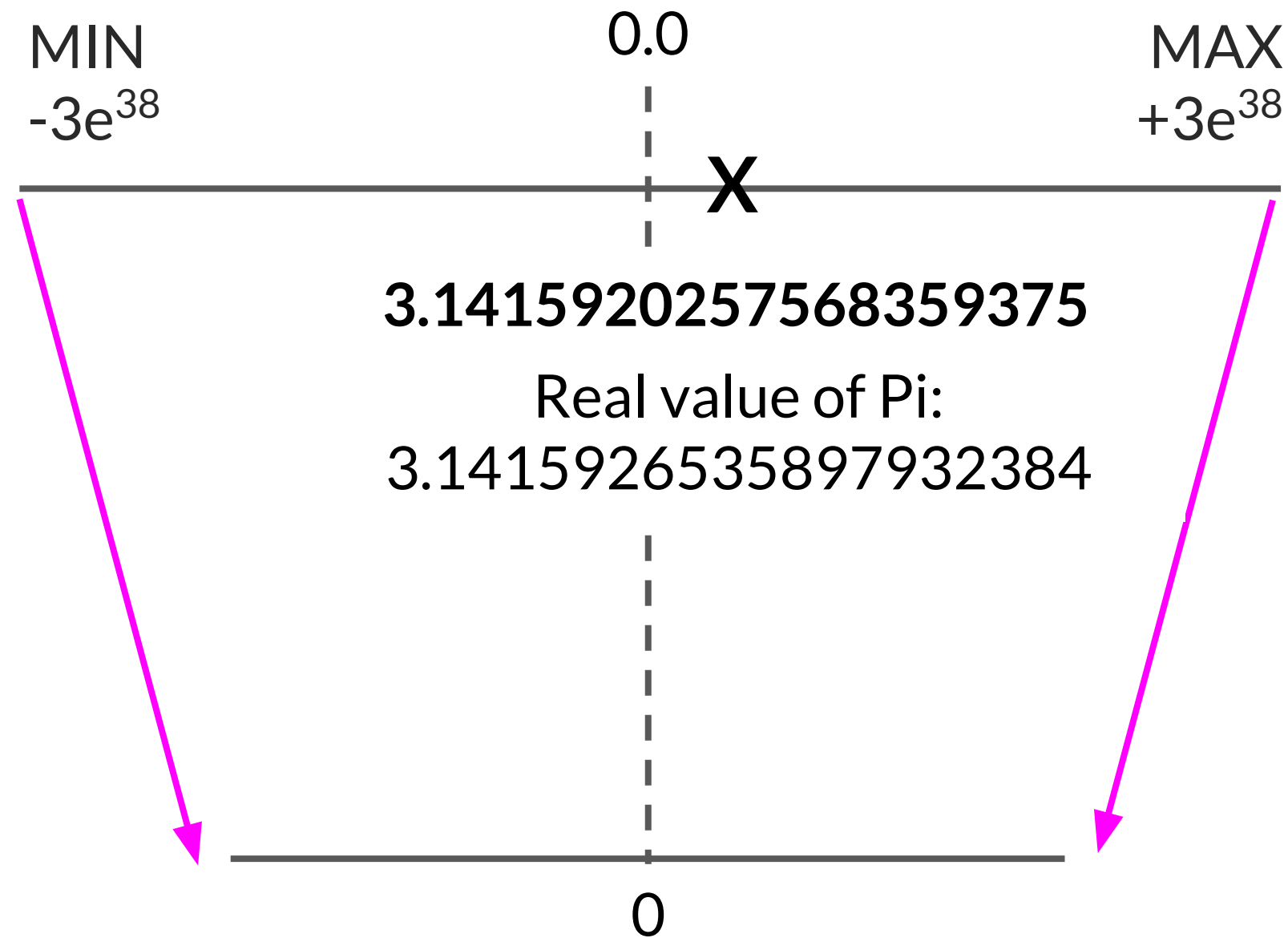
From $-3e^{38}$ to $+3e^{38}$

FP16 | BFLOAT16 | INT8

16-bit floating point | 8-bit integer

Quantization: FP32

Let's store Pi: 3.141592



FP32

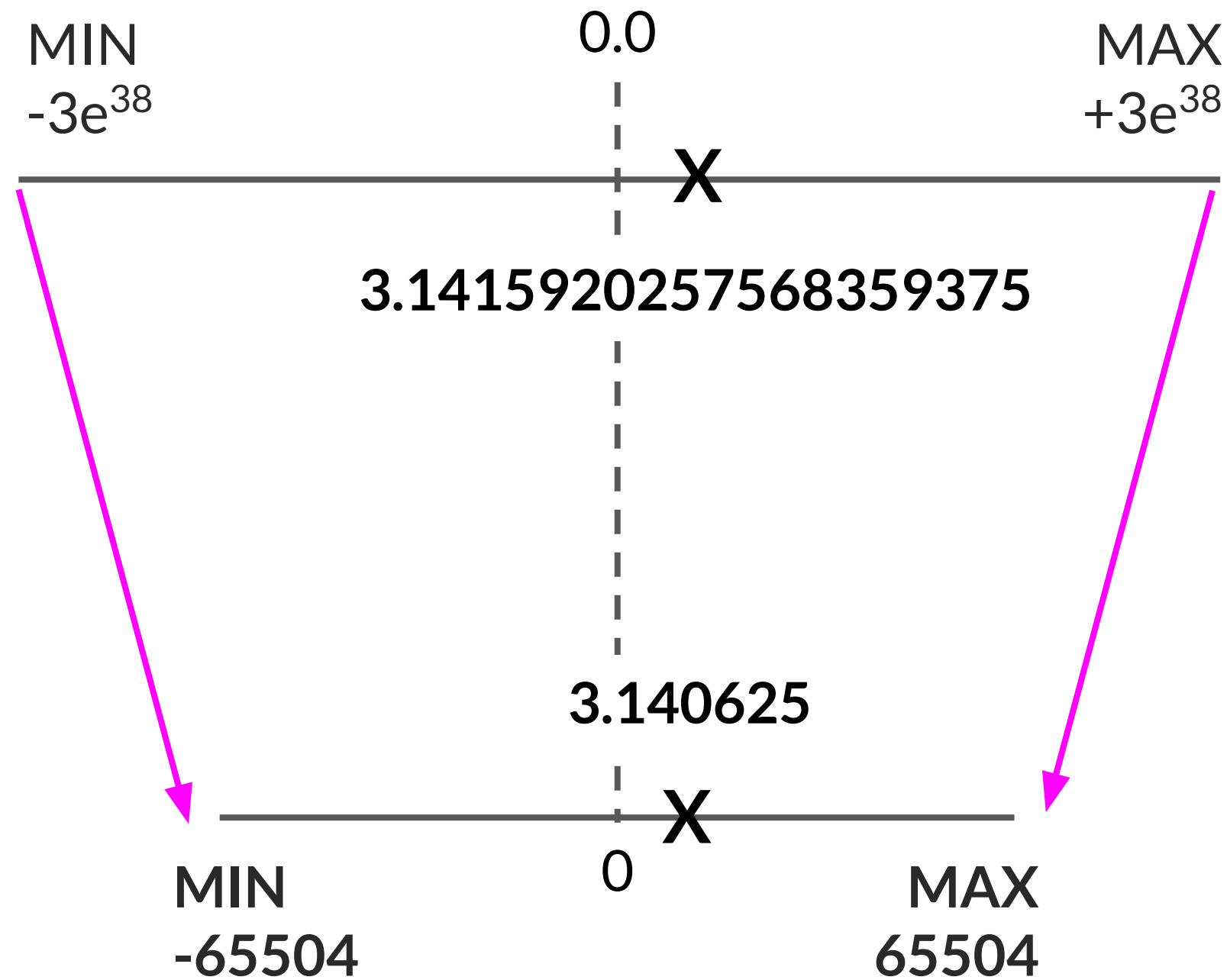
0 10000000 10010010000111111011000

Sign 1 bit Exponent 8 bits Fraction 23 bits

Mantissa / Significand
= Precision

Quantization: FP16

Let's store Pi: 3.141592



FP32 4 bytes memory

0 10000000 10010010000111111011000

Sign 1 bit Exponent 8 bits Fraction 23 bits

FP16 2 bytes memory

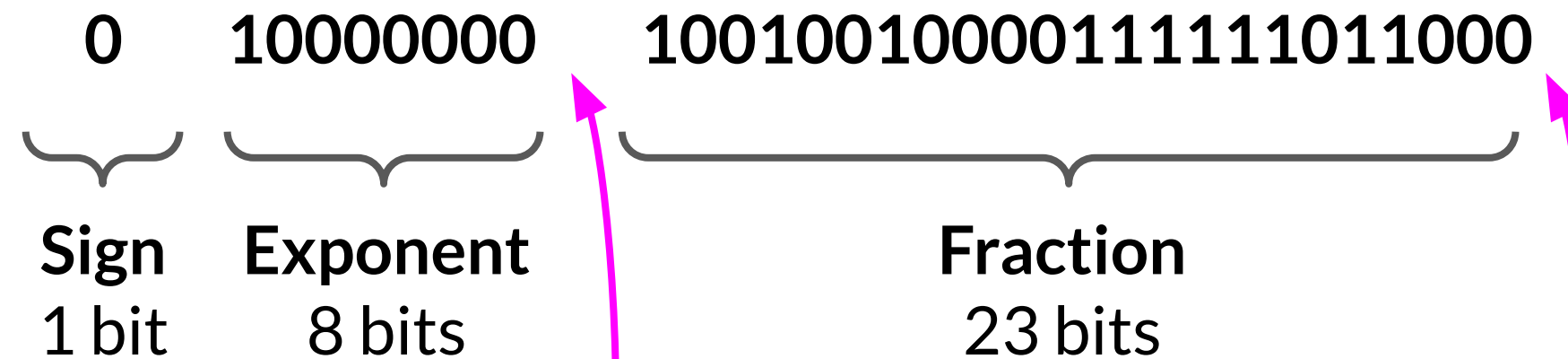
0 10000 1001001000

Sign 1 bit Exponent 5 bits Fraction 10 bits

Quantization: BFLOAT16

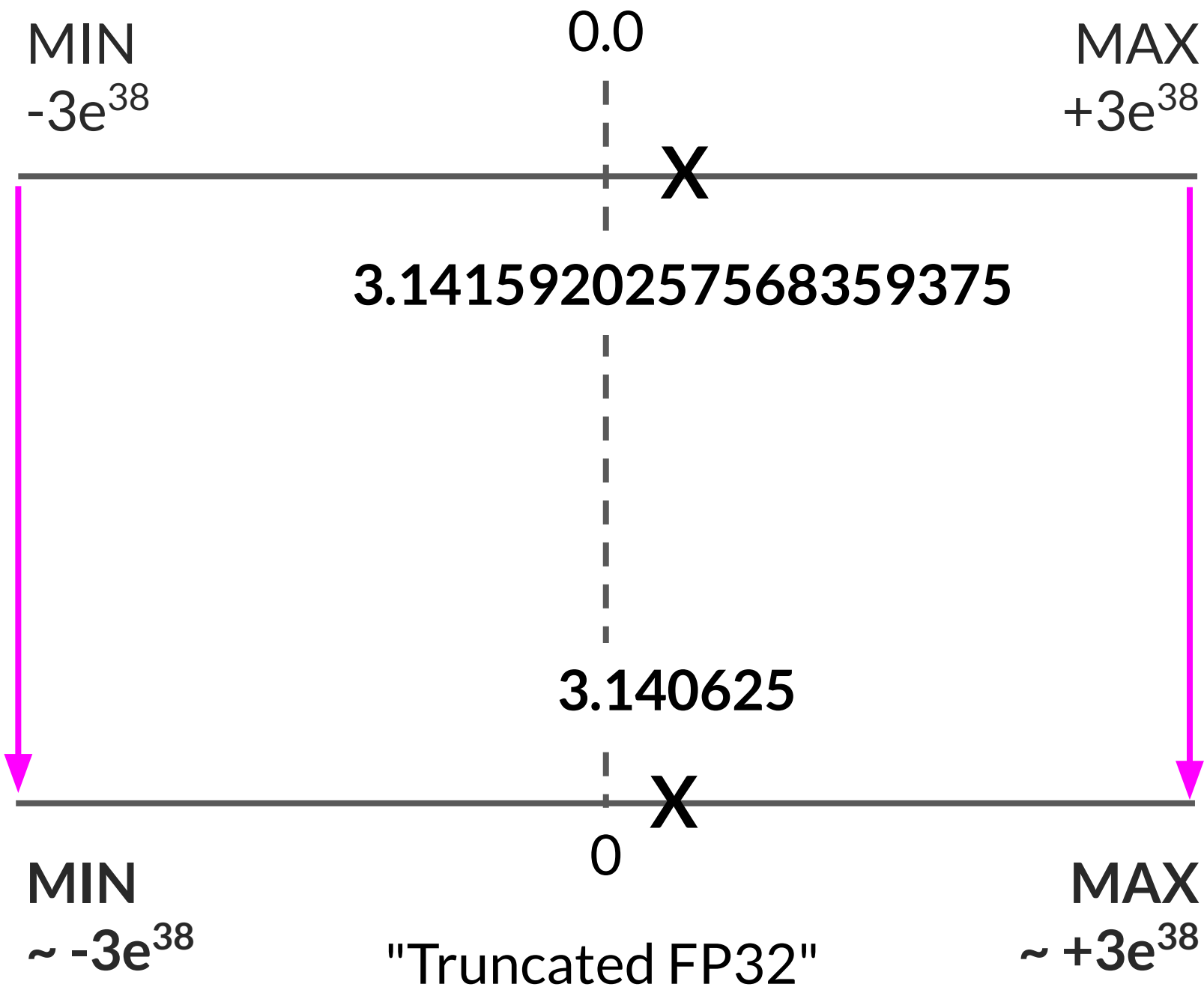
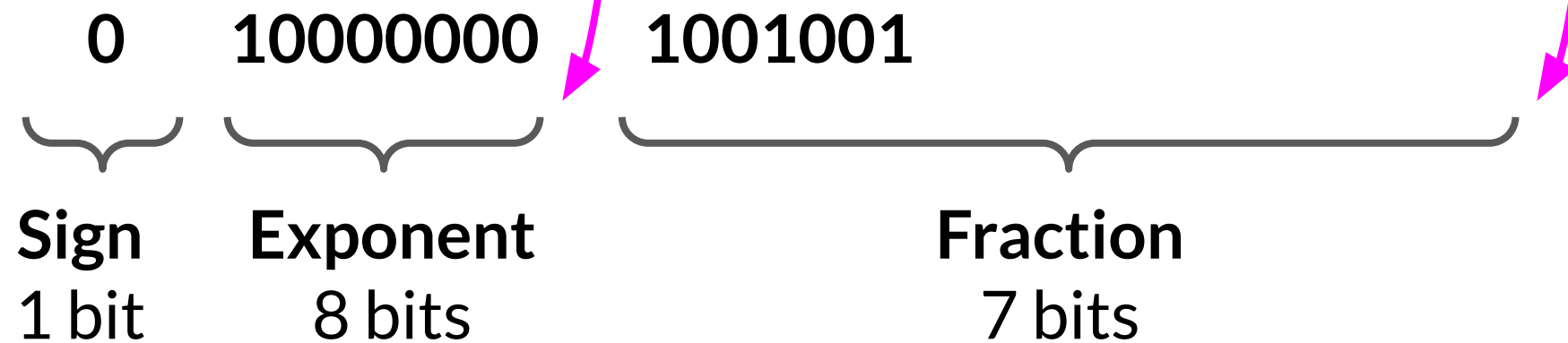
Let's store Pi: 3.141592

FP32 4 bytes memory



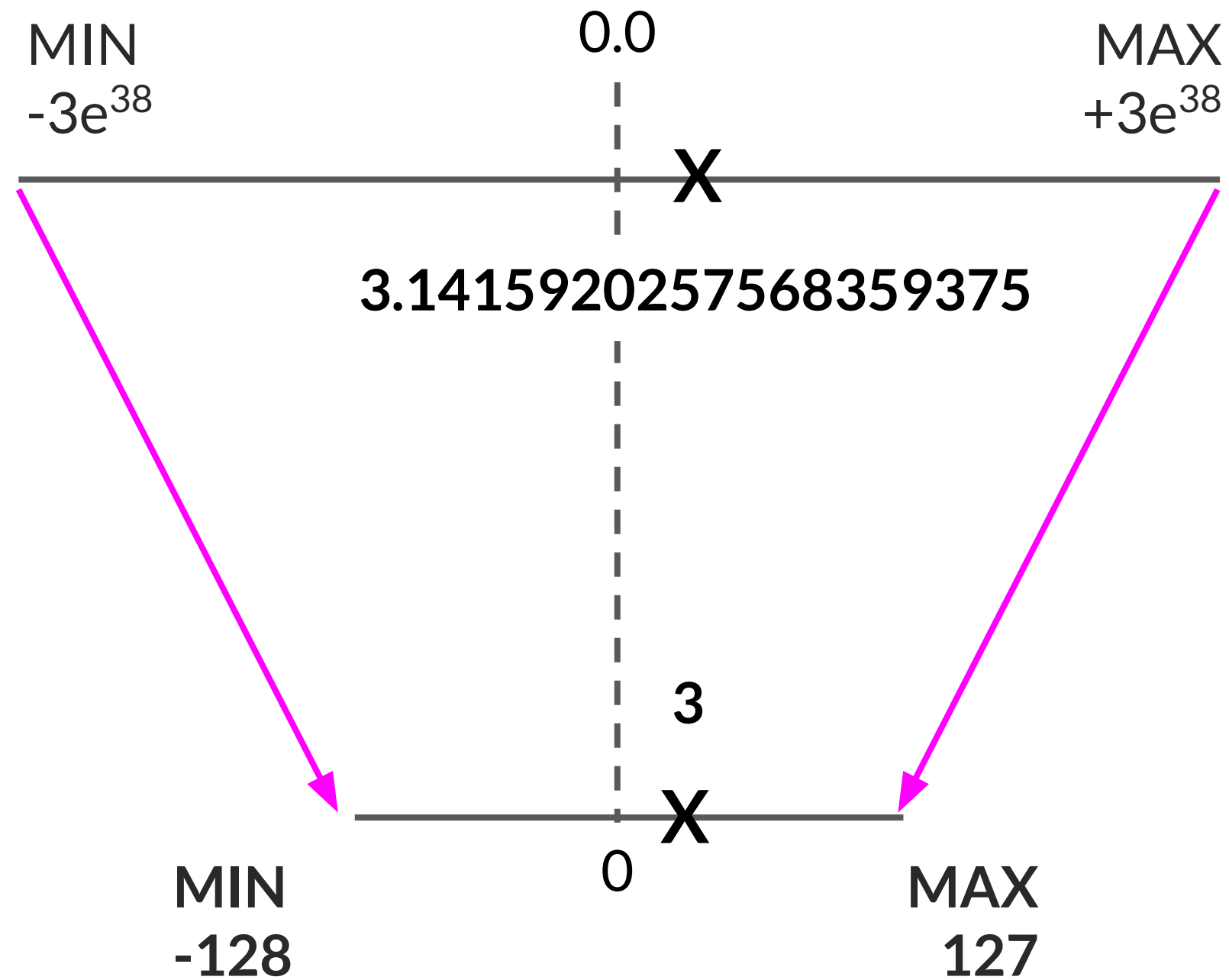
BFLOAT16 | BF16

2 bytes memory



Quantization: INT8

Let's store Pi: 3.141592



FP32 4 bytes memory

0 10000000 10010010000111111011000

Sign 1 bit Exponent 8 bits Fraction 23 bits

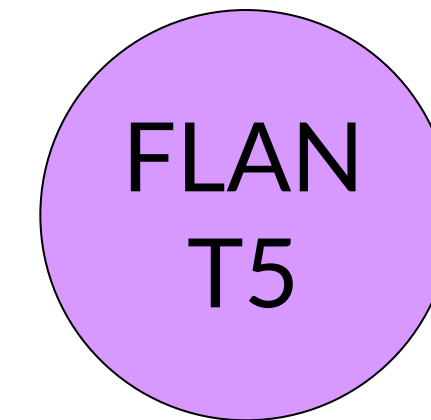
INT8 1 byte memory

0 0000011

Sign 1 bit Fraction 7 bits

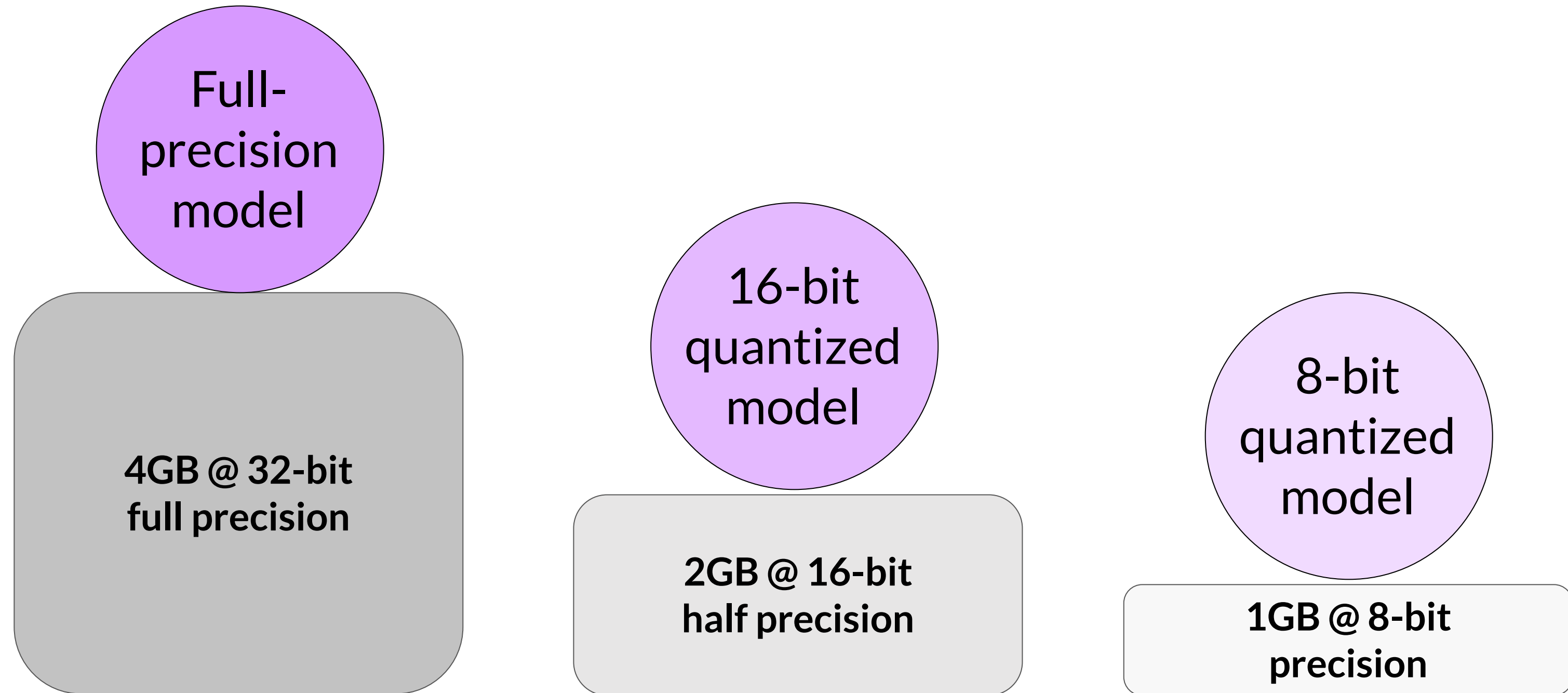
Quantization: Summary

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte



- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

Approximate GPU RAM needed to store 1B parameters



Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, <https://github.com/facebookresearch/bitsandbytes>

GPU RAM needed to train larger models

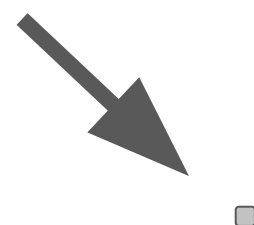
**1B param
model**

**175B param
model**

**500B param
model**

**4,200 GB @ 32-bit
full precision**

**12,000 GB @ 32-bit
full precision**



GPU RAM needed to train larger models

As model sizes get larger, you will need to split your model across multiple GPUs for training

**1B param
model**

4,200 GB @ 32-bit
full precision

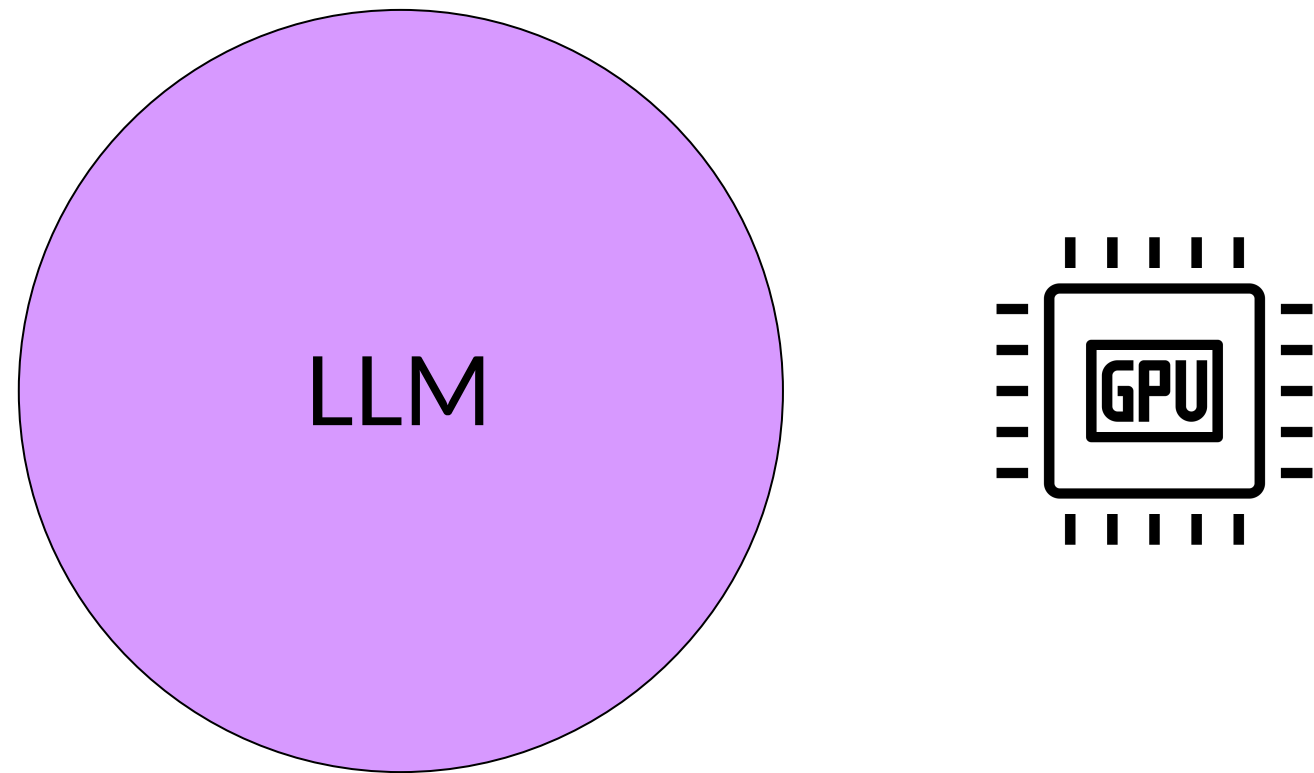
**175B param
model**

**500B param
model**

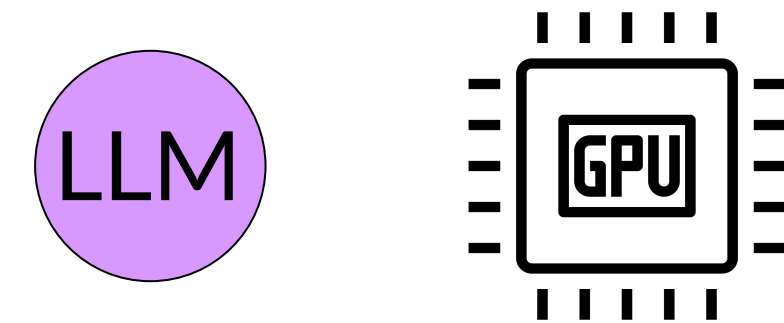
12,000 GB @ 32-bit
full precision

Efficient Multi-GPU Compute Strategies

When to use distributed compute

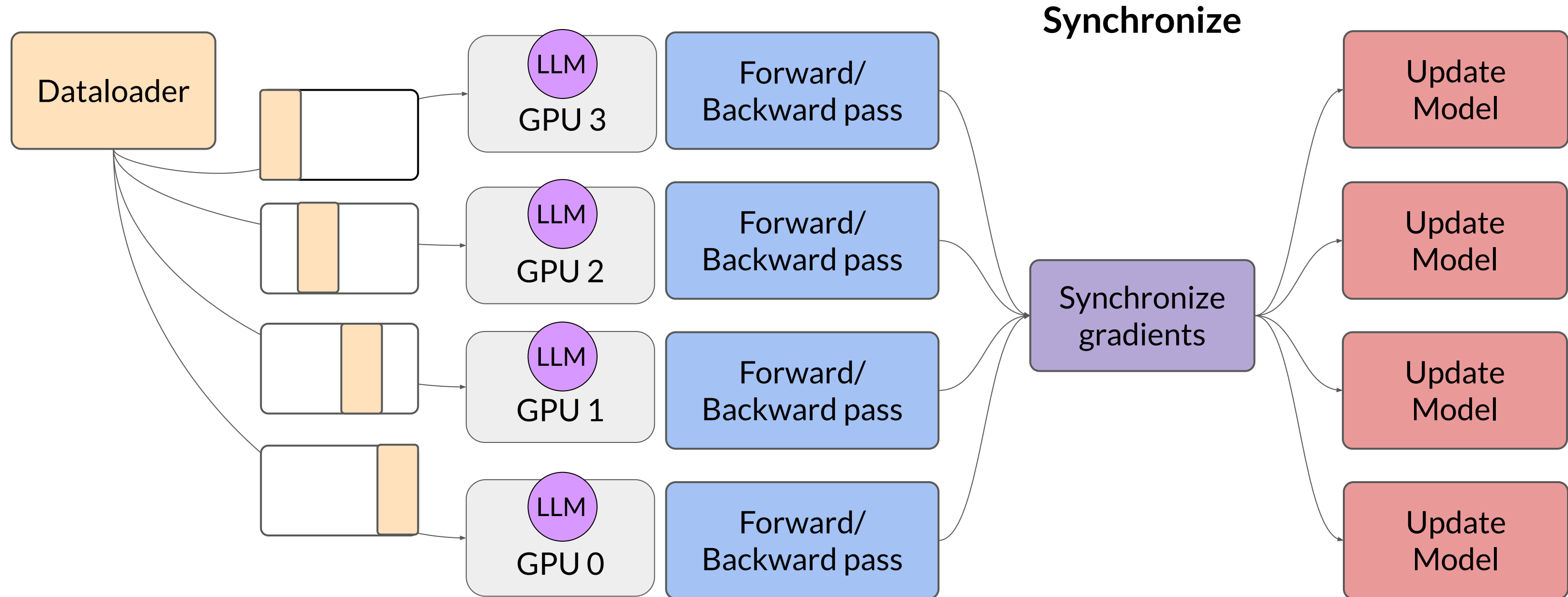


Model too big for single GPU



Model fits on GPU, train data in parallel

Distributed Data Parallel (DDP)



Fully Sharded Data Parallel (FSDP)

- Motivated by the “ZeRO” paper - zero data overlap between GPUs

ZeRO: Memory Optimizations Toward Training Trillion
Parameter Models

Samyam Rajbhandari*, Jeff Rasley*, Olatunji Ruwase, Yuxiong He
{samyamr, jerasley, olruwase, yuxhe}@microsoft.com

Sources:

Rajbhandari et al. 2019: “ZeRO: Memory Optimizations Toward Training Trillion Parameter Models”

Zhao et al. 2023: “PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel”

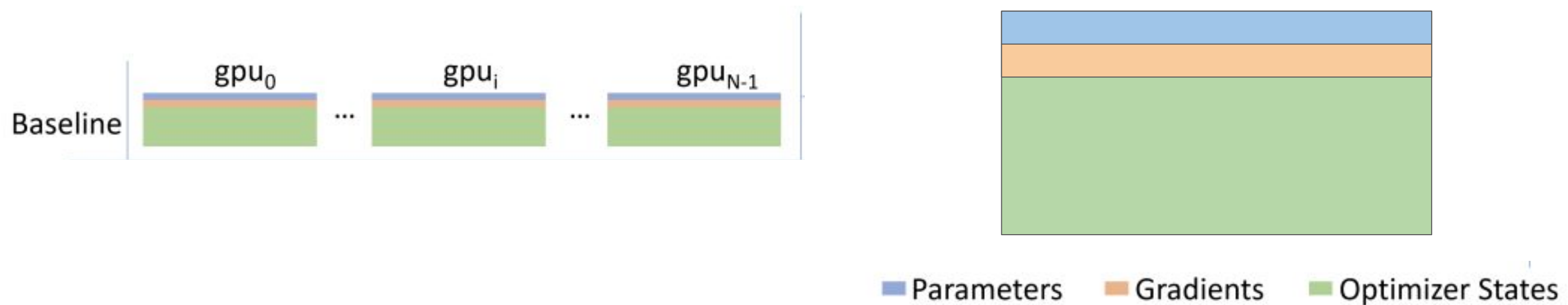
Recap: Additional GPU RAM needed for training

	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter
Adam optimizer (2 states)	+8 bytes per parameter
Gradients	+4 bytes per parameter
Activations and temp memory (variable size)	+8 bytes per parameter (high-end estimate)
TOTAL	=4 bytes per parameter +20 extra bytes per parameter

Sources: https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one#anatomy-of-models-memory, <https://github.com/facebookresearch/bitsandbytes>

Memory usage in DDP

- One full copy of model and training parameters on each GPU



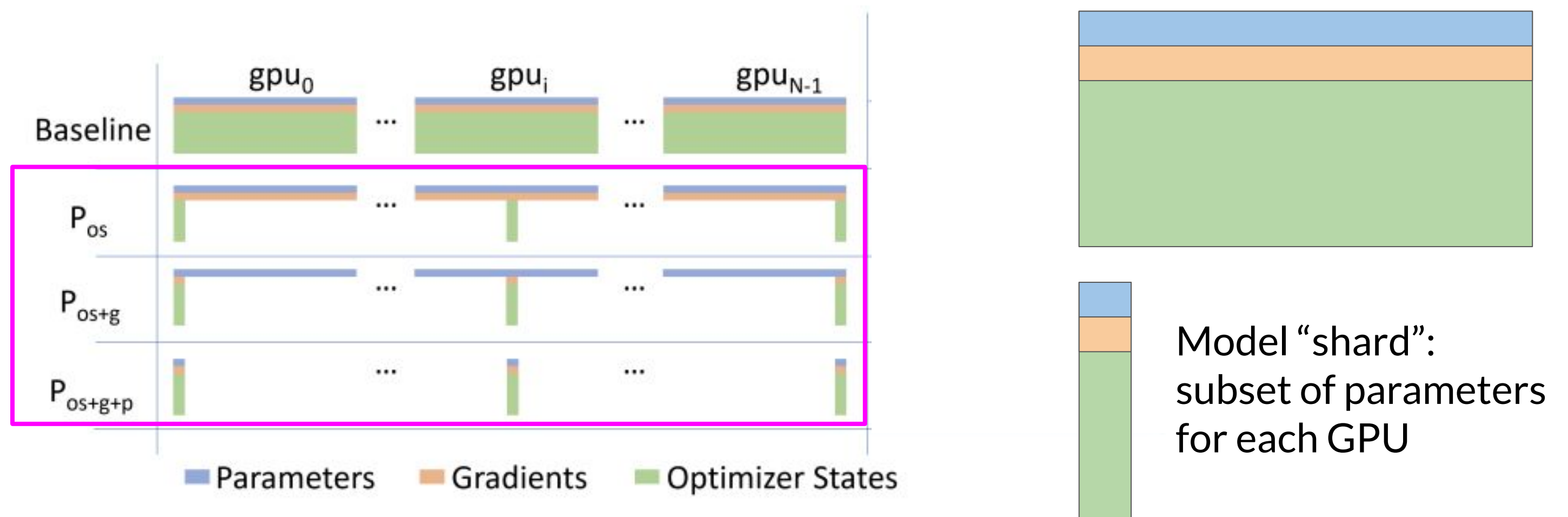
Sources:

Rajbhandari et al. 2019: "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models"

Zhao et al. 2023: "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel"

Zero Redundancy Optimizer (ZeRO)

- Reduces memory by distributing (sharding) the model parameters, gradients, and optimizer states across GPUs



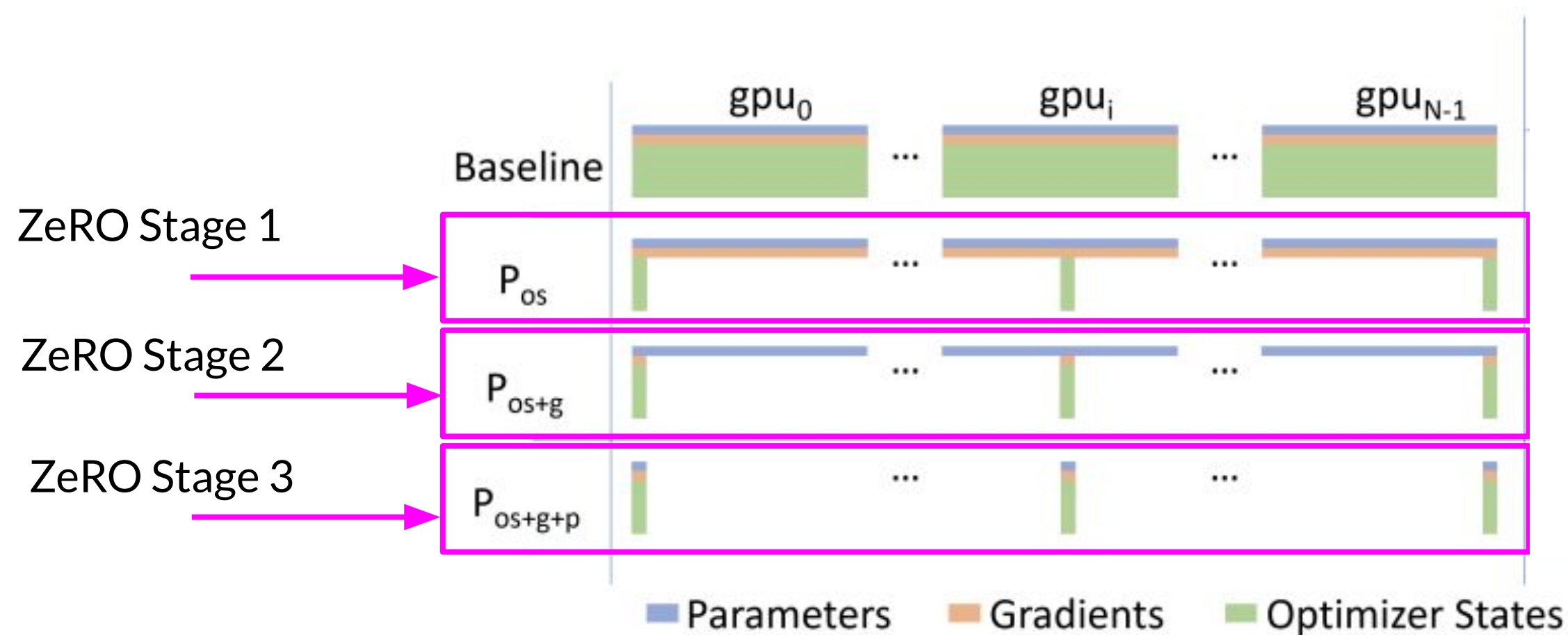
Sources:

Rajbhandari et al. 2019: "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models"

Zhao et al. 2023: "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel"

Zero Redundancy Optimizer (ZeRO)

- Reduces memory by distributing (sharding) the model parameters, gradients, and optimizer states across GPUs

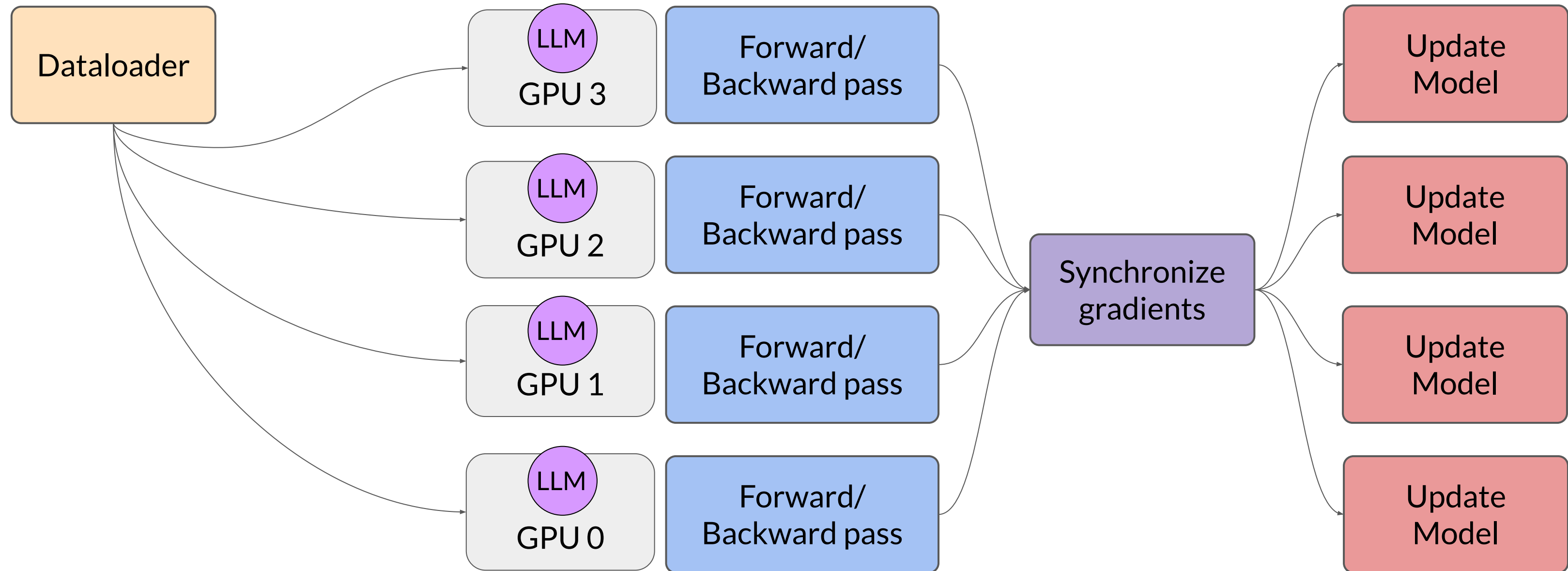


Sources:

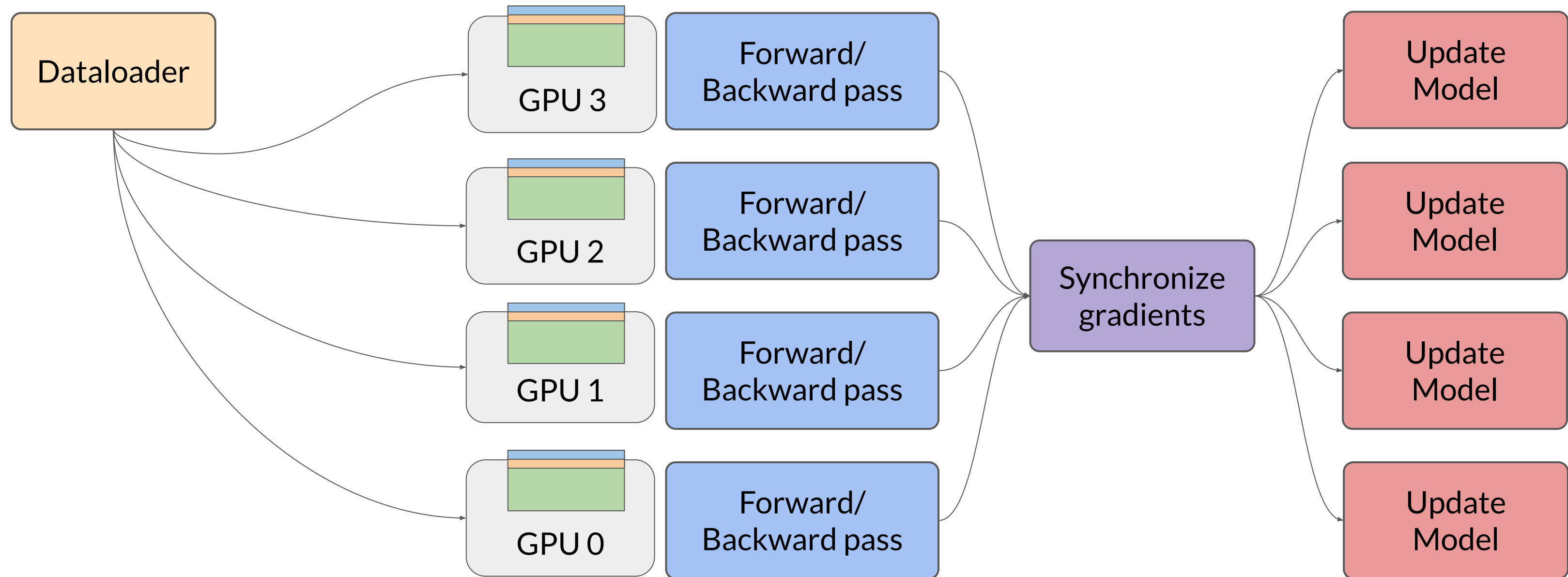
Rajbhandari et al. 2019: “ZeRO: Memory Optimizations Toward Training Trillion Parameter Models”

Zhao et al. 2023: “PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel”

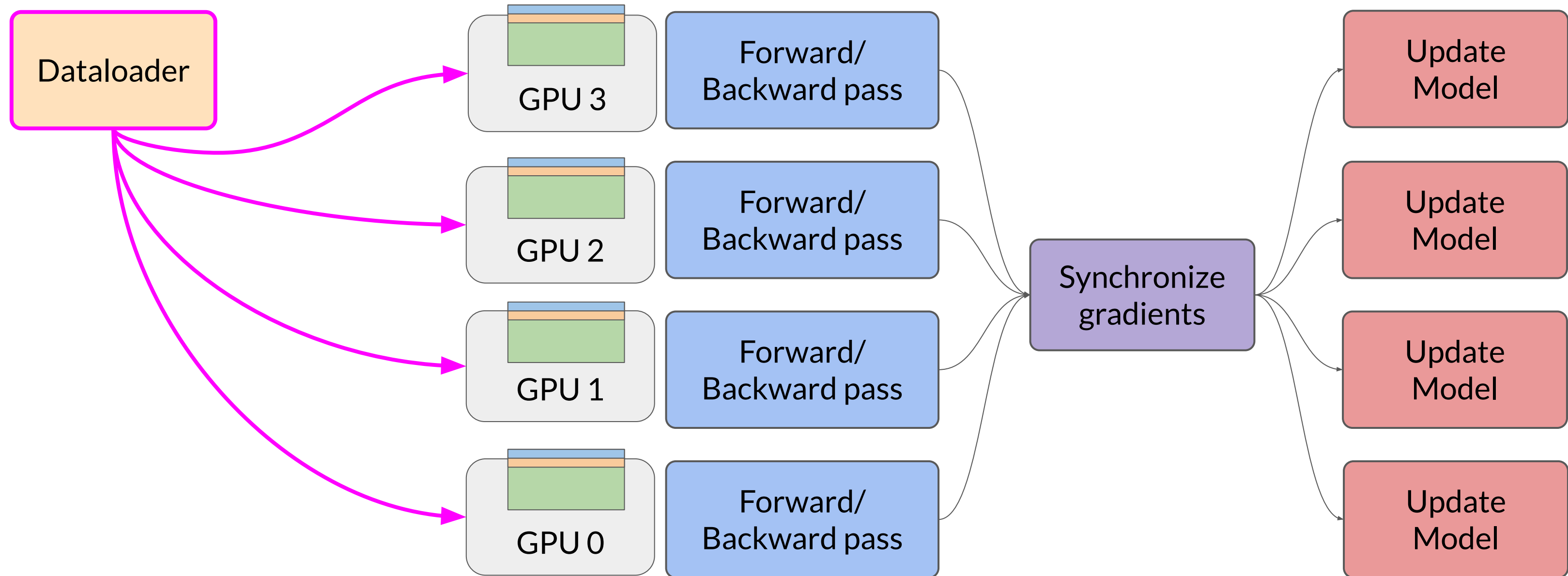
Distributed Data Parallel (DDP)



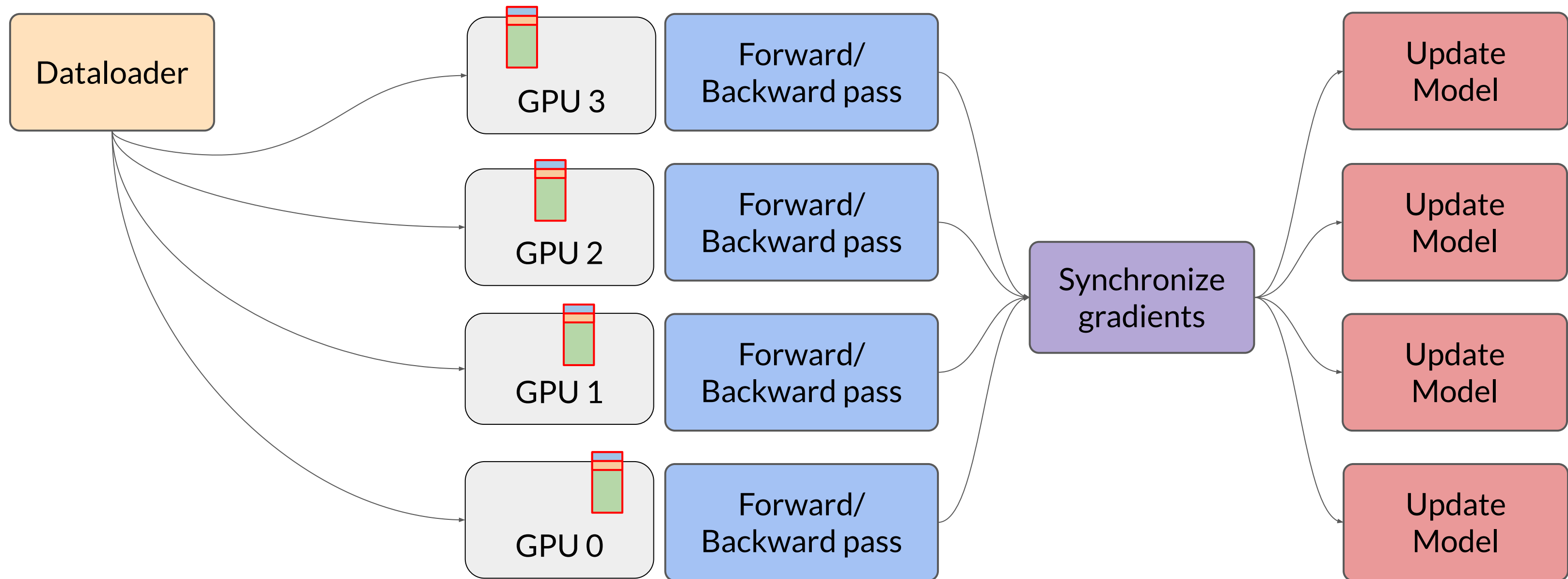
Distributed Data Parallel (DDP)



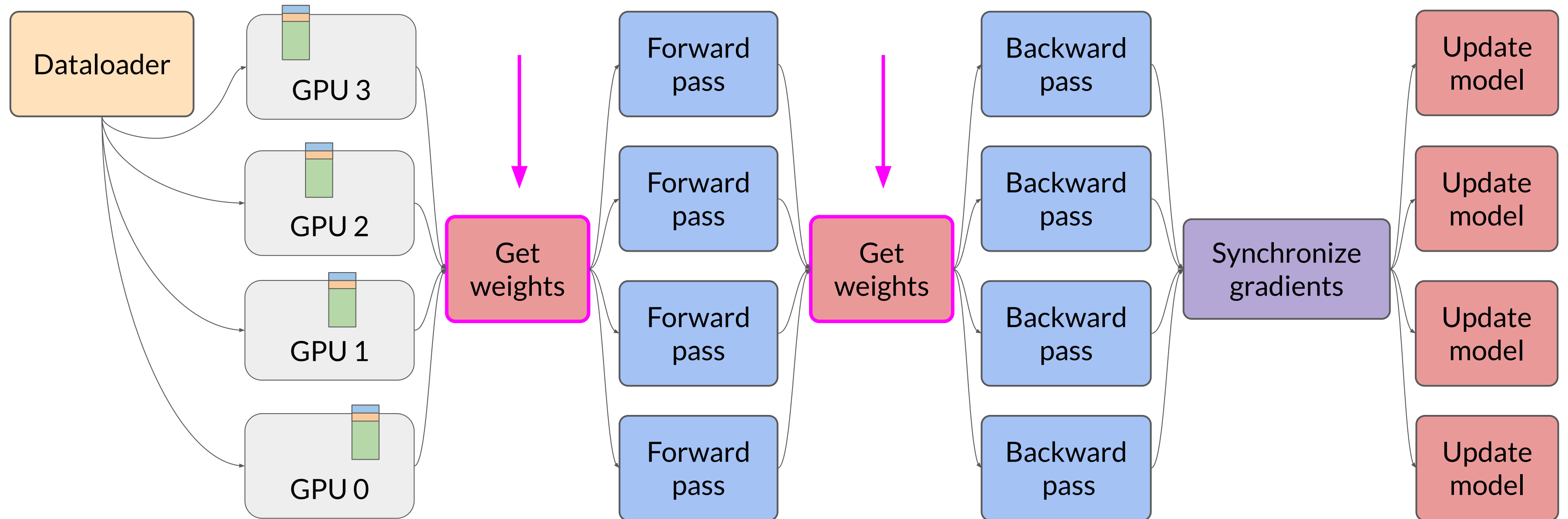
Fully Sharded Data Parallel (FSDP)



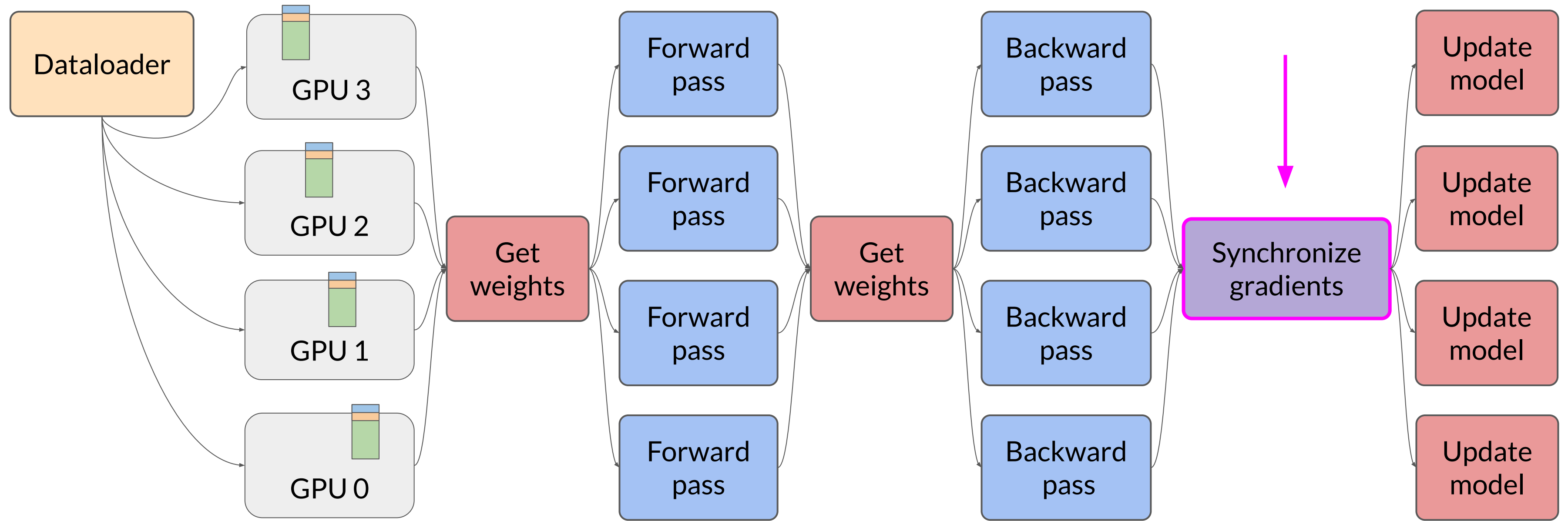
Fully Sharded Data Parallel (FSDP)



Fully Sharded Data Parallel (FSDP)



Fully Sharded Data Parallel (FSDP)

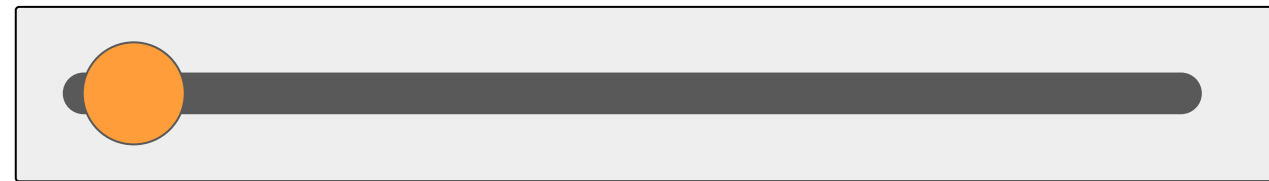


Fully Sharded Data Parallel (FSDP)

- Helps to reduce overall GPU memory utilization
- Supports offloading to CPU if needed
- Configure level of sharding via `sharding_factor`

Full replication (no sharding)

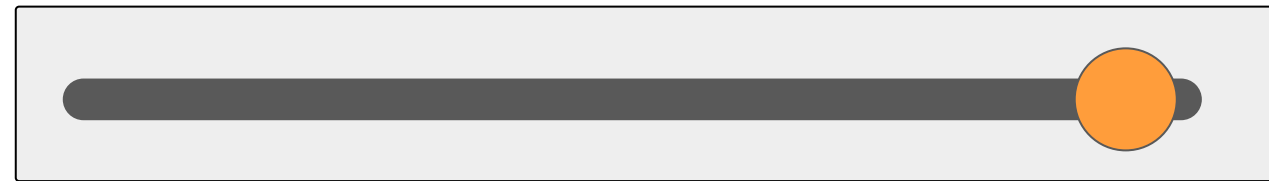
1 GPU



max. number of GPUs

Full sharding

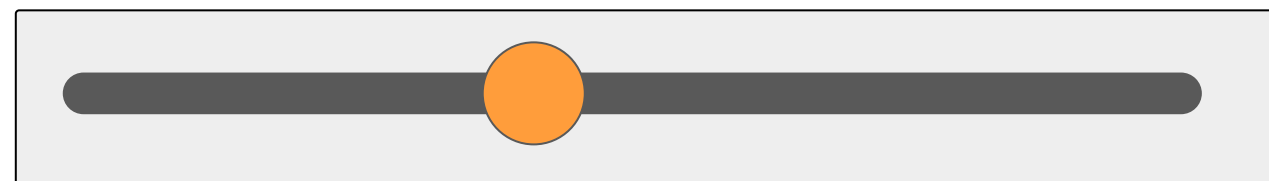
1 GPU



max. number of GPUs

Hybrid sharding

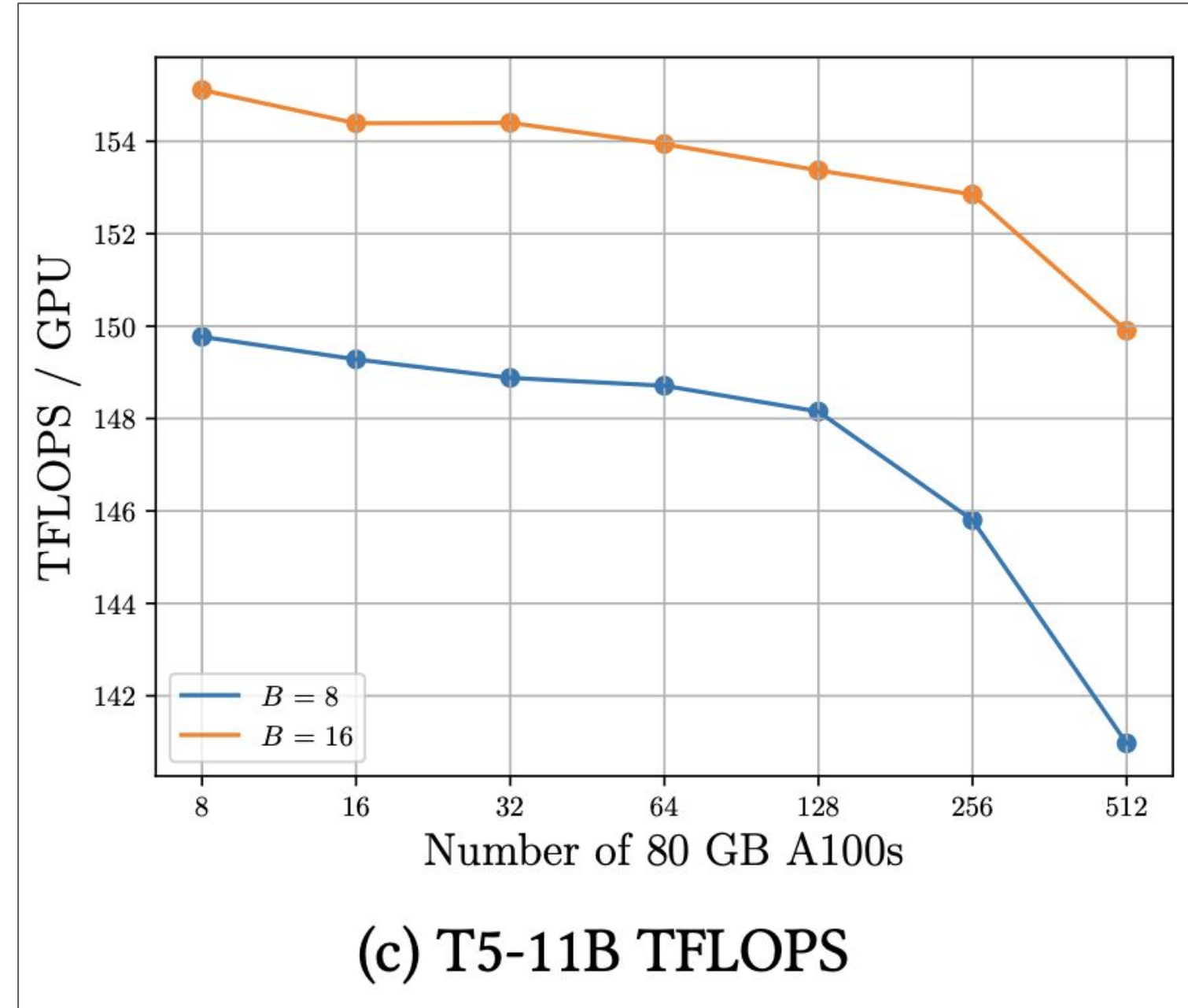
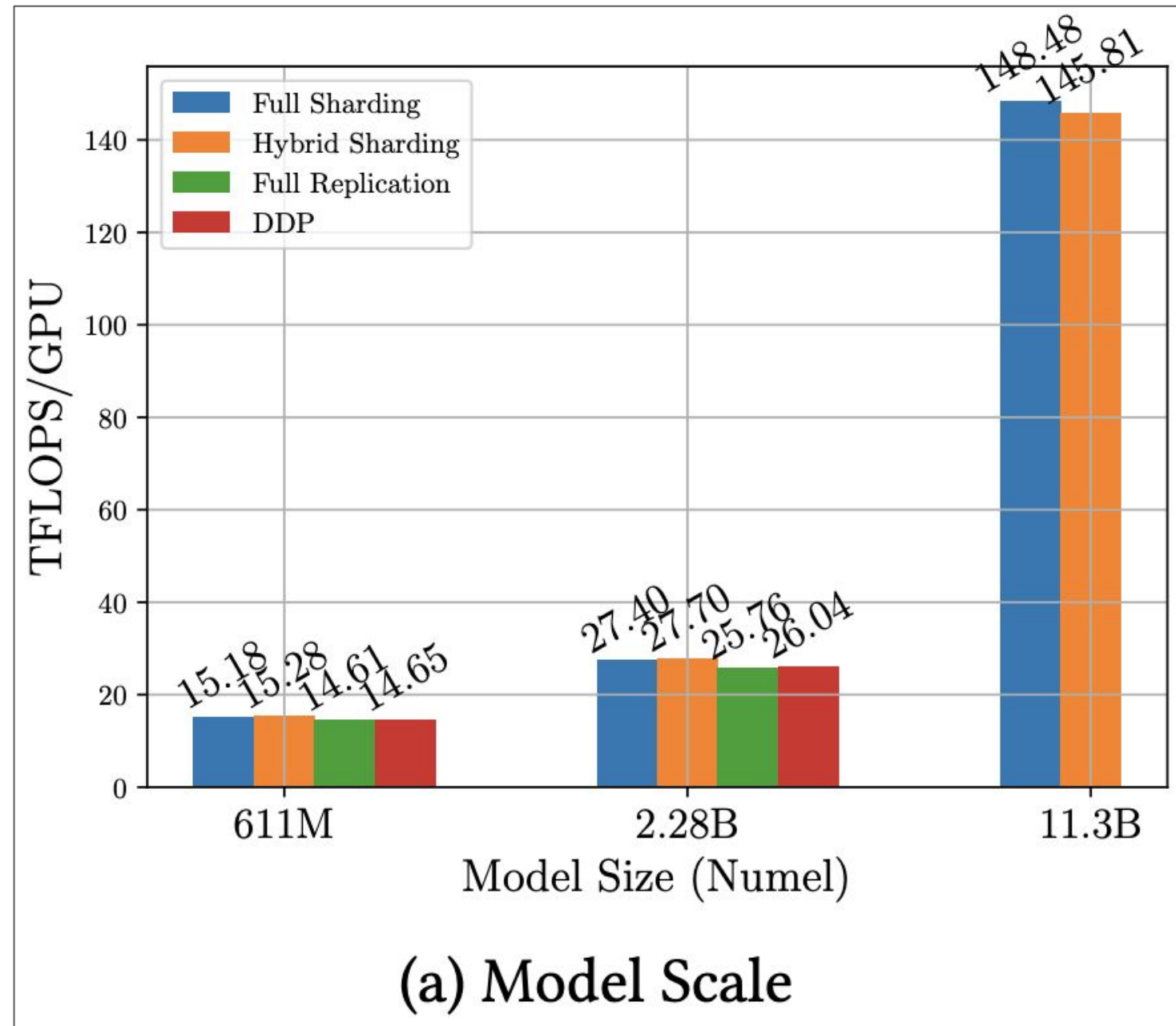
1 GPU



max. number of GPUs

Impact of using FSDP

Note: 1 teraFLOP/s = 1,000,000,000,000 (one trillion) floating point operations per second



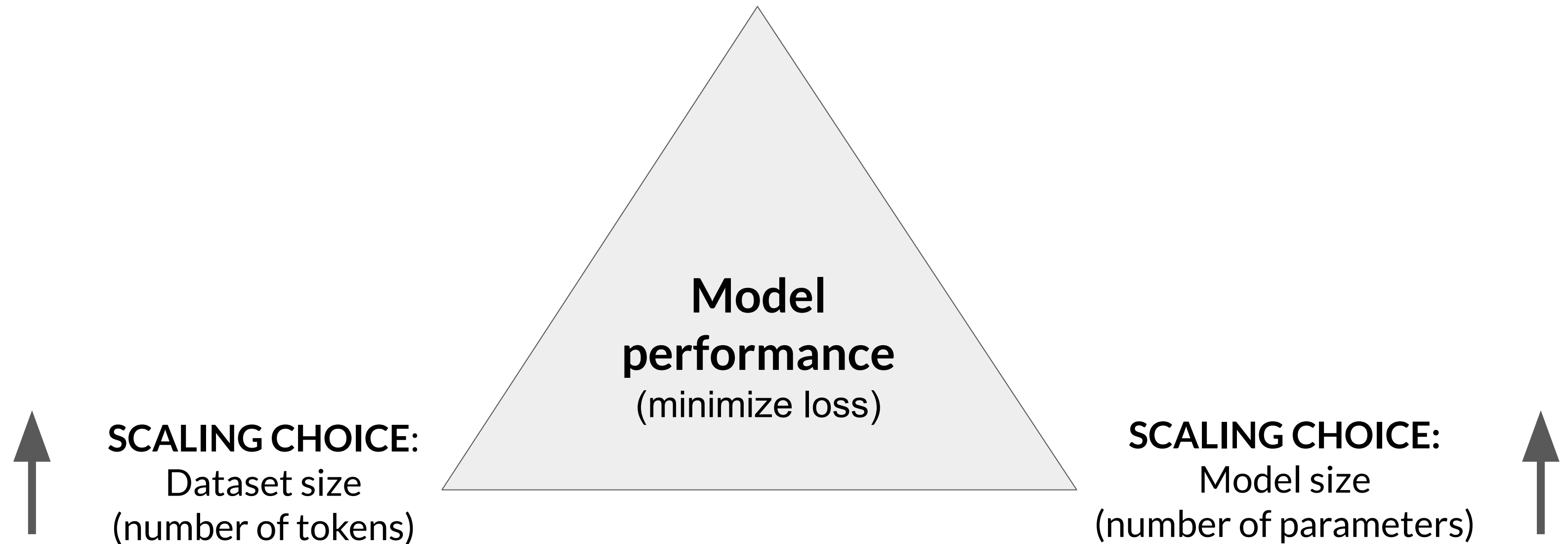
Zhao et al. 2023: "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel"

Scaling laws and compute-optimal models

Scaling choices for pre-training

Goal: maximize model performance

CONSTRAINT:
Compute budget
(GPUs, training time, cost)

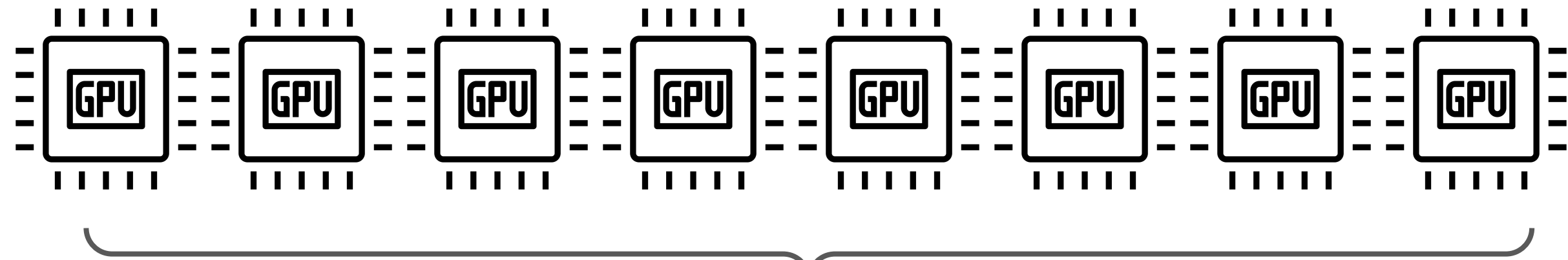


Compute budget for training LLMs

1 “petaflop/s-day” =

floating point operations performed at rate of 1 petaFLOP per second for one day

NVIDIA V100s



Note: 1 petaFLOP/s = 1,000,000,000,000,000
(one quadrillion) floating point operations per second

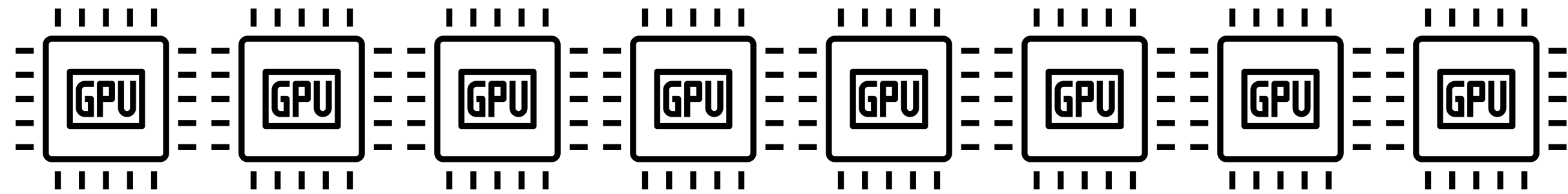
1 petaflop/s-day is these chips running at full efficiency for 24 hours

Compute budget for training LLMs

1 “petaflop/s-day” =

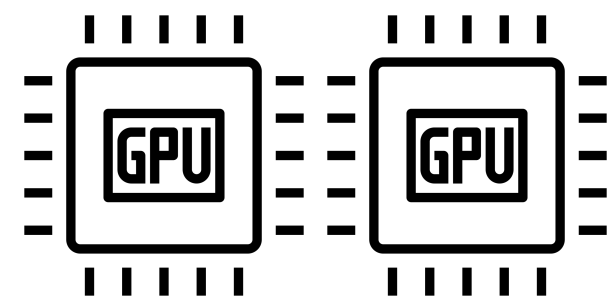
floating point operations performed at rate of 1 petaFLOP per second for one day

NVIDIA V100s



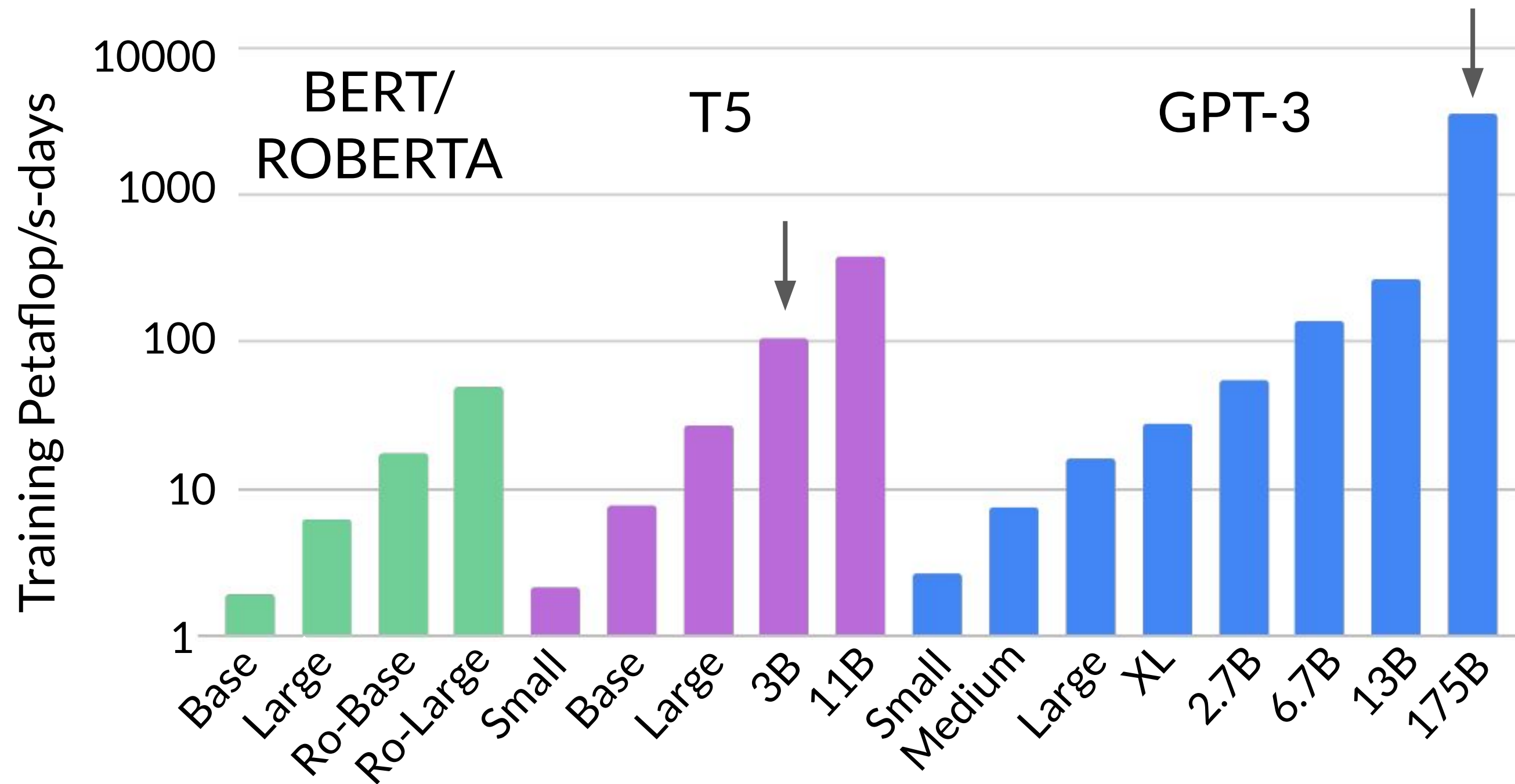
OR

NVIDIA A100s



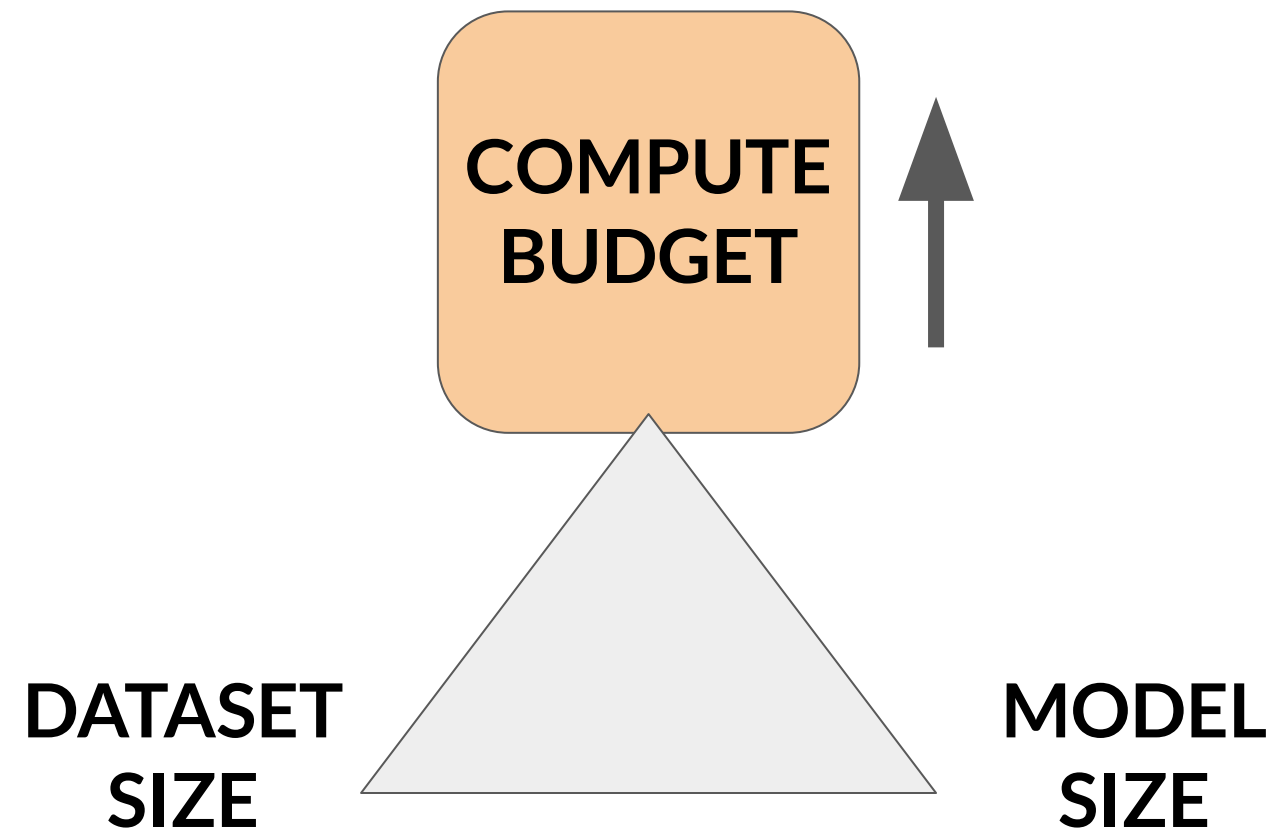
1 petaflop/s-day is these chips running at full efficiency for 24 hours

Number of petaflop/s-days to pre-train various LLMs



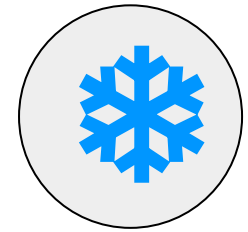
Source: Brown et al. 2020, "Language Models are Few-Shot Learners"

Compute budget vs. model performance

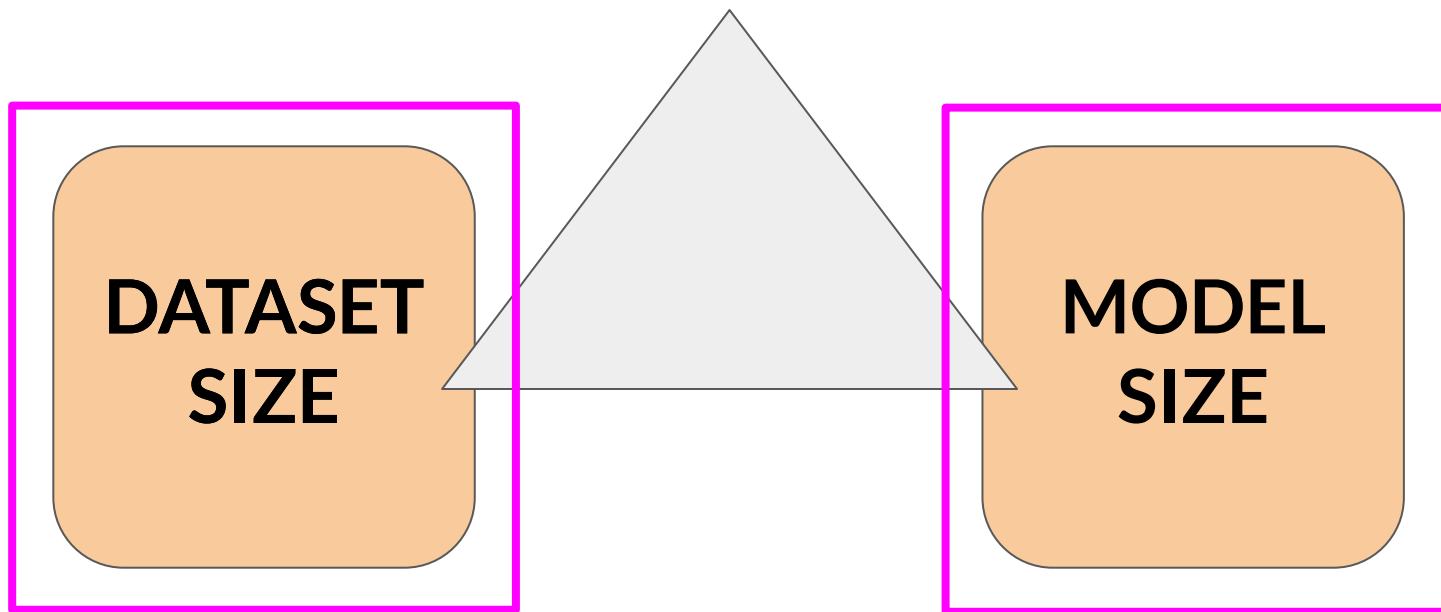


Source: Kaplan et al. 2020, "Scaling Laws for Neural Language Models"

Dataset size and model size vs. performance



**COMPUTE
BUDGET**

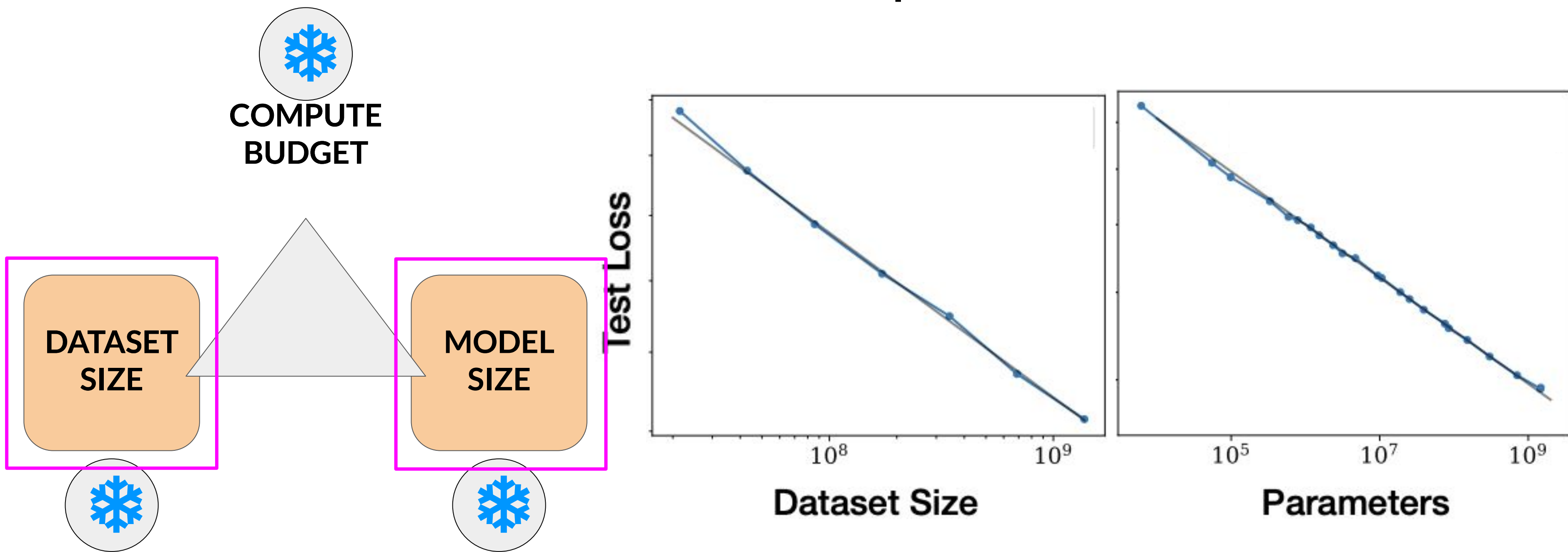


Compute resource constraints

- Hardware
- Project timeline
- Financial budget

Source: Kaplan et al. 2020, “Scaling Laws for Neural Language Models”

Dataset size and model size vs. performance



Source: Kaplan et al. 2020, "Scaling Laws for Neural Language Models"

Chinchilla paper

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

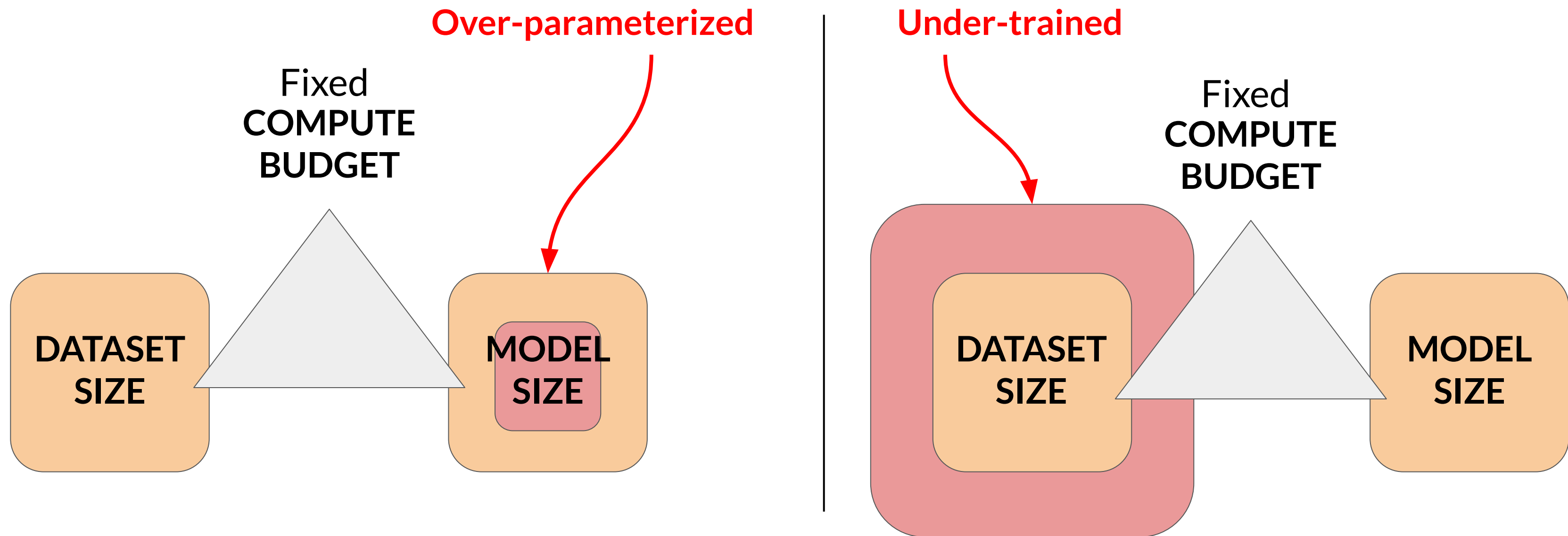
*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Jordan et al. 2022

Compute optimal models

- Very large models may be **over-parameterized** and **under-trained**
- Smaller models trained on more data could perform as well as large models



Chinchilla scaling laws for model and dataset size

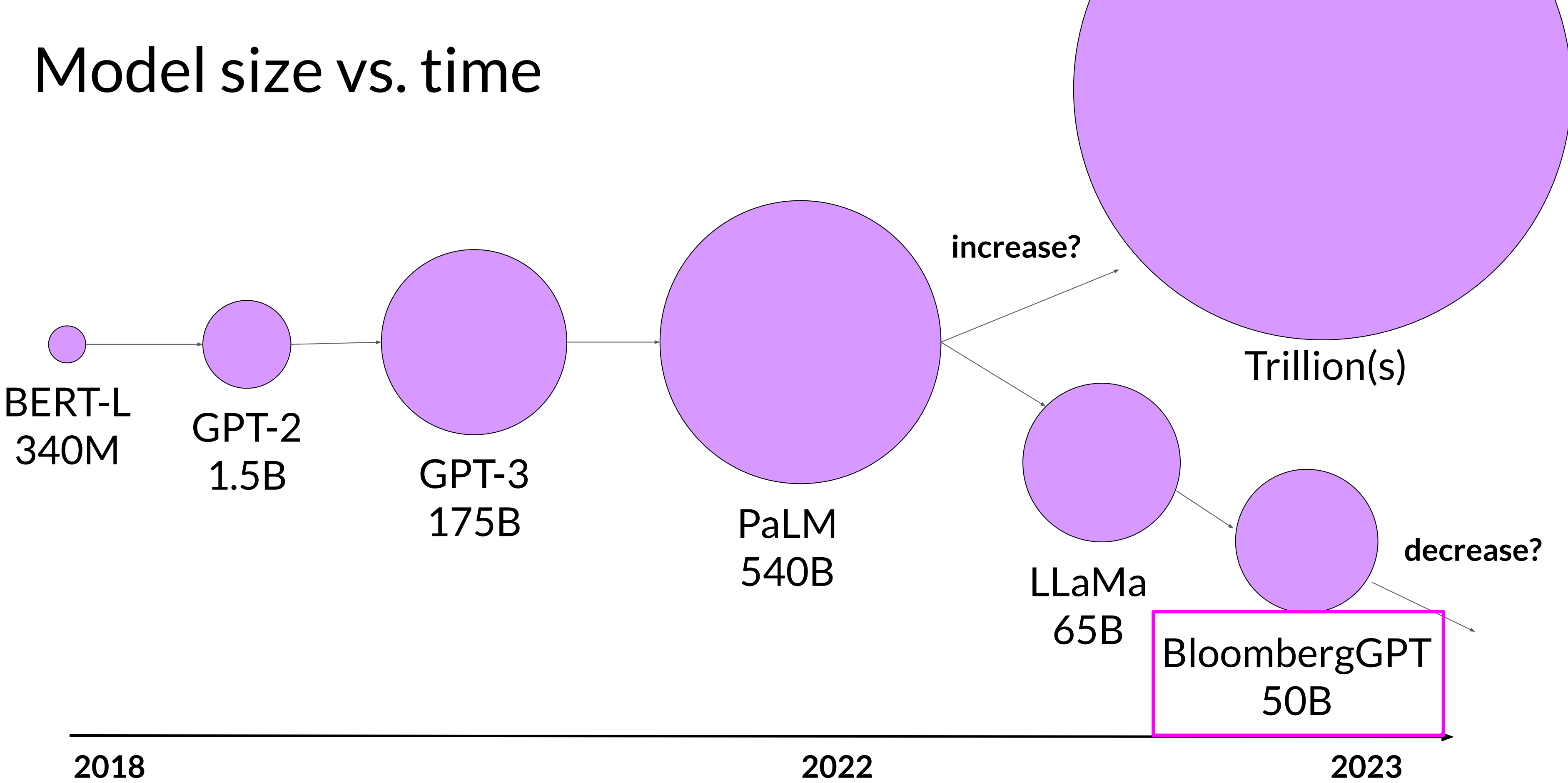
Model	# of parameters	Compute-optimal* # of tokens (~20x)	Actual # tokens
Chinchilla	70B	~1.4T	1.4T
LLaMA-65B	65B	~1.3T	1.4T
GPT-3	175B	~3.5T	300B
OPT-175B	175B	~3.5T	180B
BLOOM	176B	~3.5T	350B

Compute optimal training datasize
is **~20x** number of parameters

Sources: Hoffmann et al. 2022, "Training Compute-Optimal Large Language Models"
Touvron et al. 2023, "LLaMA: Open and Efficient Foundation Language Models"

* assuming models are trained to be
compute-optimal per Chinchilla paper

Model size vs. time



Pre-training for domain adaptation

Pre-training for domain adaptation

Legal language

Pre-training for domain adaptation

Legal language

The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.

The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.

Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.

Pre-training for domain adaptation

Legal language

The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.

The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.

Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.

Medical language

After a strenuous workout, the patient experienced severe myalgia that lasted for several days.

After the biopsy, the doctor confirmed that the tumor was malignant and recommended immediate treatment.

Sig: 1 tab po qid pc & hs



Take one tablet by mouth four times a day, after meals, and at bedtime.

BloombergGPT: domain adaptation for finance

BloombergGPT: A Large Language Model for Finance

Shijie Wu^{1,*}, Ozan İrsoy^{1,*}, Steven Lu^{1,*}, Vadim Dabravolski¹, Mark Dredze^{1,2}, Sebastian Gehrmann¹, Prabhanjan Kambadur¹, David Rosenberg¹, Gideon Mann¹

¹ Bloomberg, New York, NY USA

² Computer Science, Johns Hopkins University, Baltimore, MD USA

gmann16@bloomberg.net

Abstract

The use of NLP in the realm of financial technology is broad and complex, with applications ranging from sentiment analysis and named entity recognition to question answering. Large Language Models (LLMs) have been shown to be effective on a variety of tasks; however, no LLM specialized for the financial domain has been reported in literature. In this work, we present BLOOMBERGGPT, a 50 billion parameter language model that is trained on a wide range of financial data. We construct a 363 billion token dataset based on Bloomberg's extensive data sources, perhaps the largest domain-specific dataset yet, augmented with 345 billion tokens from general purpose datasets. We validate BLOOMBERGGPT on standard LLM benchmarks, open financial benchmarks, and a suite of internal benchmarks that most accurately reflect our intended usage. Our mixed dataset training leads to a model that outperforms existing models on financial tasks by significant margins without sacrificing performance on general LLM benchmarks. Additionally, we explain our modeling choices, training process, and evaluation methodology. As a next step, we plan to release training logs (Chronicles) detailing our experience in training BLOOMBERGGPT.

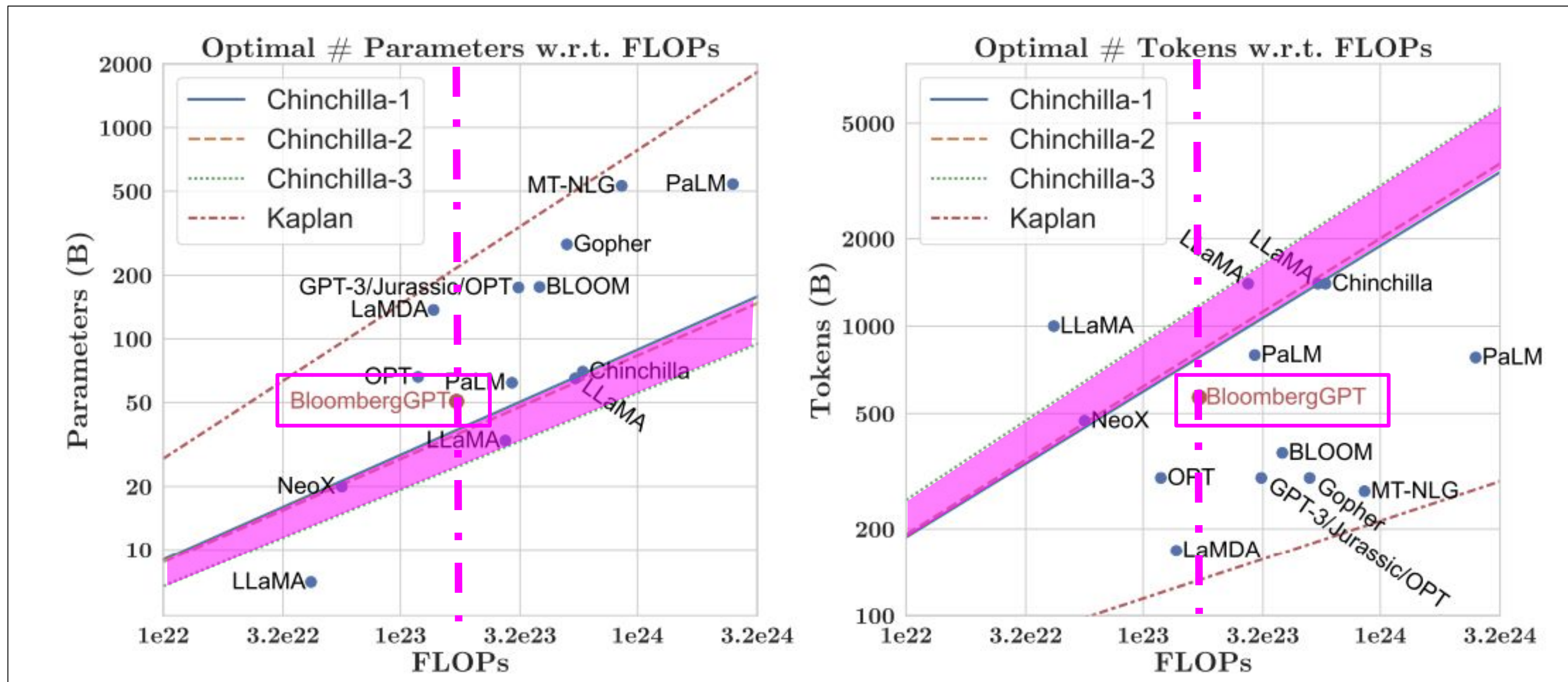
~51%

**Financial
(Public & Private)**

~49%

**Other
(Public)**

BloombergGPT relative to other LLMs



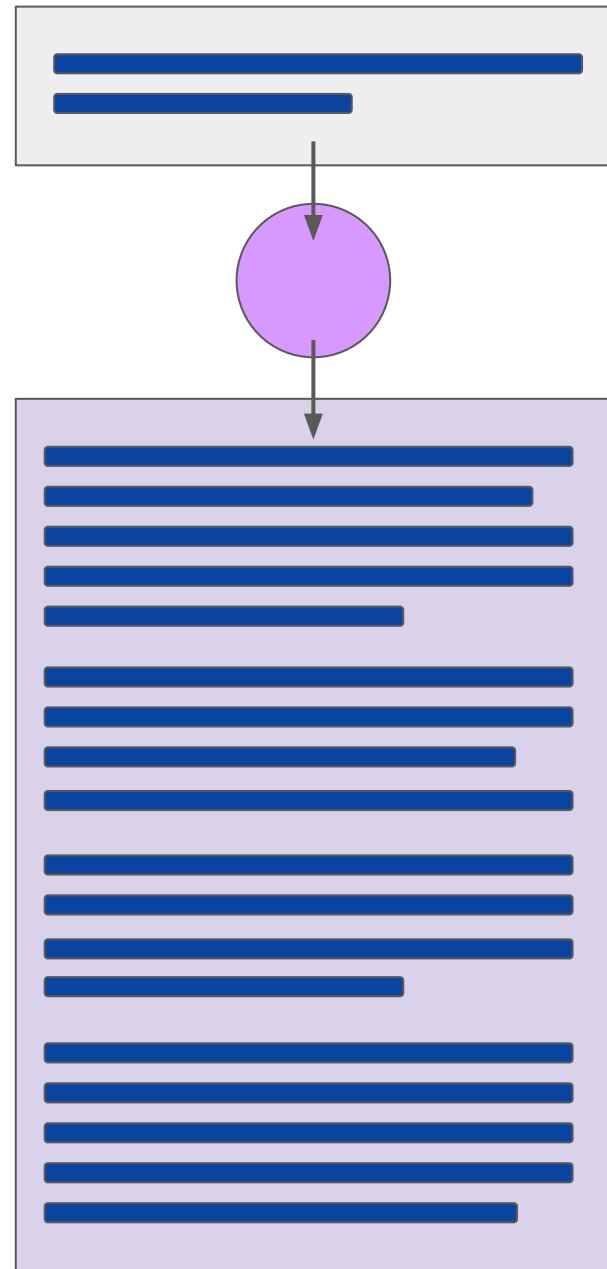
Source: Wu et al. 2023, "BloombergGPT: A Large Language Model for Finance"

Key takeaways

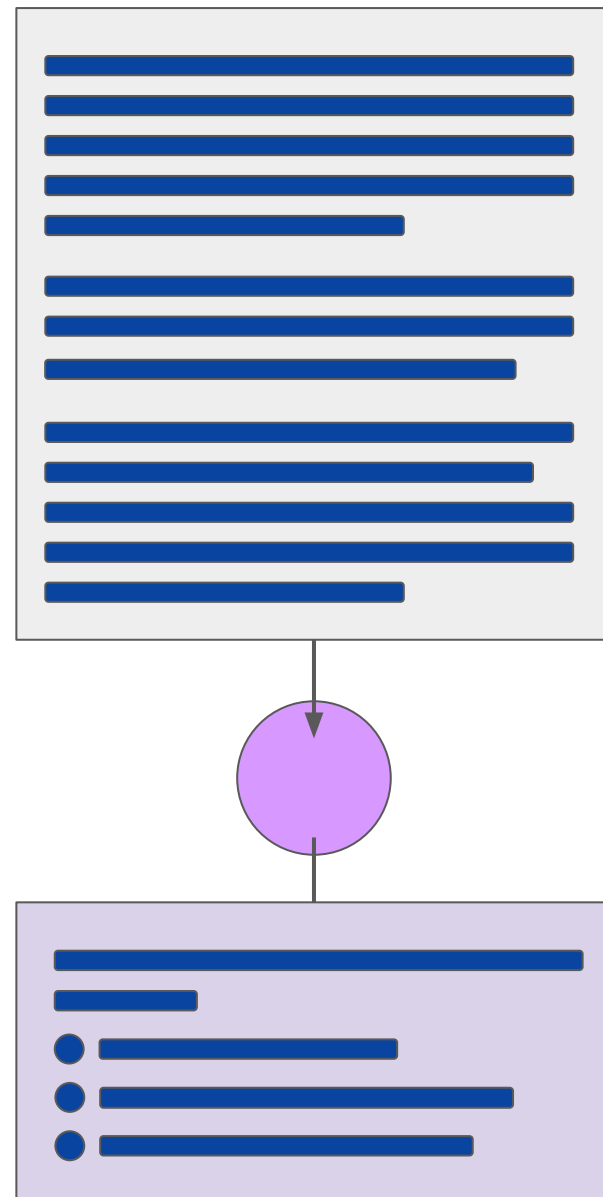


LLM use cases & tasks

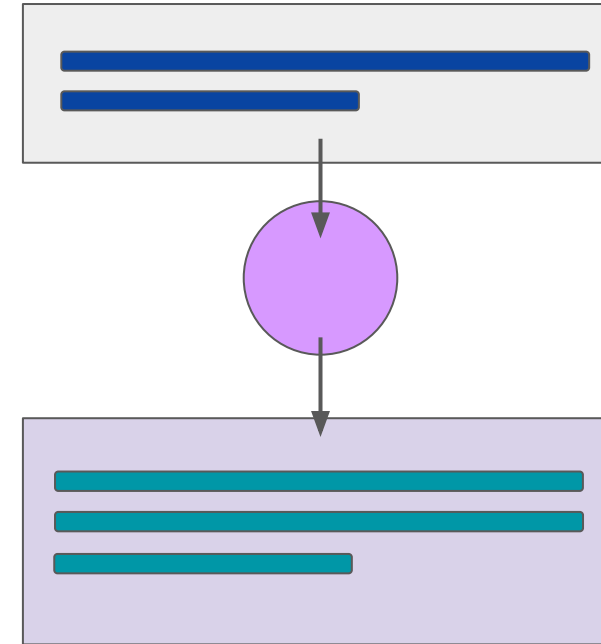
Essay Writing



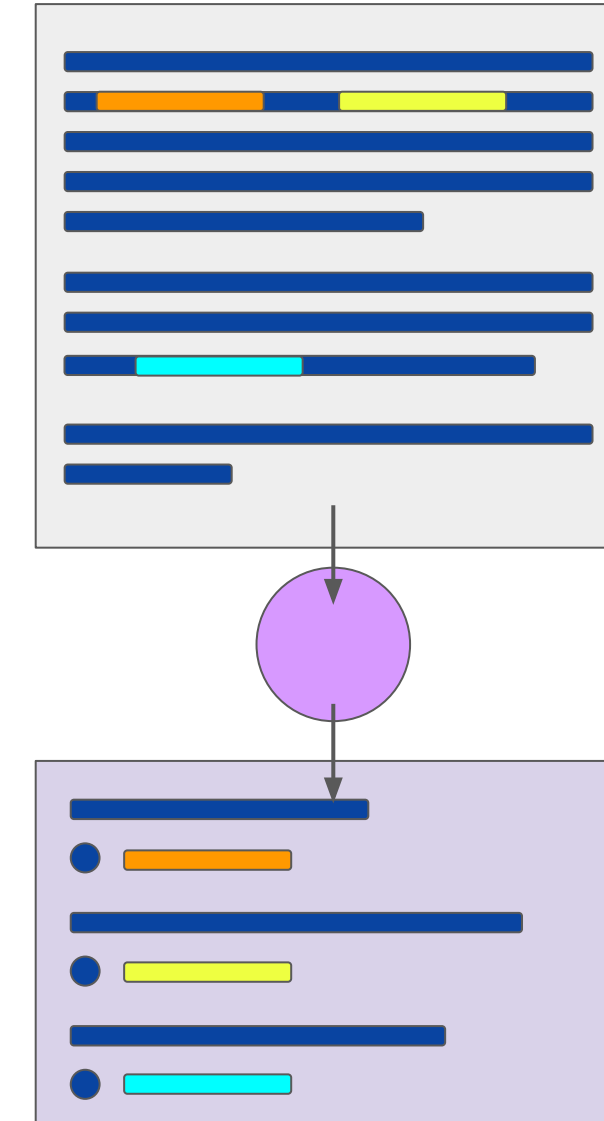
Summarization



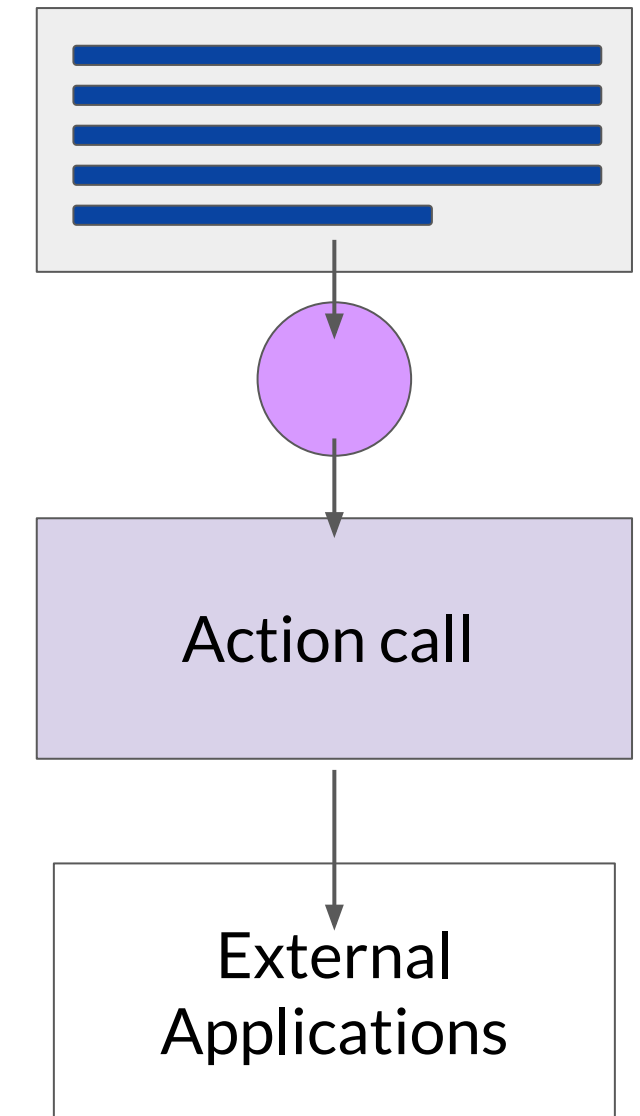
Translation



Information retrieval



Invoke APIs and actions



Generative AI project lifecycle

