LLM: Backbone of generative AI

GenAI 2025

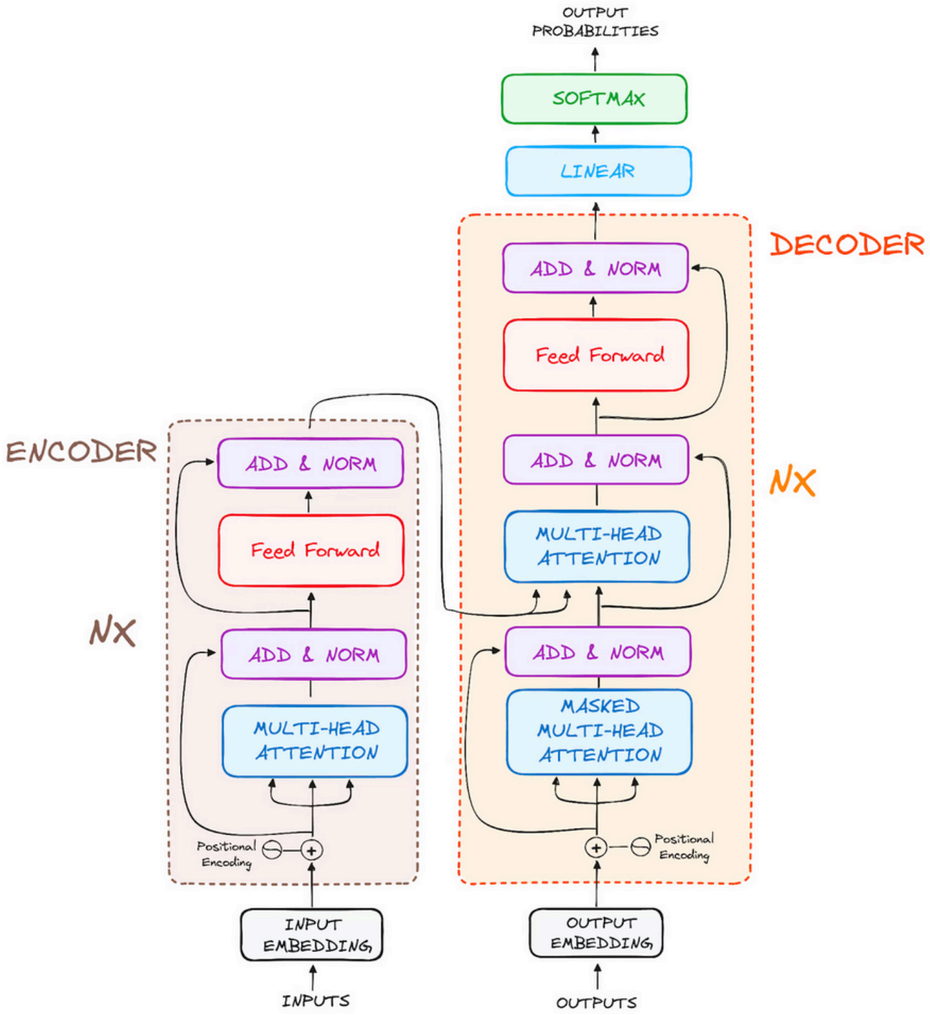
Linda Belkessa
Computer Science State Eng & MsC, PhD candidate @GRETTIA, CLEAR-DOC Fellow
linda.belkessa@univ-eiffel.fr

https://www.linkedin.com/in/linda-belkessa/

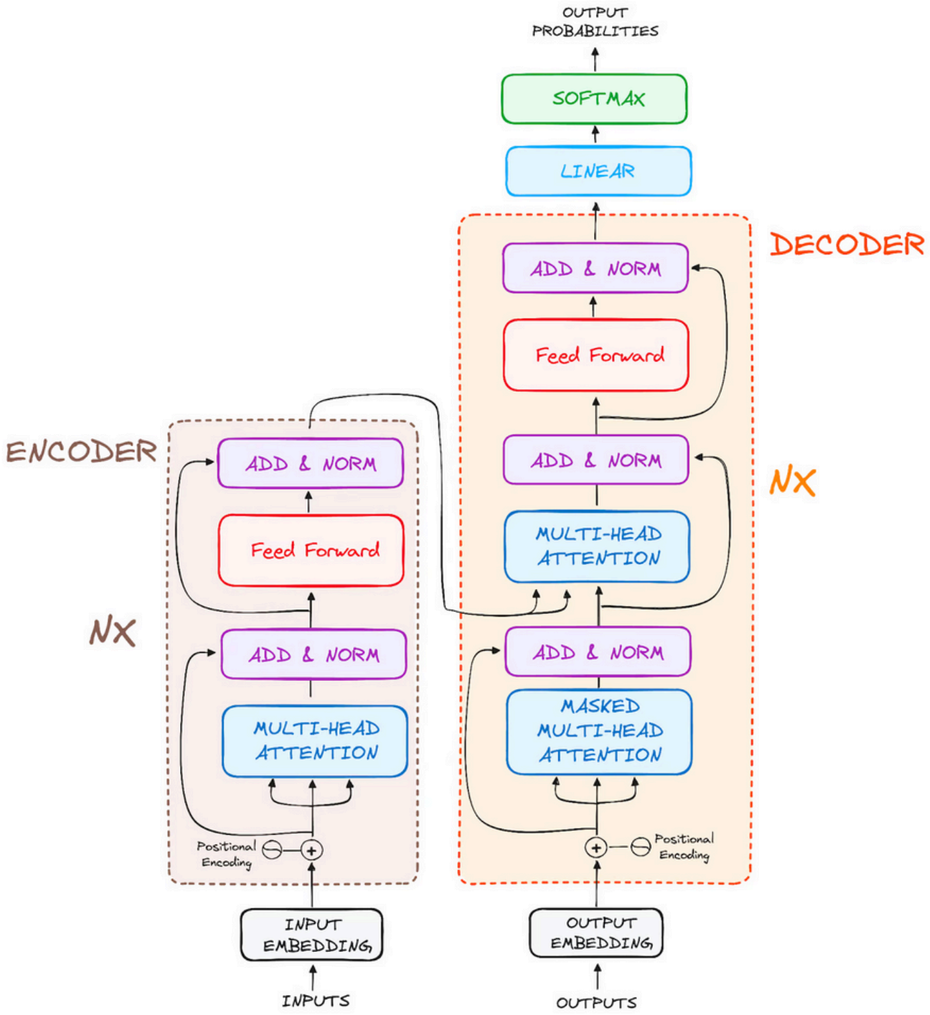
Lynda-Starkus


No, we’re not talking about optimus prime

THIS is a transformer !



THIS is a transformer !



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 lukaszkaiser@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 be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

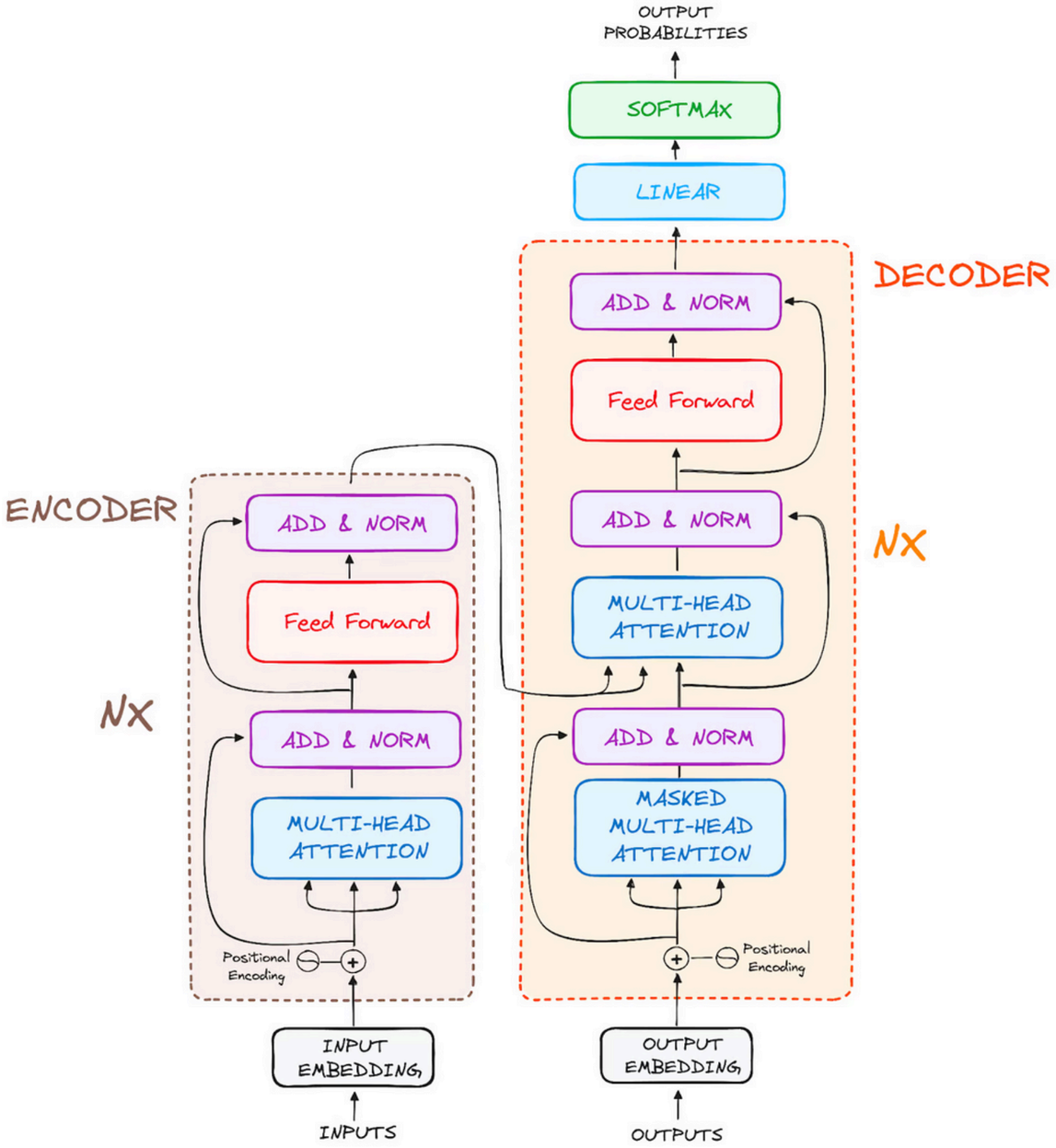
†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

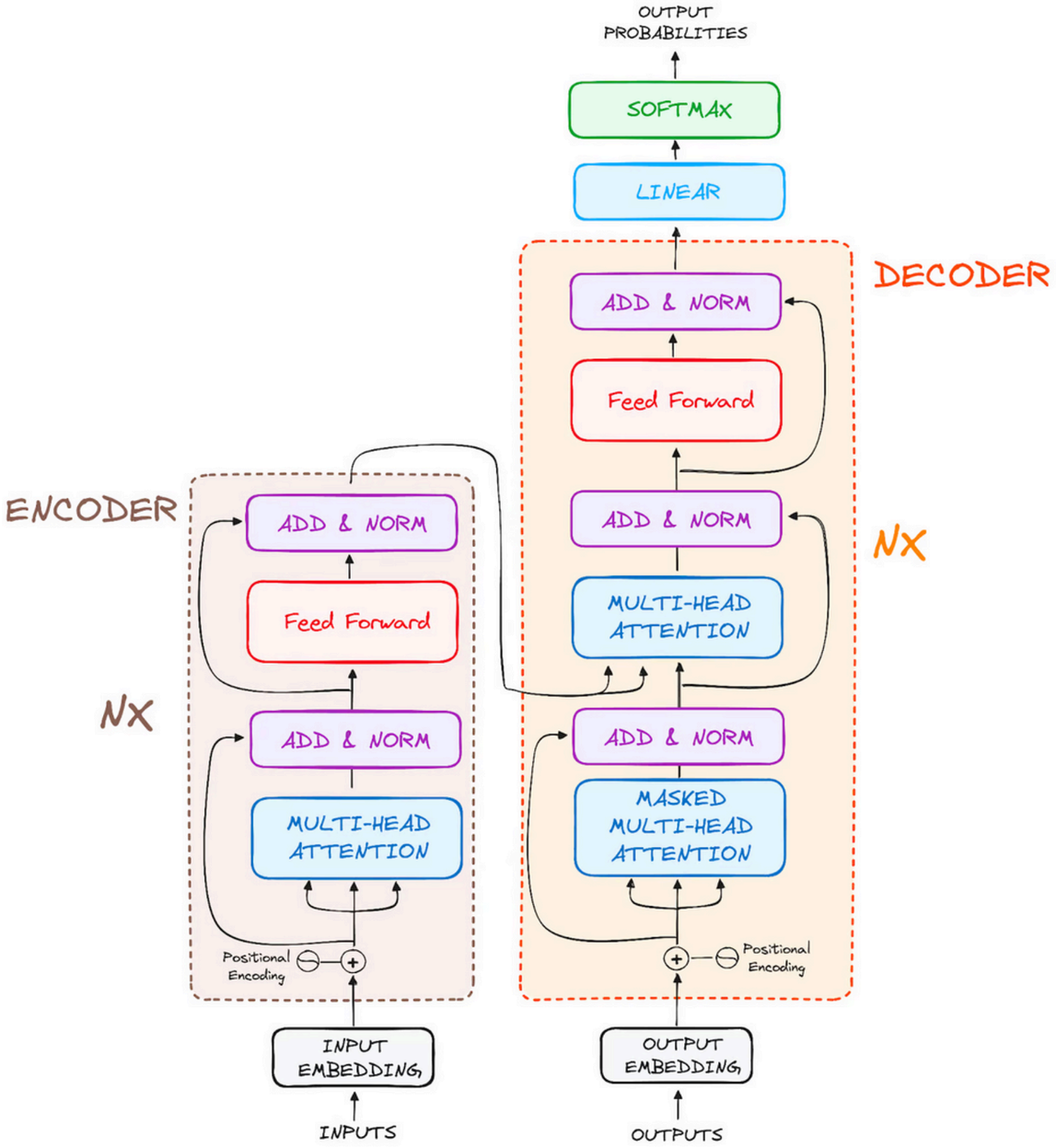
THIS is a transformer !



THIS is a transformer !

Transformers are current state-of-the-art **NLP** models with **encoder-decoder** architecture

They extract the **context** of **very large sequences** of text (or other inputs)

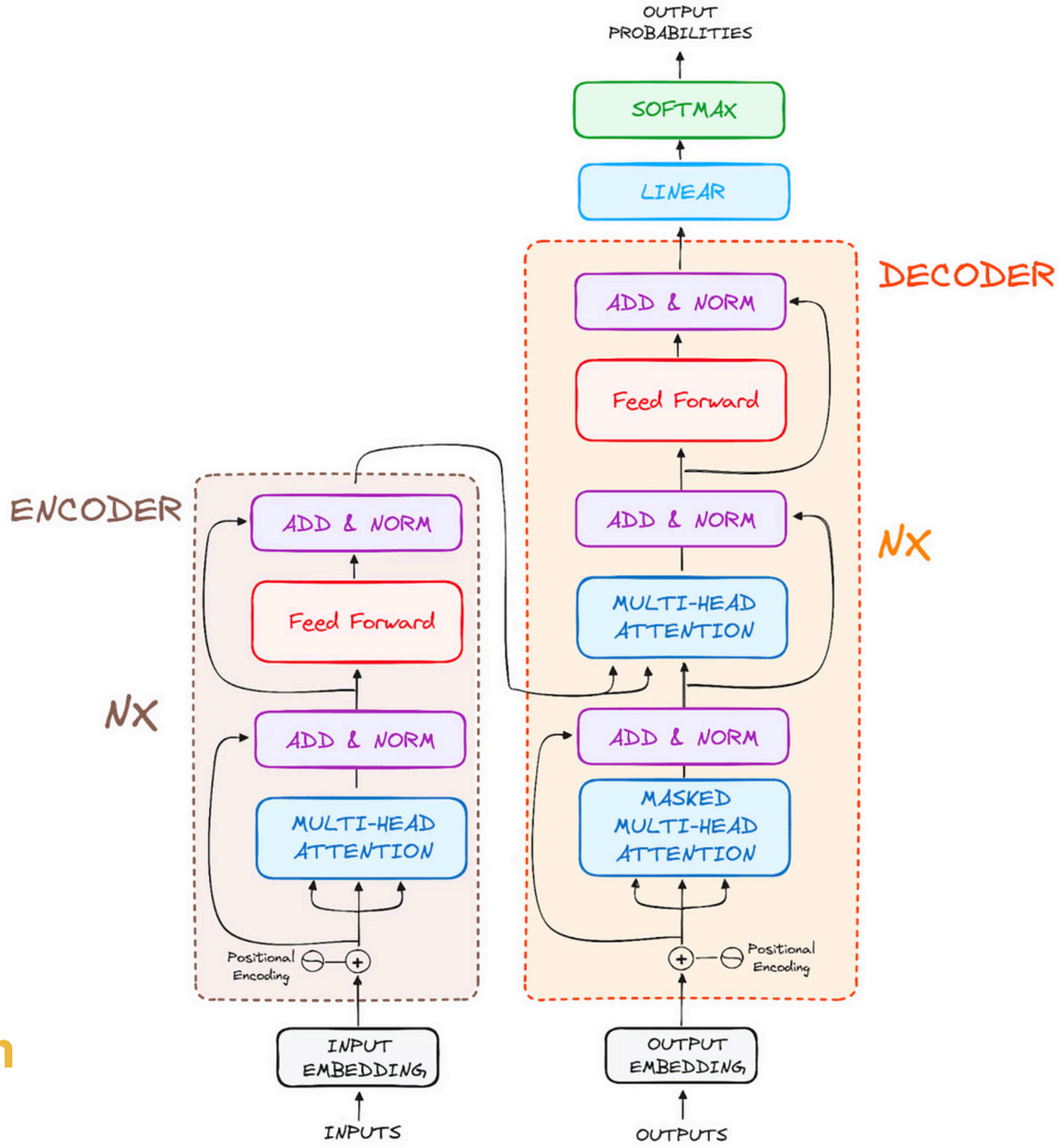


THIS is a transformer !

Transformers are current state-of-the-art **NLP** models with **encoder-decoder** architecture

They extract the **context** of **very large sequences** of text (or other inputs)

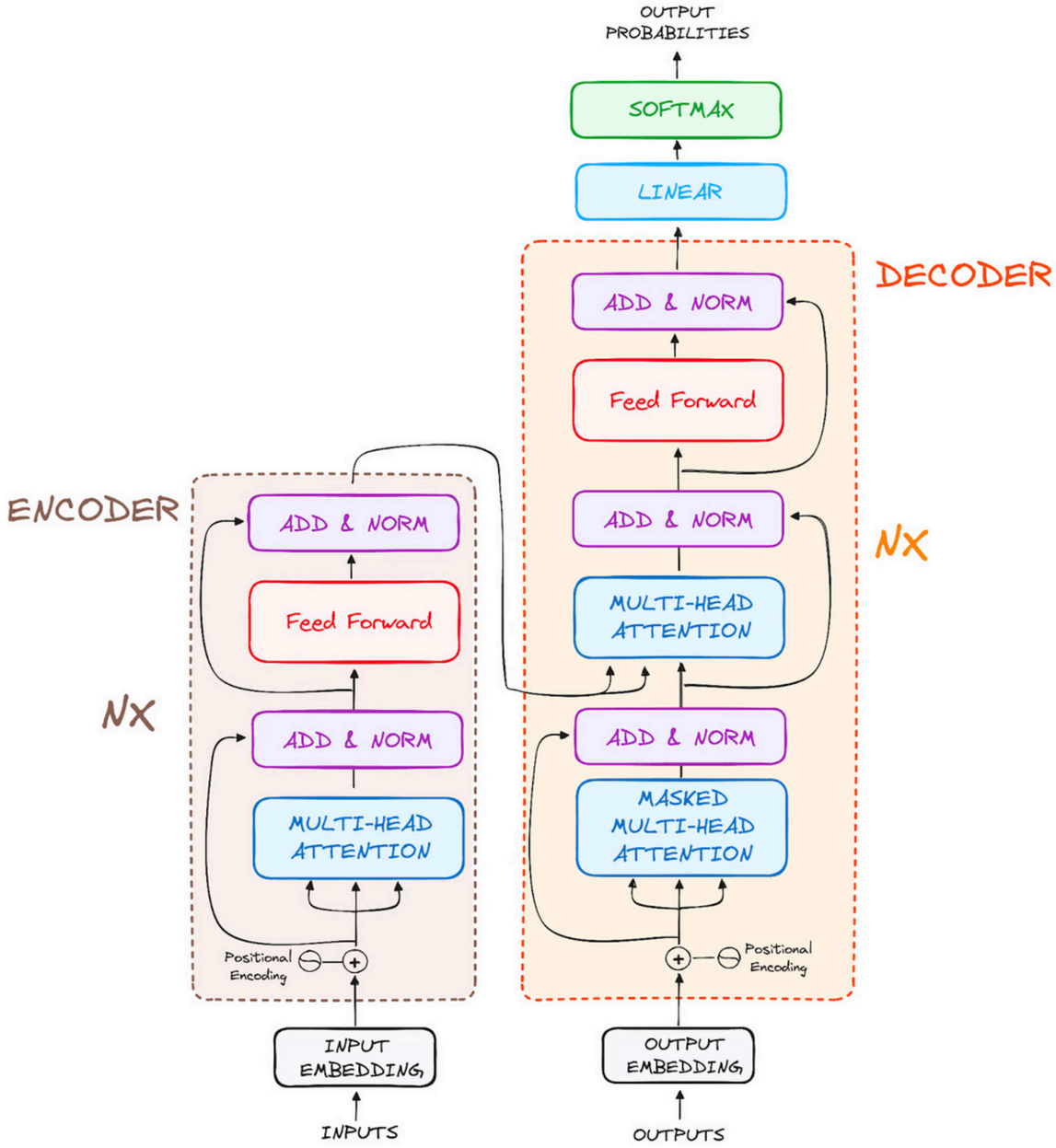
Unlike recurrent networks *(process the input sequentially, one element at a time)*, they don't use recurrence but rely on **attention mechanism**



Architecture overview of transformers

Transformers were originally made for transduction (neural translation) of inputs to a different type of outputs

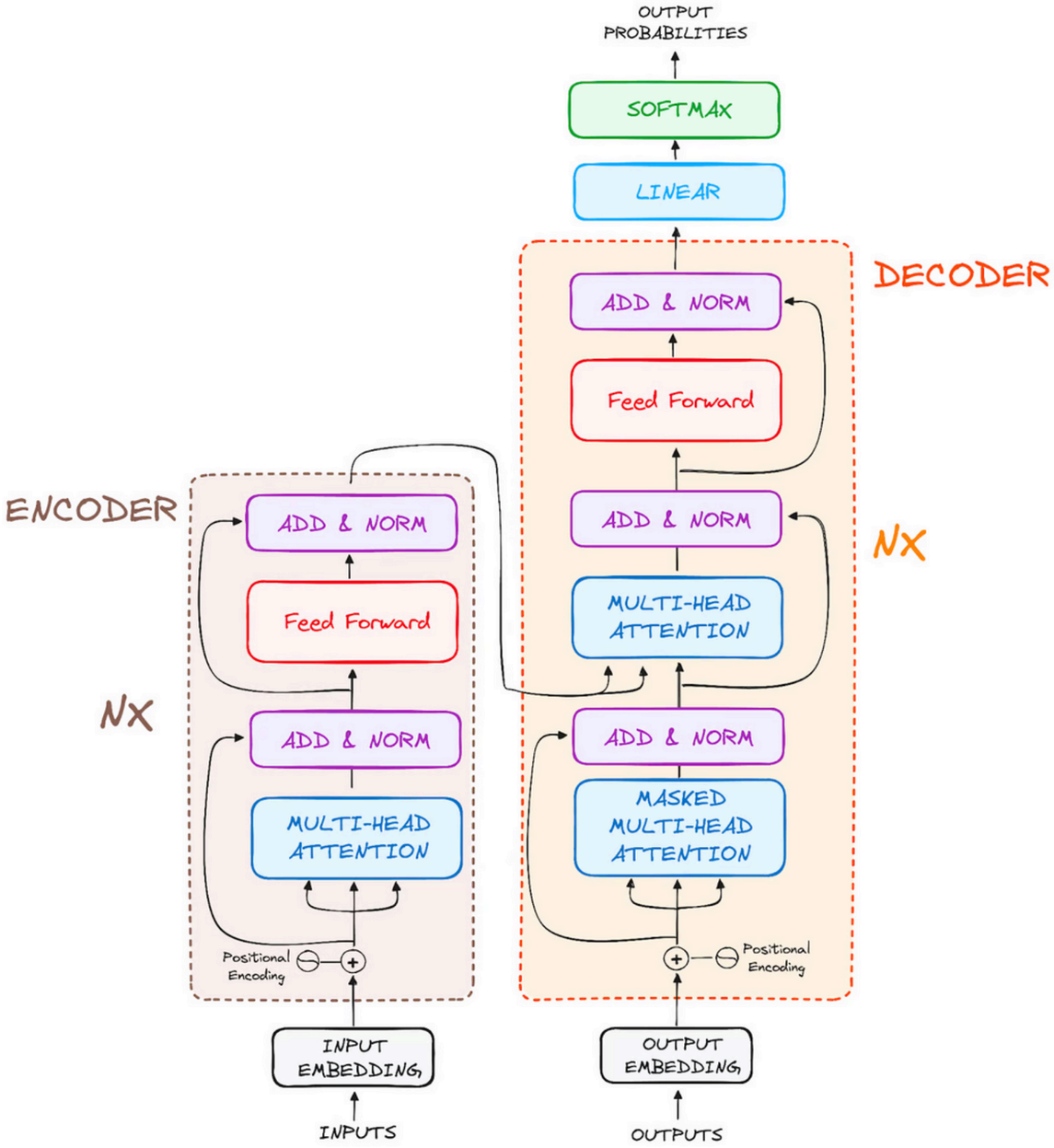
For example, using english to **generate** spanish text



Architecture overview of transformers

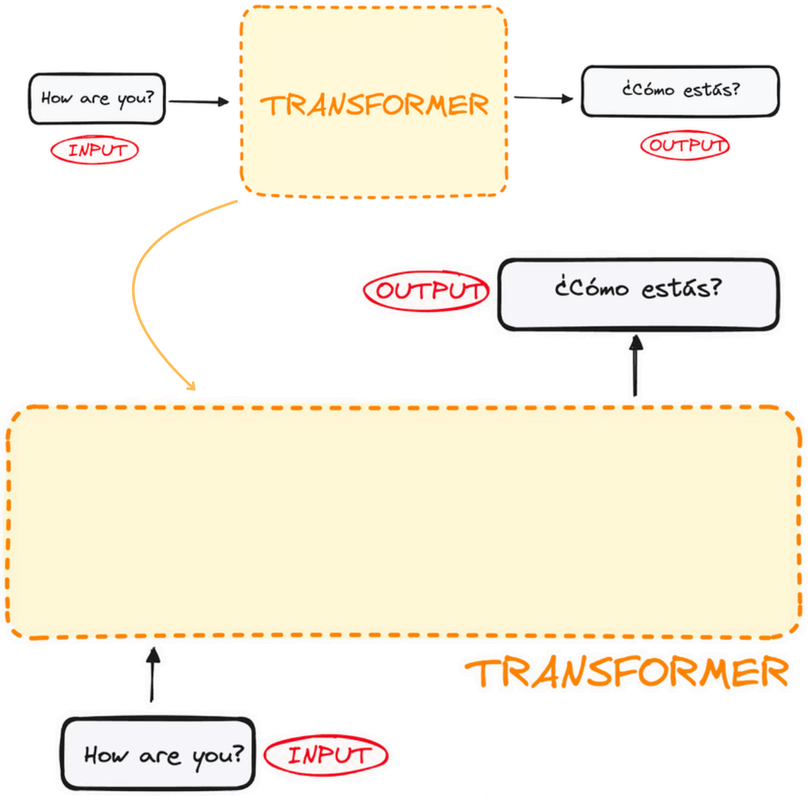
Transformers were originally made for transduction (neural translation) of inputs to a different type of outputs

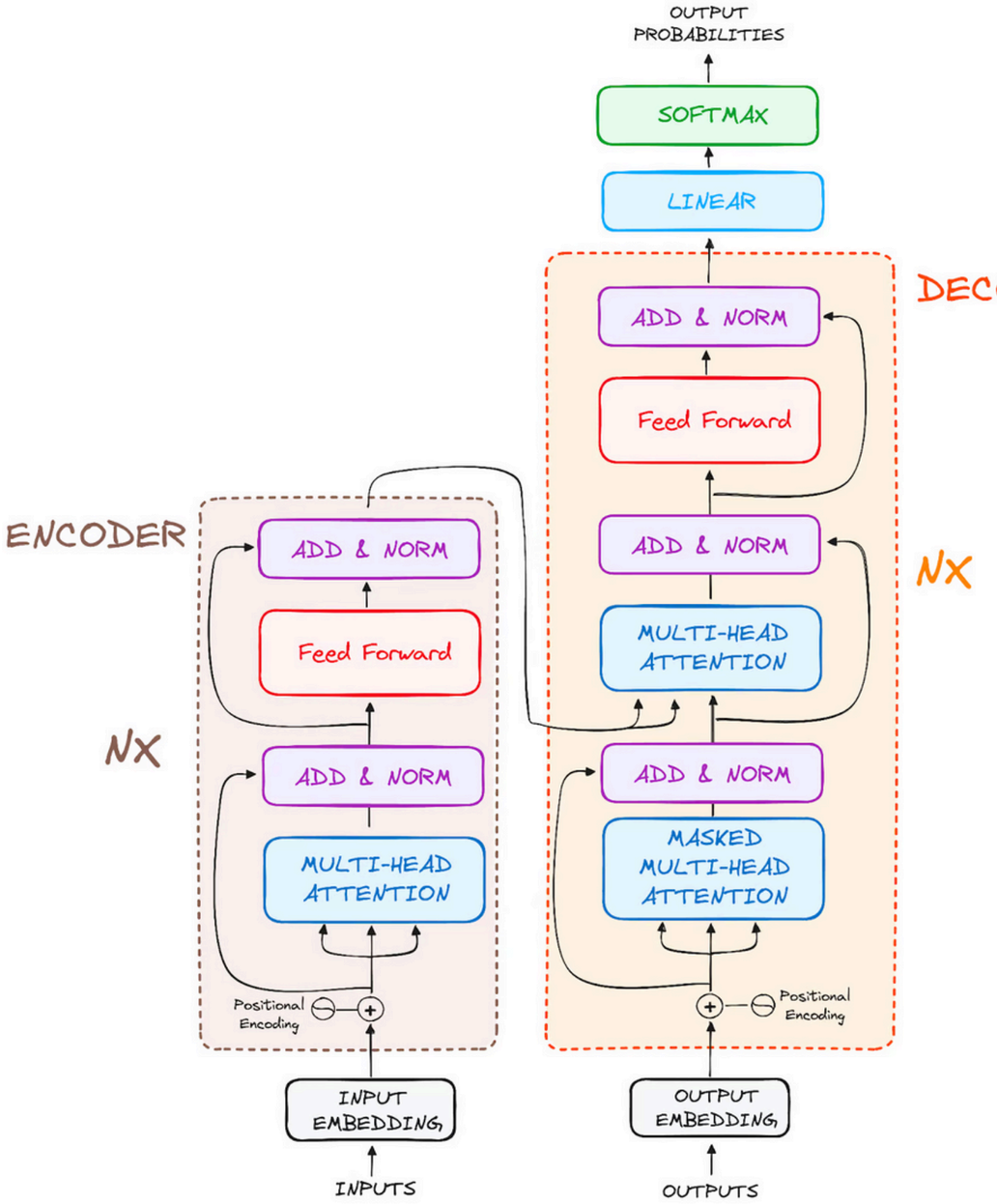
For example, using english to **generate** spanish text



Architecture overview of transformers

The model is composed of two main components

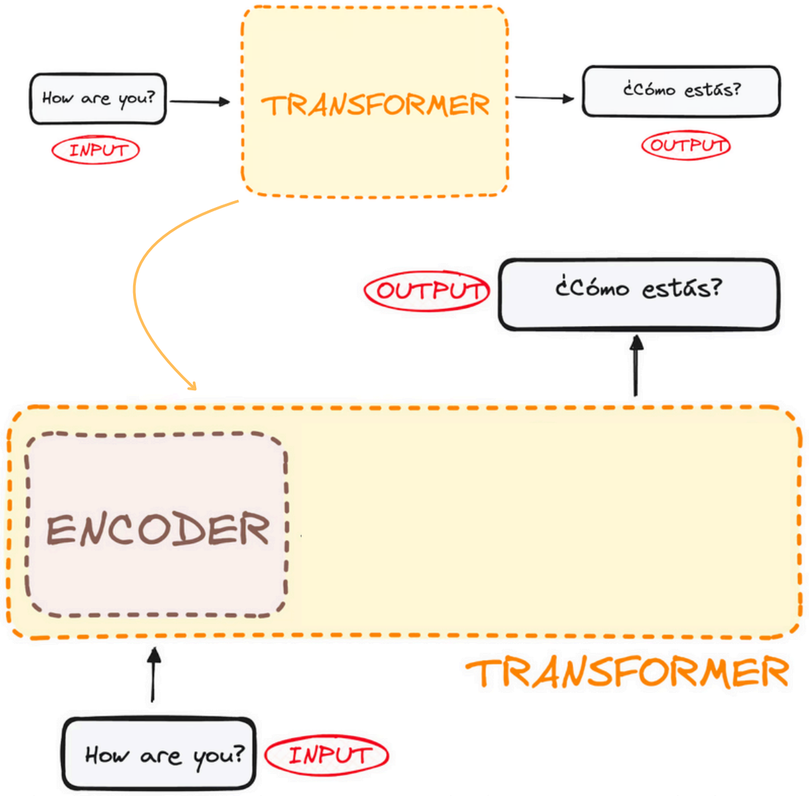


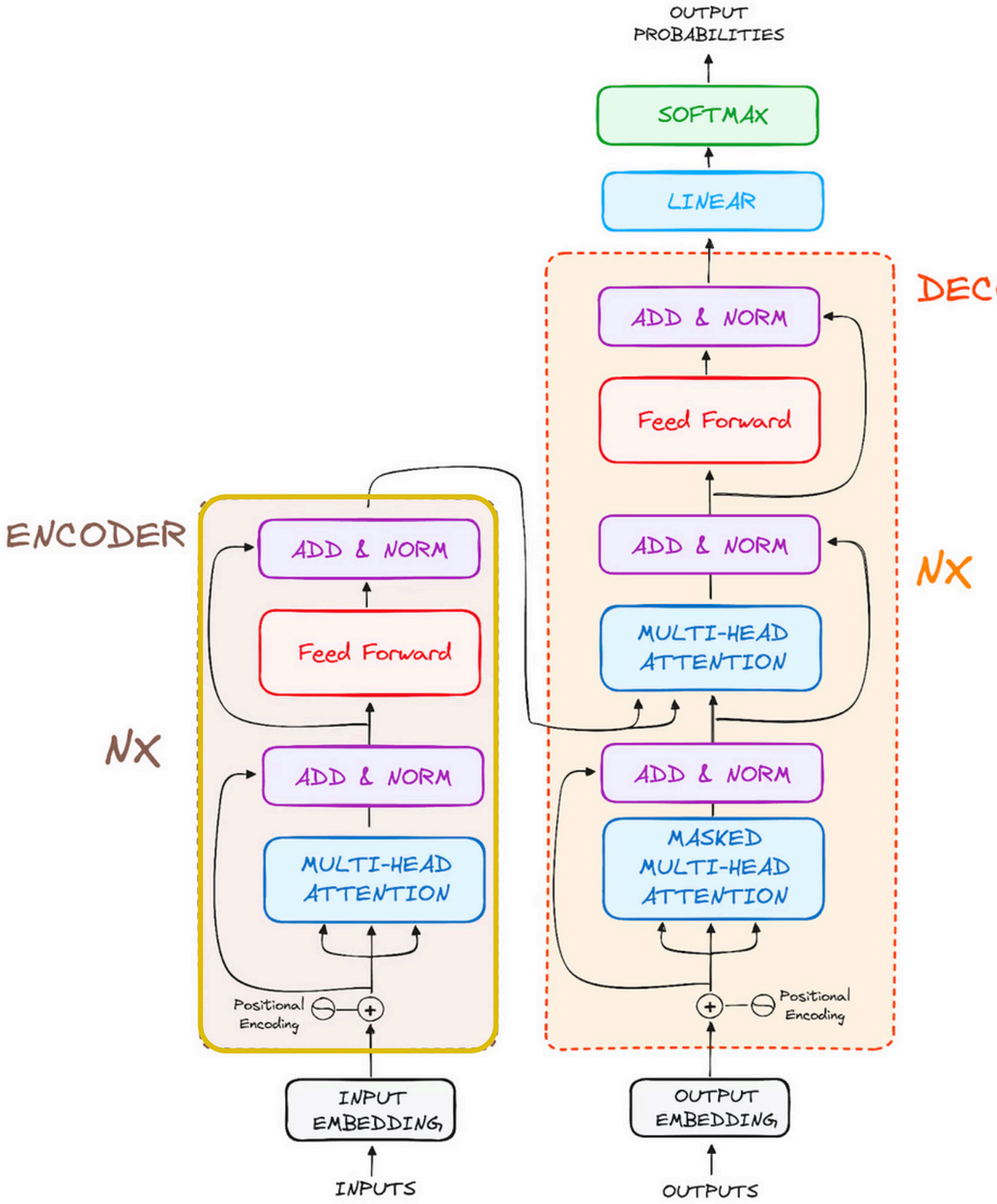


Architecture overview of transformers

The model is composed of two main components

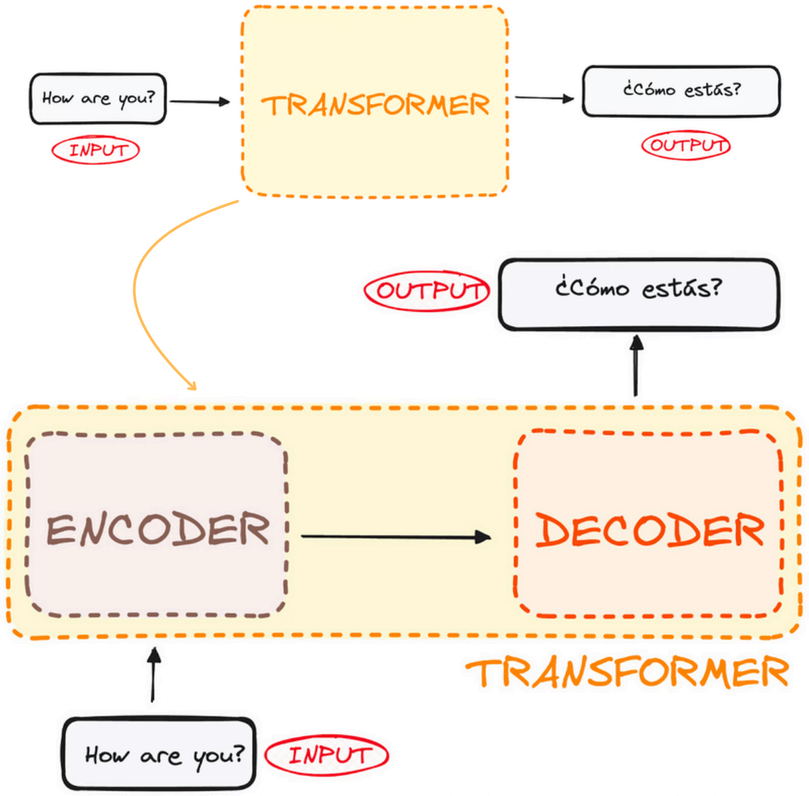
Encoder: takes a matrix representation of inputs

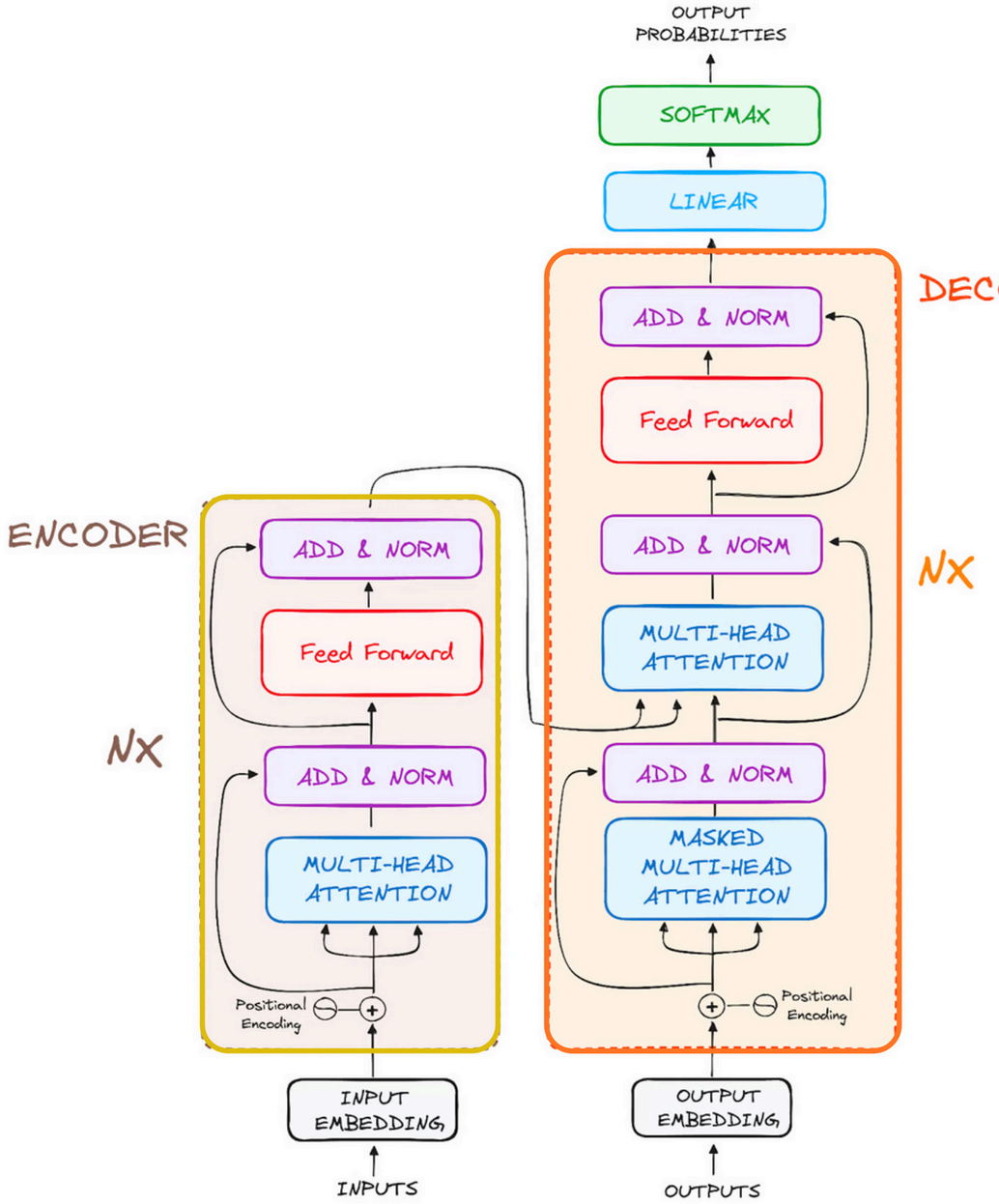




Architecture overview of transformers

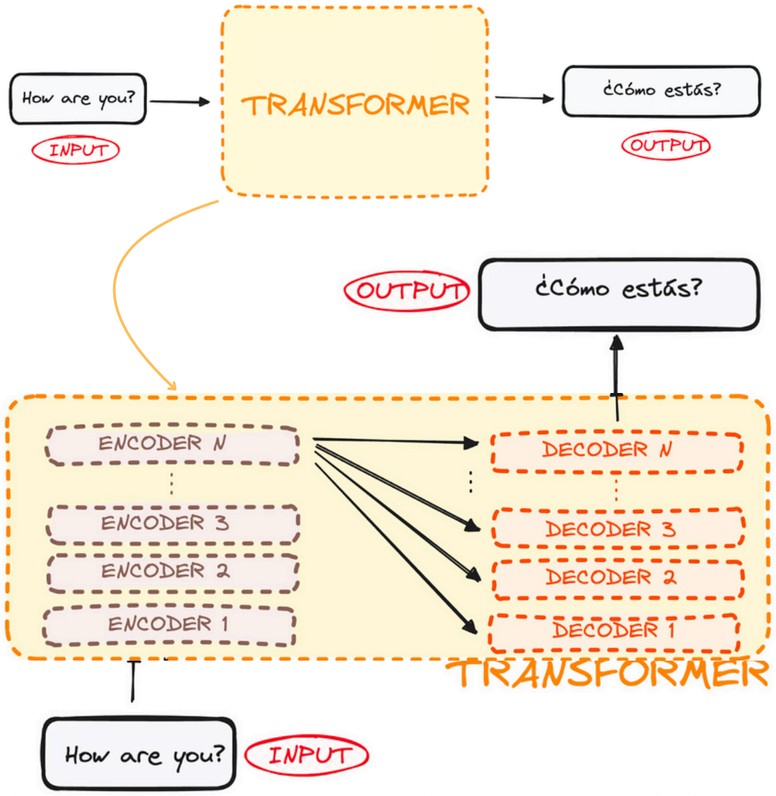
The model is composed of two main components
Encoder: takes a matrix representation of inputs
Decoder: iteratively generates an output given encoded representations

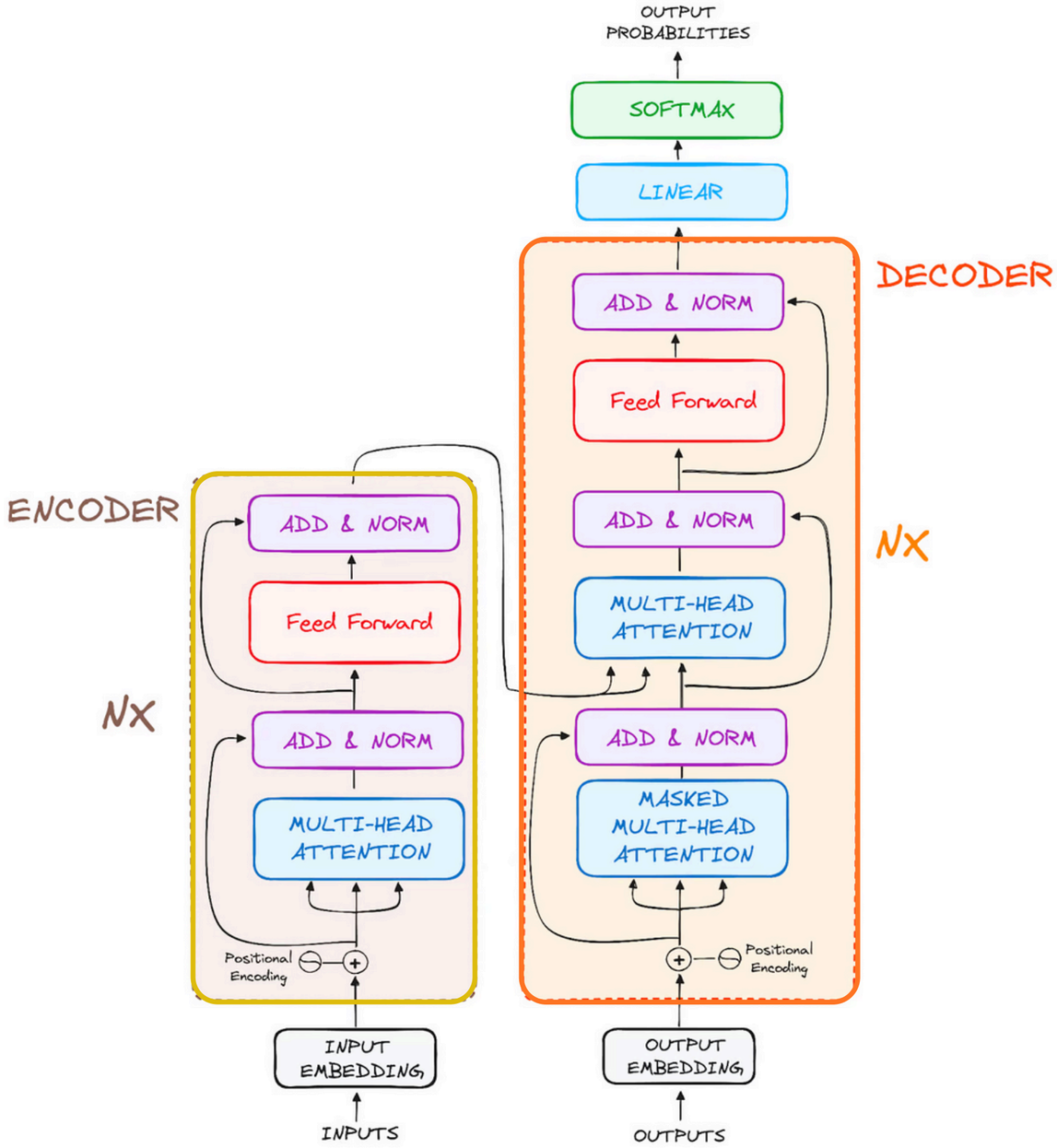




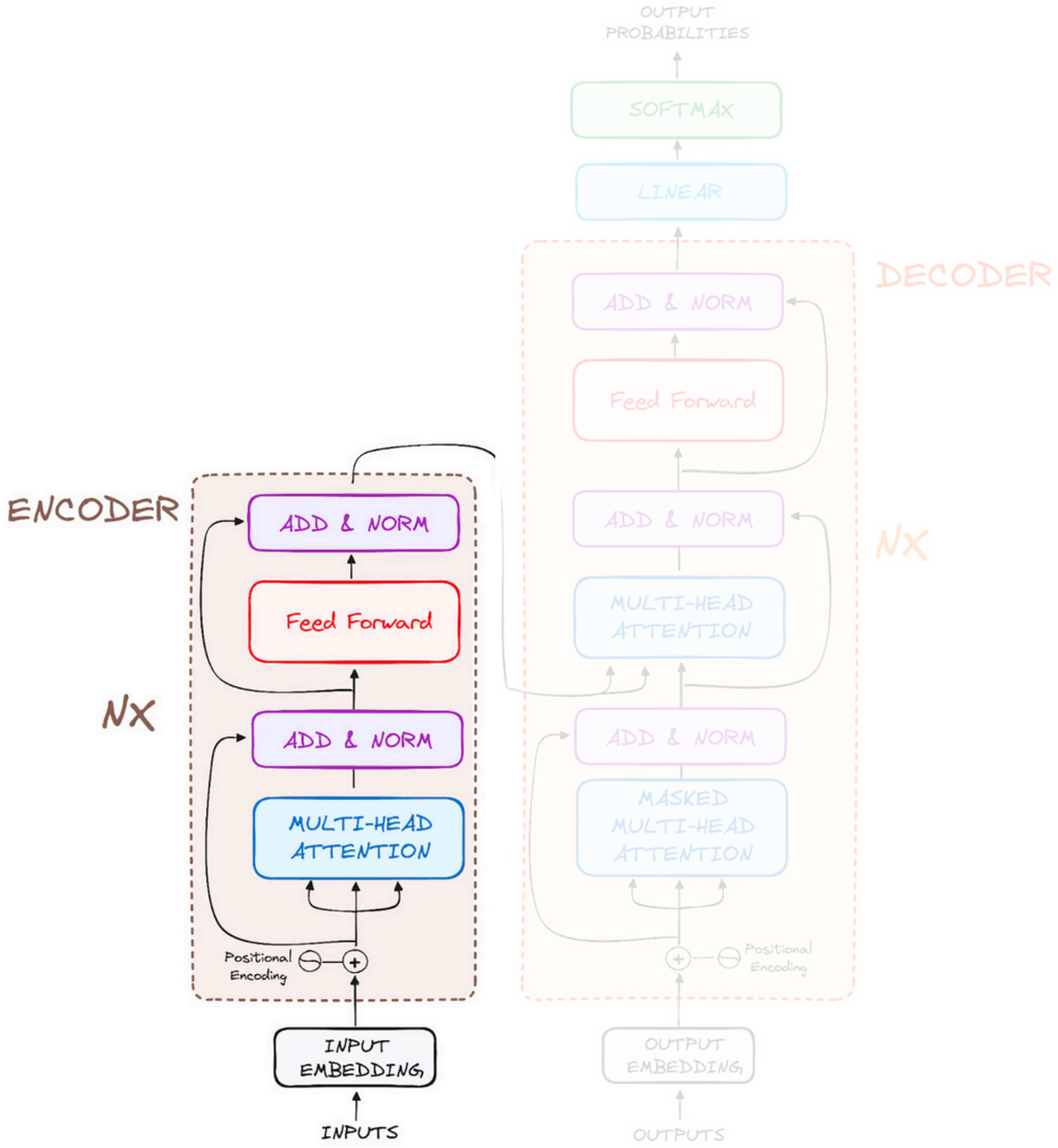
Architecture overview of transformers

Actually each component is a **stack** of N layers (encoders-decoders)

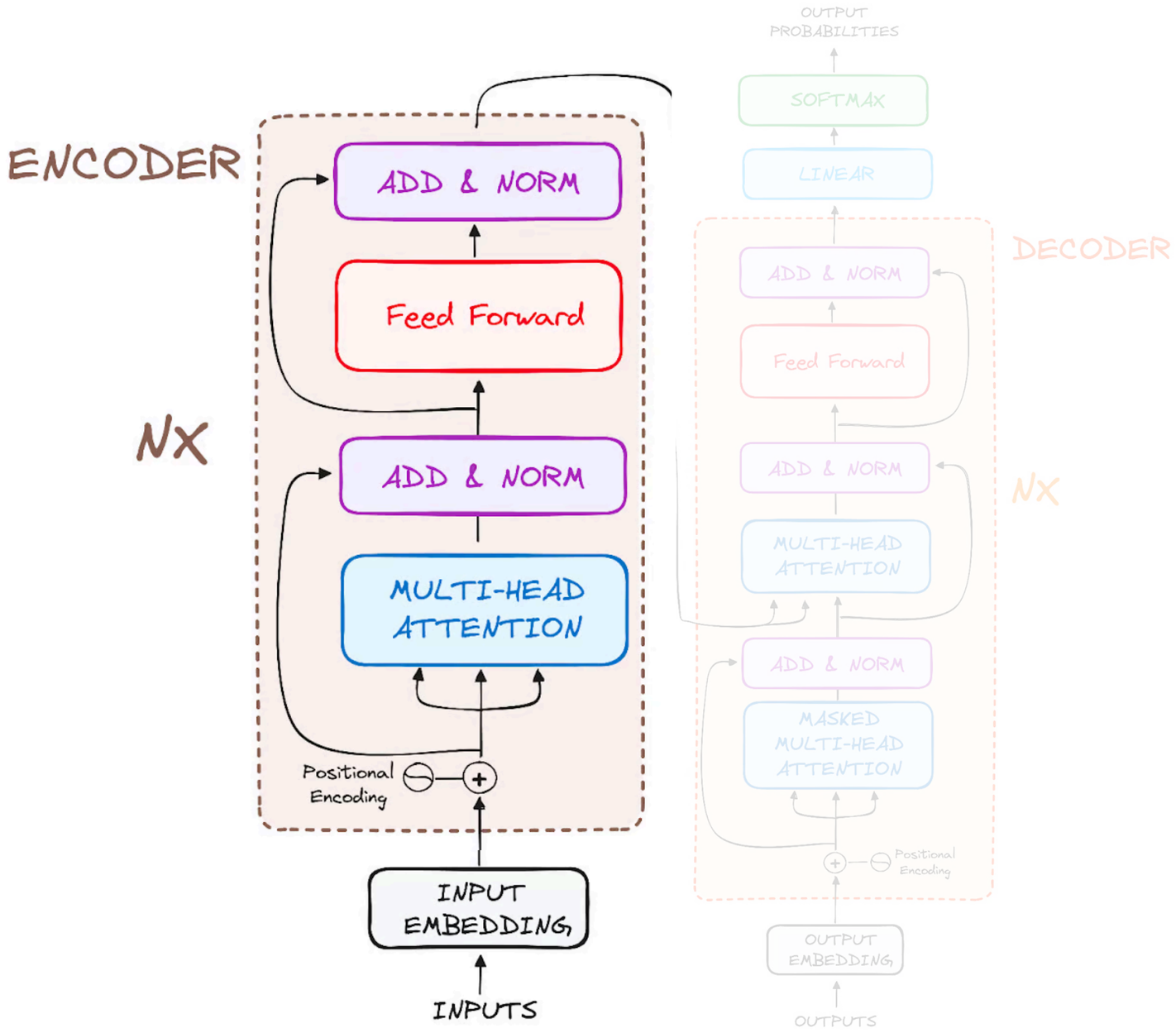




Encoder architecture



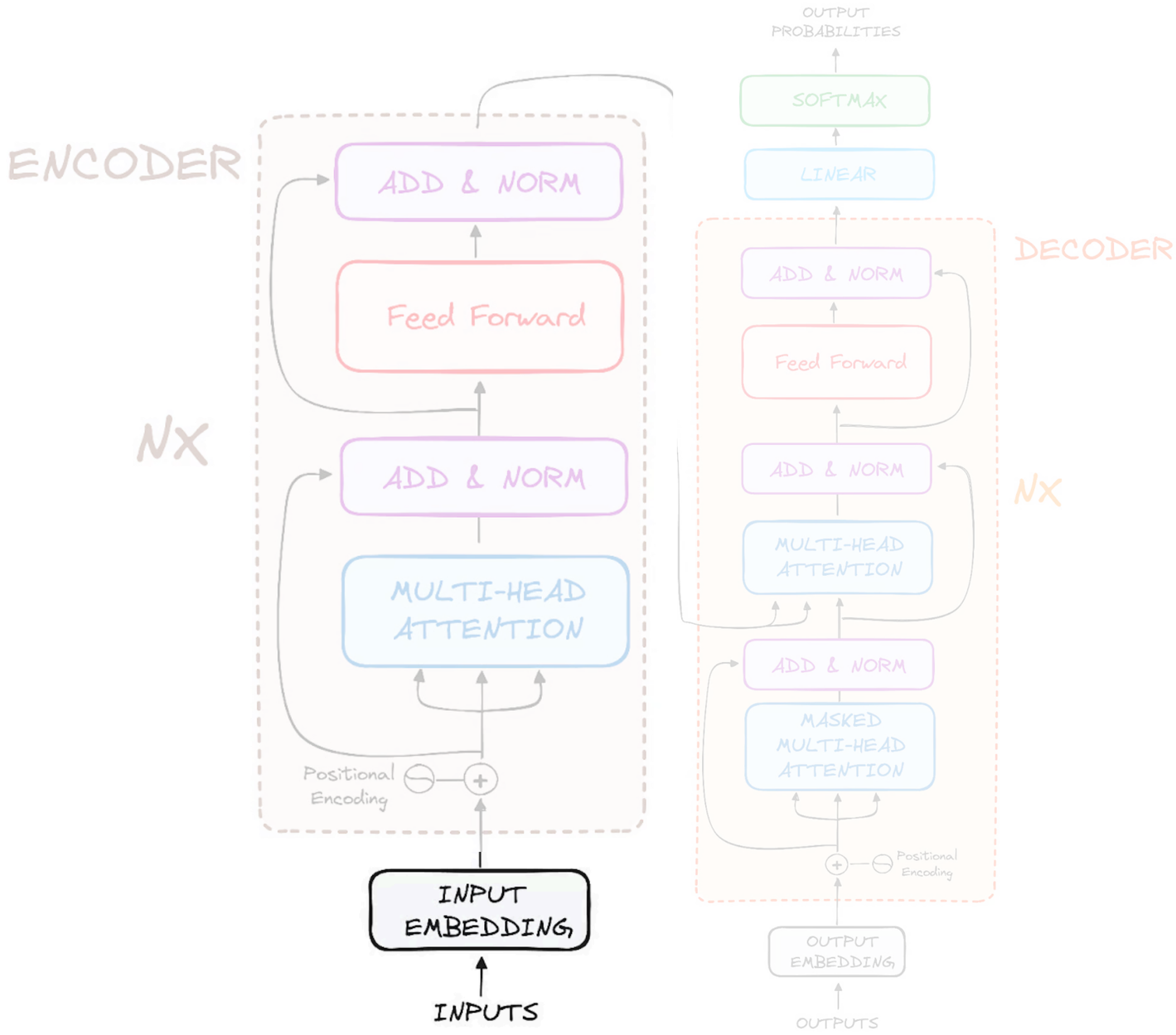
Encoder architecture



Encoder architecture

Input Embeddings

- Happens only at this stage
- Convert words **(tokens)** into vectors
- Capture the semantics of tokens and represent them as **numerical vectors**

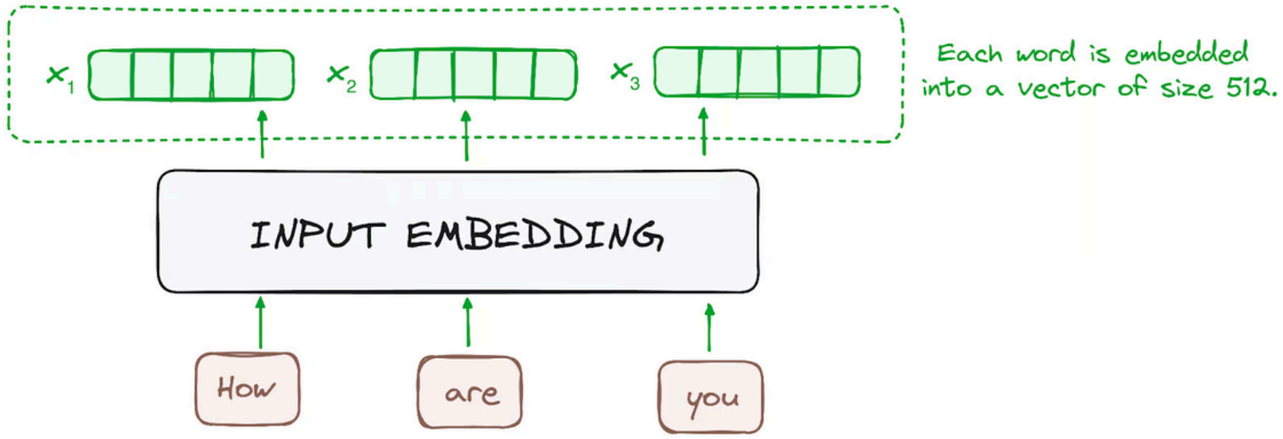


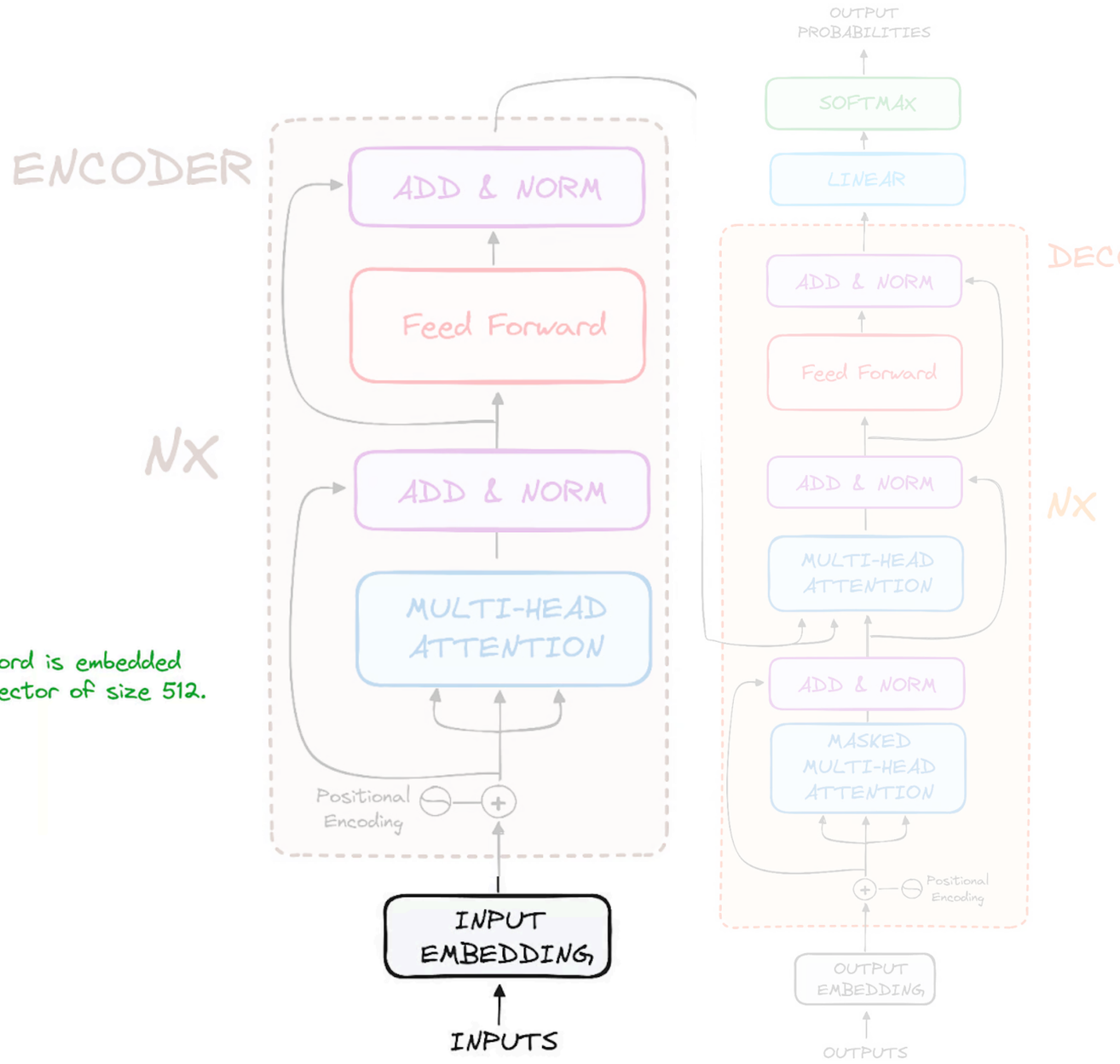
Encoder architecture

Input Embeddings

- Happens only at this stage
- Convert words **(tokens)** into vectors
- Capture the semantics of tokens and represent them as **numerical vectors**

Gives to the next encoder a **list of vectors (of fixed size 512)**

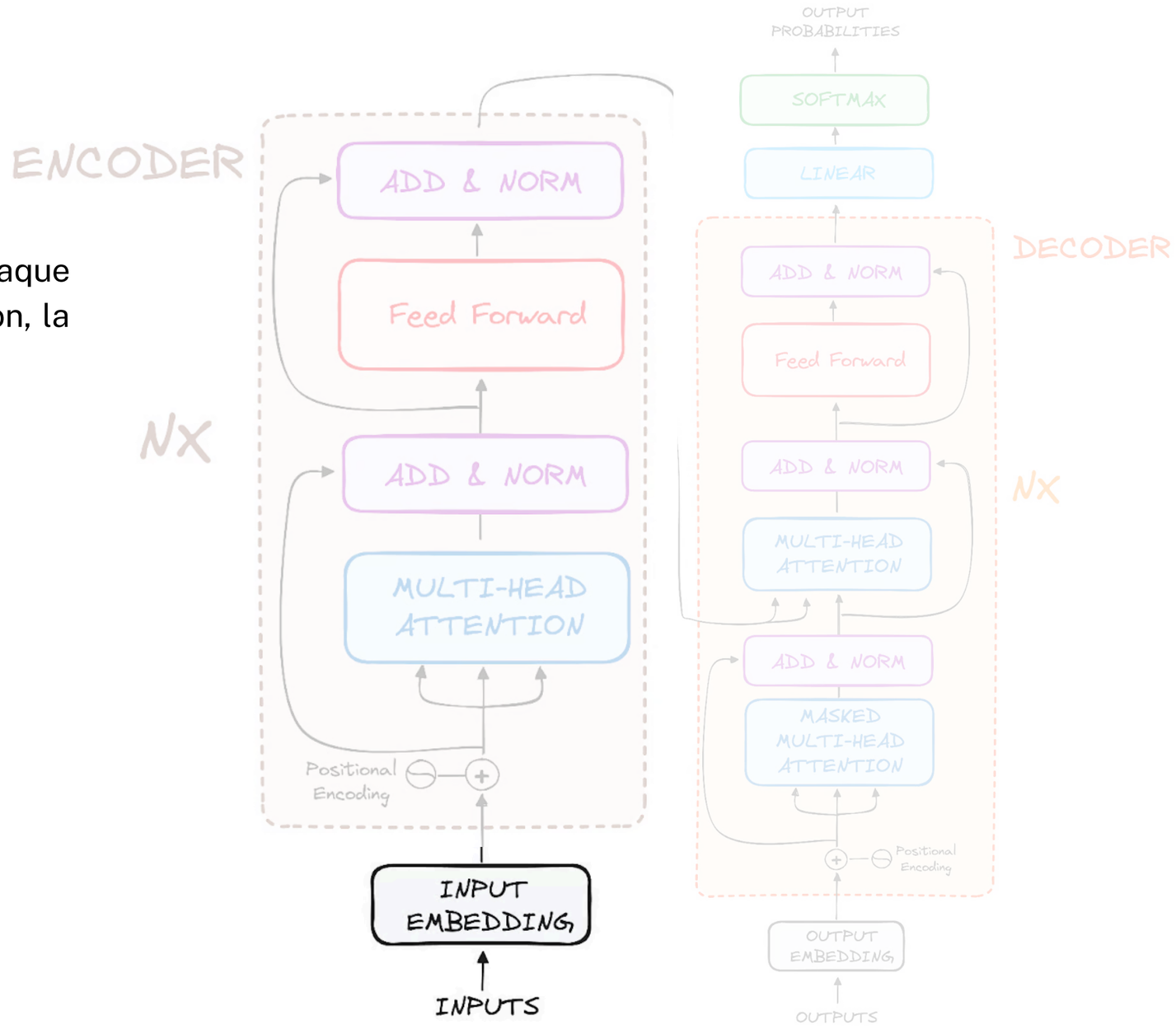




Encoder architecture

Input Embeddings

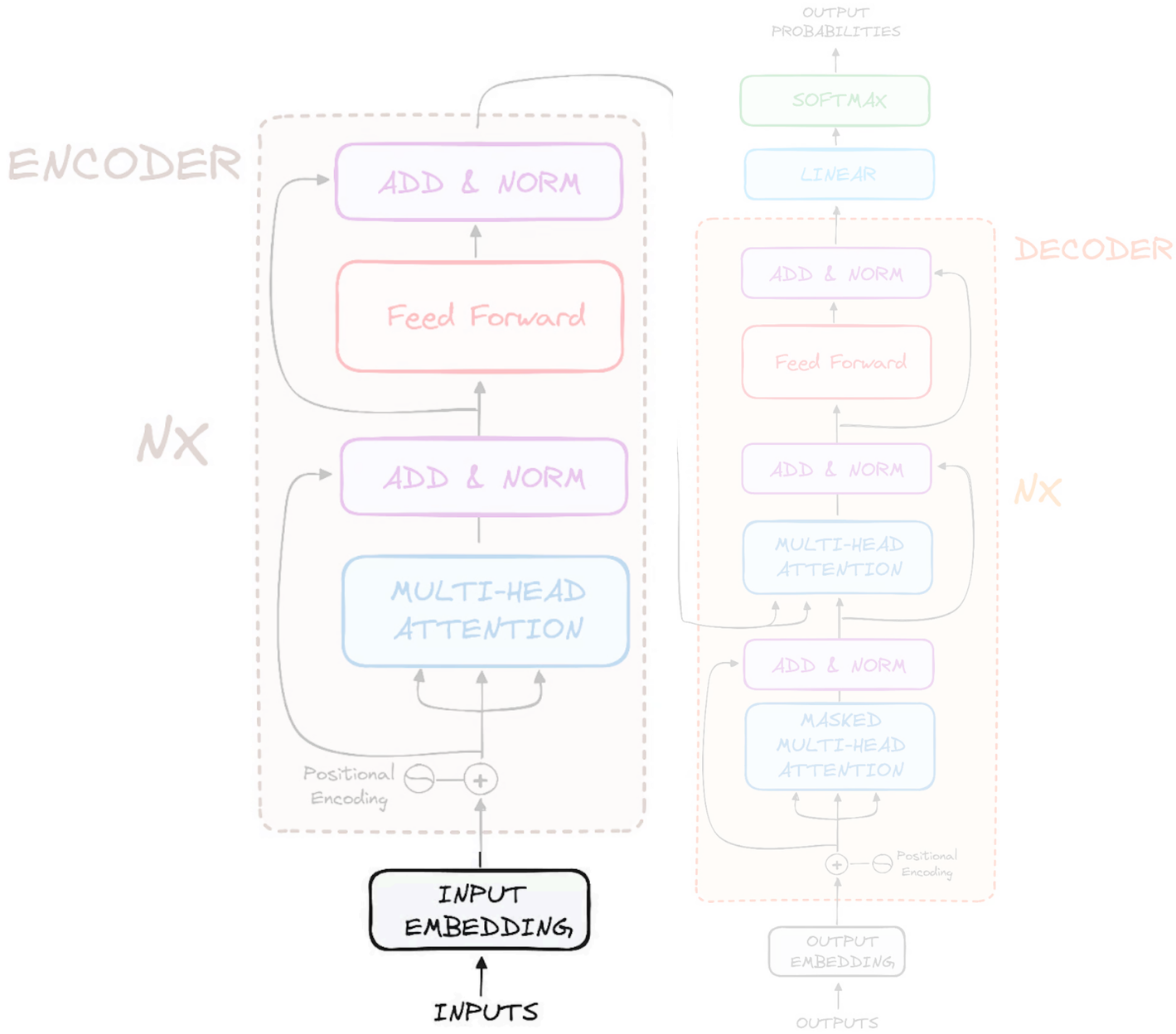
“La perfection du caractère consiste à passer chaque journée comme si c'était la dernière, à éviter l'agitation, la torpeur et l'hypocrisie.” - Marc Aurèle



Encoder architecture

Input Embeddings

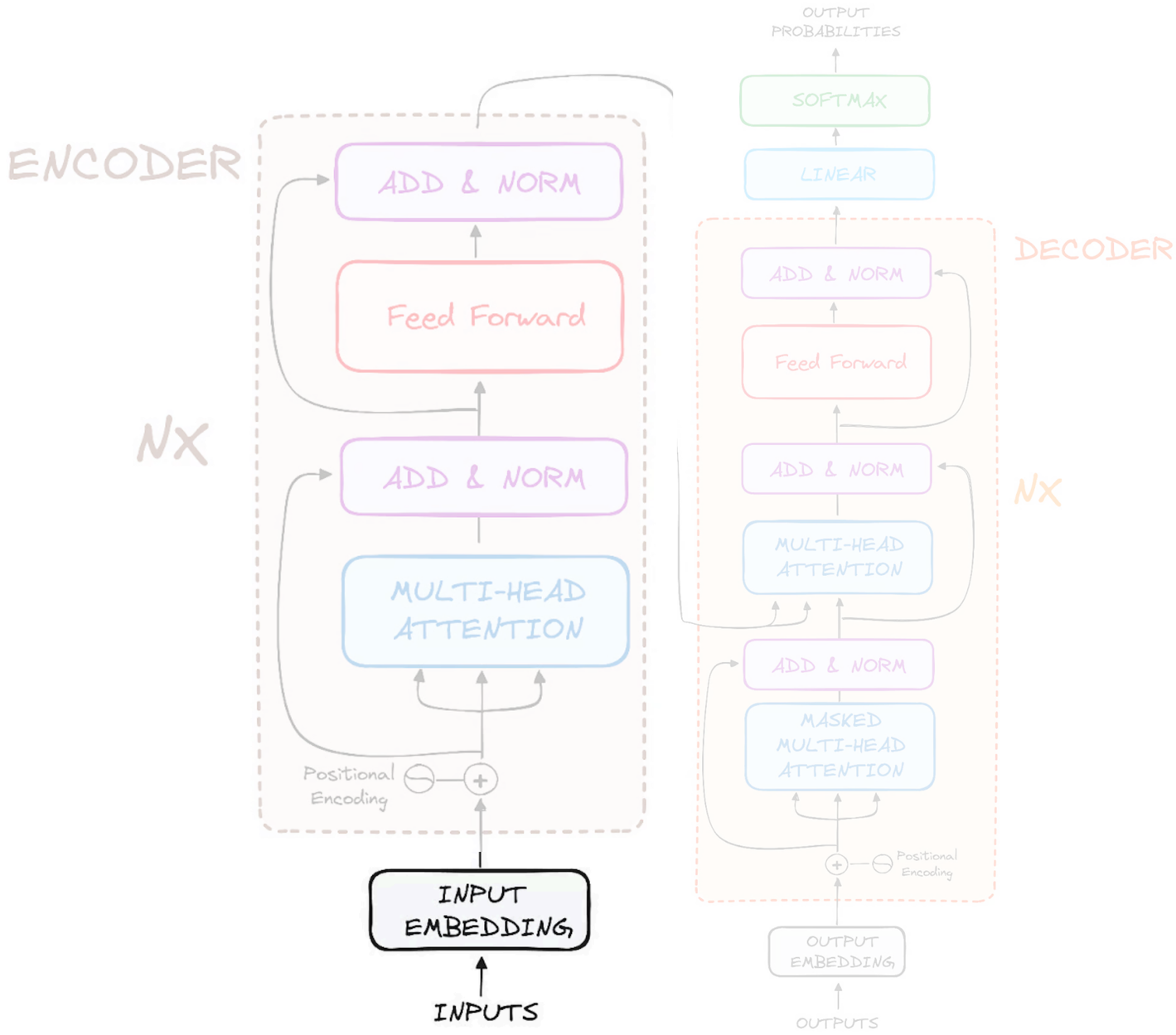
La	0
perfection	1
du	2
caractère	3
chaque	18
journée	19
dernière	30
éviter	35
l'agitation	36
l'hypocrisie	40



Encoder architecture

Input Embeddings

La	0
perfection	1
du	2
caractère	3
chaque	18
journée	19
dernière	30
éviter	35
l'agitation	36
l'hypocrisie	40

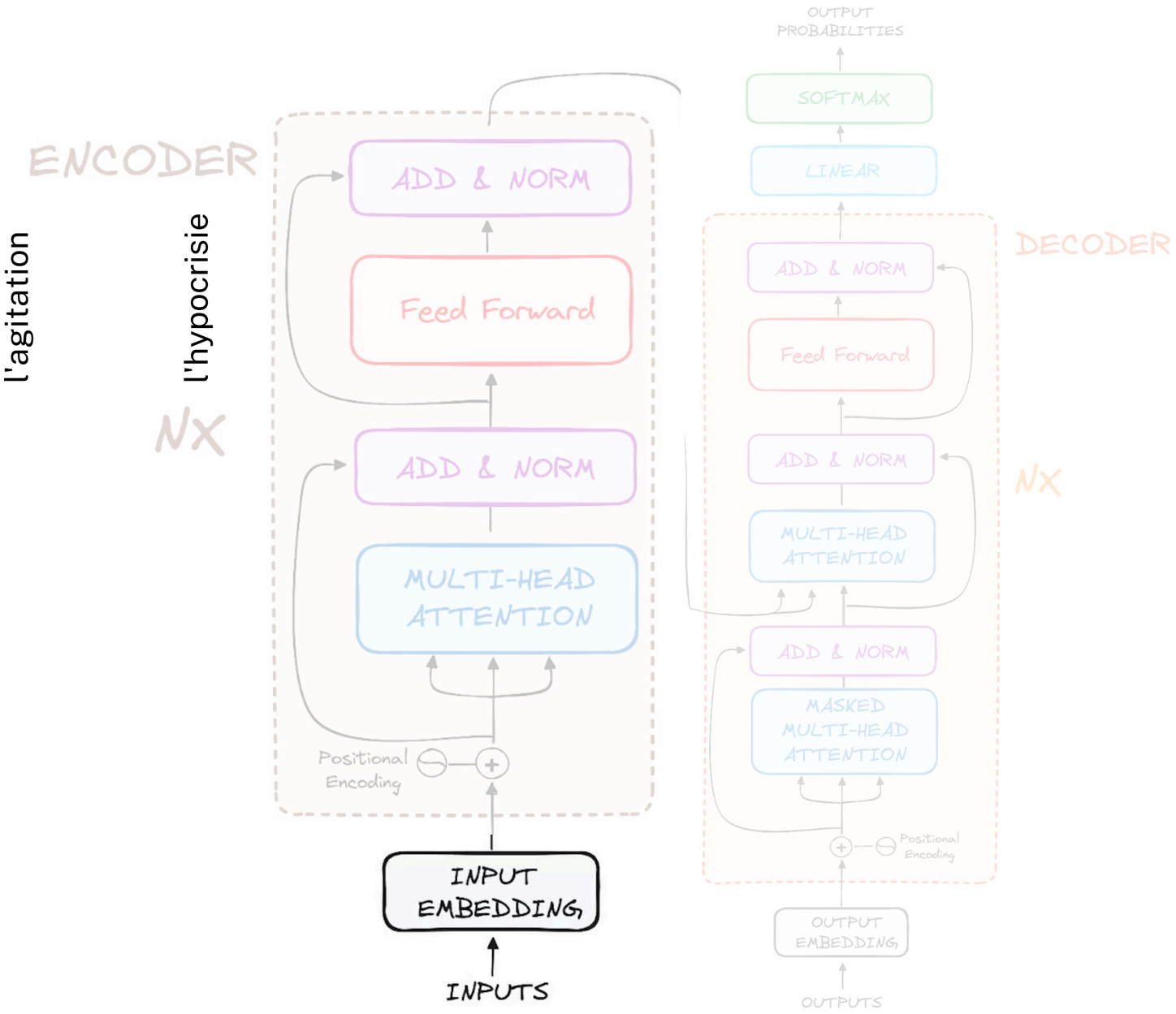


Encoder architecture

Input Embeddings

La perfection du caractère chaque journée dernière éviter l'agitation l'hypocrisie

0.87
-0.64
0.81
-0.35
⋮
-0.22
0.54



Encoder architecture

Input Embeddings

La perfection du caractère chaque journée dernière éviter l'agitation l'hypocrisie

0.87 — Linguistic feature 1

-0.64 — Linguistic feature 2

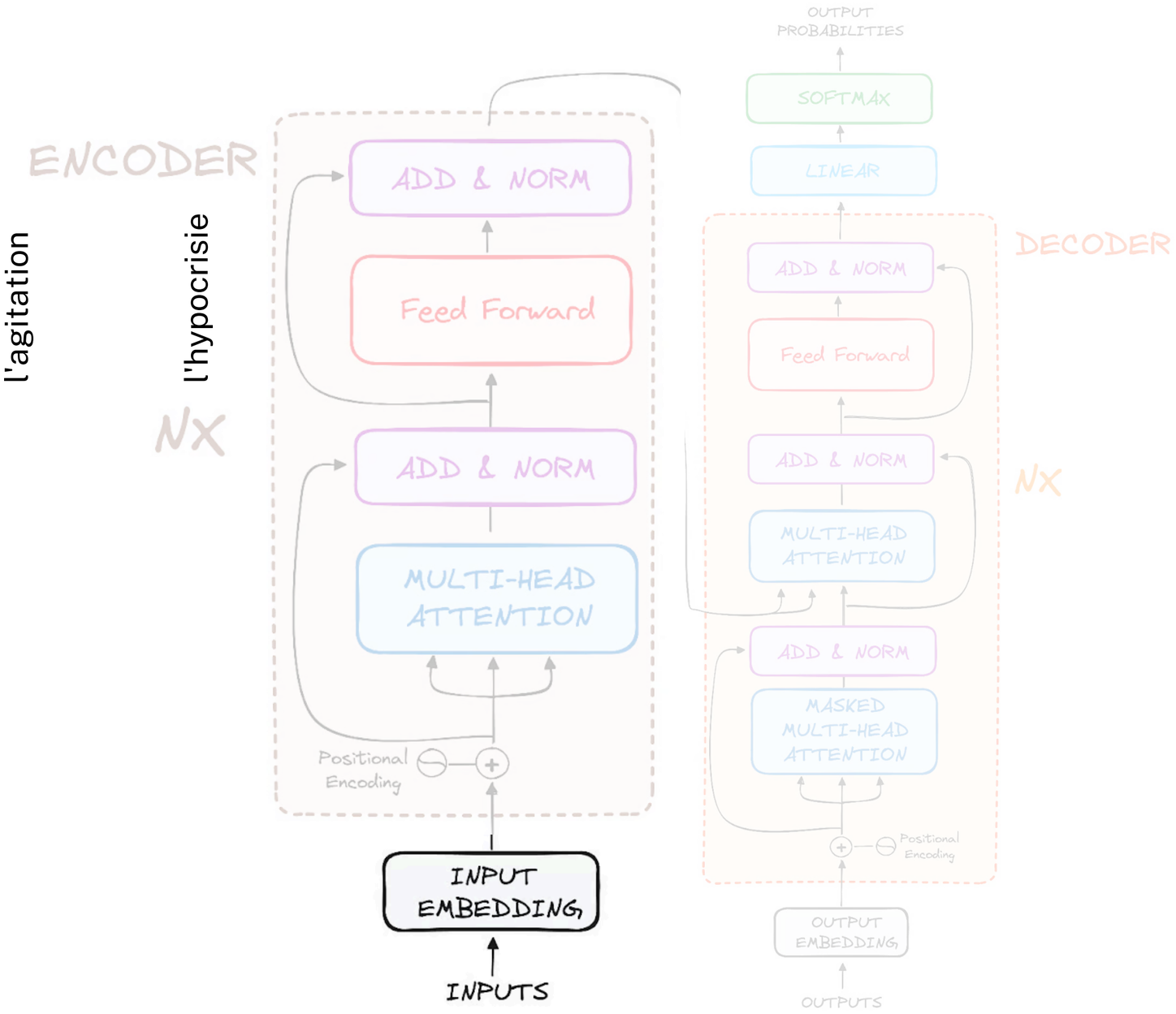
0.81

-0.35

⋮

-0.22

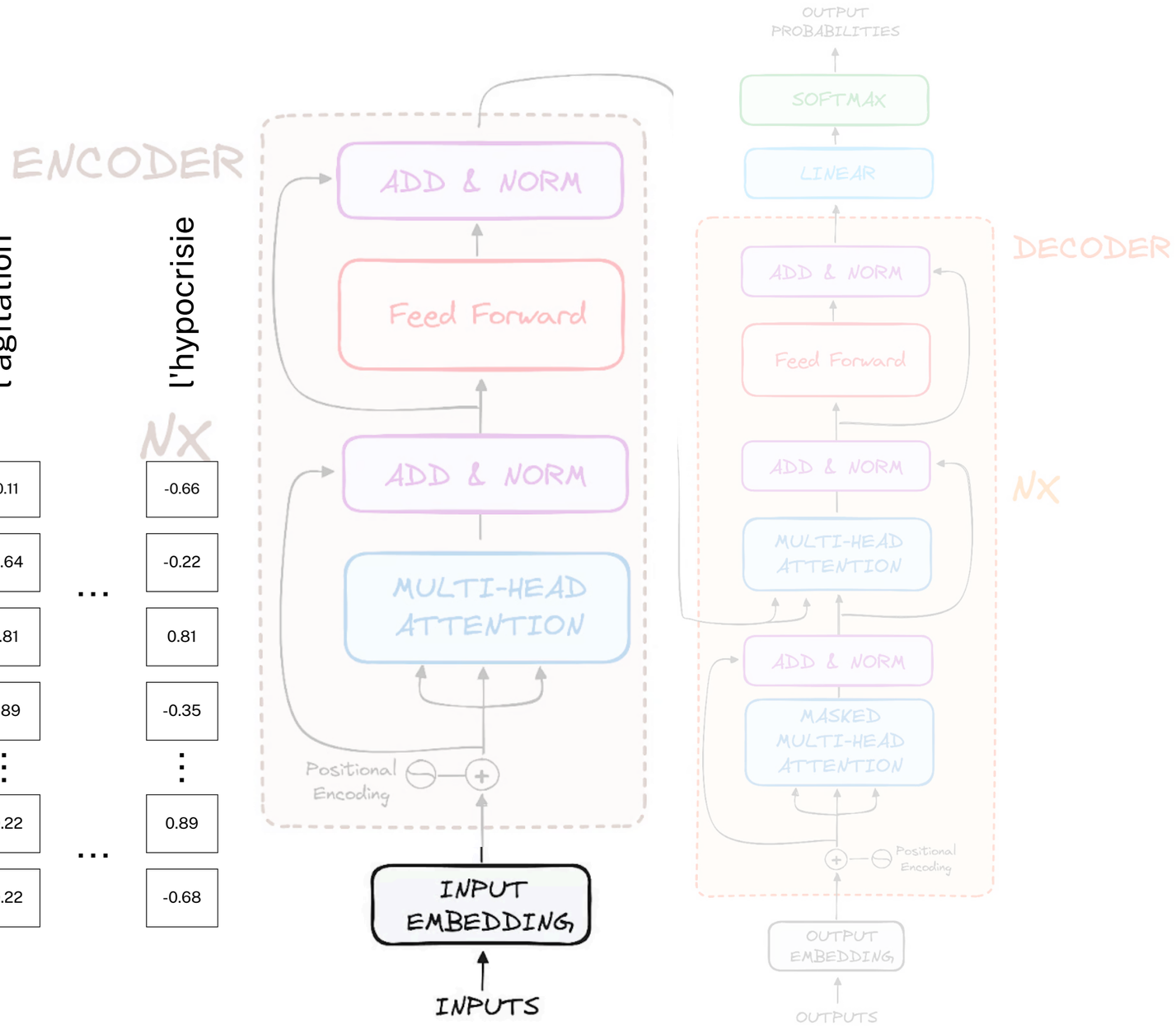
0.54 — Linguistic feature 512



Encoder architecture

Input Embeddings

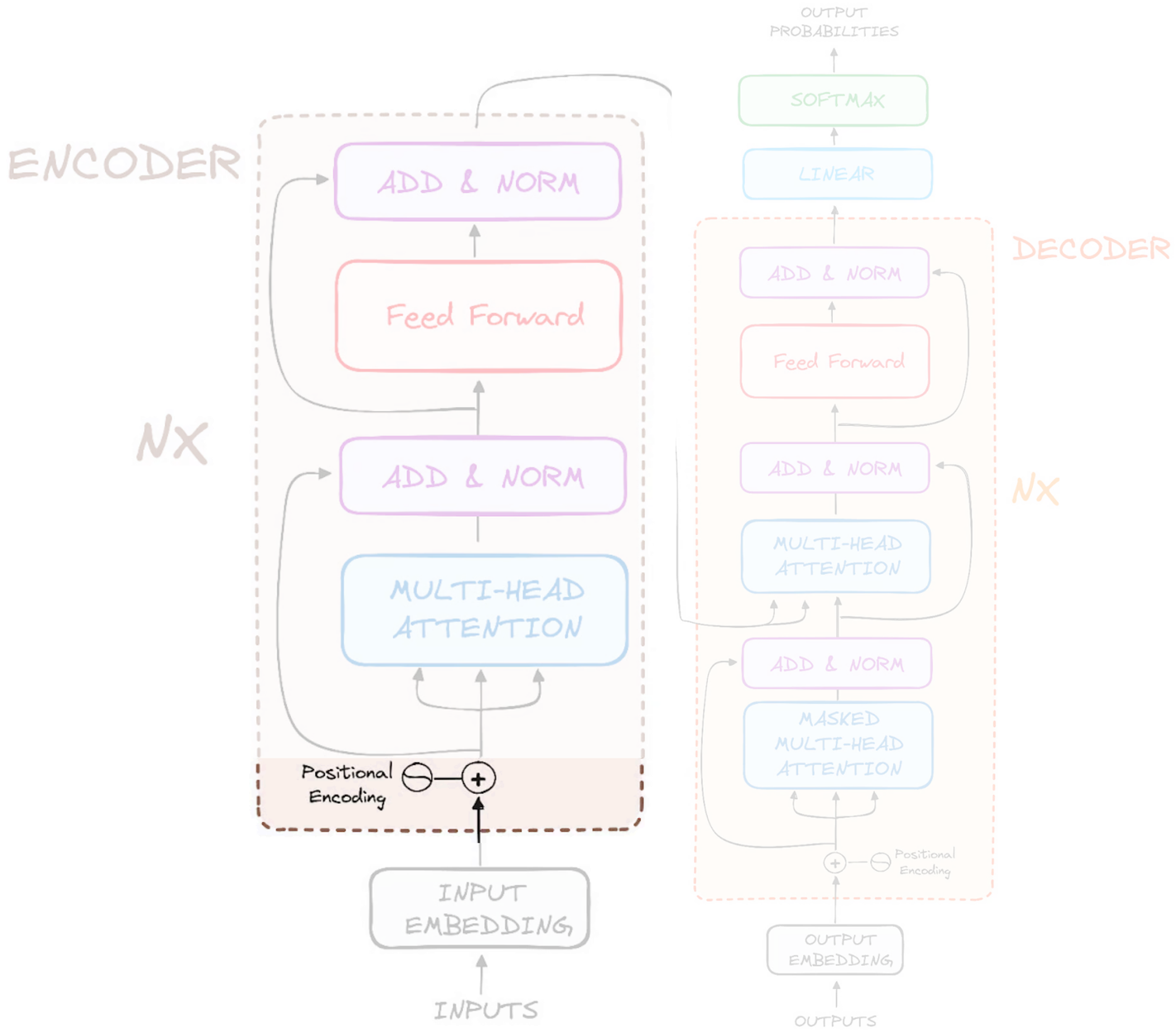
La	perfection	du	caractère	…	chaque	journée	…	dernière	…	éviter	l'agitation	…	l'hypocrisie
0.87	0.23	0.30			-0.68	0.87		0.12			-0.11		-0.66
-0.64	0.52	-0.68			-0.64	-0.02		0.89			-0.64		-0.22
0.81	-0.36	-0.25			0.81	0.03		-0.22			0.81		0.81
-0.35	0.88	0.94			-0.35	-0.22		0.05			0.89		-0.35
⋮	⋮	⋮			⋮	⋮		⋮			⋮		⋮
-0.22	-0.10	0.32			-0.22	-0.20		-0.22			-0.22		0.89
0.54	-0.56	0.64			0.81	-0.68		0.54			-0.22		-0.68



Encoder architecture

Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

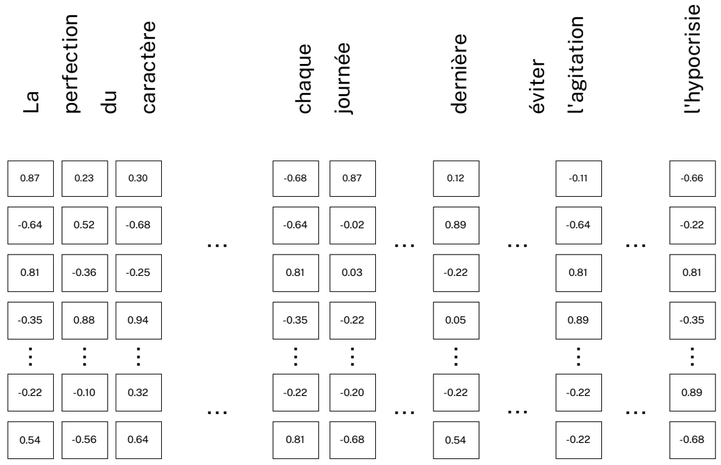


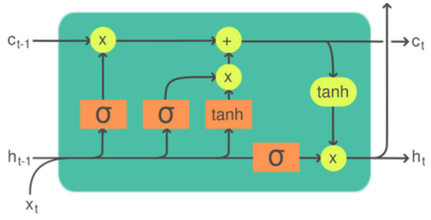
Encoder architecture

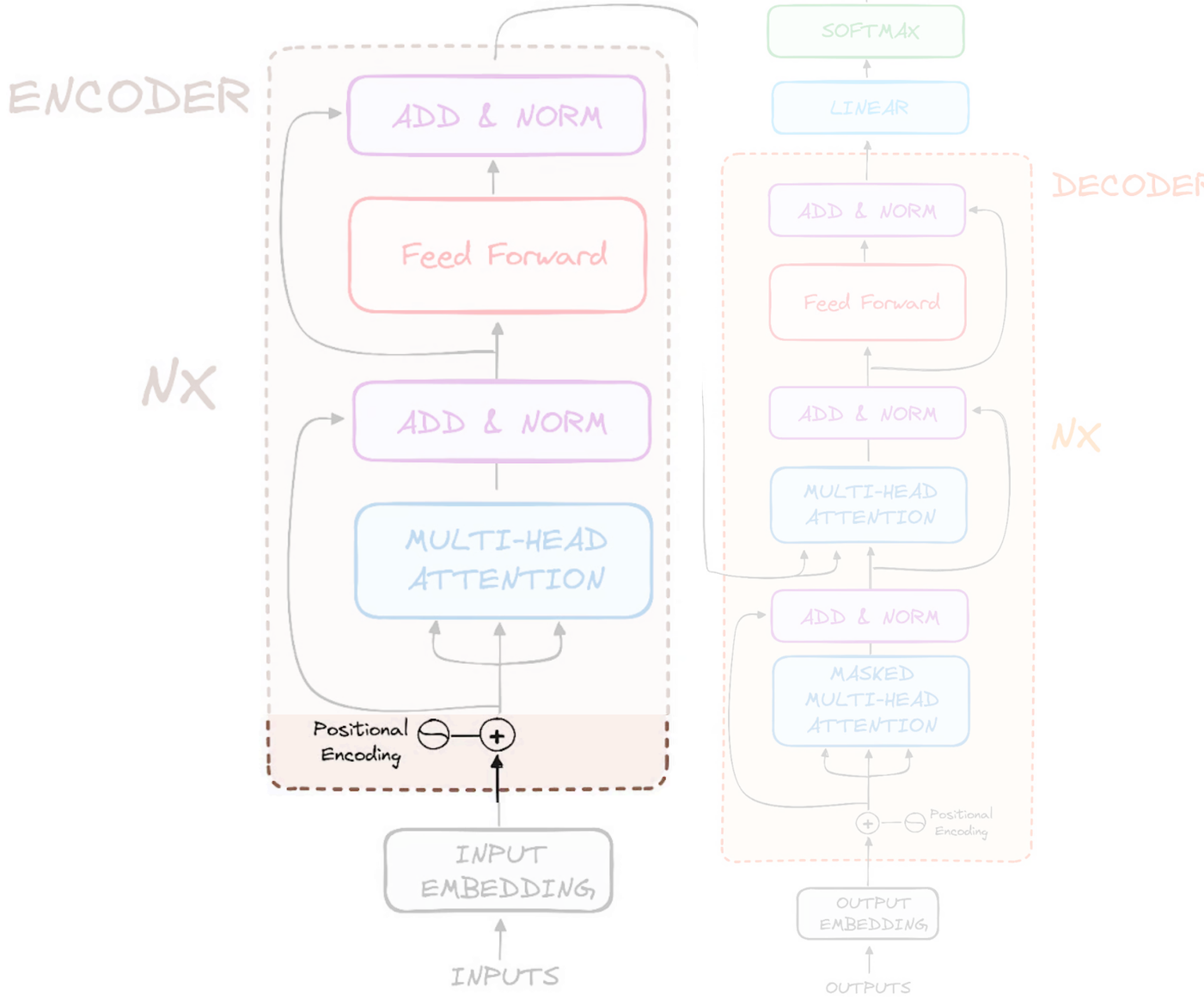
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





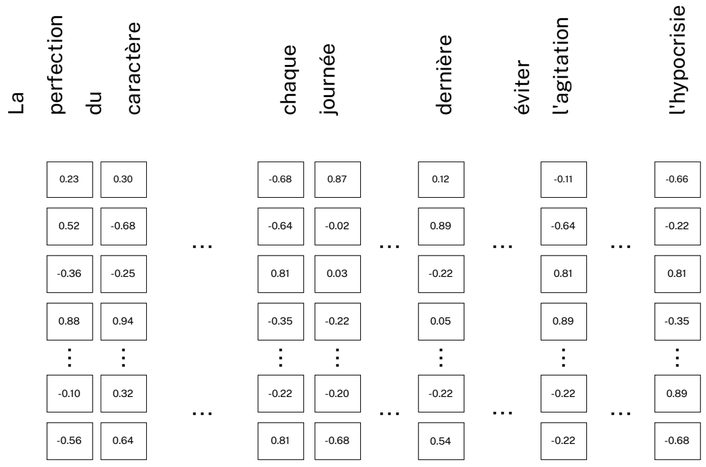


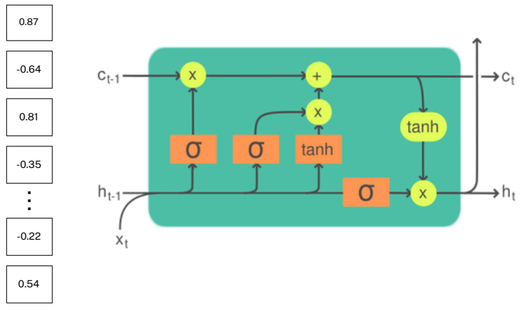
Encoder architecture

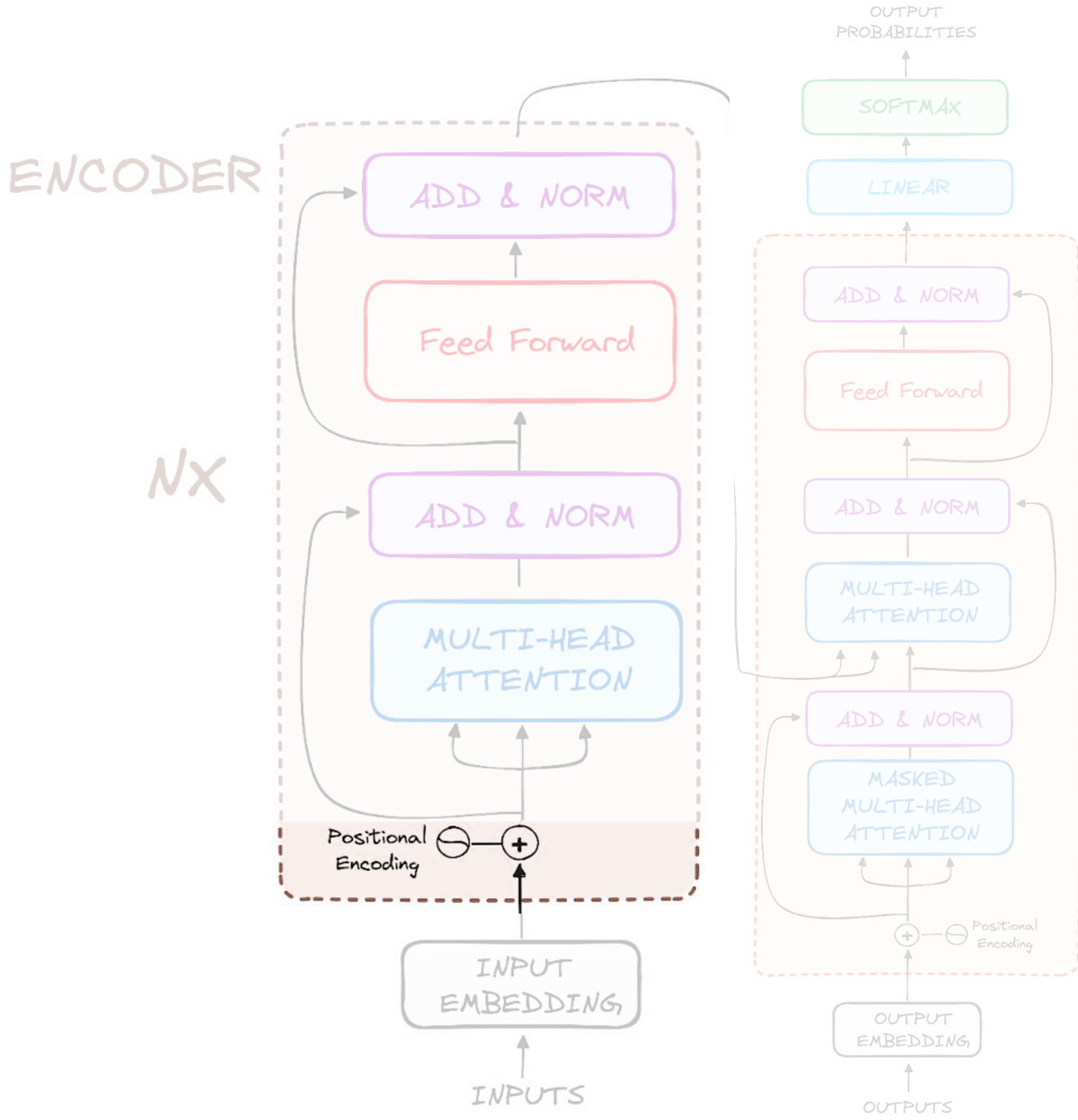
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





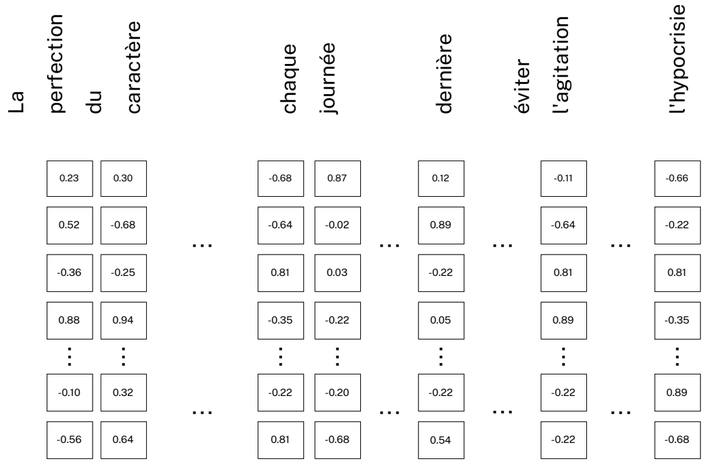


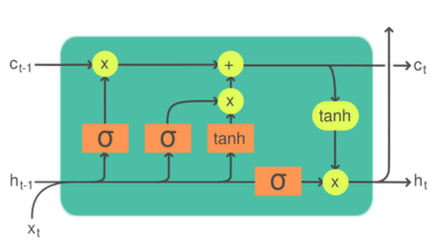
Encoder architecture

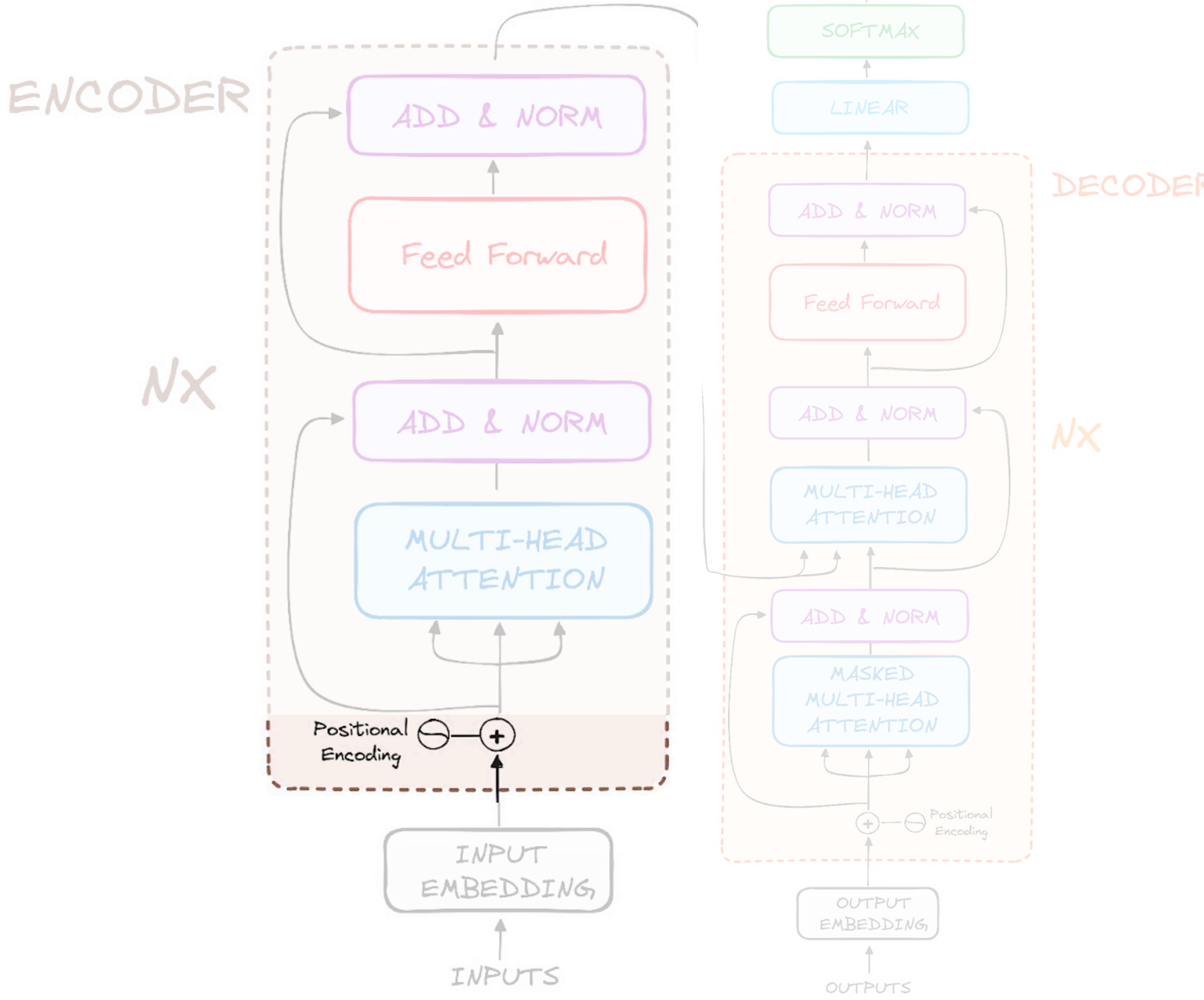
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





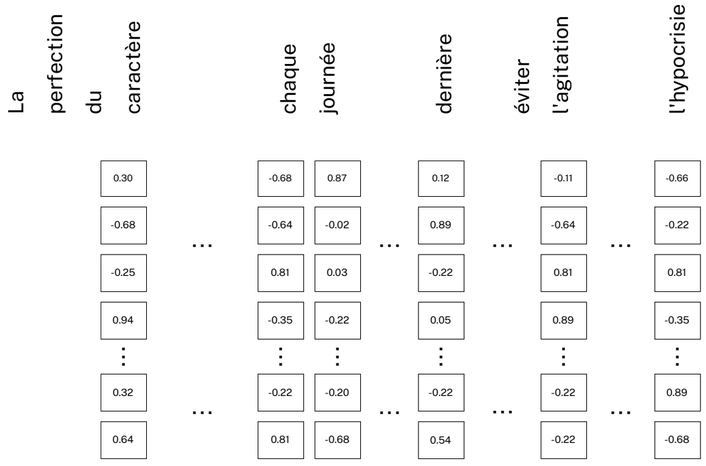


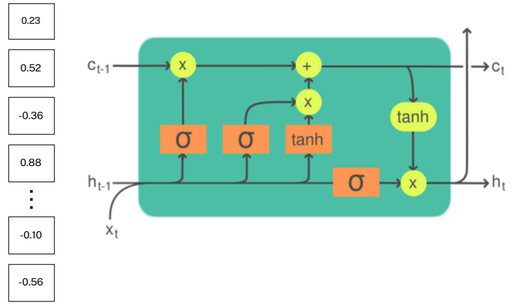
Encoder architecture

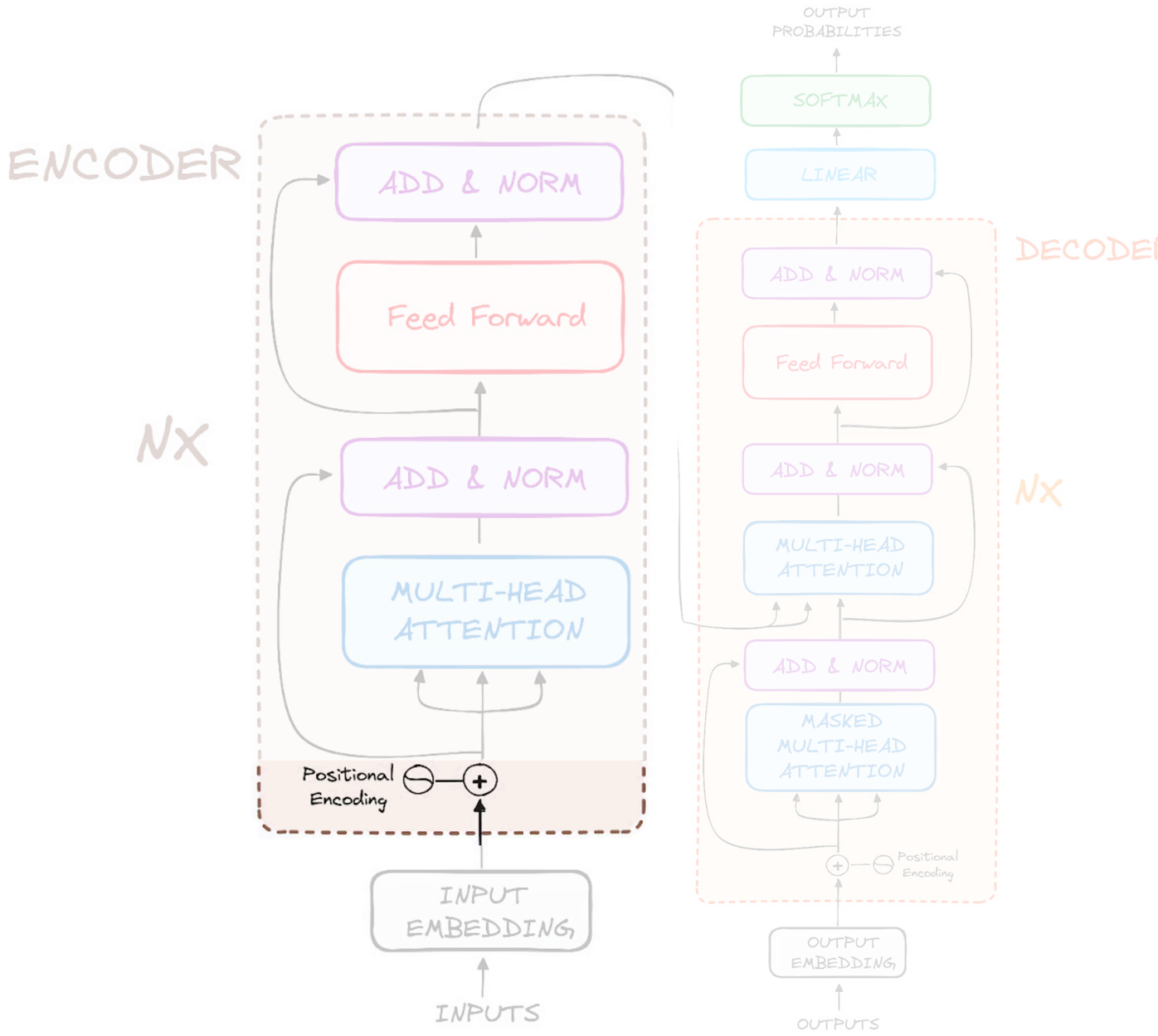
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





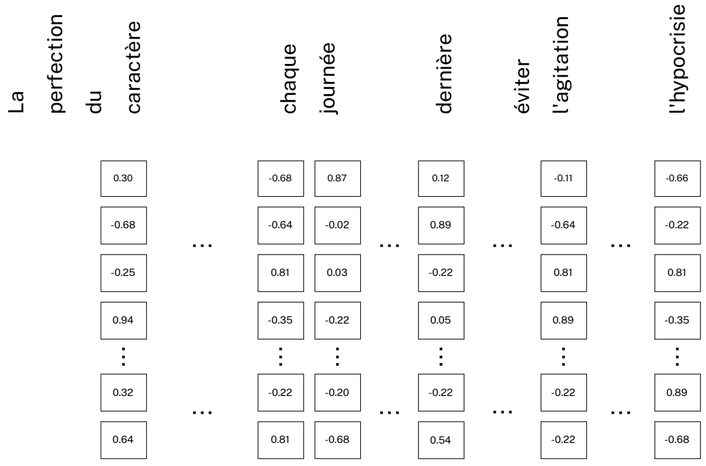


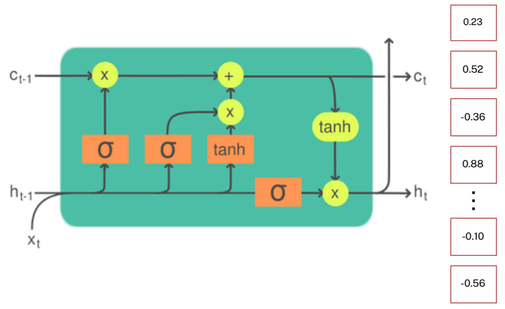
Encoder architecture

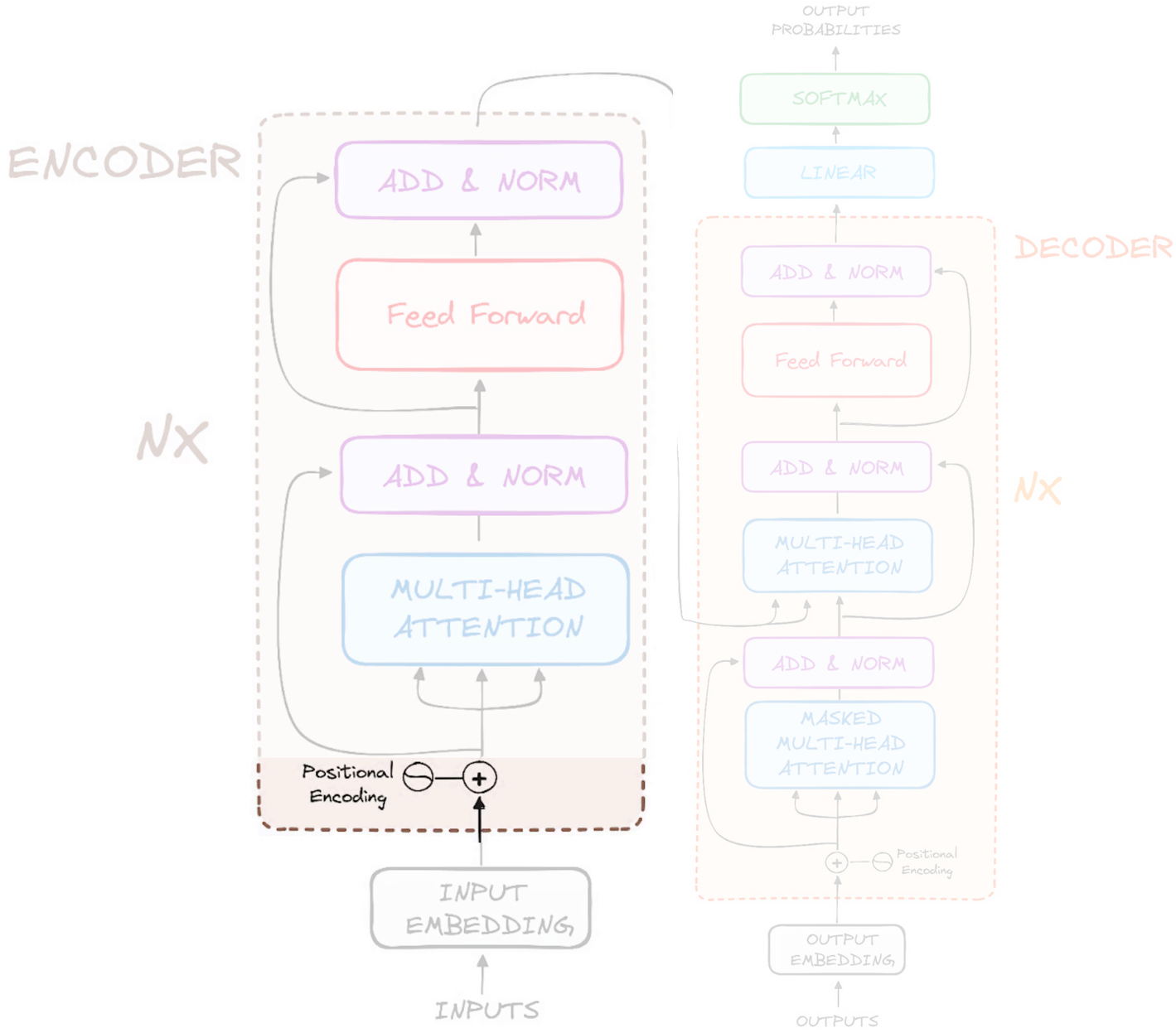
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





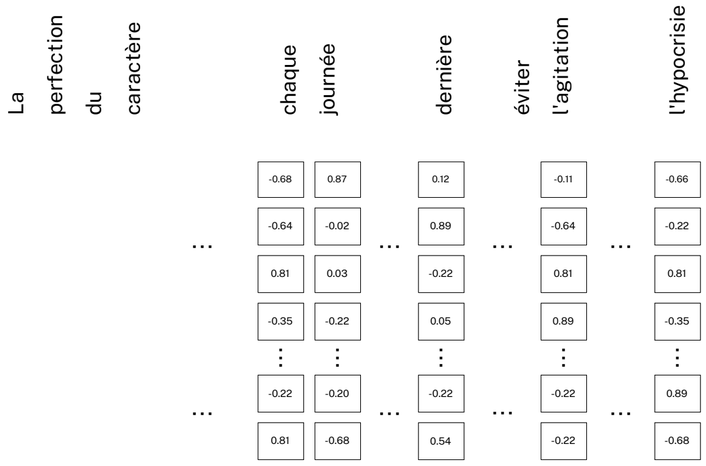


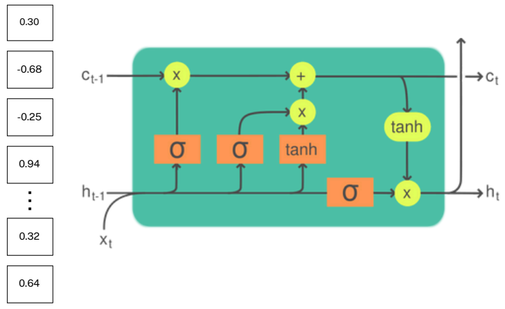
Encoder architecture

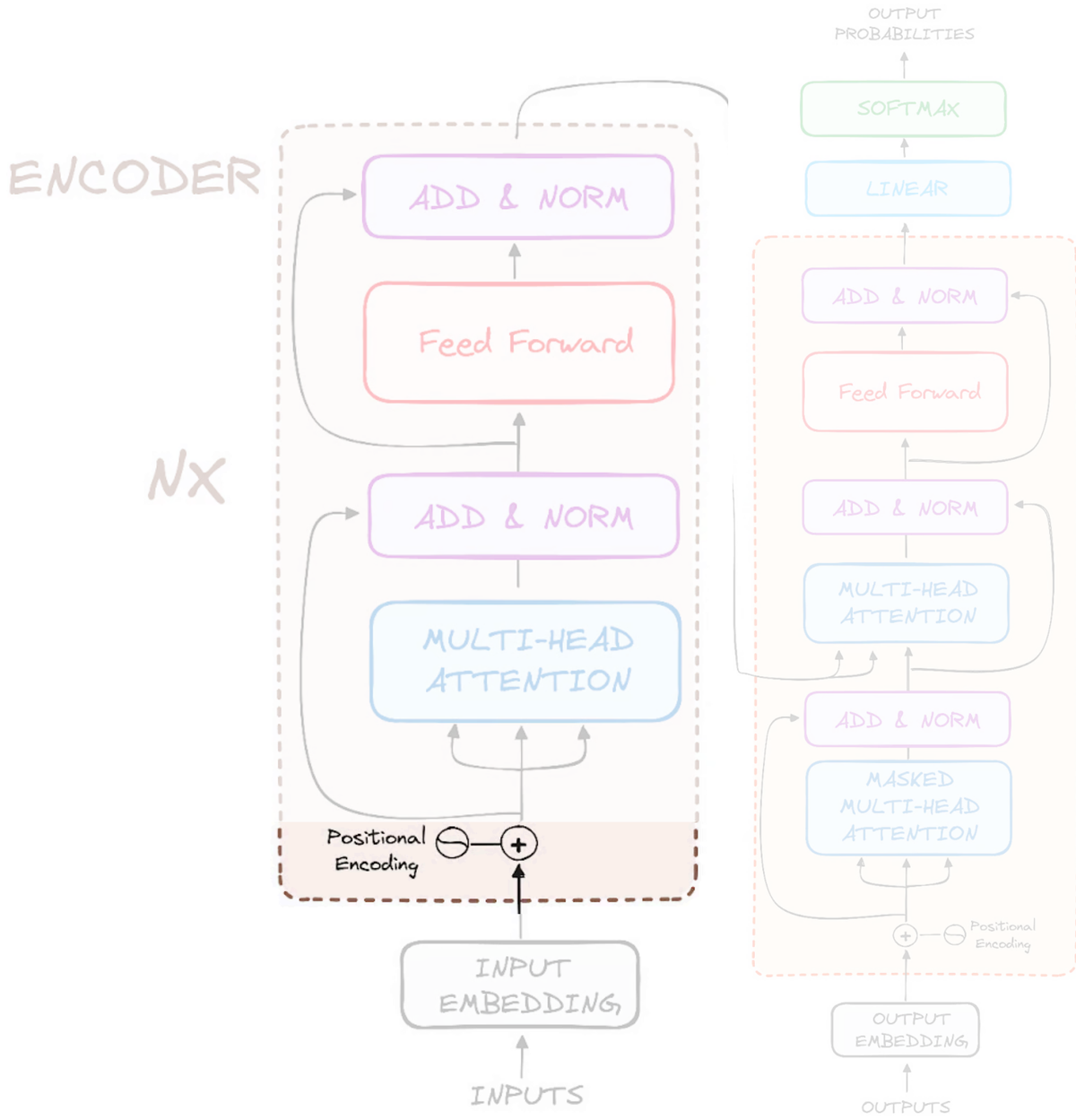
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





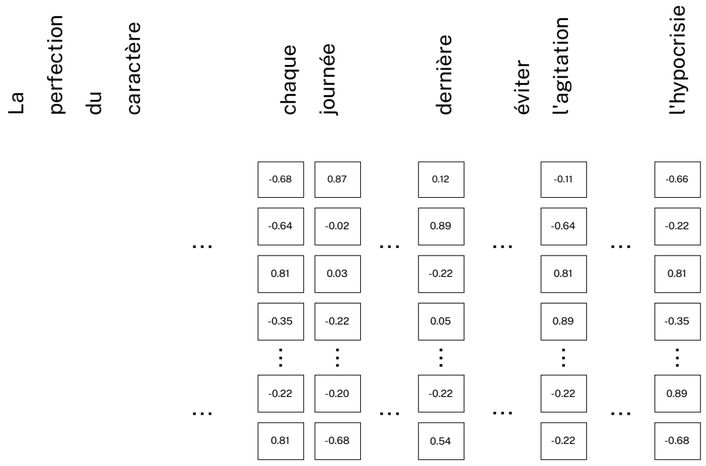


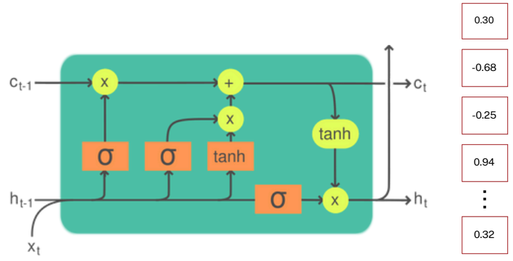
Encoder architecture

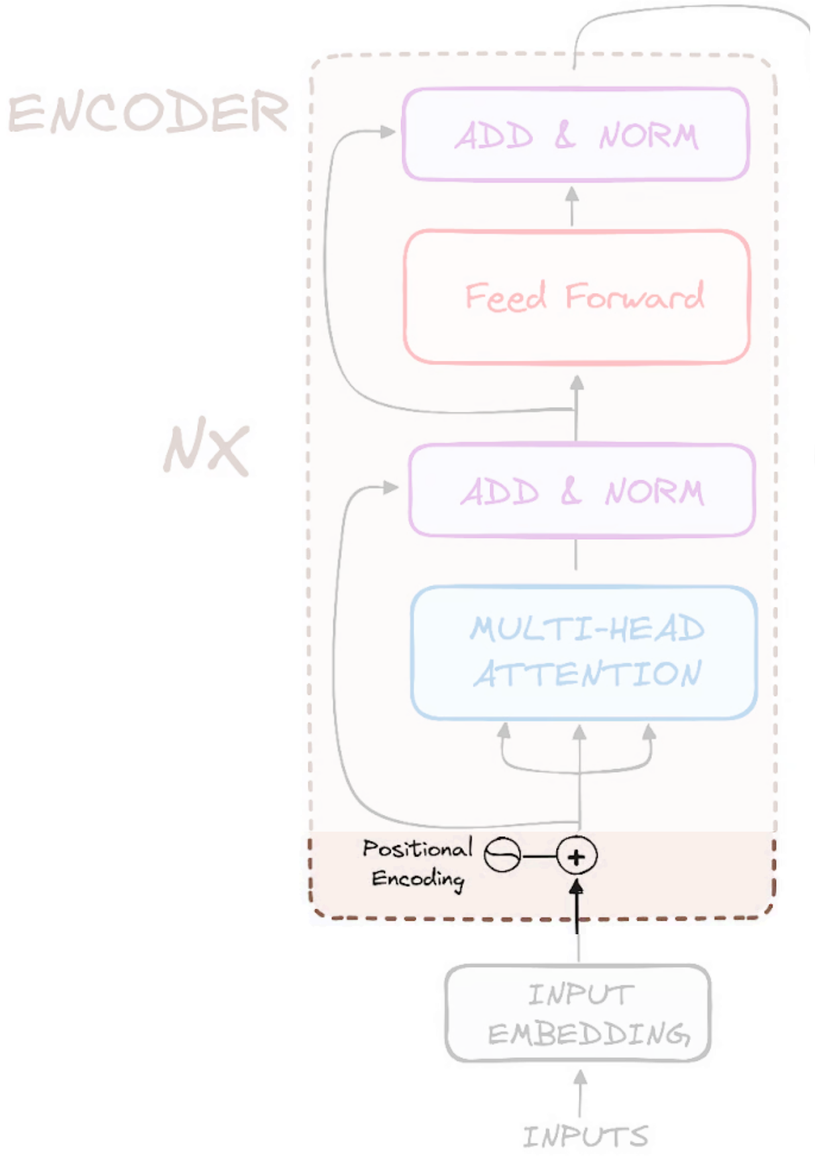
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





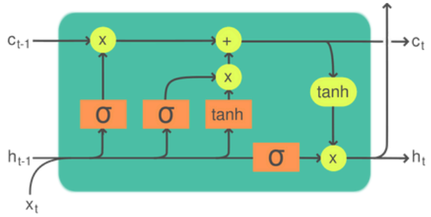


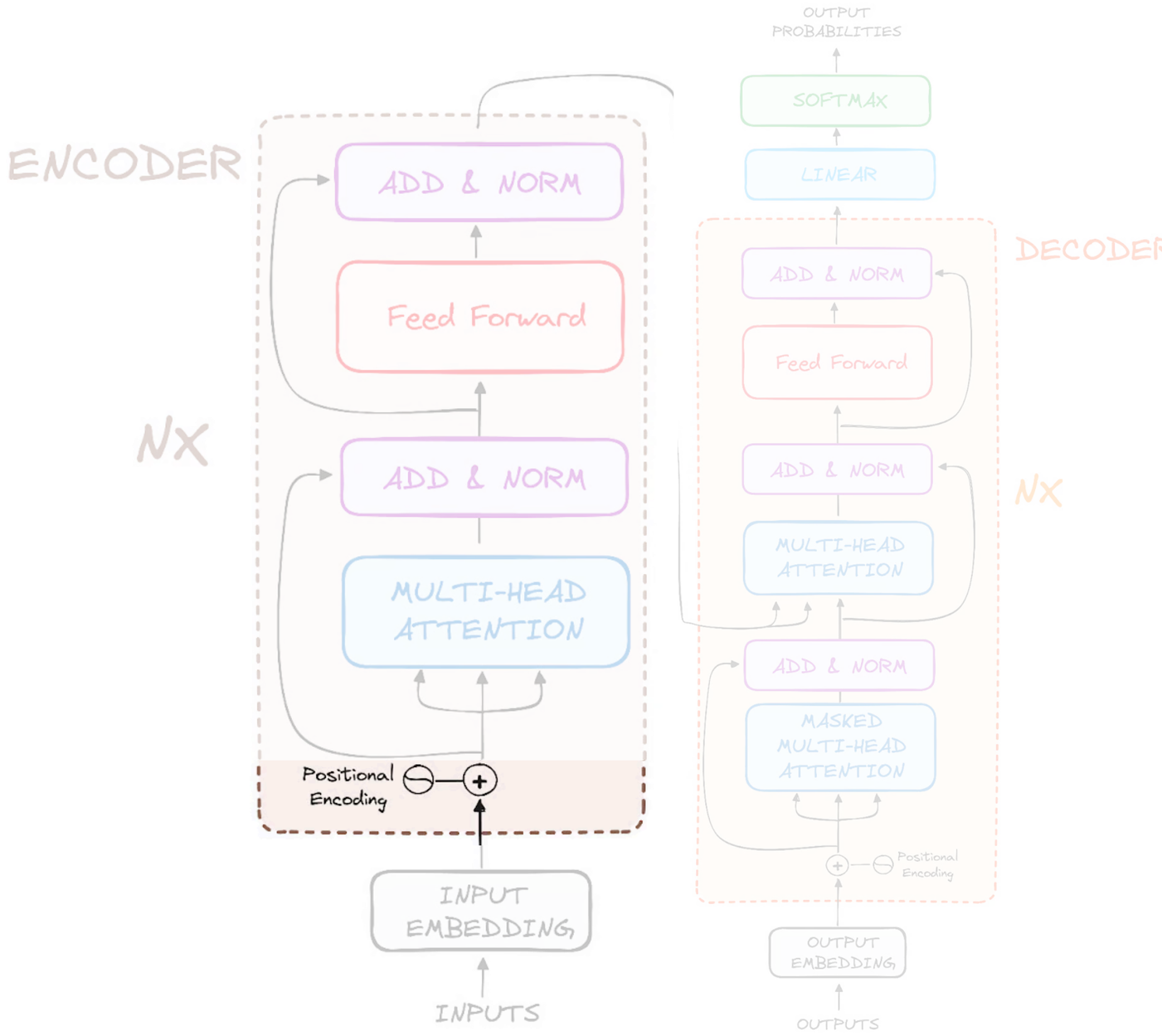
Encoder architecture

Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs





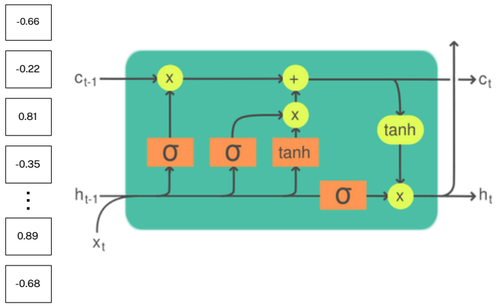
Encoder architecture

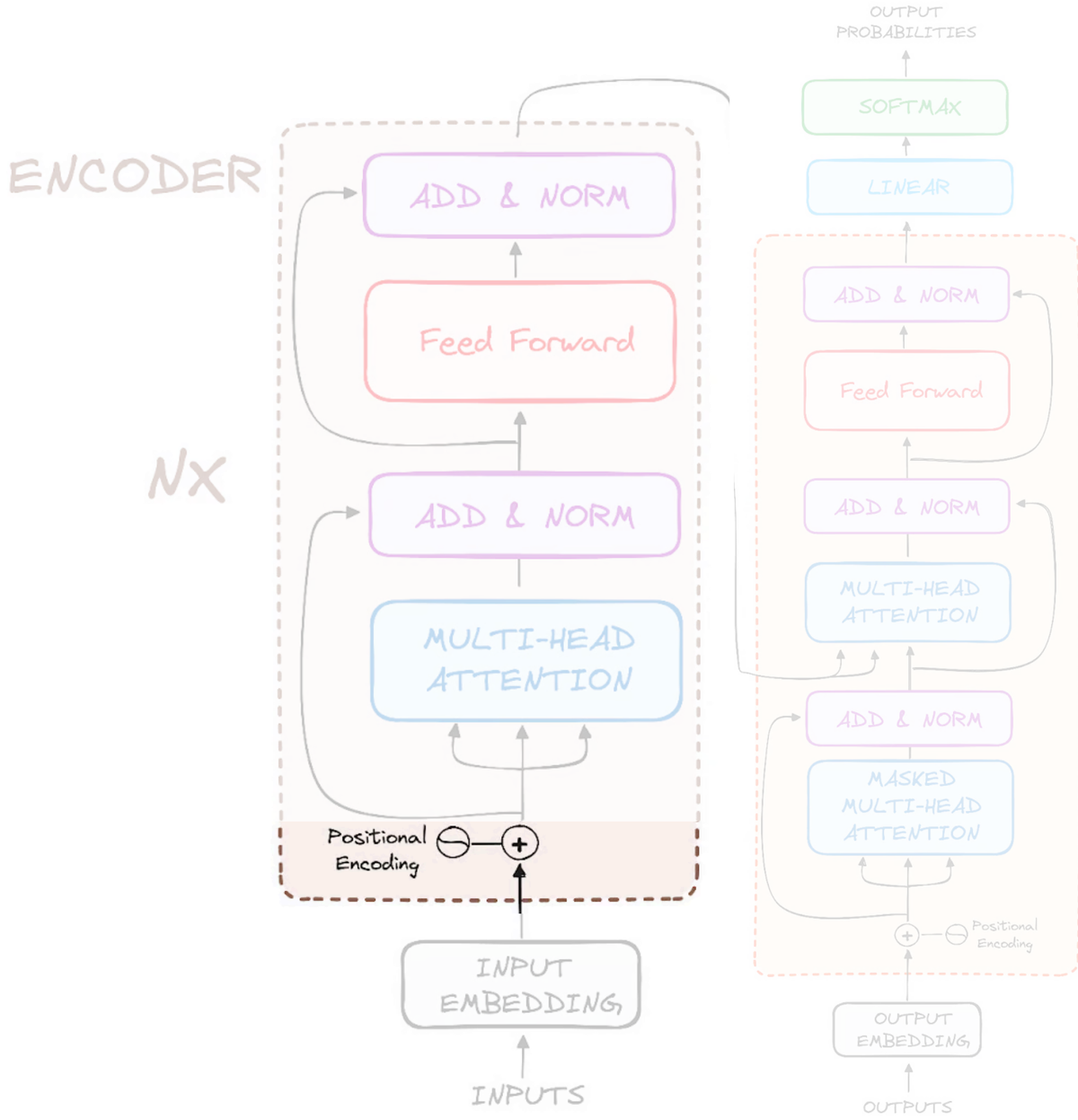
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs

La perfection du caractère chaque journée dernière éviter l'agitation l'hypocrisie





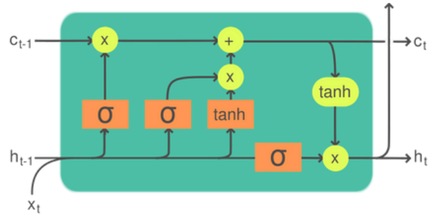
Encoder architecture

Positional Encoding

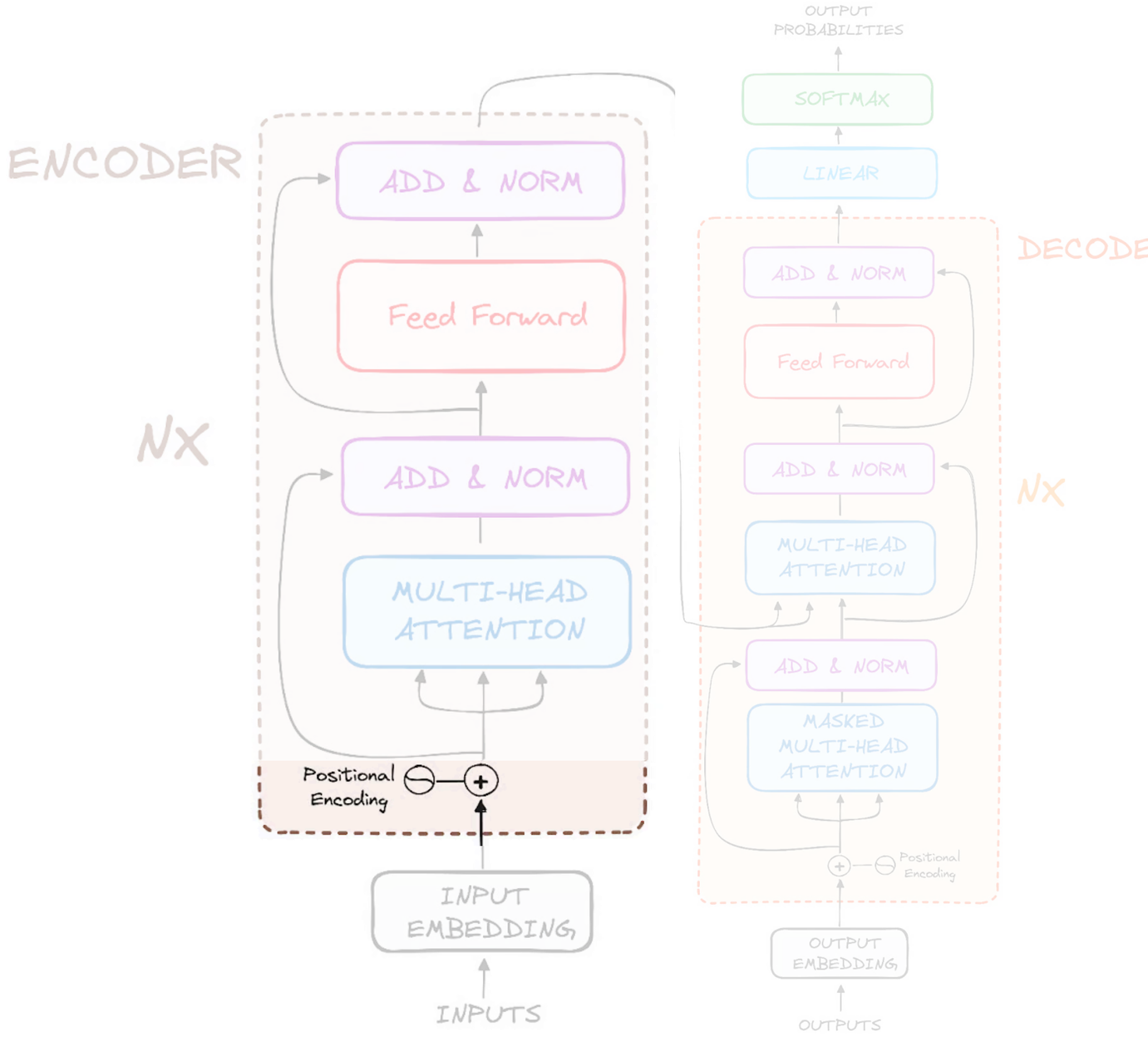
- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs

La perfection du caractère chaque journée dernière éviter l'agitation l'hypocrisie



-0.66
-0.22
0.81
-0.35
⋮
0.89
-0.68



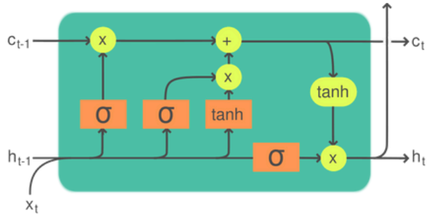
Encoder architecture

Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In RNNs/LSTMs

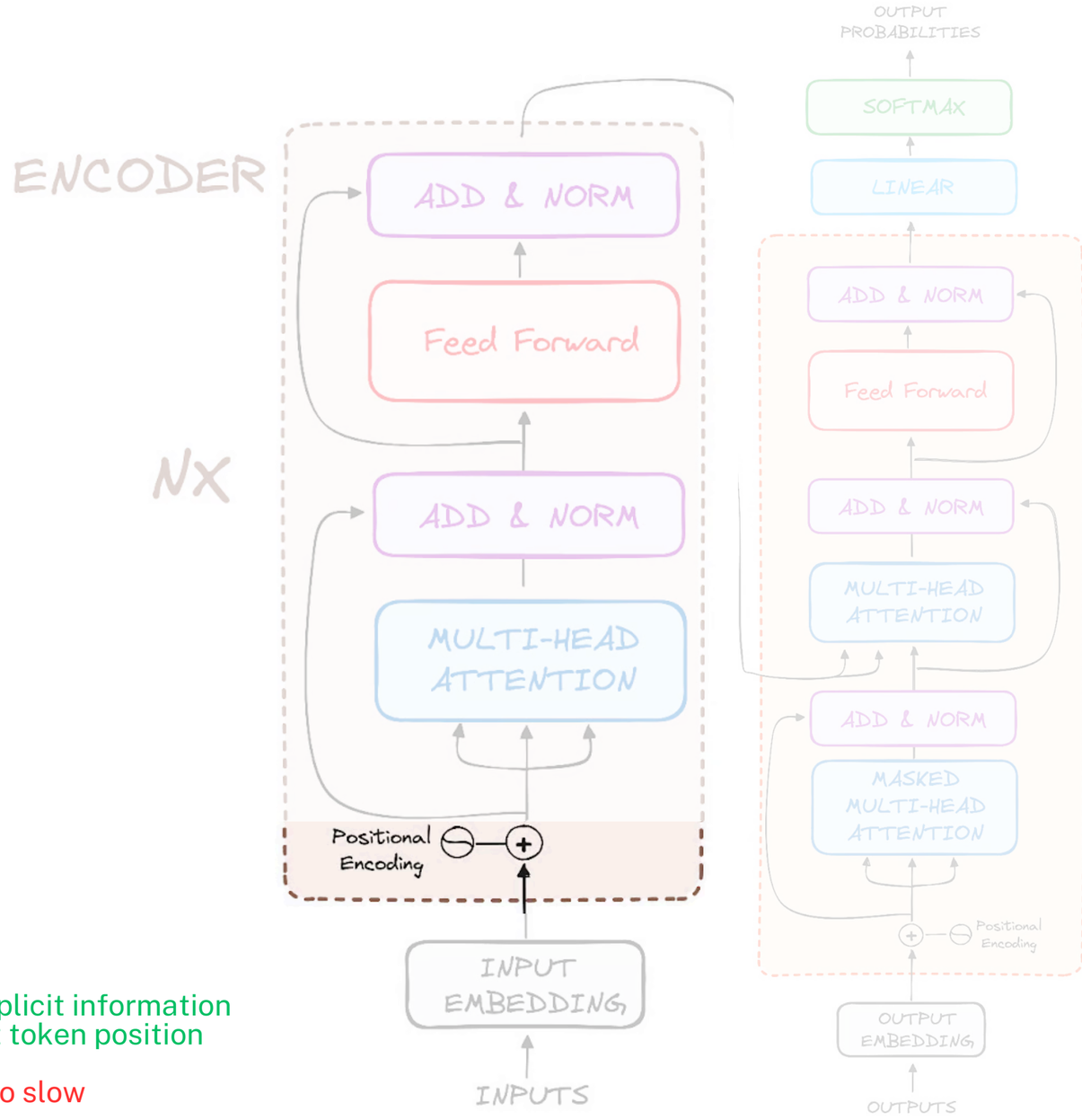
La perfection du caractère chaque journée dernière éviter l'agitation l'hypocrisie



-0.66
-0.22
0.81
-0.35
⋮
0.89
-0.68

++ Implicit information about token position

- - Too slow

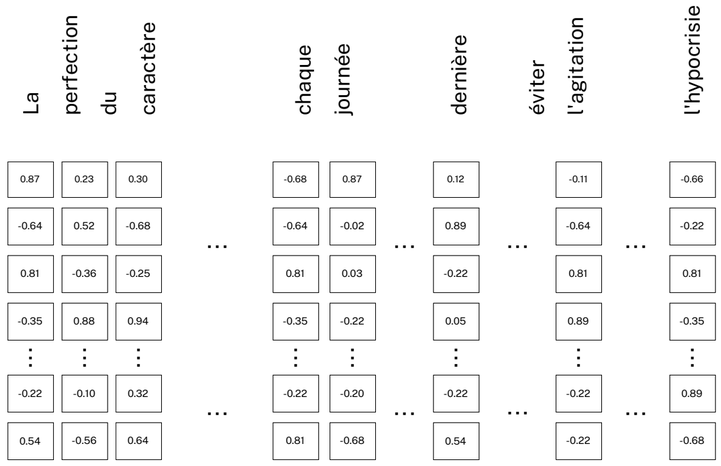


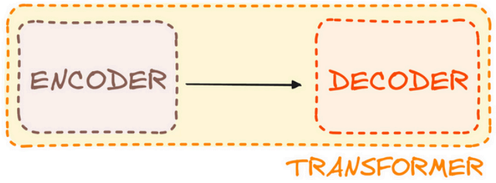
Encoder architecture

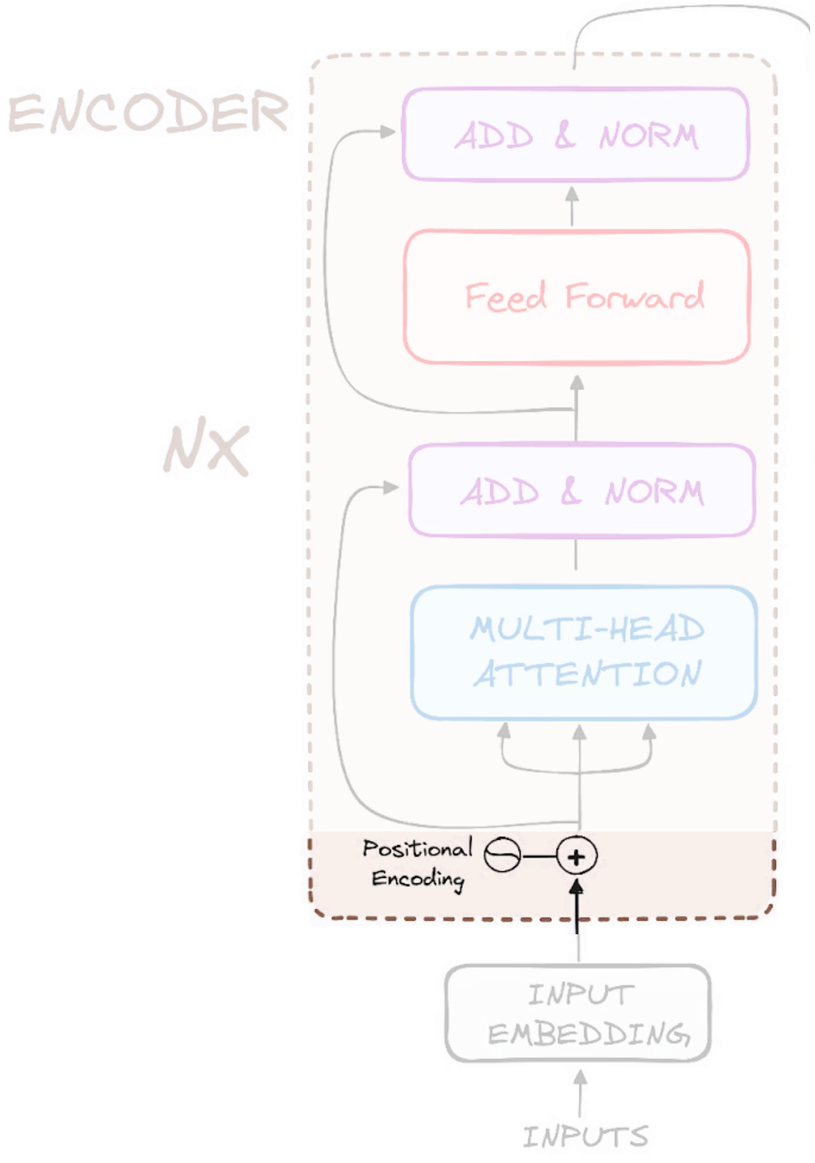
Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In Transformers





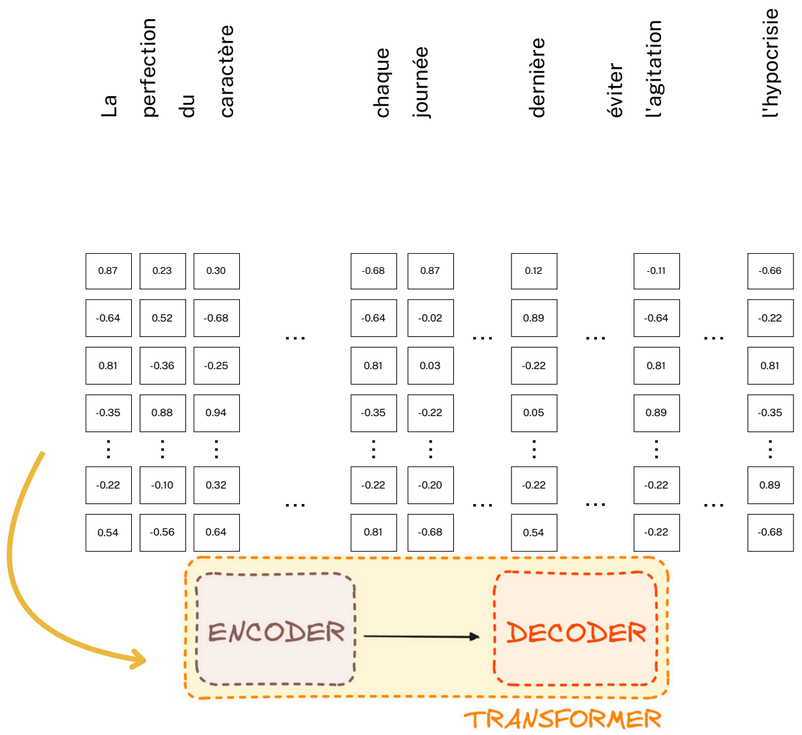


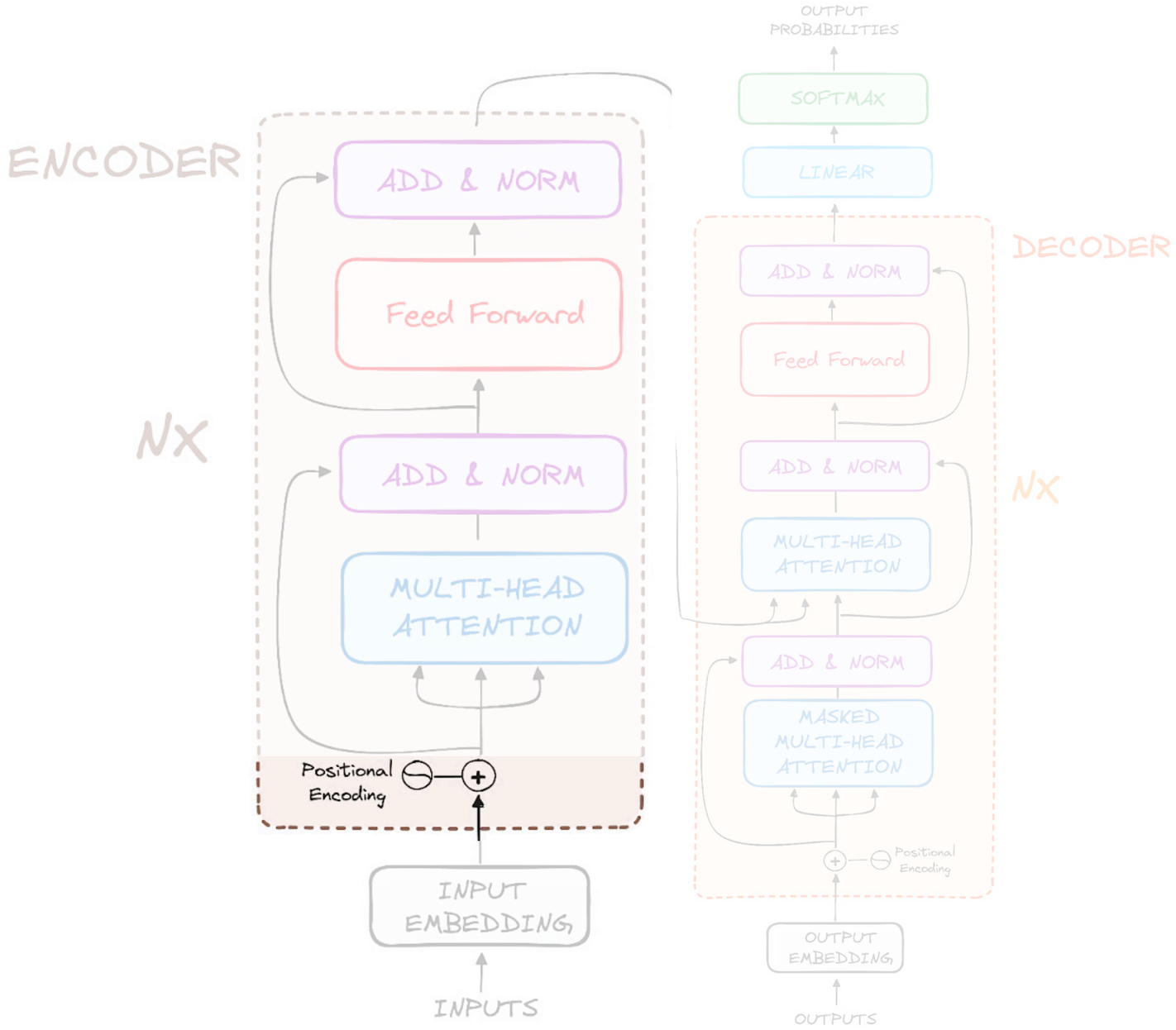
Encoder architecture

Positional Encoding

- Is added to the input embeddings
- Gives information about **position** of token (because it does not use recurrence)

In Transformers





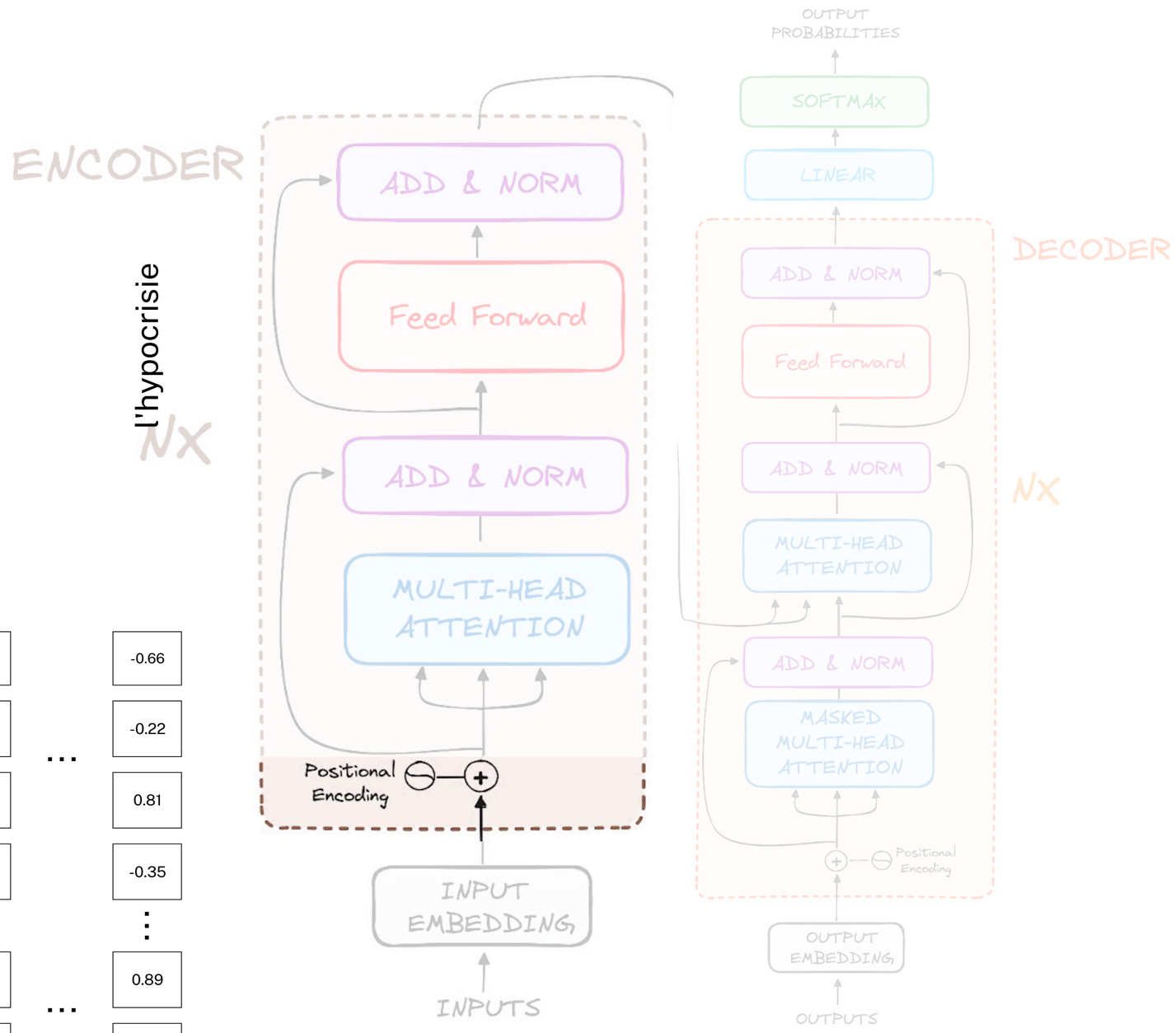
How to know which token came first and which came last ?

Encoder architecture

Positional Encoding

La perfection du caractère ... chaque journée ... dernière ... éviter l'agitation ... l'hypocrisie

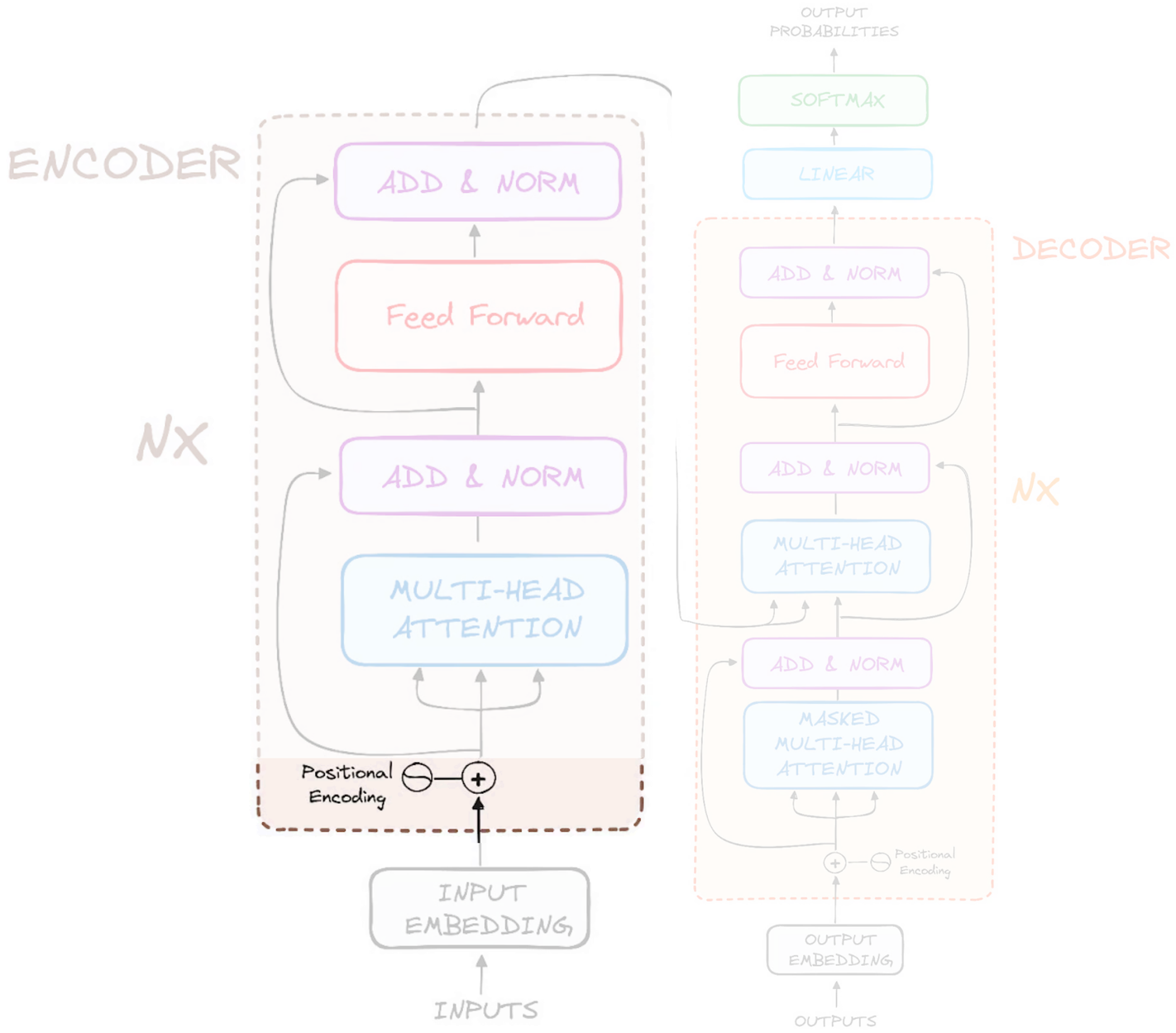
0.87	0.23	0.30	...	-0.68	0.87	...	0.12	...	-0.11	...	-0.66
-0.64	0.52	-0.68		-0.64	-0.02		0.89		-0.64		-0.22
0.81	-0.36	-0.25		0.81	0.03		-0.22		0.81		0.81
-0.35	0.88	0.94		-0.35	-0.22		0.05		0.89		-0.35
⋮	⋮	⋮		⋮	⋮		⋮		⋮		⋮
-0.22	-0.10	0.32	...	-0.22	-0.20	...	-0.22	...	-0.22	...	0.89
0.54	-0.56	0.64		0.81	-0.68		0.54		-0.22		-0.68



Encoder architecture

Positional Encoding

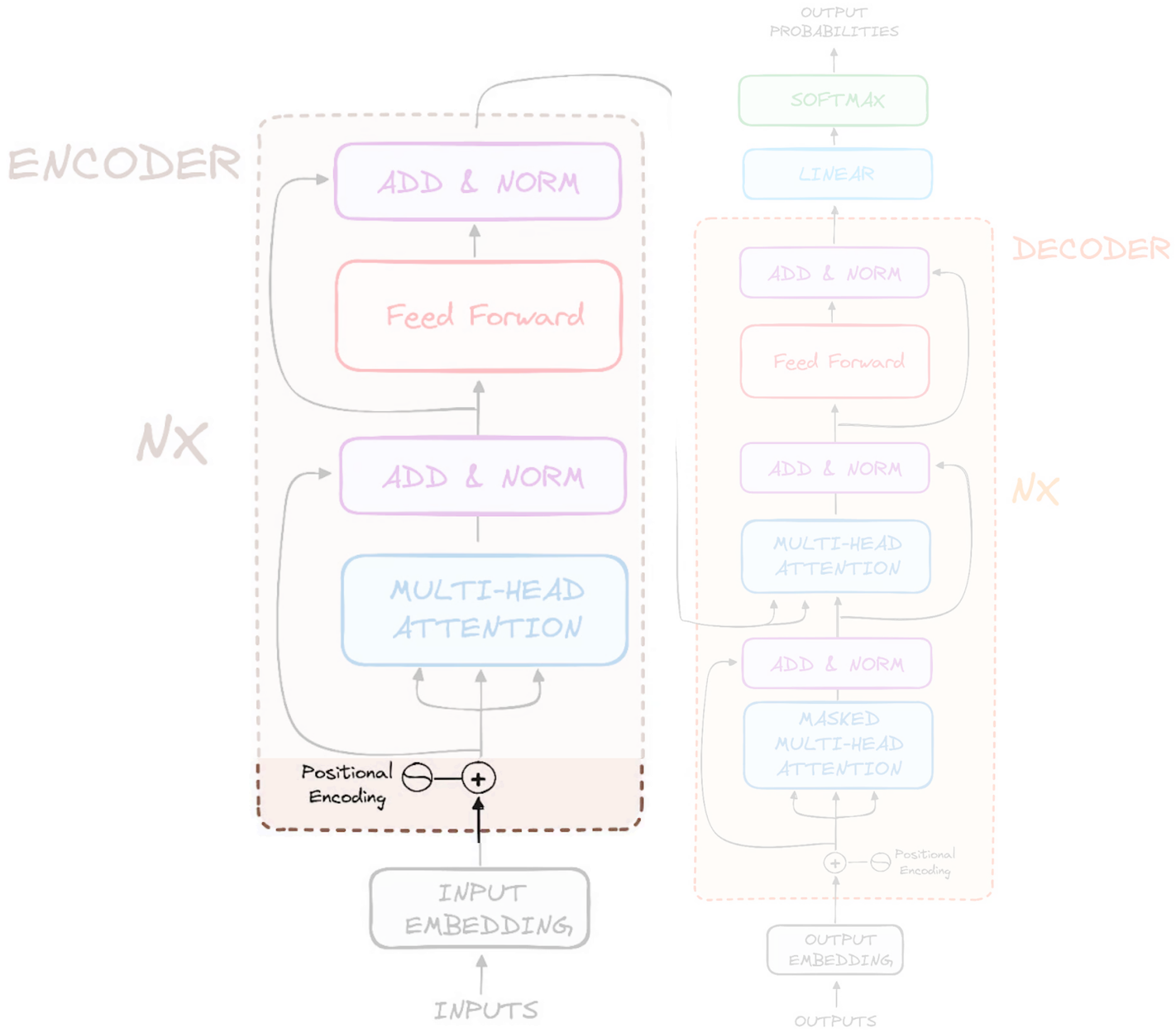
e0	e1	e2
0.87	0.23	0.30
-0.64	0.52	-0.68
0.81	-0.36	-0.25
-0.35	0.88	0.94
⋮	⋮	⋮
-0.22	-0.10	0.32
0.54	-0.56	0.64



Encoder architecture

Positional Encoding (intuitive solution)

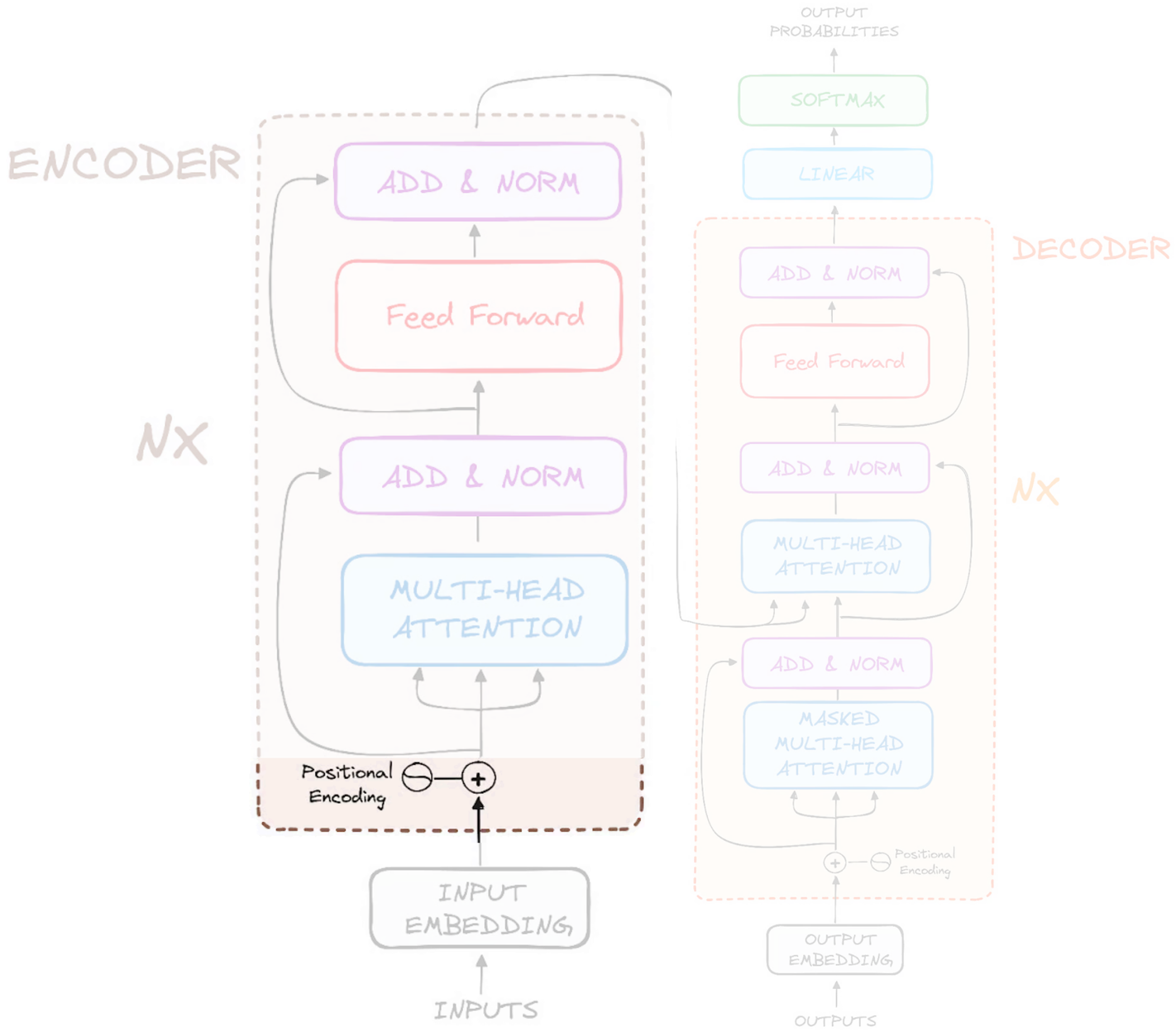
e0	p0	e1	p1	e2	p2
0.87	0	0.23	1	0.30	2
-0.64	0	0.52	1	-0.68	2
0.81	0	-0.36	1	-0.25	2
-0.35	0	0.88	1	0.94	2
⋮	+ ⋮	⋮	+ ⋮	⋮	+ ⋮
-0.22	0	-0.10	1	0.32	2
0.54	0	-0.56	1	0.64	2



Encoder architecture

Positional Encoding (intuitive solution)

e0	p0	e1	p1	e2	p2
0.87	0	0.23	1	0.30	2
-0.64	0	0.52	1	-0.68	2
0.81	0	-0.36	1	-0.25	2
-0.35	0	0.88	1	0.94	2
⋮	+ ⋮	⋮	+ ⋮	⋮	+ ⋮
-0.22	0	-0.10	1	0.32	2
0.54	0	-0.56	1	0.64	2



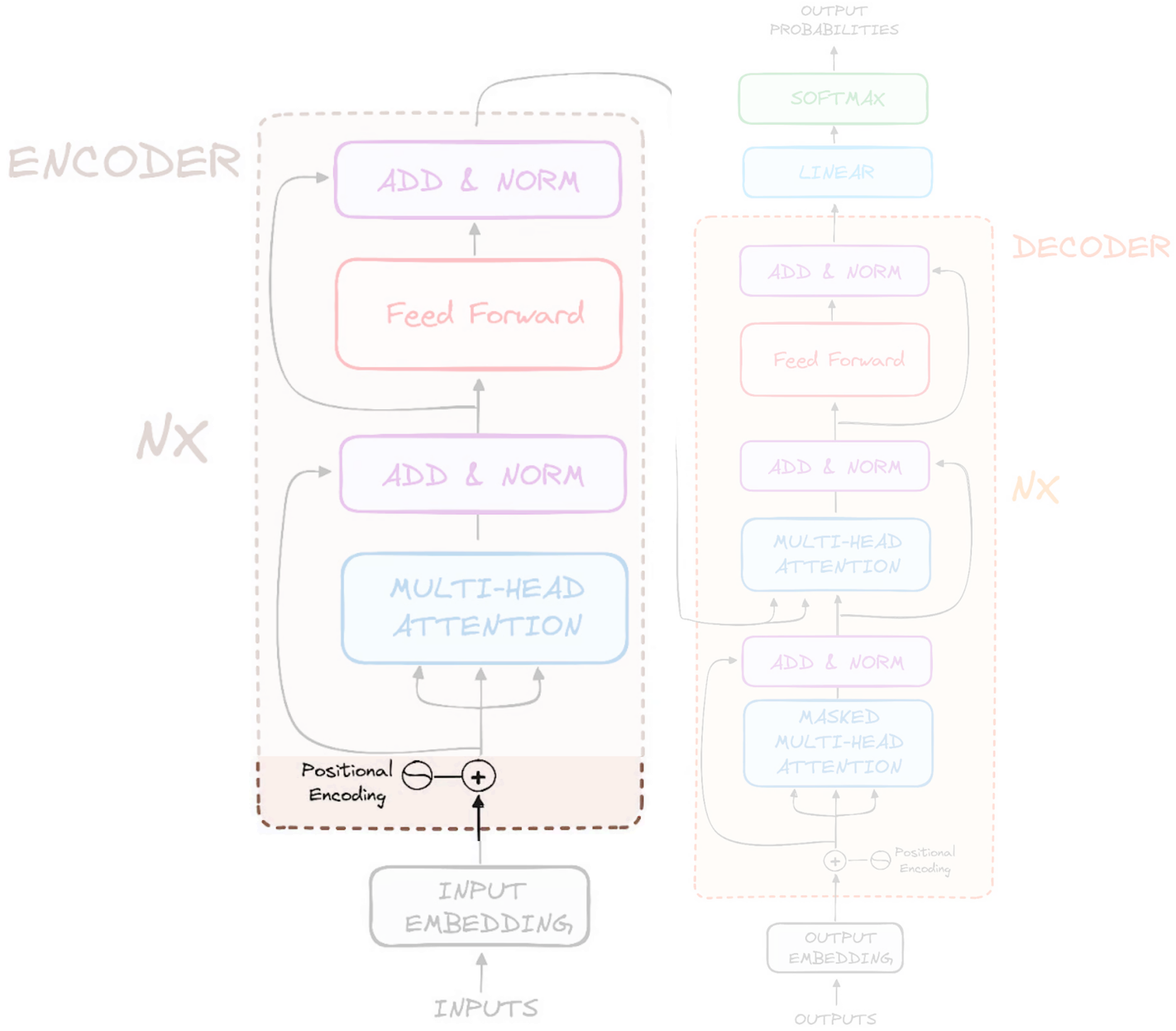
This addition alters the embedding information, Not Good !

Encoder architecture

Positional Encoding (intuitive solution)

Normalize between 0 and 1 ? 1/(N-1)

e0	p0	e1	p1	e2	p2
0.87	0	0.23	0.33	0.30	0.66
-0.64	0	0.52	0.33	-0.68	0.66
0.81	0	-0.36	0.33	-0.25	0.66
-0.35	0	0.88	0.33	0.94	0.66
⋮	+ ⋮	⋮	+ ⋮	⋮	+ ⋮
-0.22	0	-0.10	0.33	0.32	0.66
0.54	0	-0.56	0.33	0.64	0.66

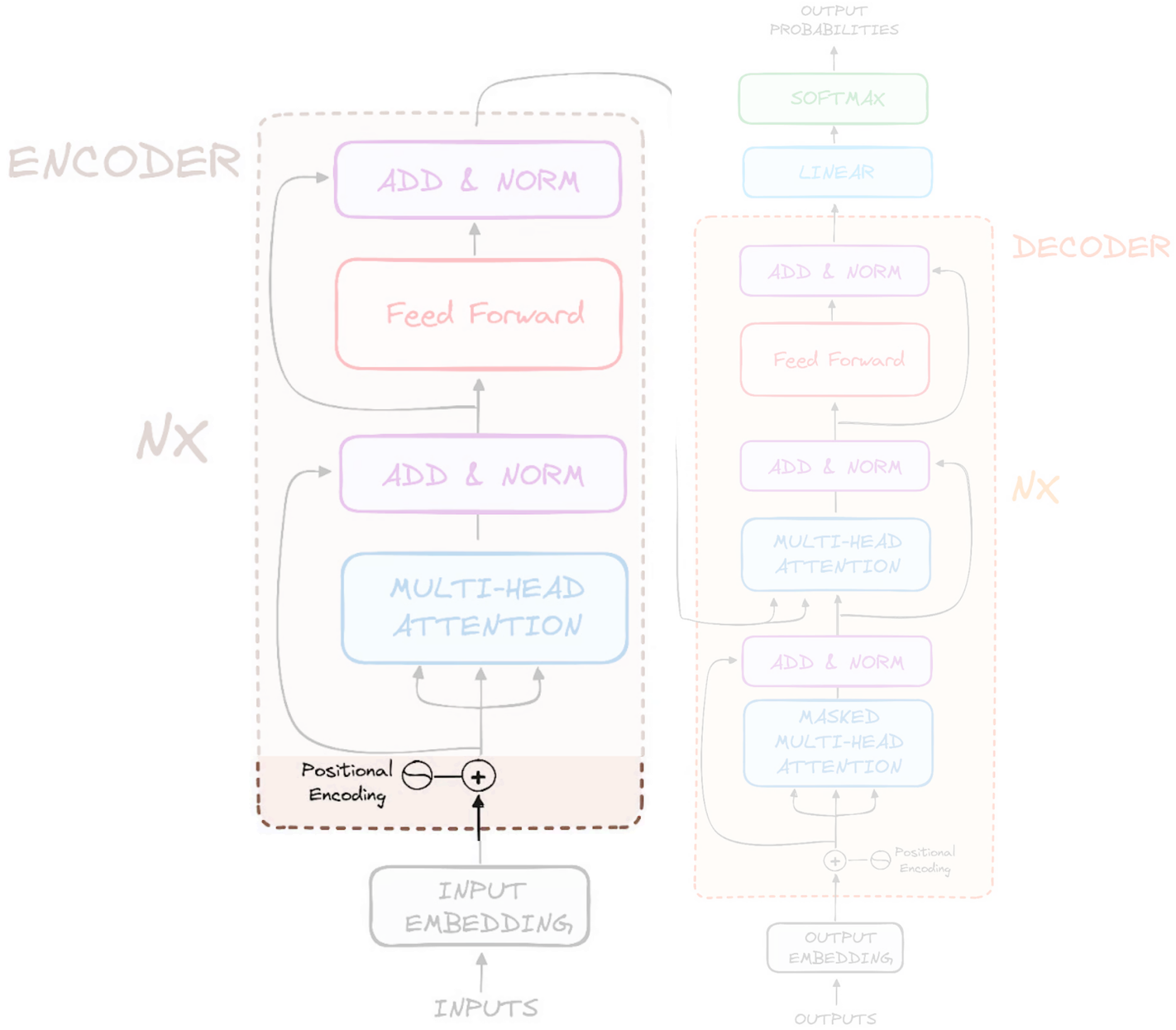


Encoder architecture

Positional Encoding (intuitive solution)

Normalize between 0 and 1 ? 1/(N-1)

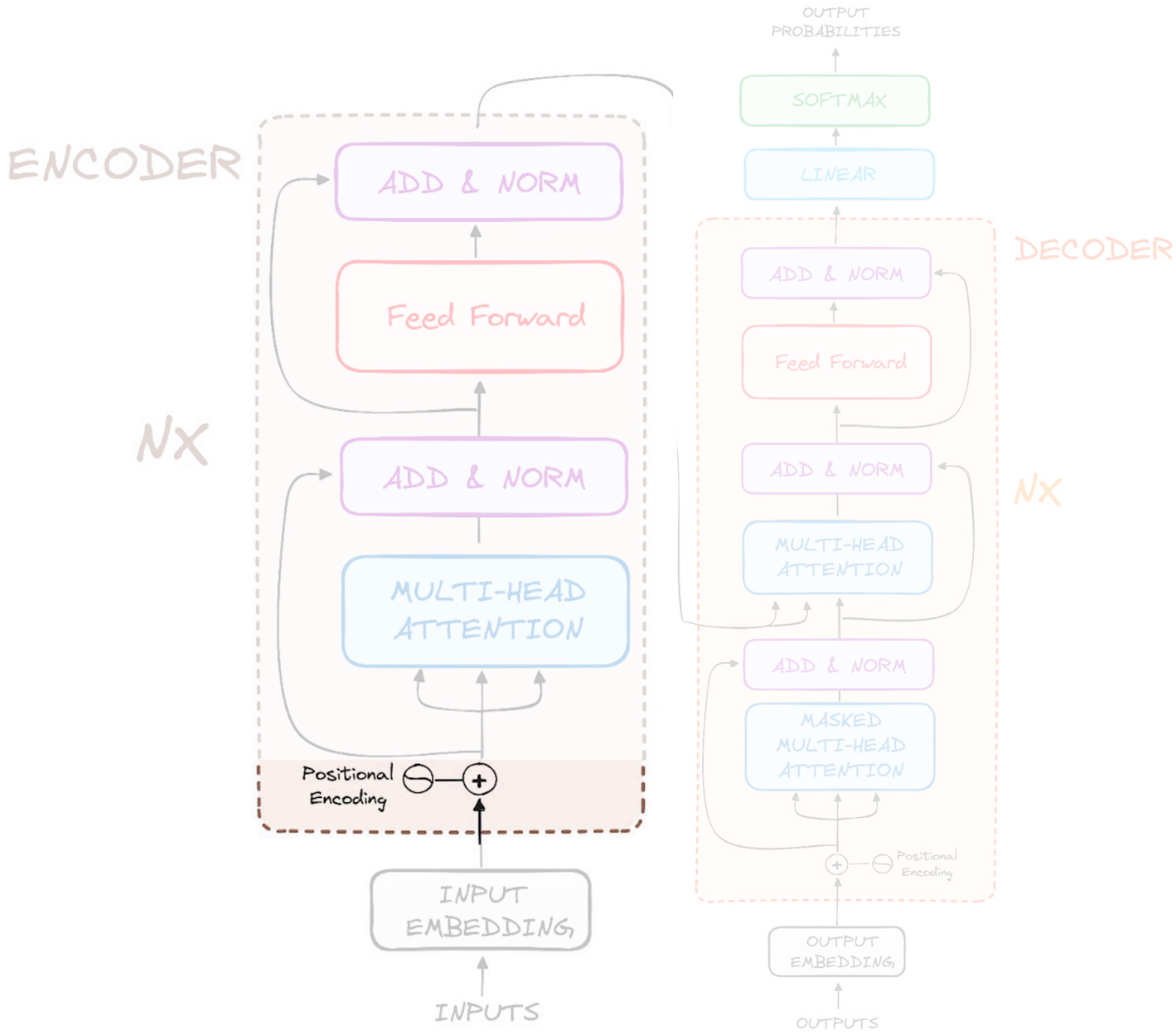
e0	p0	e1	p1	e2	p2
0.87	0	0.23	0.33	0.30	0.66
-0.64	0	0.52	0.33	-0.68	0.66
0.81	0	-0.36	0.33	-0.25	0.66
-0.35	0	0.88	0.33	0.94	0.66
⋮	+ ⋮	⋮	+ ⋮	⋮	+ ⋮
-0.22	0	-0.10	0.33	0.32	0.66
0.54	0	-0.56	0.33	0.64	0.66



We need to know the length N beforehand
For different sentences lengths, same position value changes !

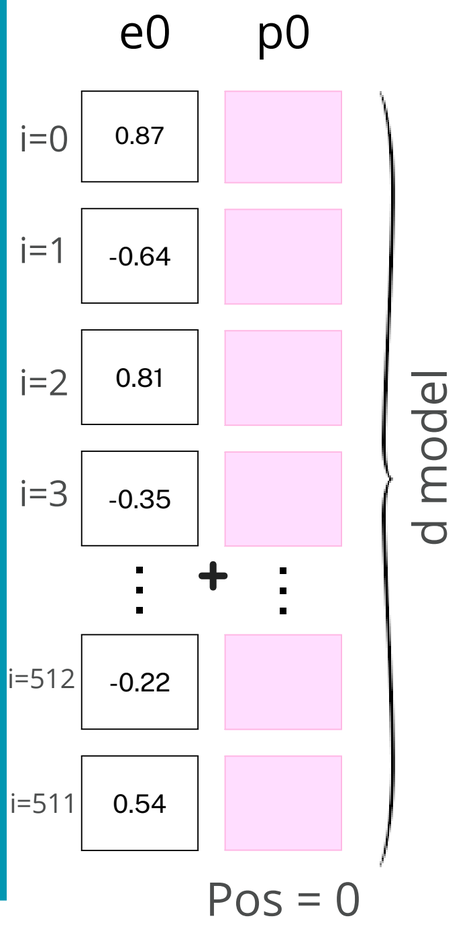
Encoder architecture

Positional Encoding (Wave frequencies)

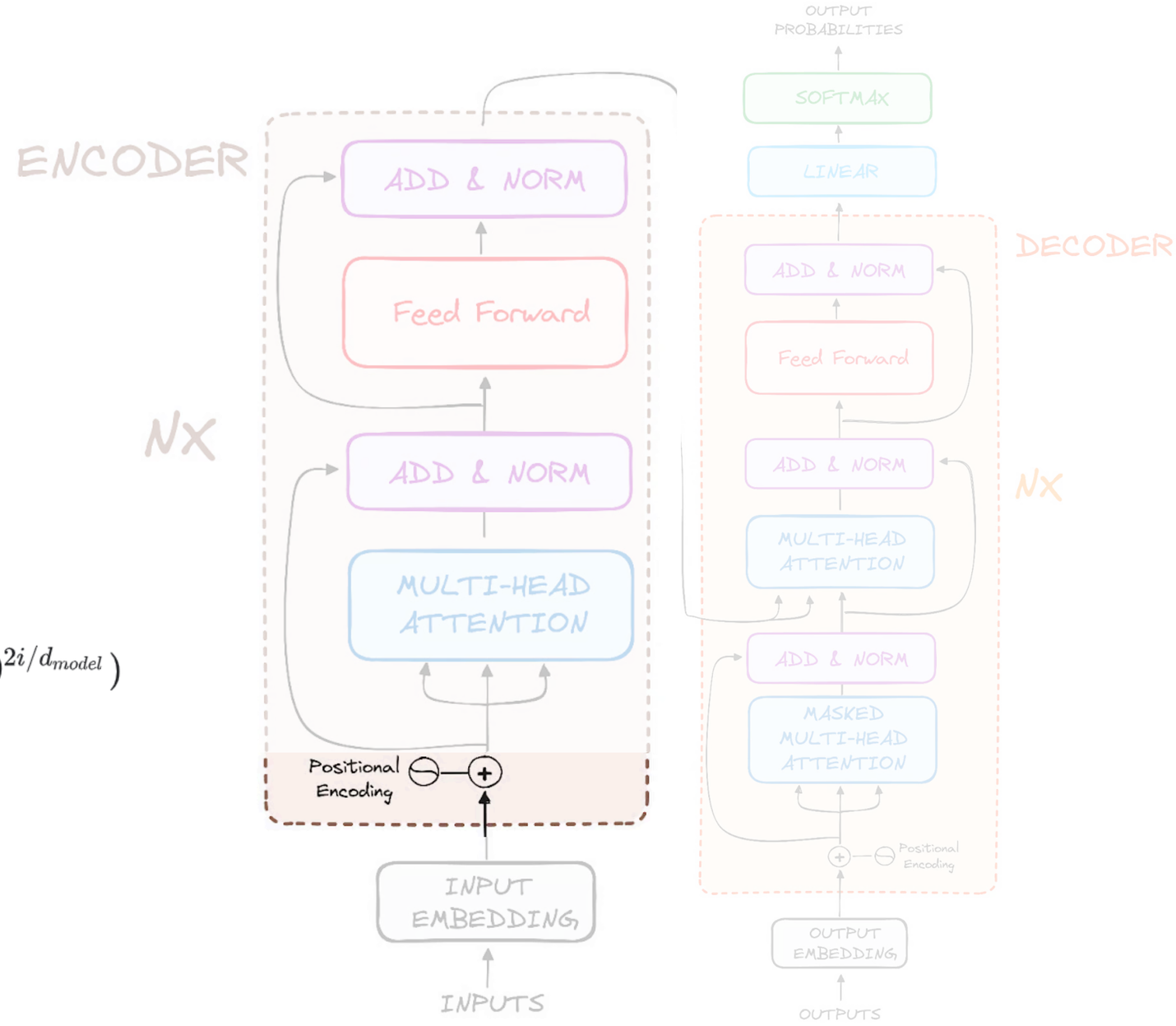


Encoder architecture

Positional Encoding (Wave frequencies)

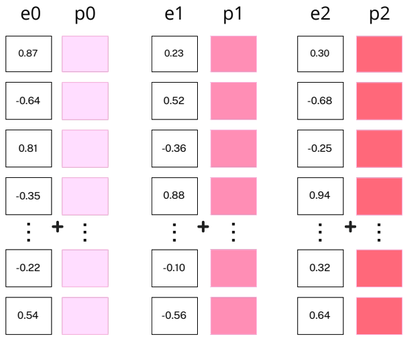


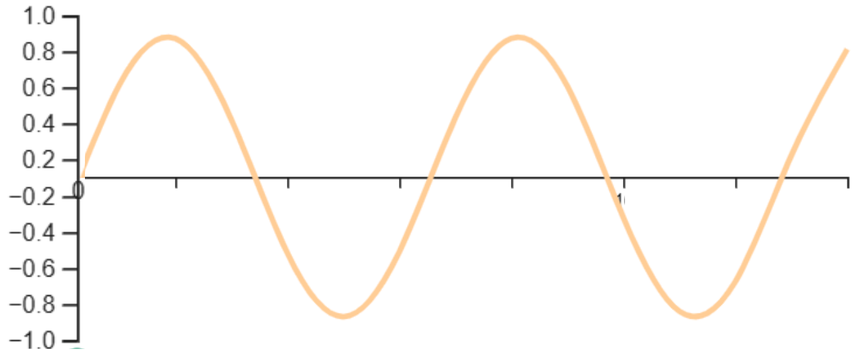
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



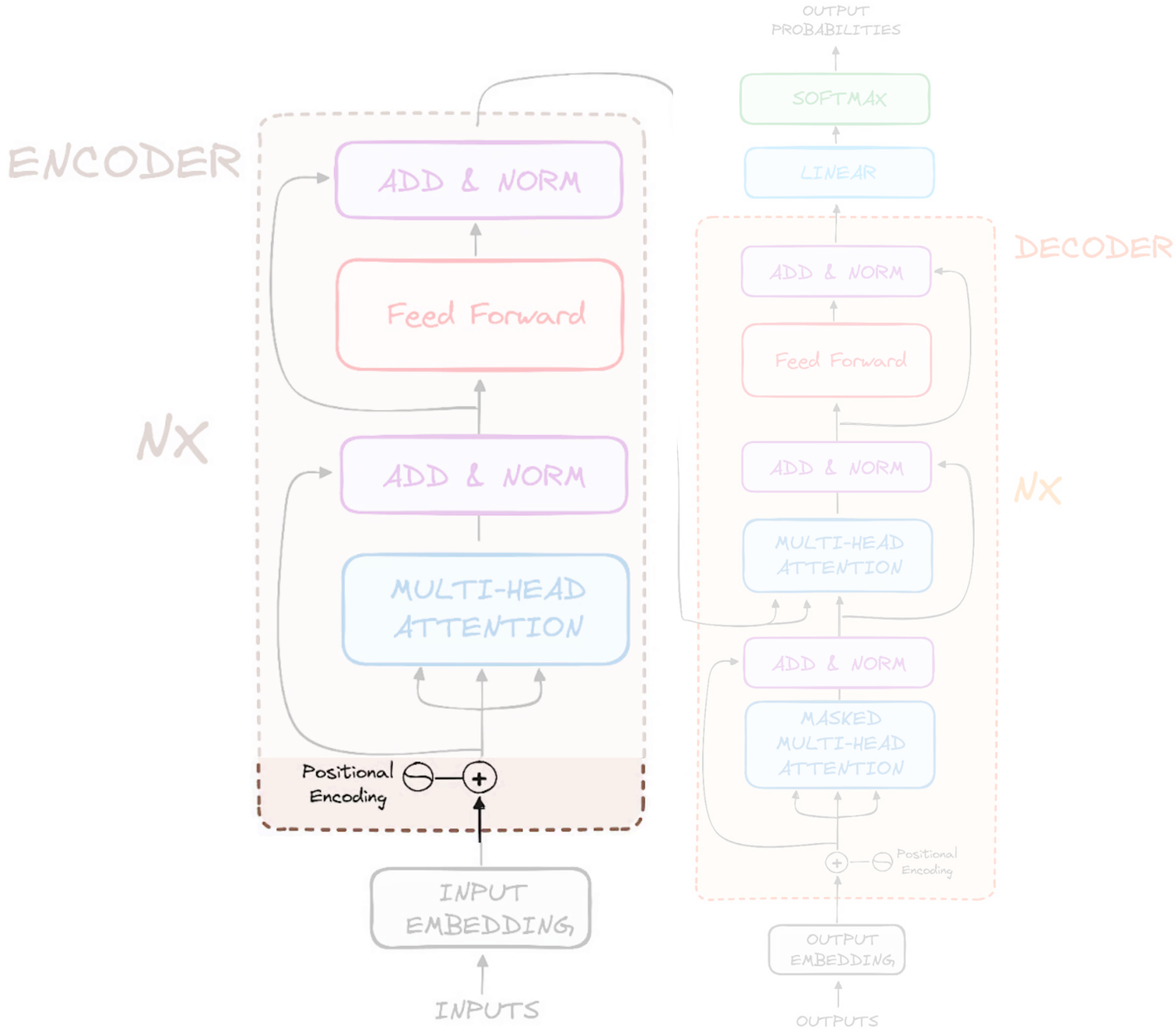
Encoder architecture

Positional Encoding (Wave frequencies)



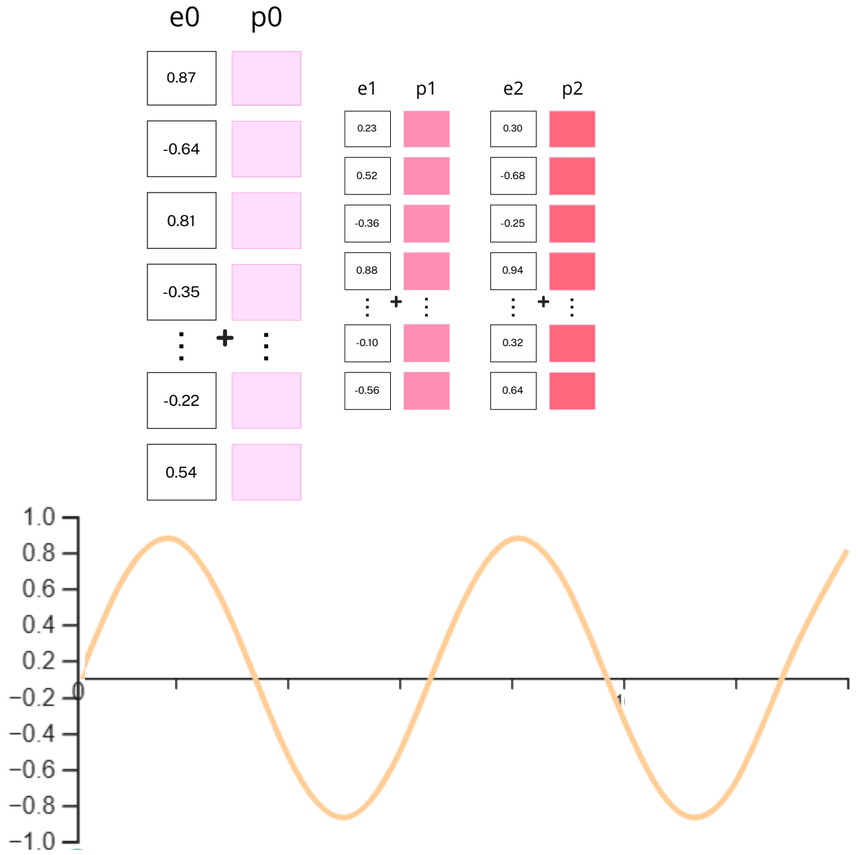


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

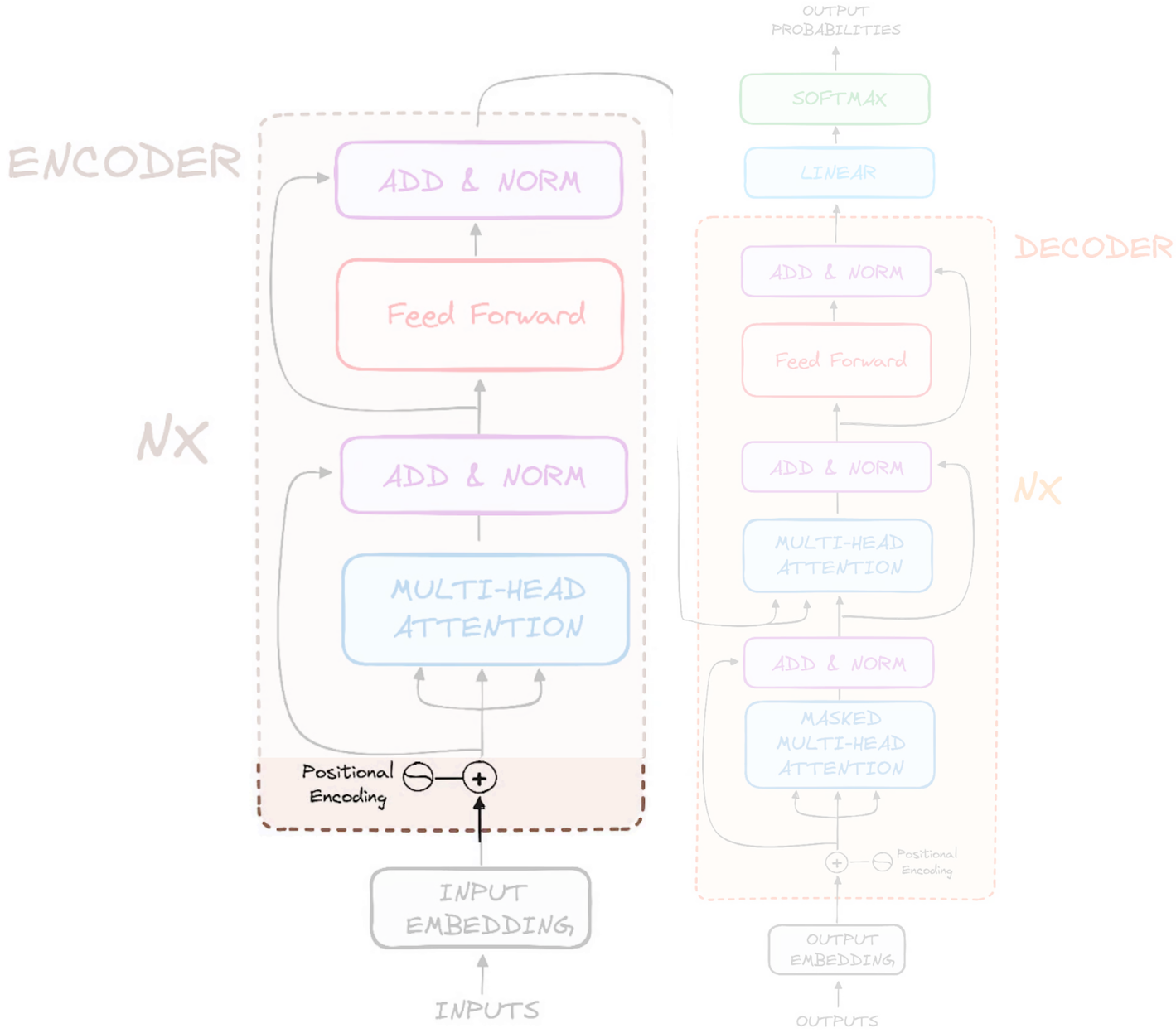


Encoder architecture

Positional Encoding (Wave frequencies)

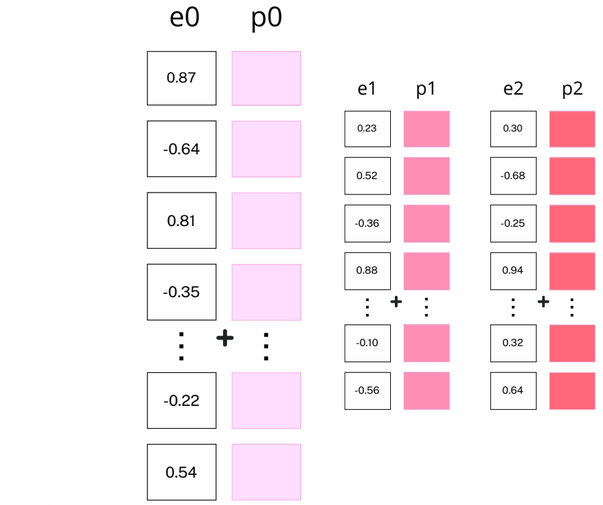


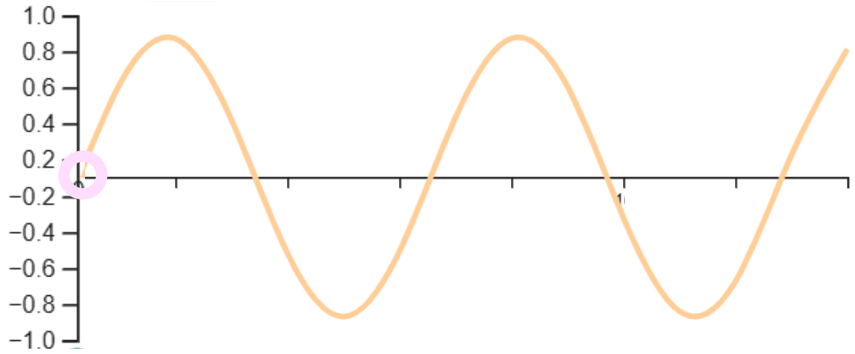
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



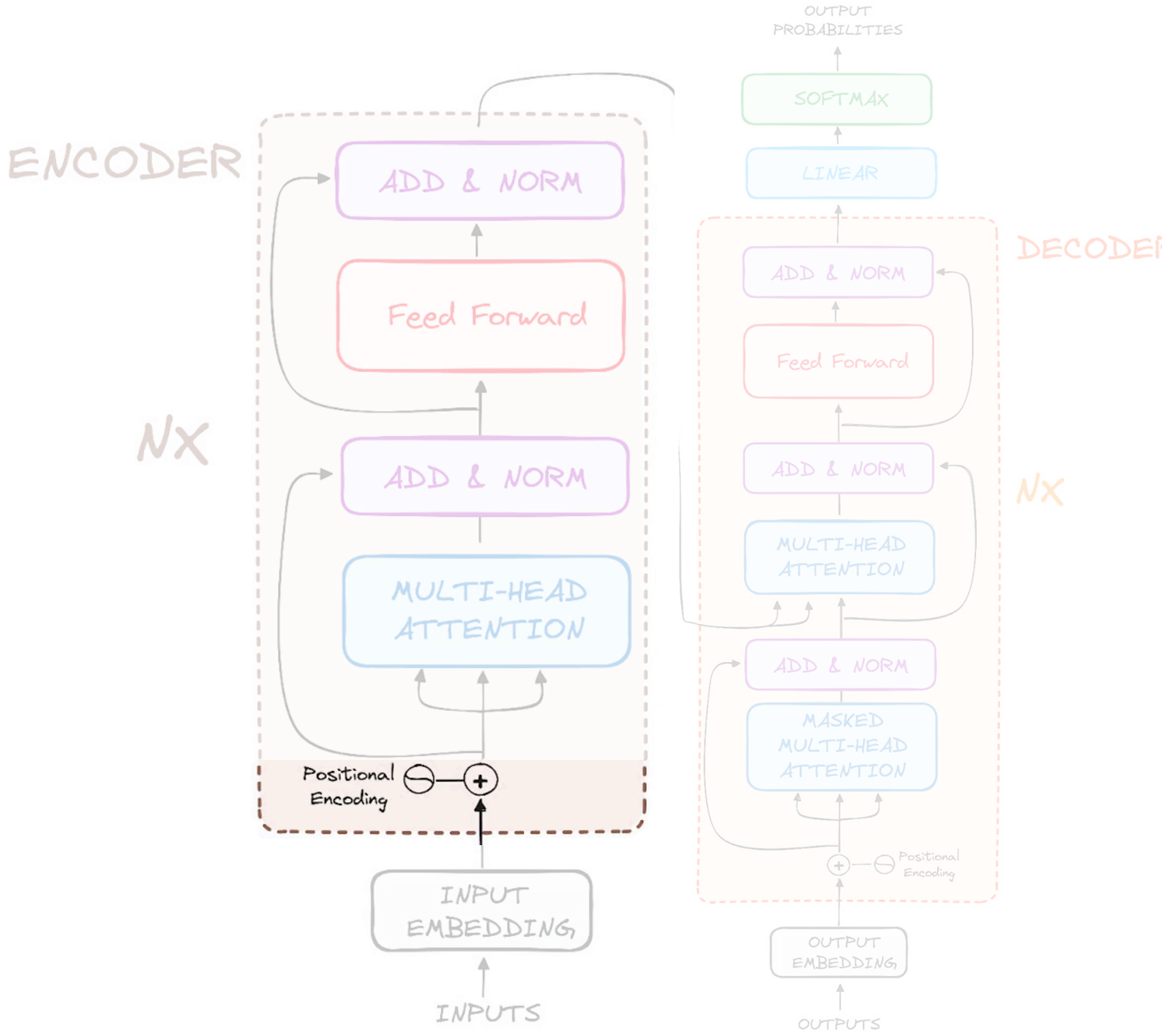
Encoder architecture

Positional Encoding (Wave frequencies)



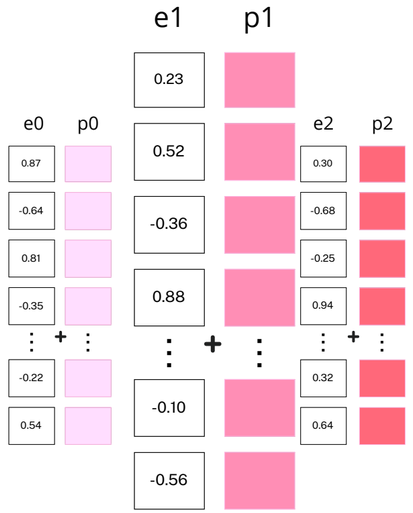


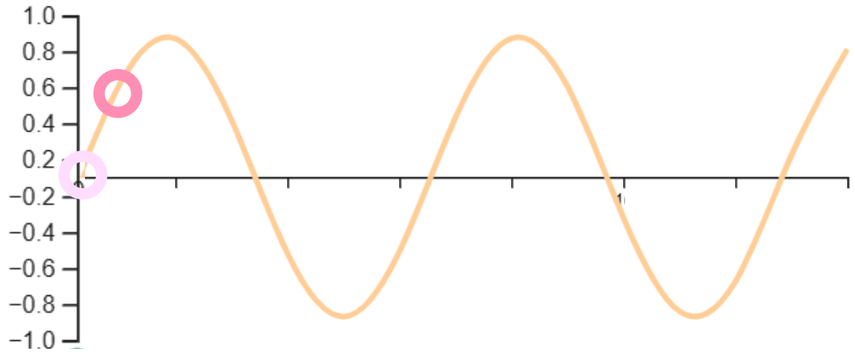
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



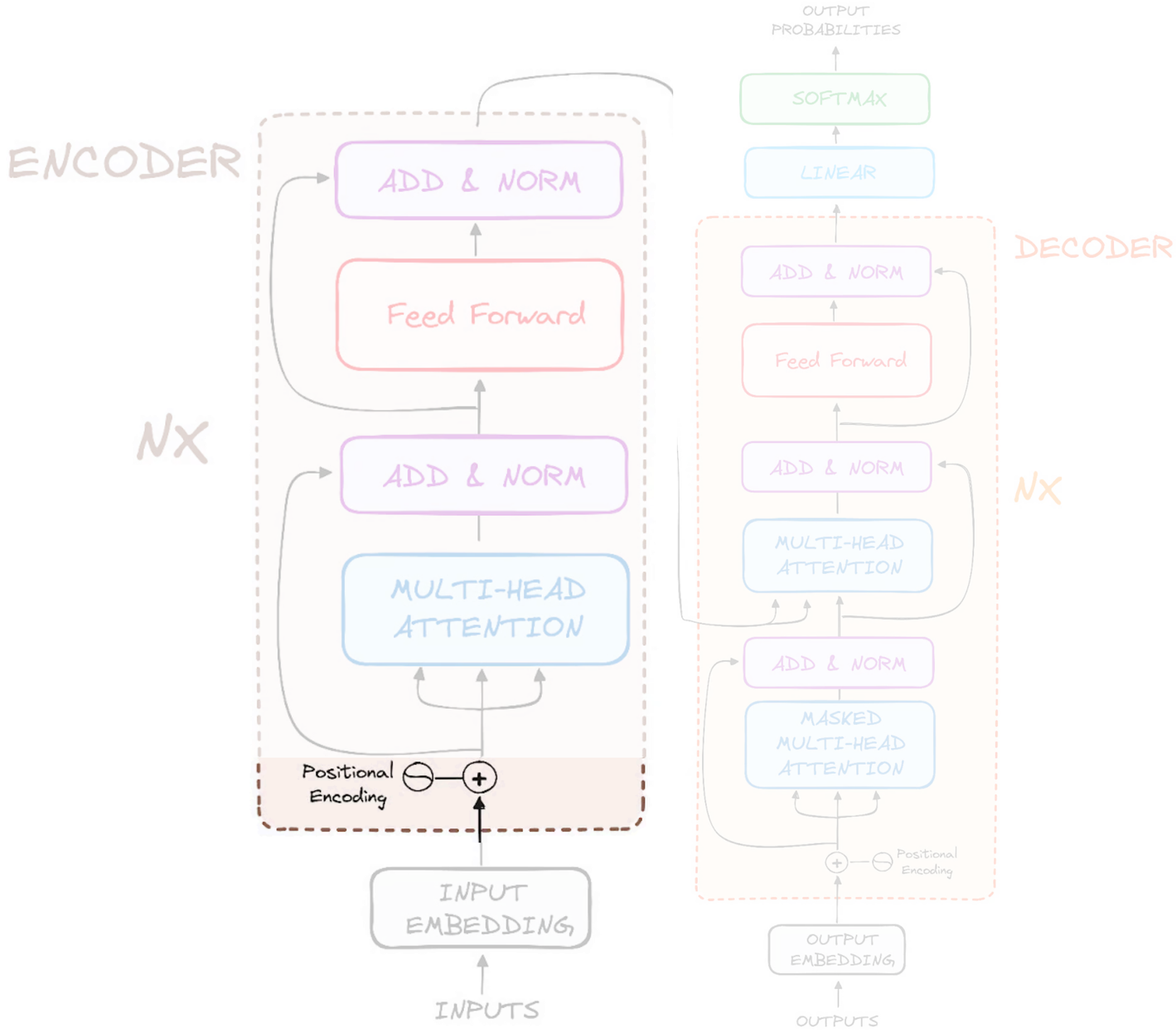
Encoder architecture

Positional Encoding (Wave frequencies)



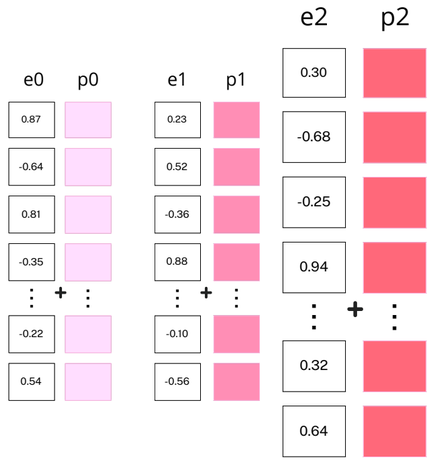


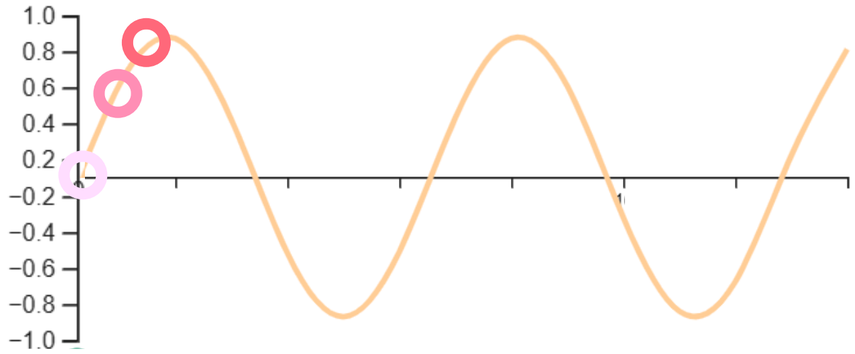
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

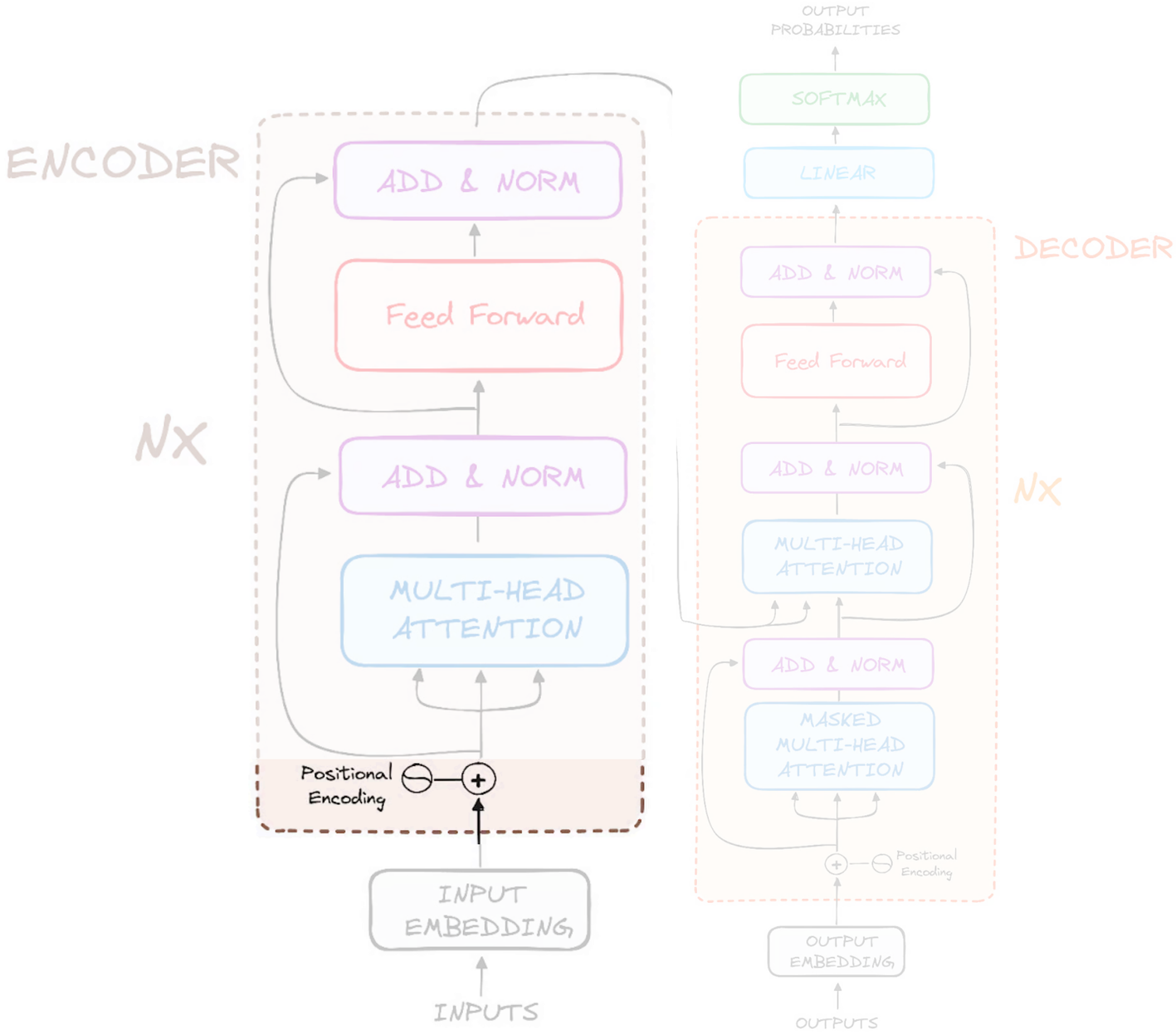


Encoder architecture

Positional Encoding (Wave frequencies)



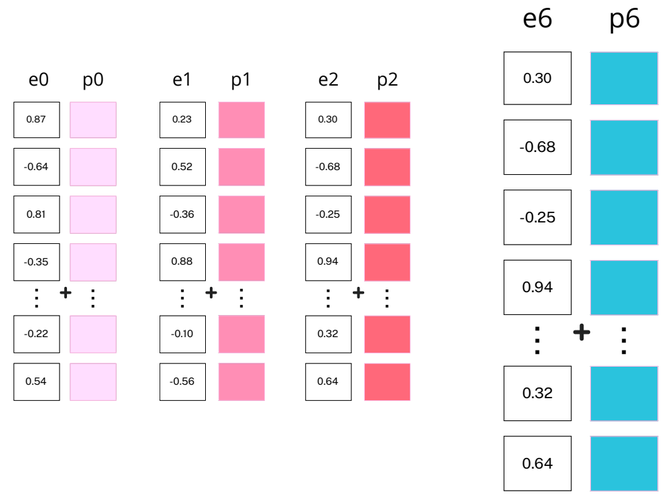


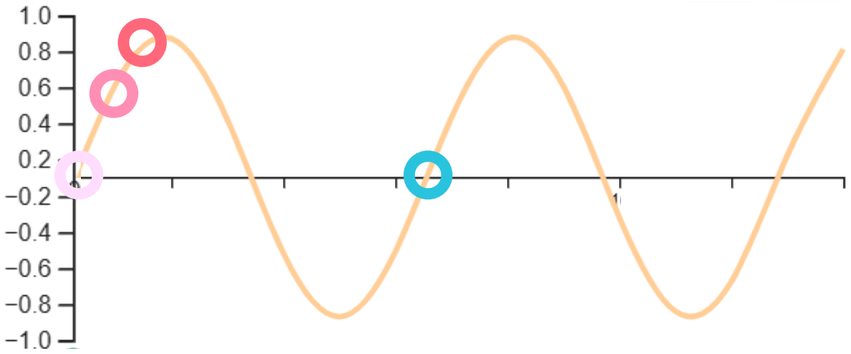


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

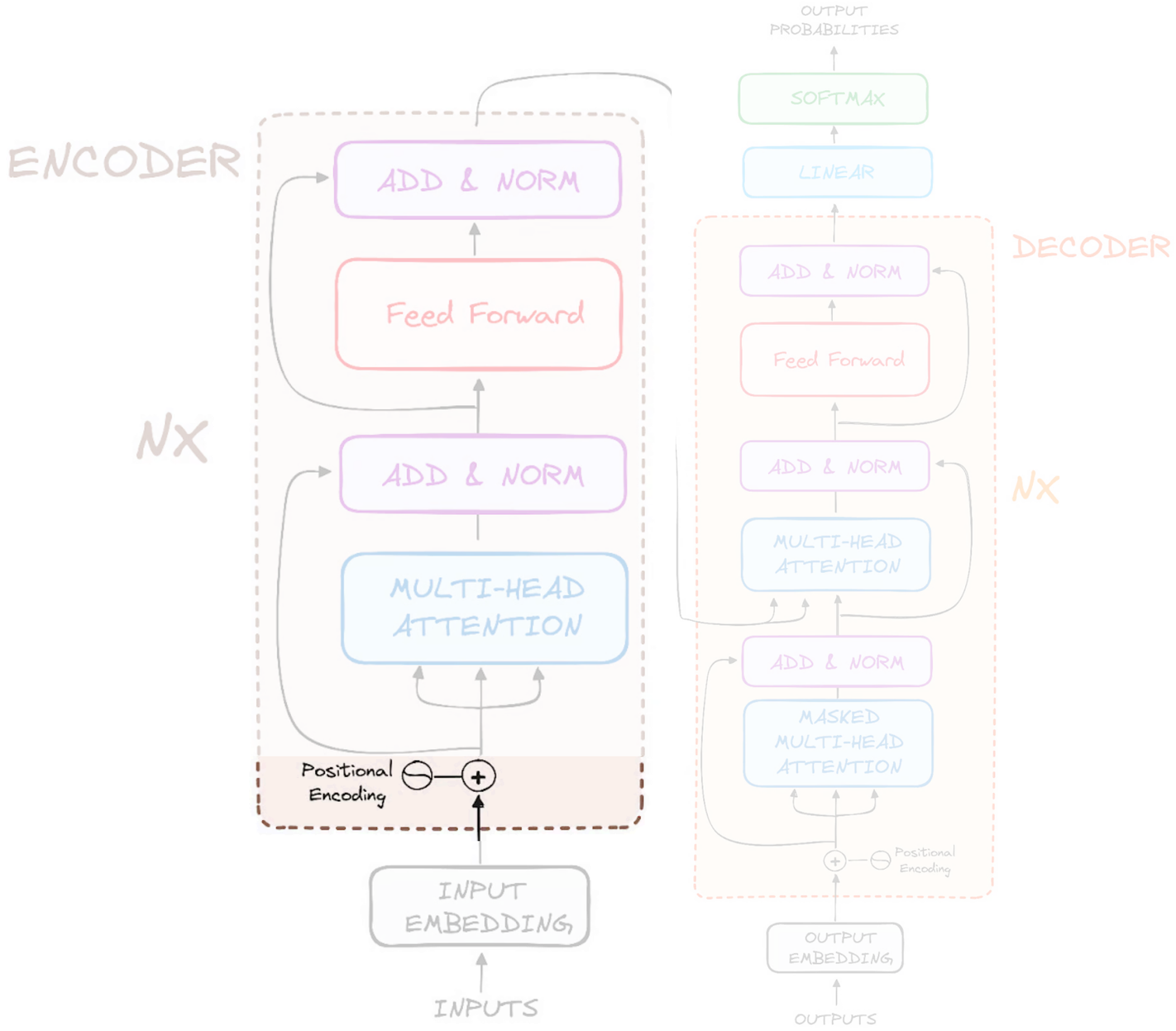
Encoder architecture

Positional Encoding (Wave frequencies)



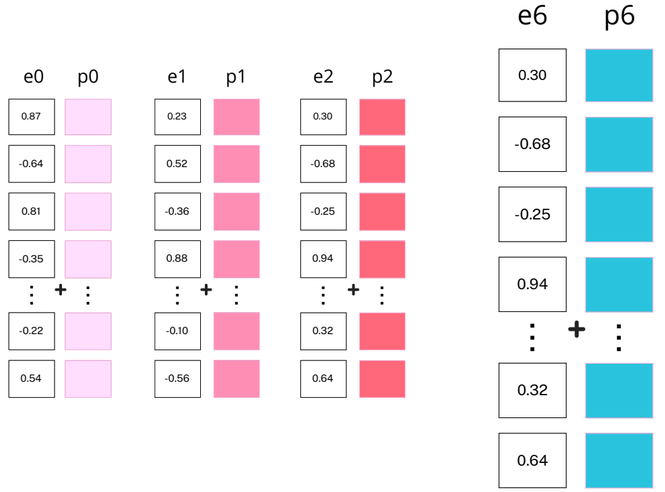


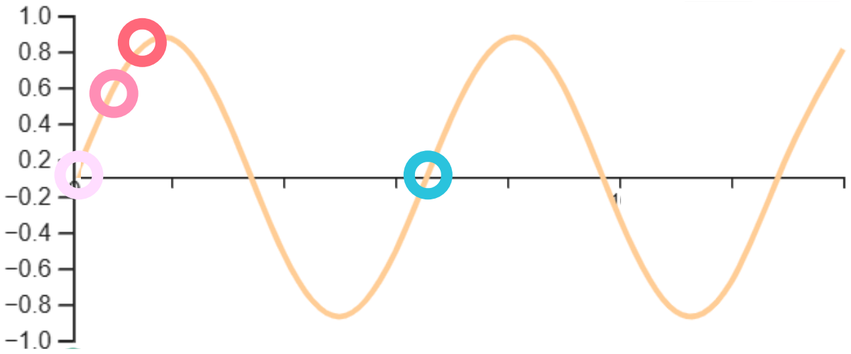
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



Encoder architecture

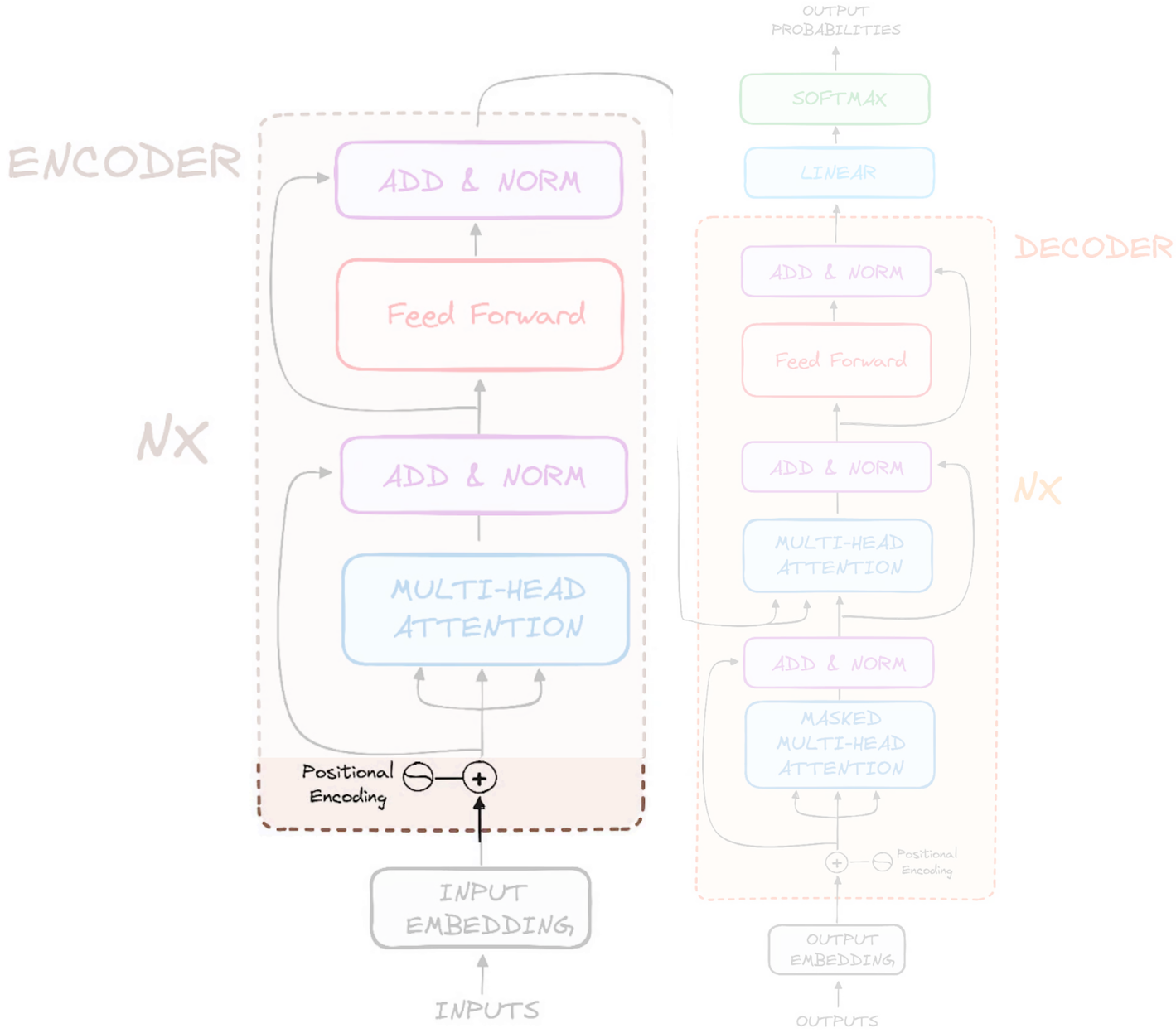
Positional Encoding (Wave frequencies)





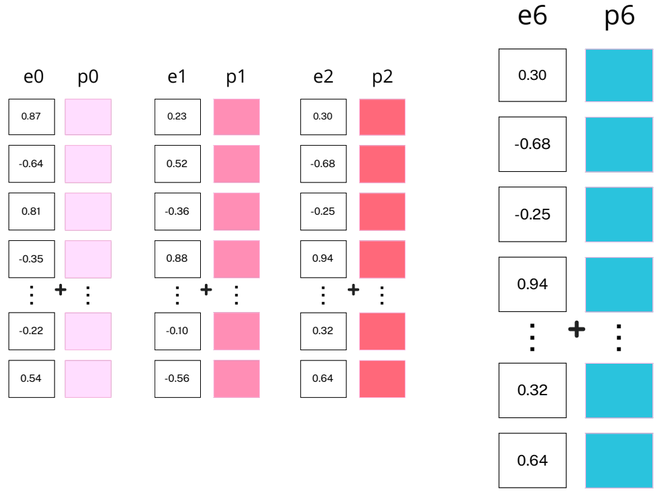
Same position for pos 1 and 6 !

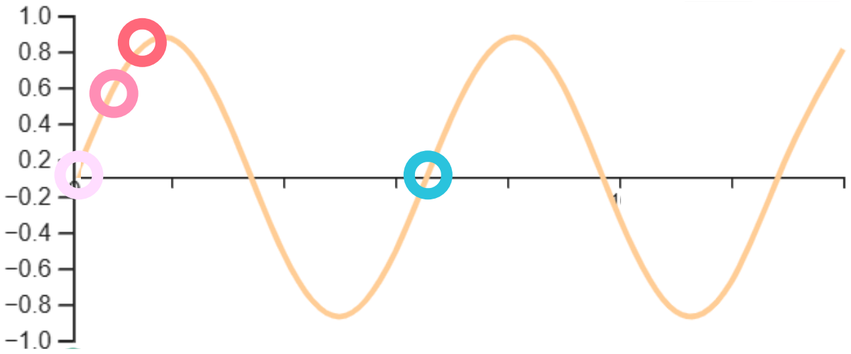
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



Encoder architecture

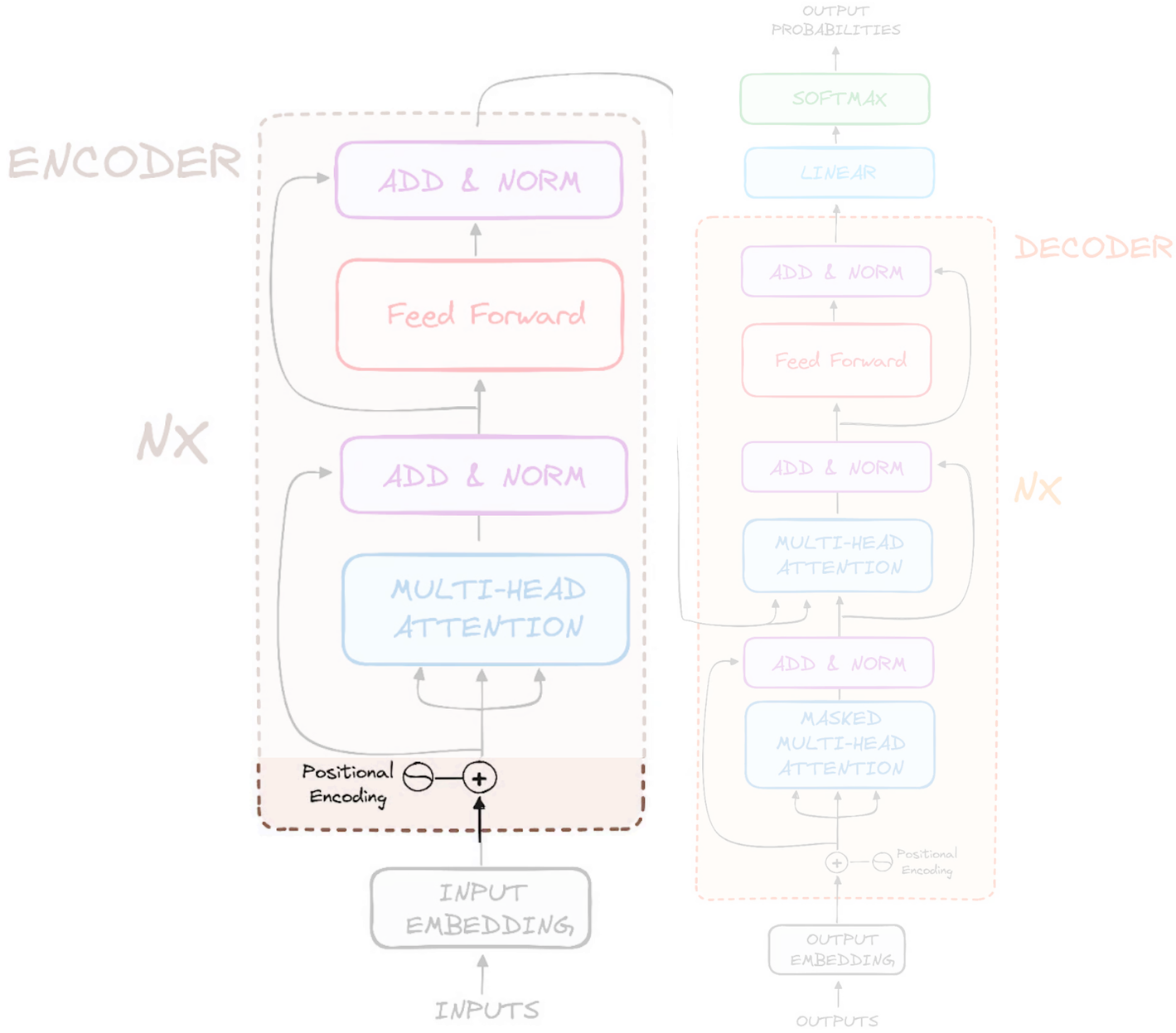
Positional Encoding (Wave frequencies)





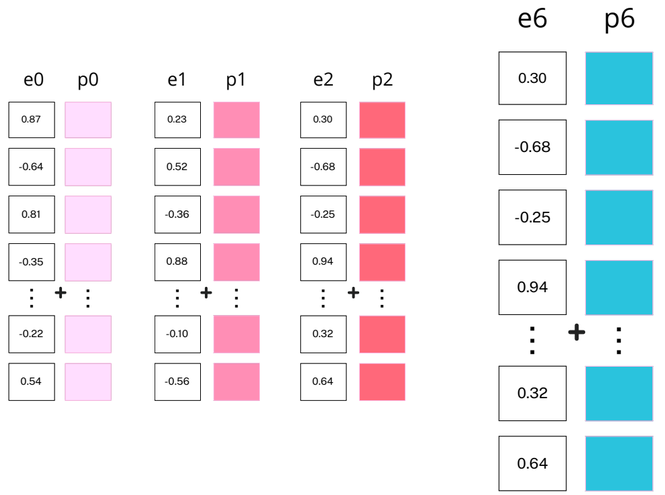
Same position for pos 1 and 6 !

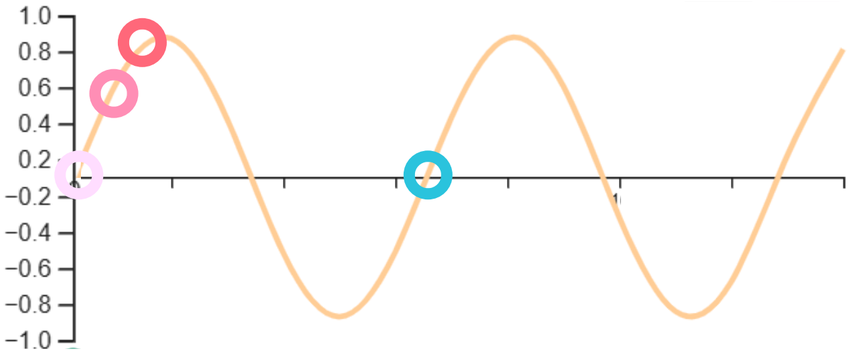
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



Encoder architecture

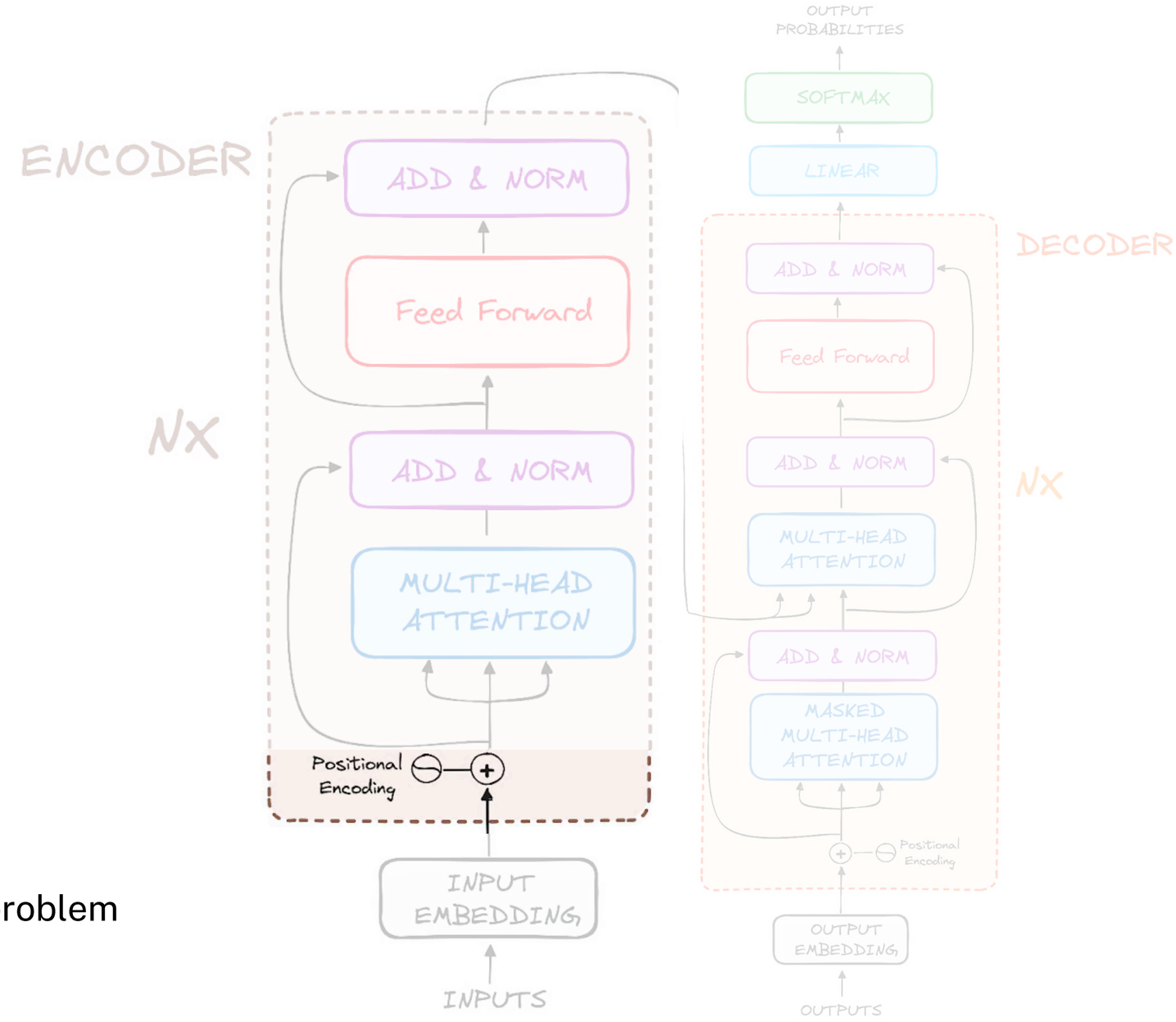
Positional Encoding (Wave frequencies)





But the i frequency controller will solve the problem

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



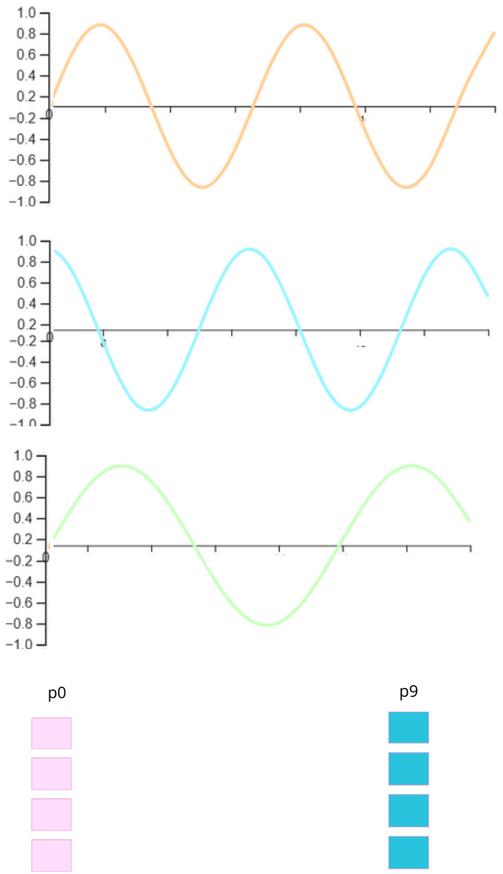
Encoder architecture

Positional Encoding (Wave frequencies)

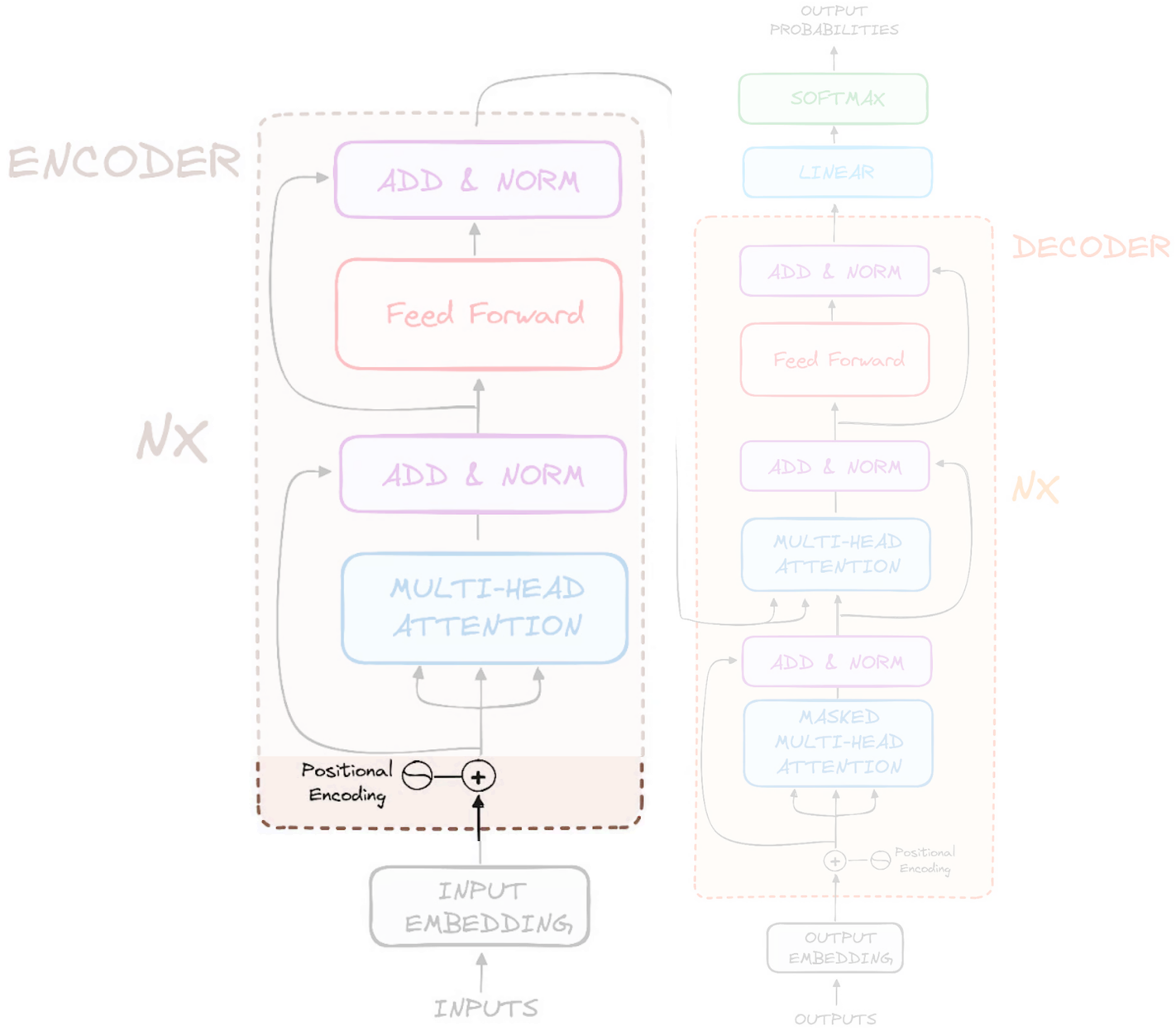
i = 0

i = 1

i = 2



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$



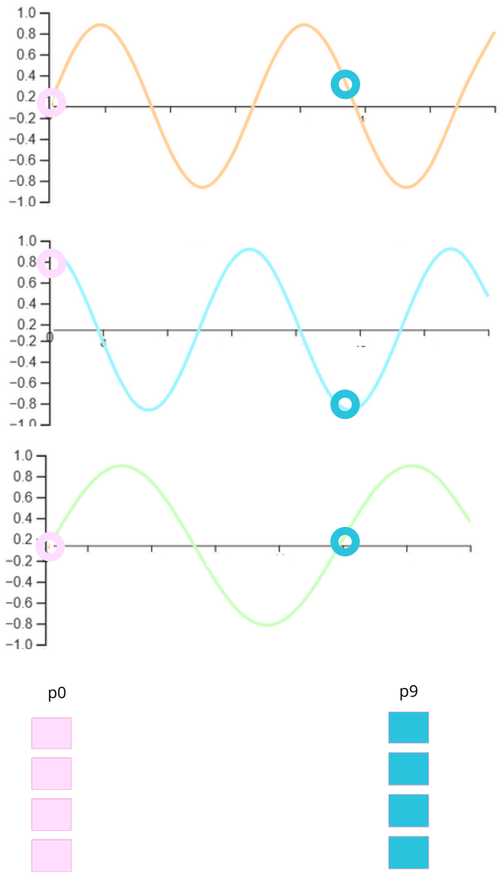
Encoder architecture

Positional Encoding (Wave frequencies)

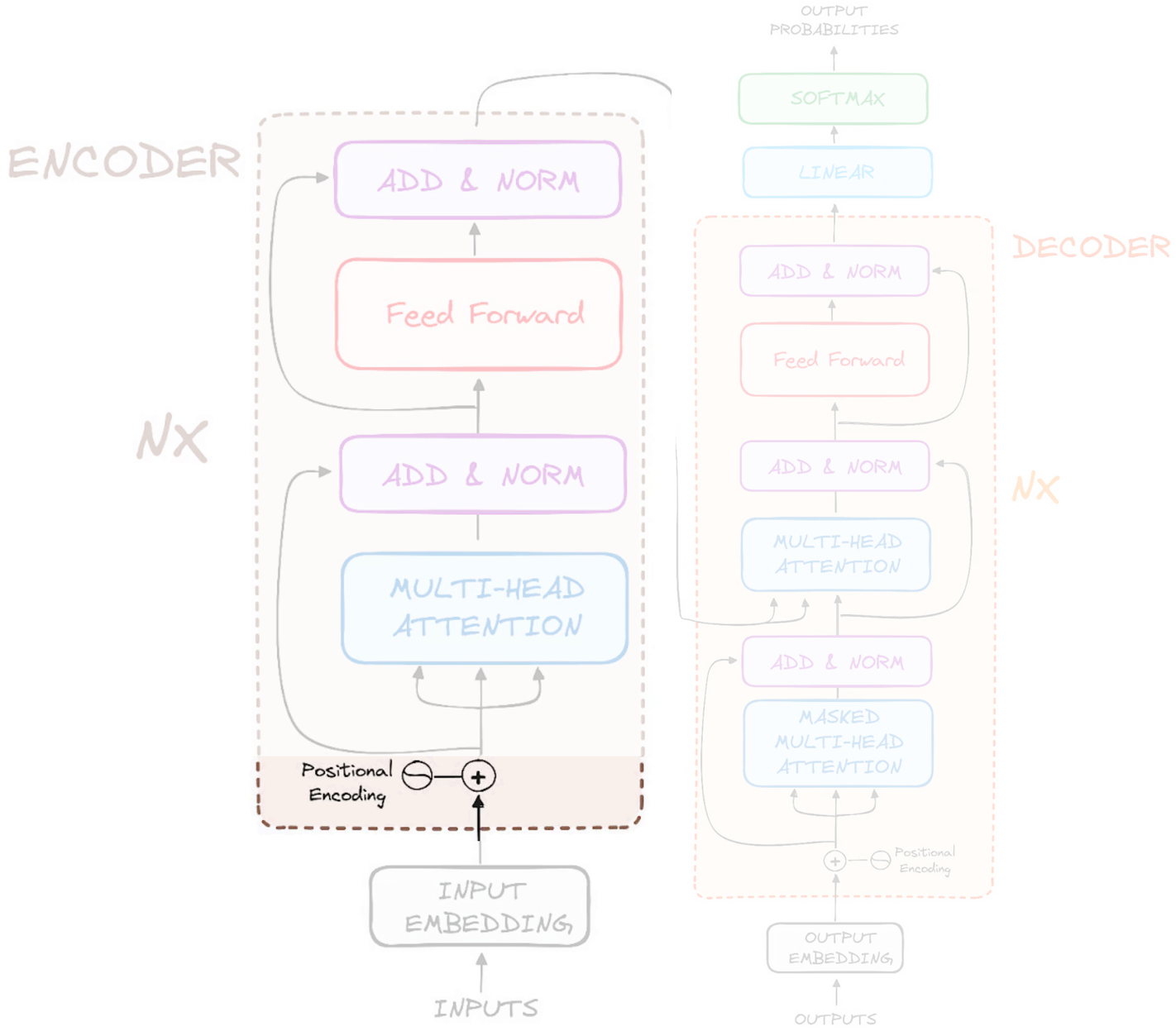
i = 0

i = 1

i = 2



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

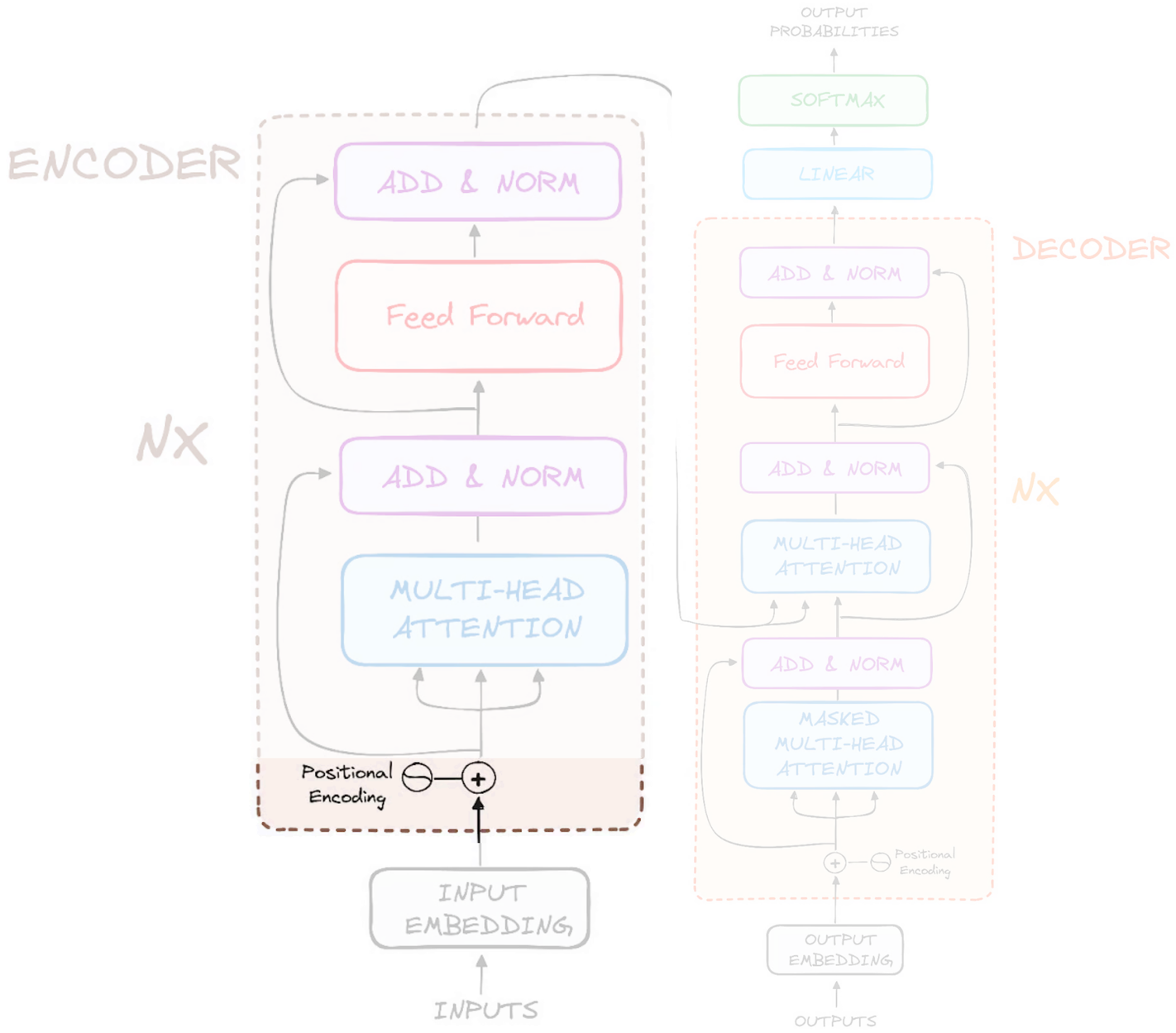


Encoder architecture

Positional Encoding (Wave frequencies)

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

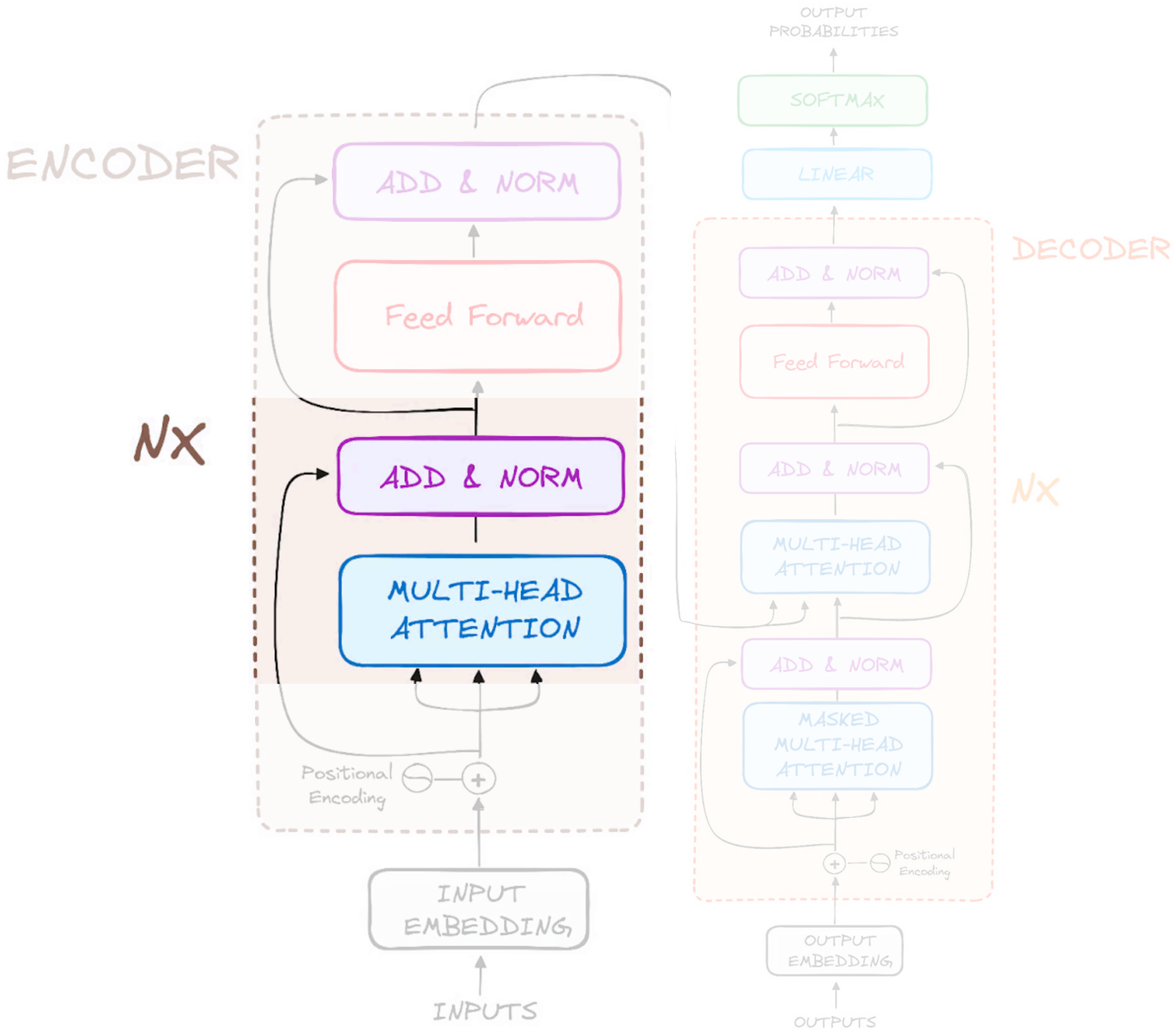
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$



Encoder architecture

Multi-headed self-attention mechanism

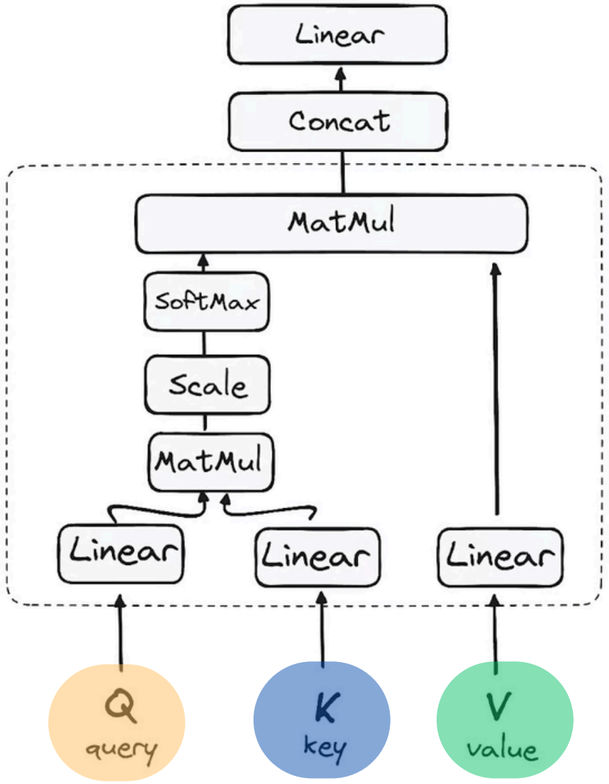
- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match

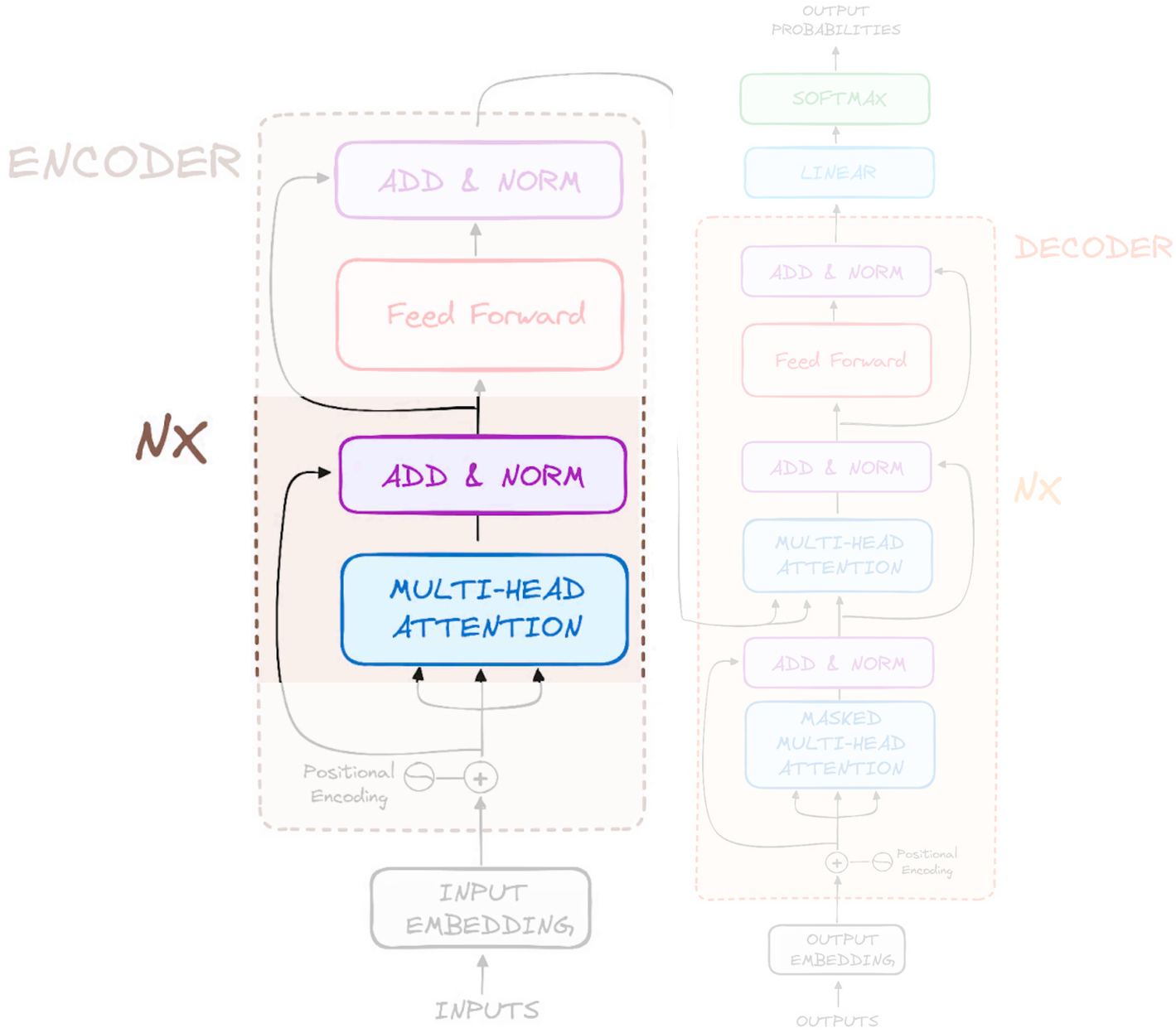


Encoder architecture

Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match

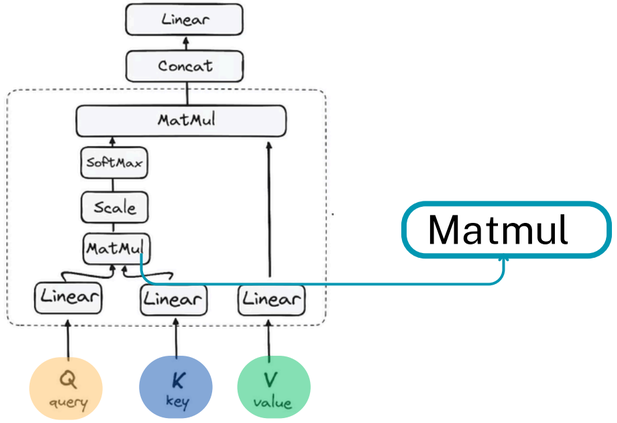


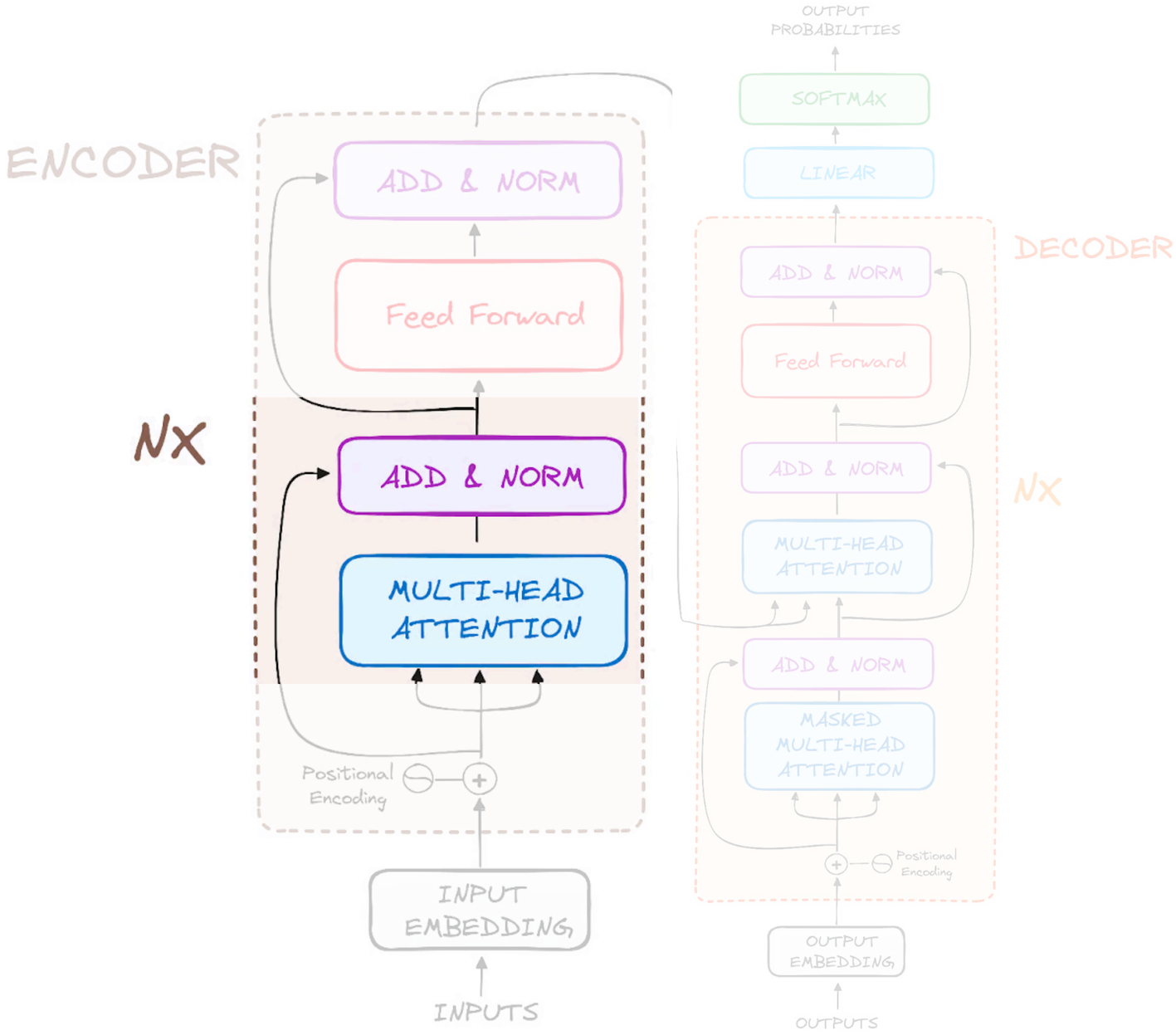


Encoder architecture

Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match

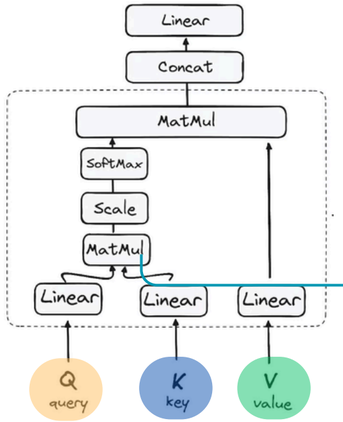




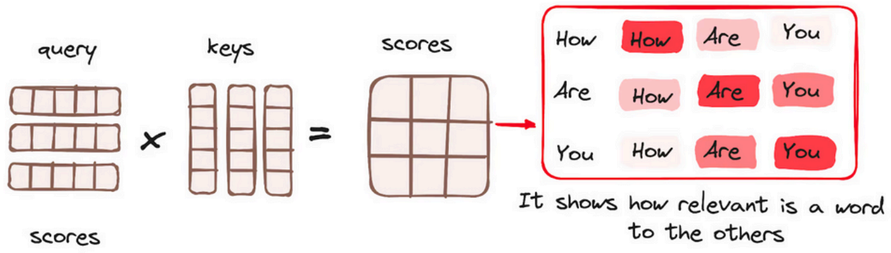
Encoder architecture

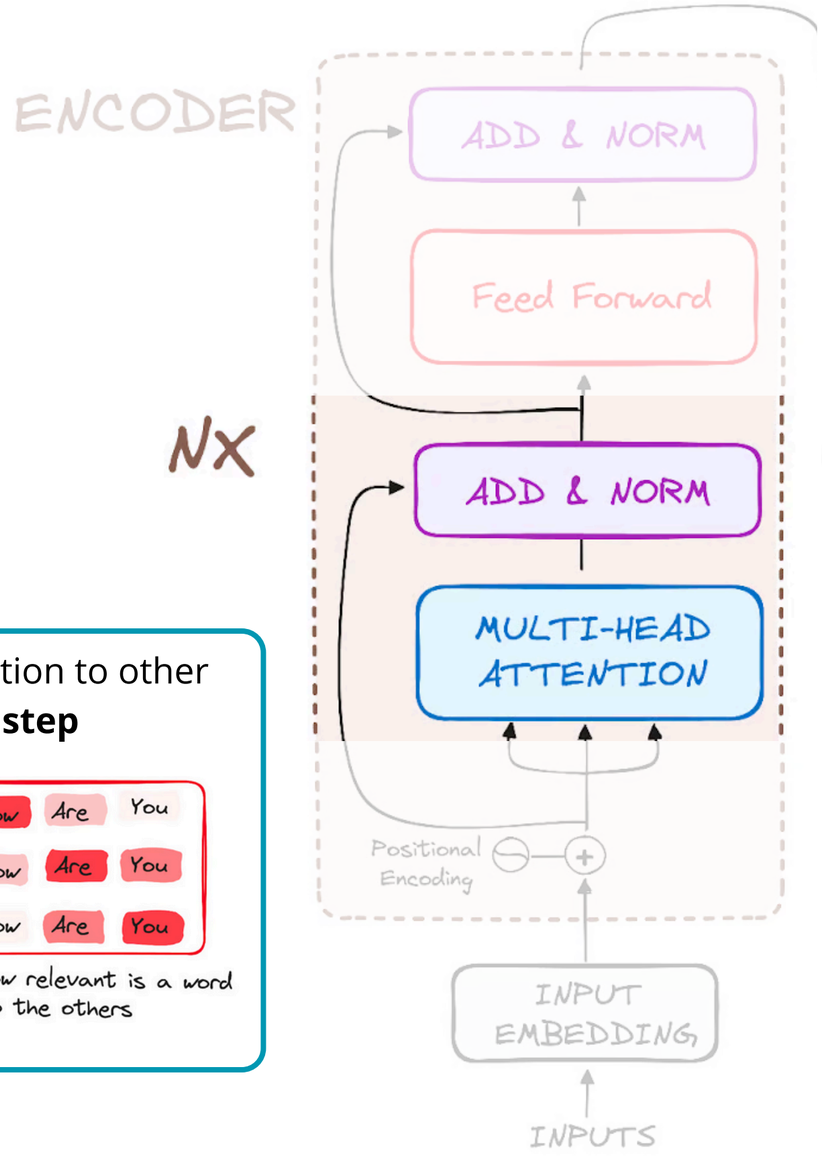
Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match



Each word is assigned a score in relation to other words within the same **time step**

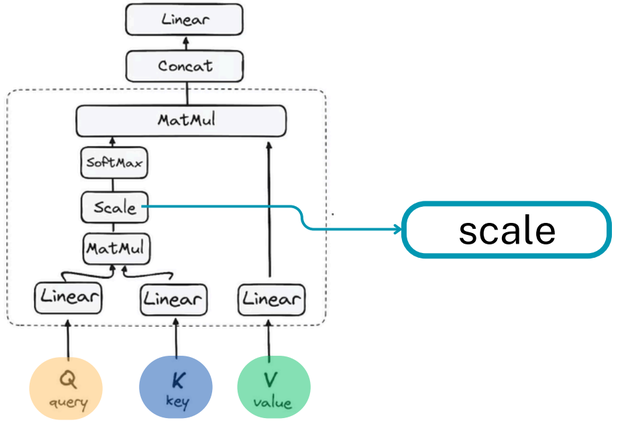


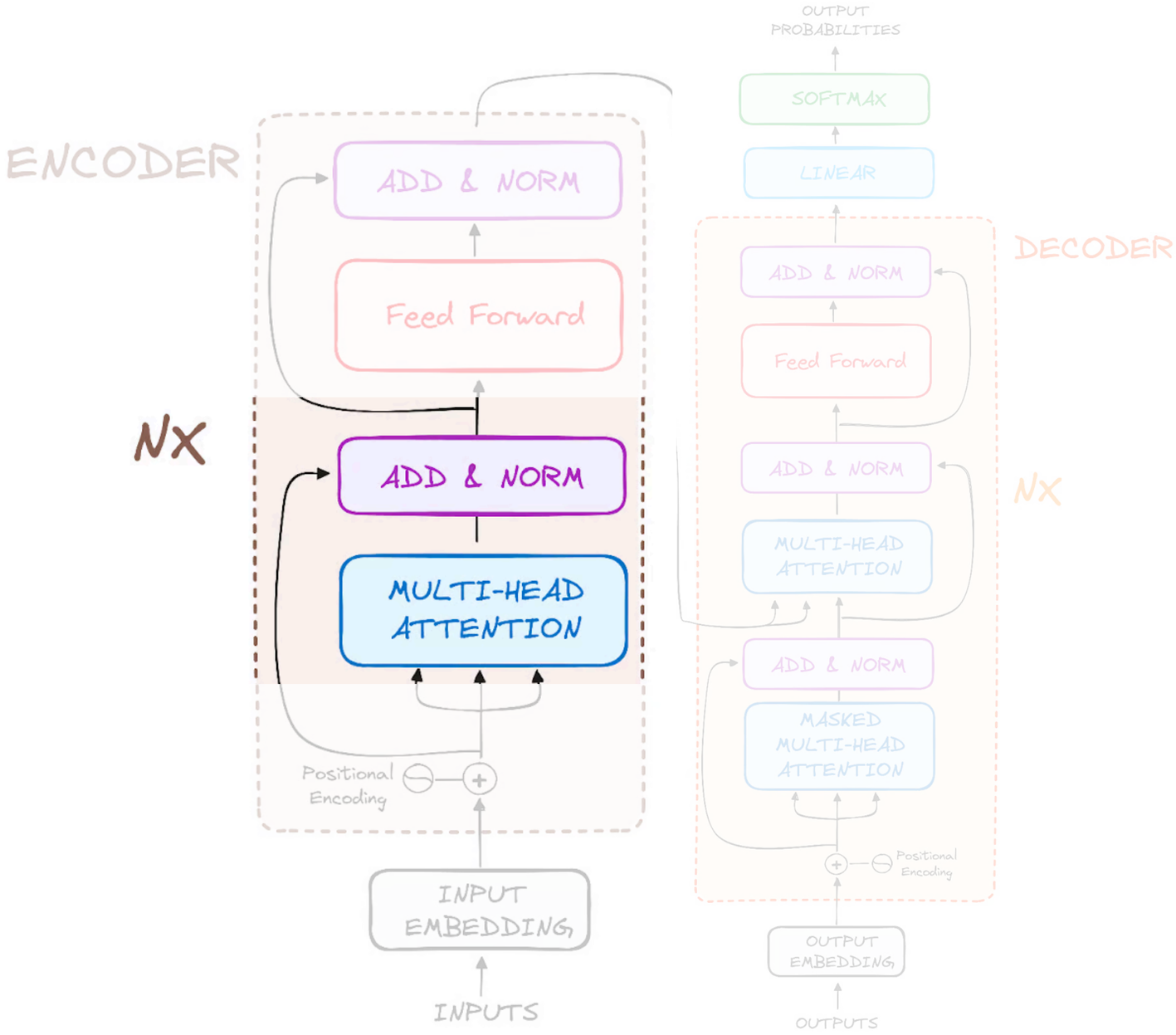


Encoder architecture

Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match

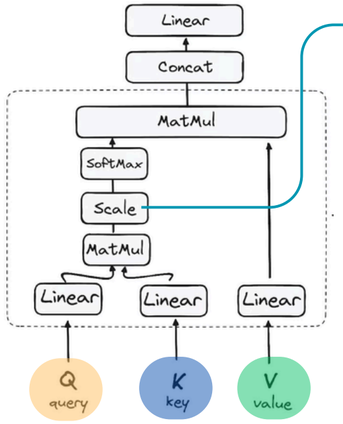




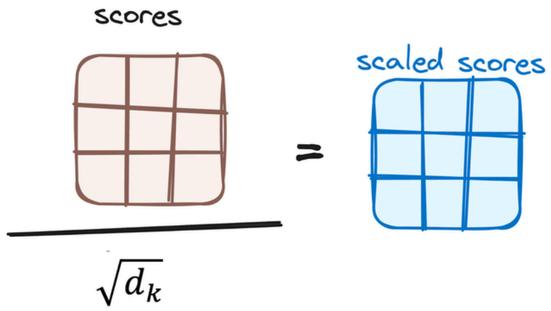
Encoder architecture

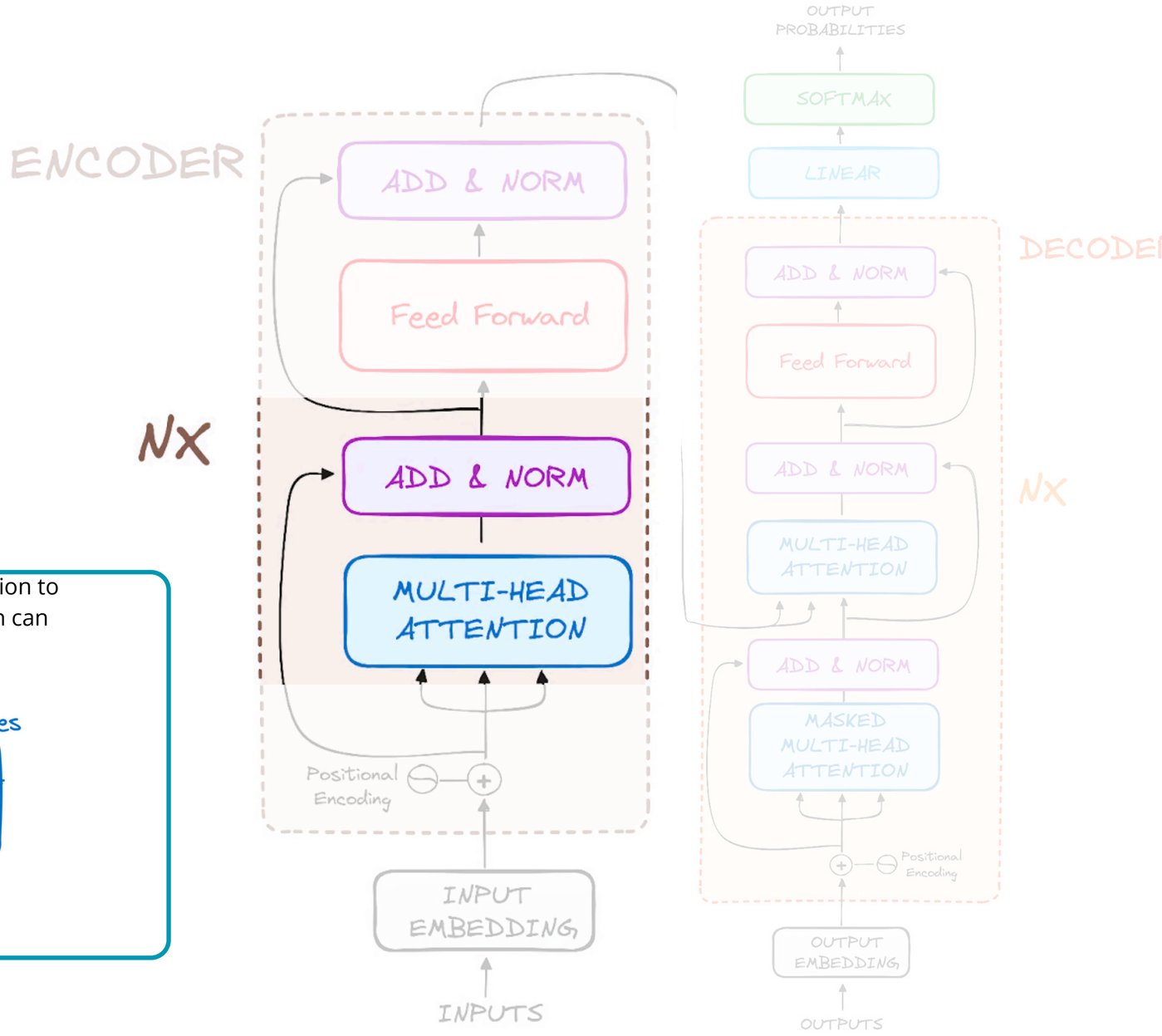
Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match



Scaling down by dividing by sqrt of dimension to have more stable gradients (multiplication can lead to very large outputs)

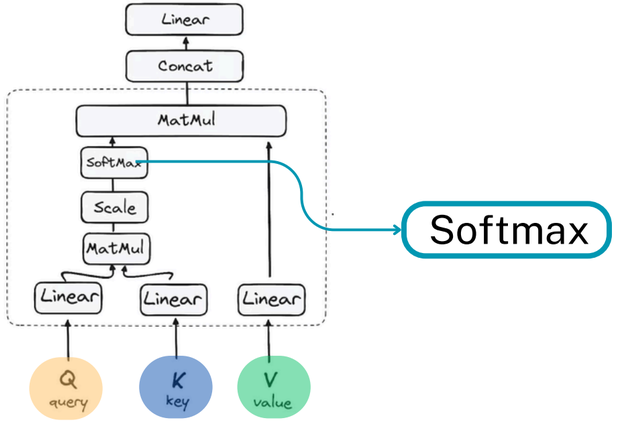


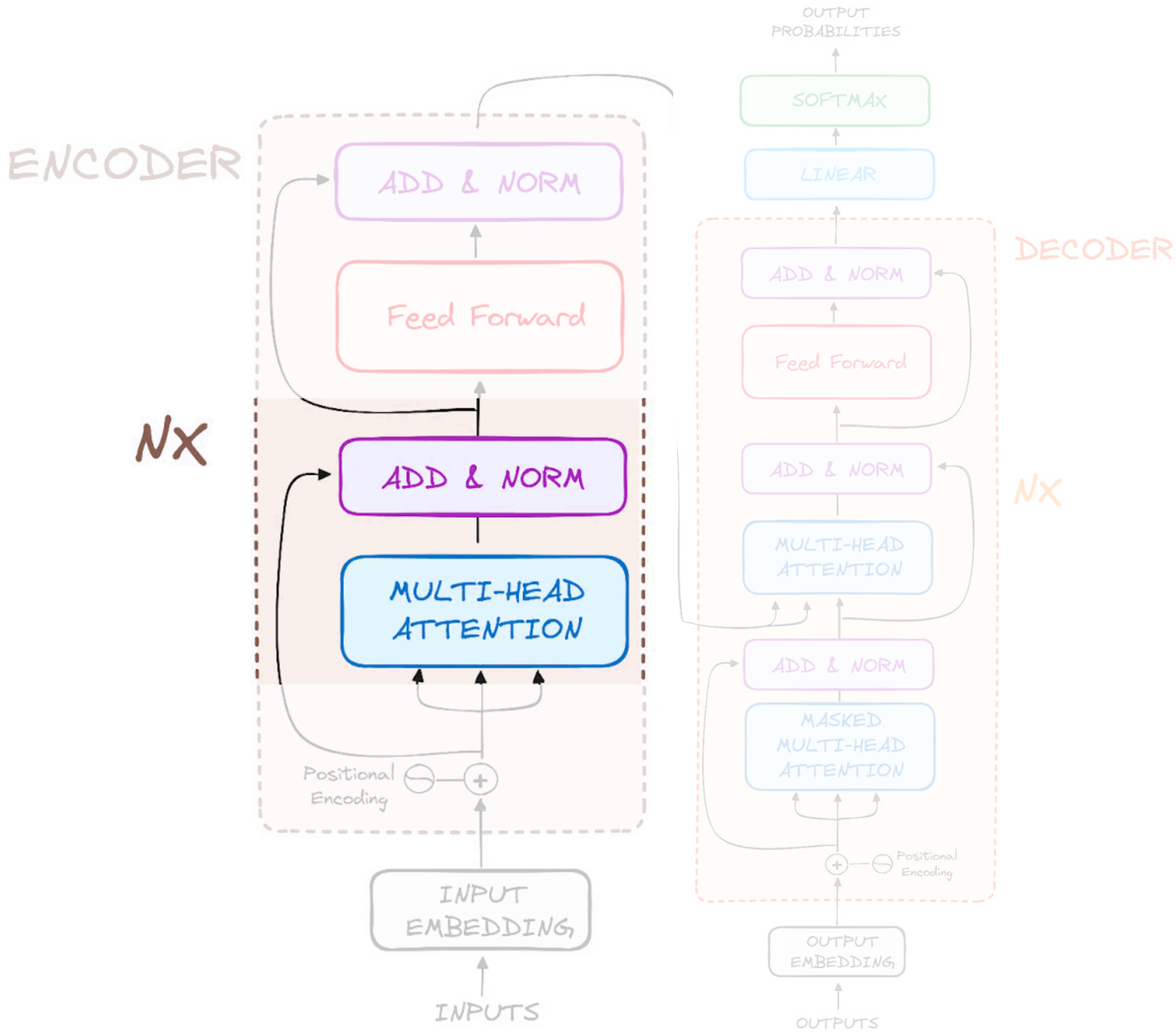


Encoder architecture

Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match

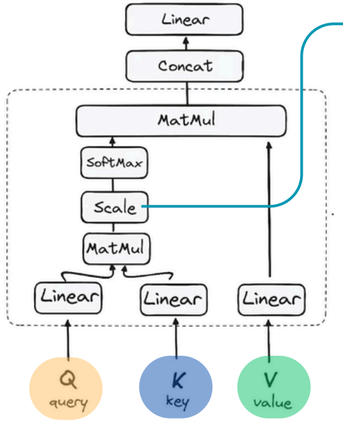




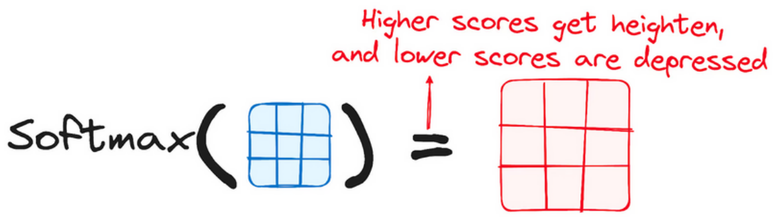
Encoder architecture

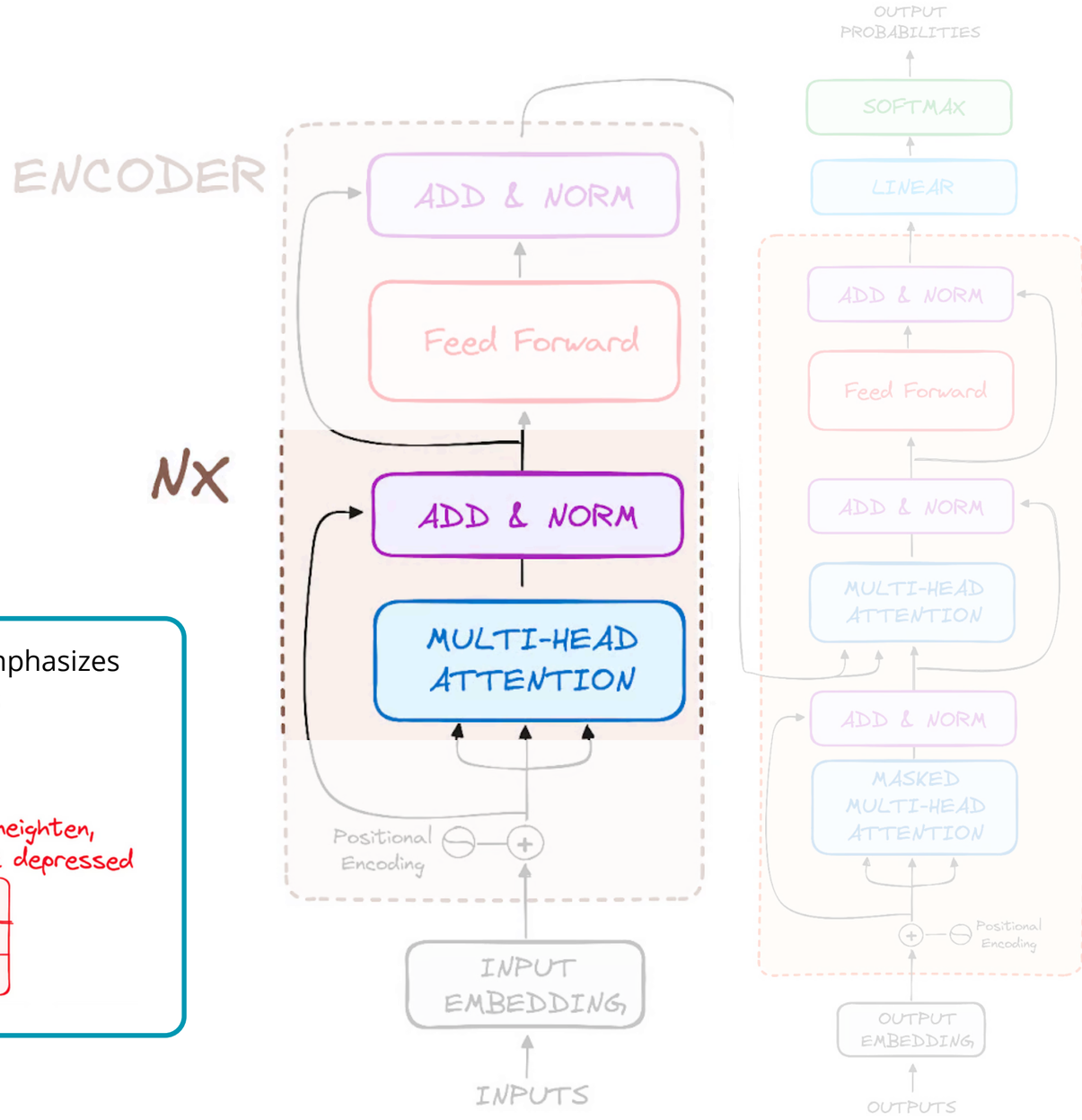
Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match



Probability values ranging from 0 to 1emphasizes higher scores and lower scores

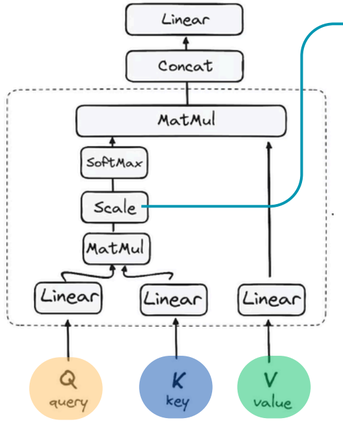




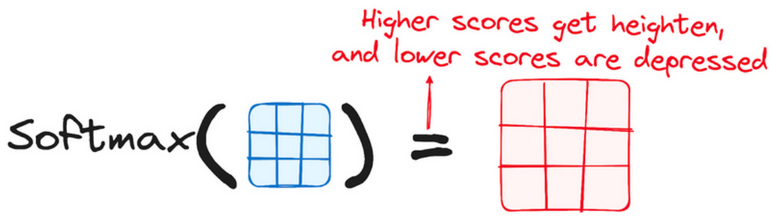
Encoder architecture

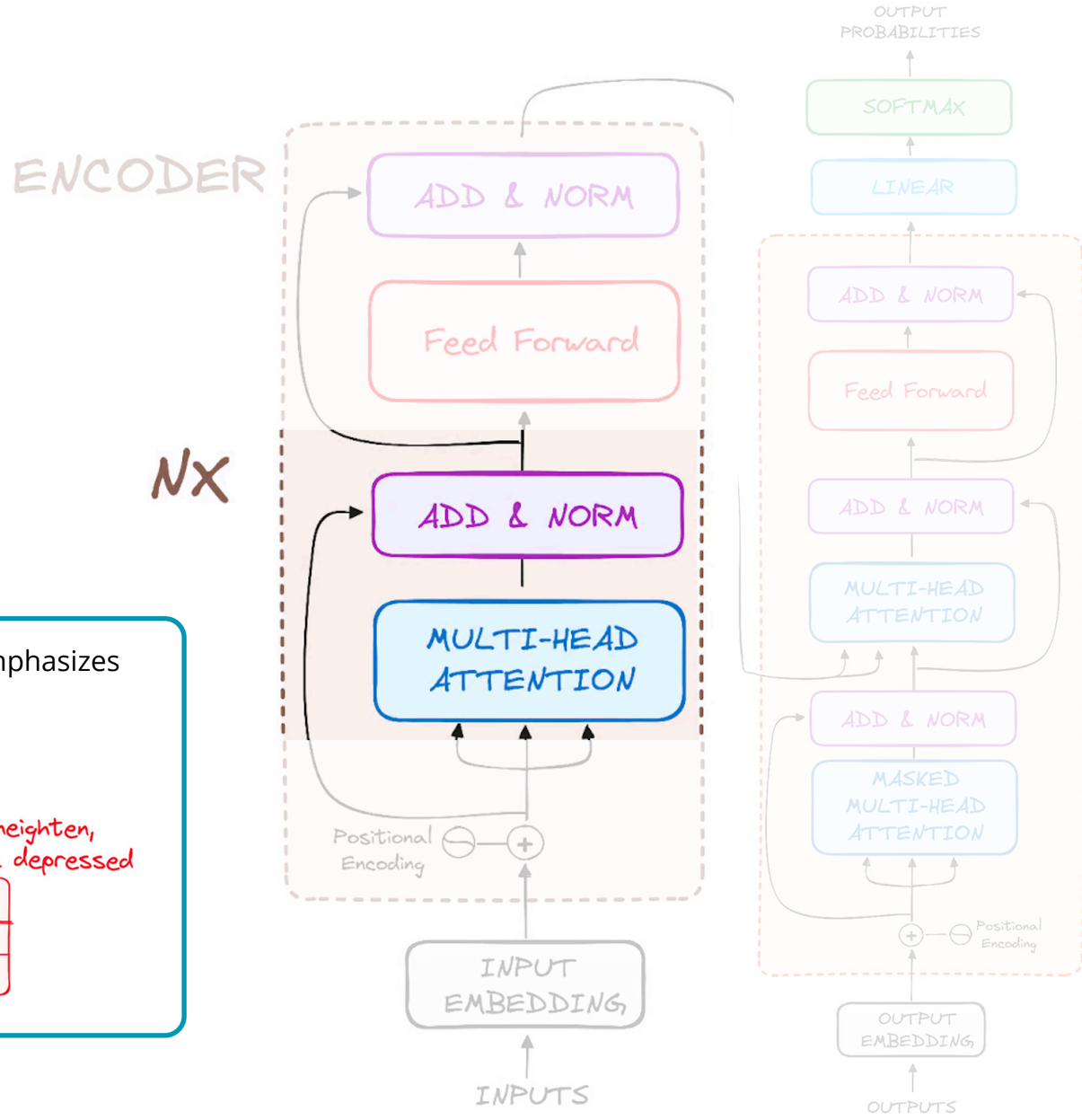
Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match



Probability values ranging from 0 to 1emphasizes higher scores and lower scores

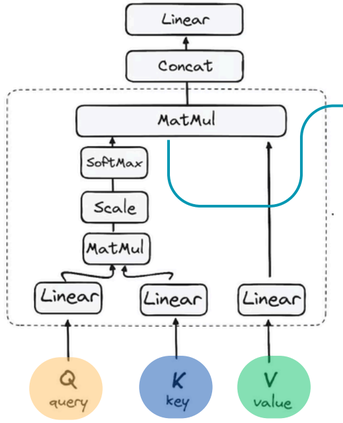




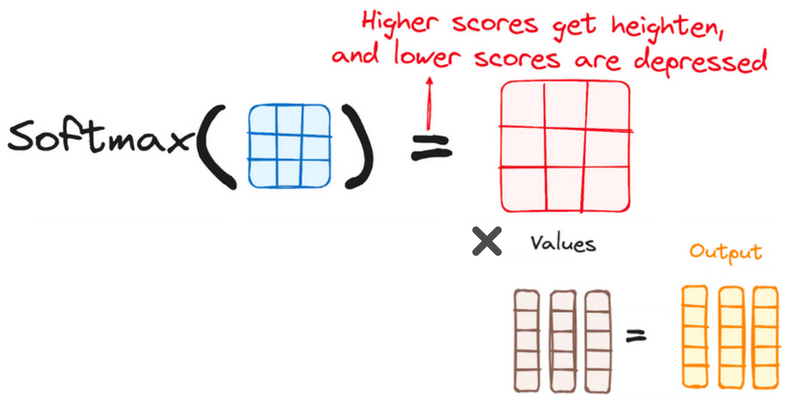
Encoder architecture

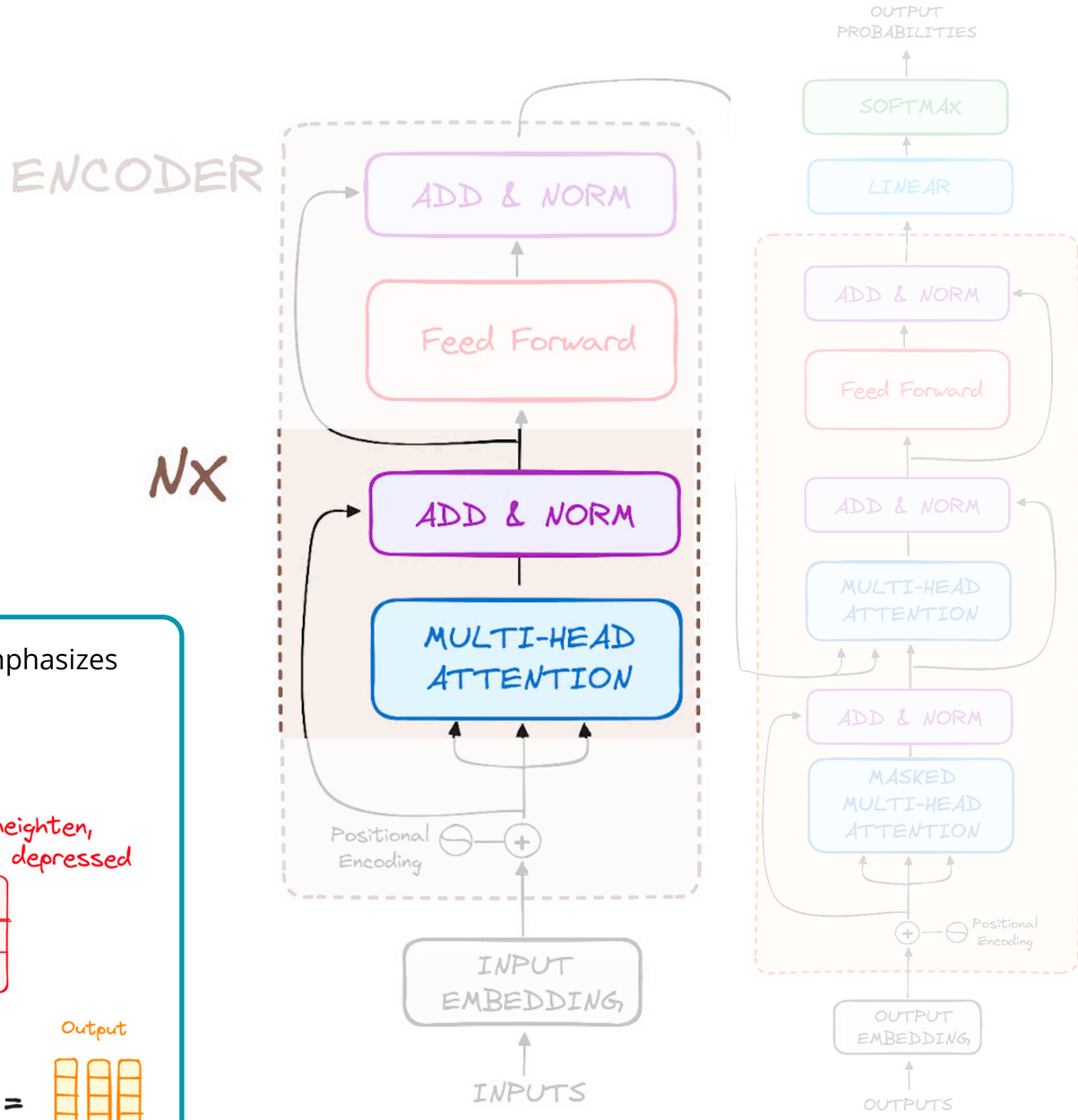
Multi-headed self-attention mechanism

- **Self-attention** allows to relate each word in the sequence to the others
- Calculates an **attention score** using:
  - **Query (Q)** : a token in the sequence
  - **Key (K)** : another token in the sequence
  - **Value (V)** : how well do **Q** and **K** match



Probability values ranging from 0 to 1emphasizes higher scores and lower scores

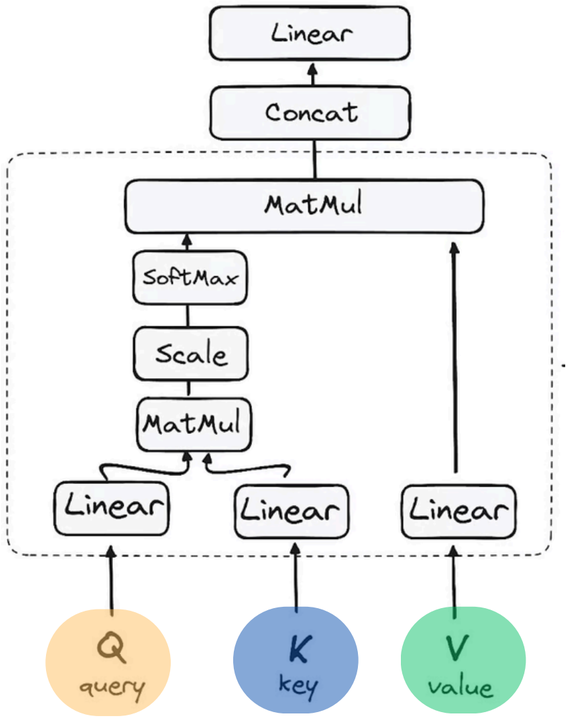


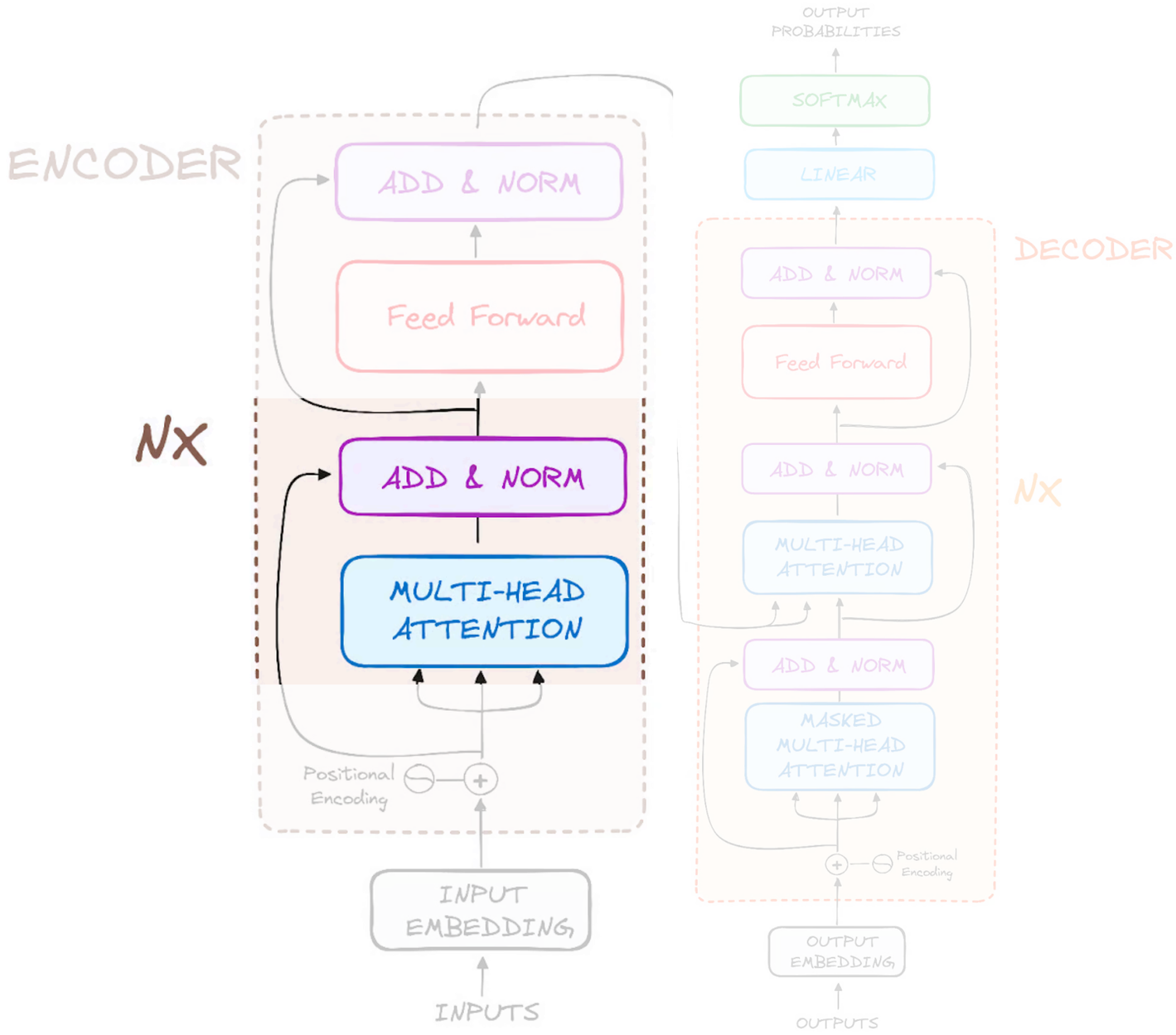


Encoder architecture

Multi-headed self-attention mechanism

- Note that we add skip (residual) connections to avoid **vanishing gradient** problem

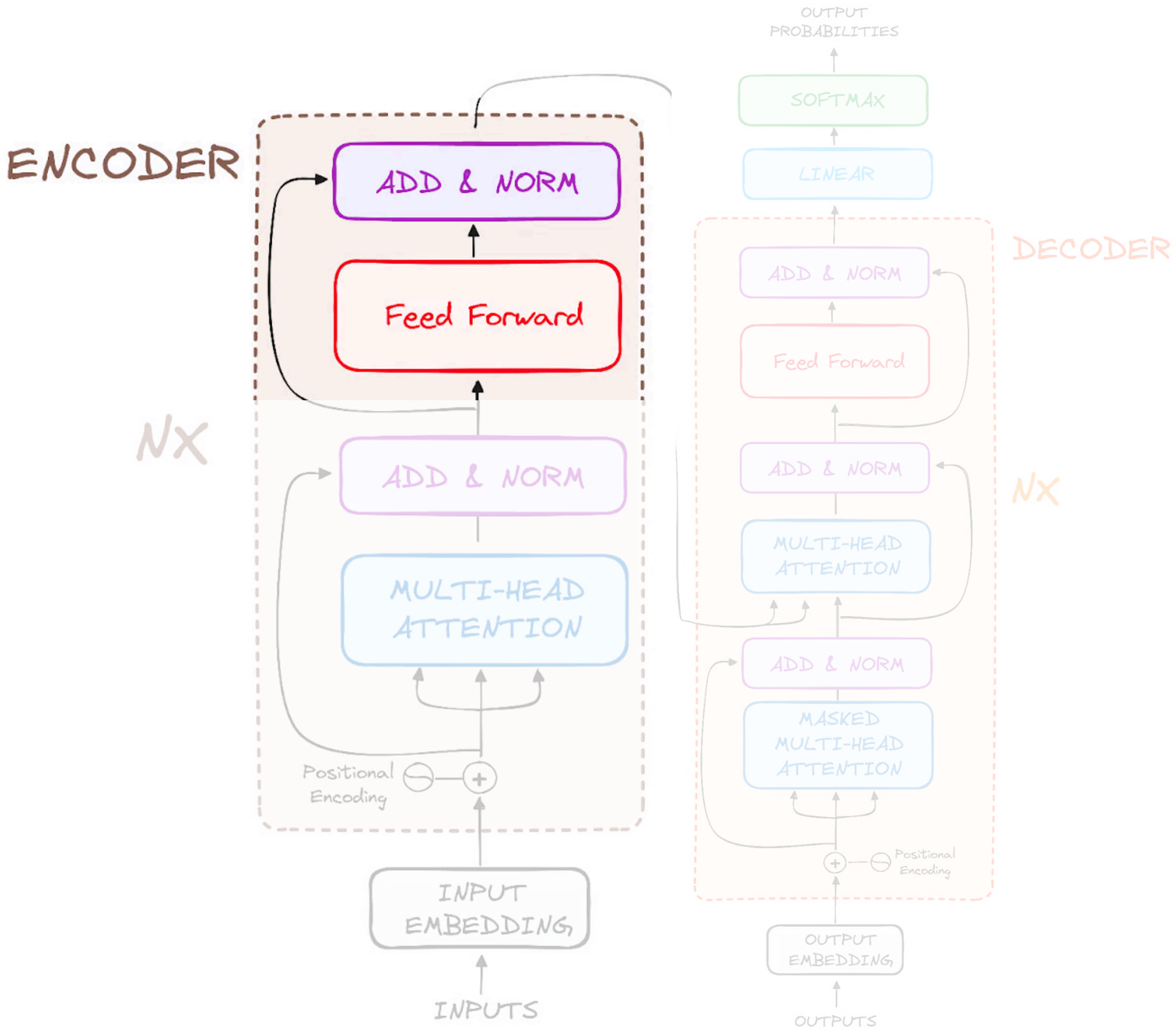




Encoder architecture

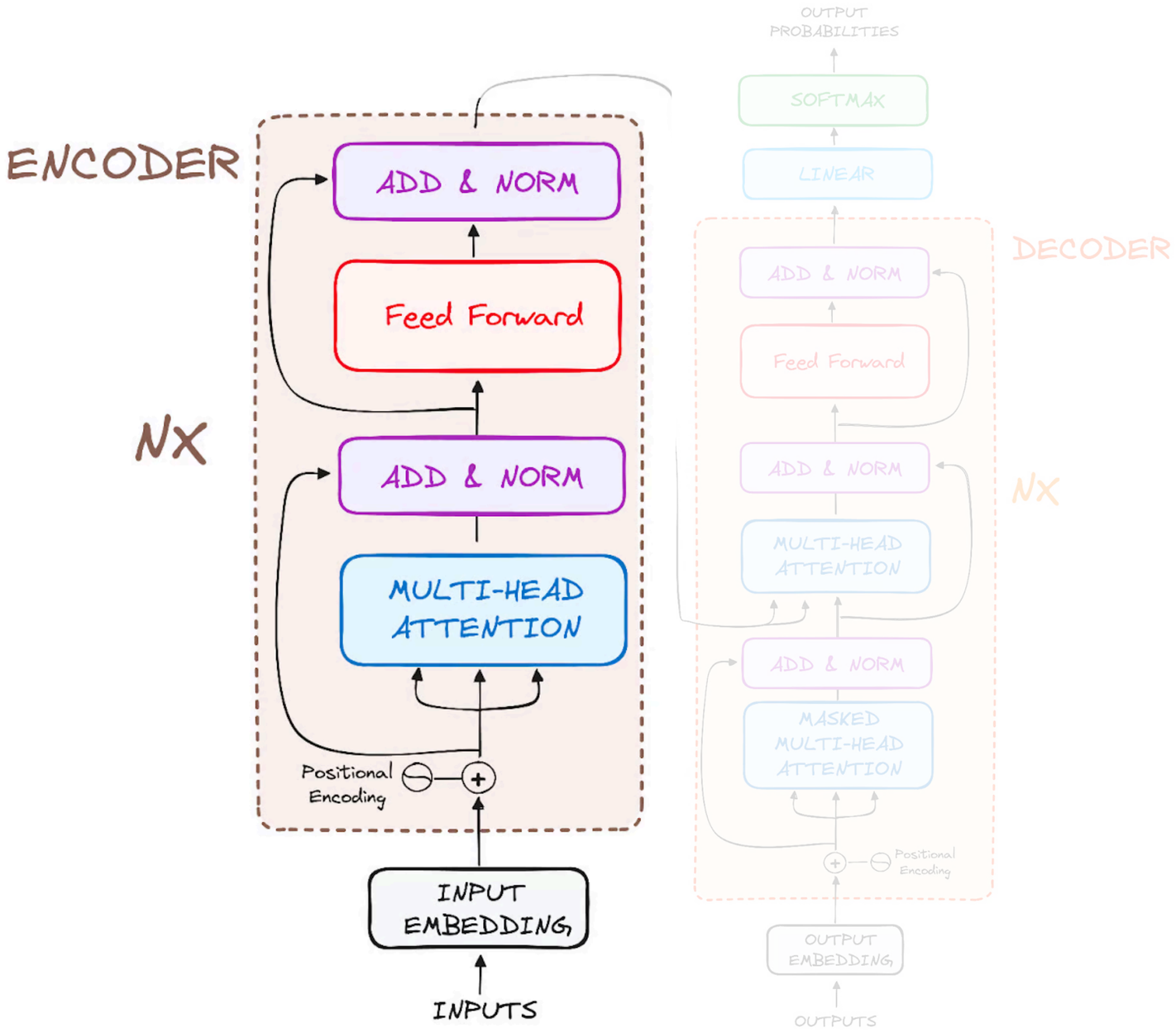
Feed Forward network

- Acts as a bridge between encoder and decoder

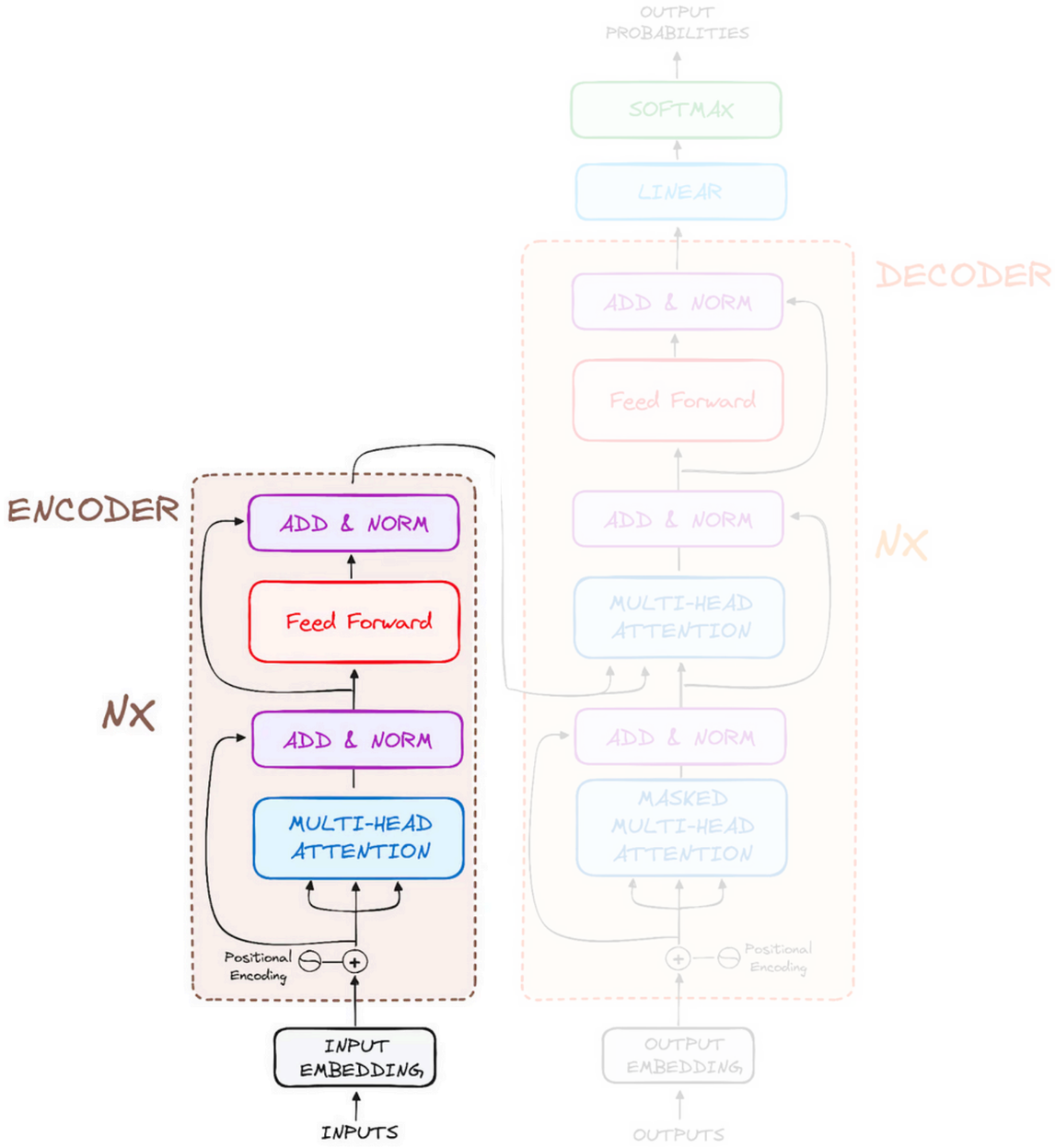


Encoder architecture

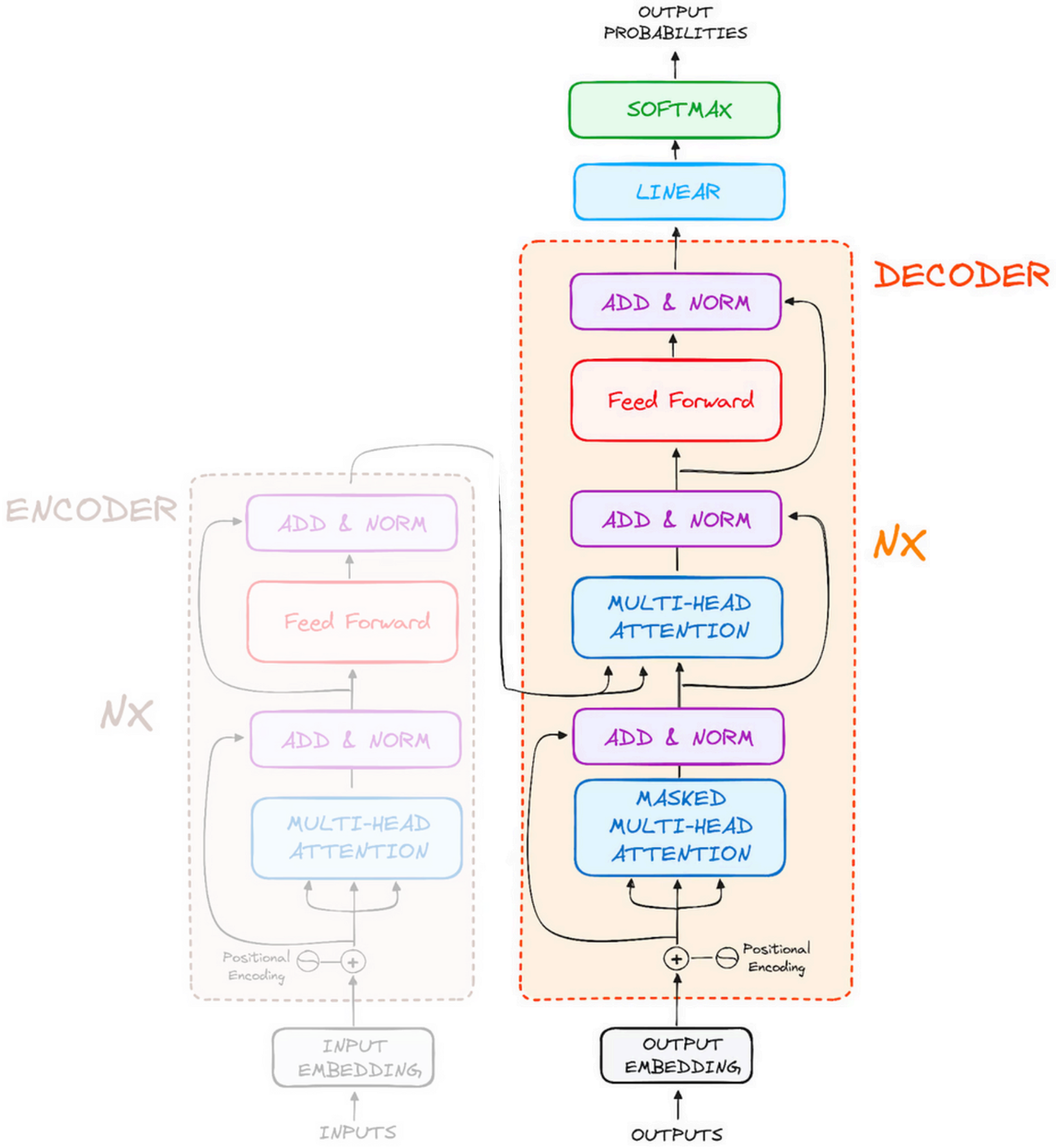
So, encoder output is a rich contextual representation of tokens and their semantics



Encoder architecture

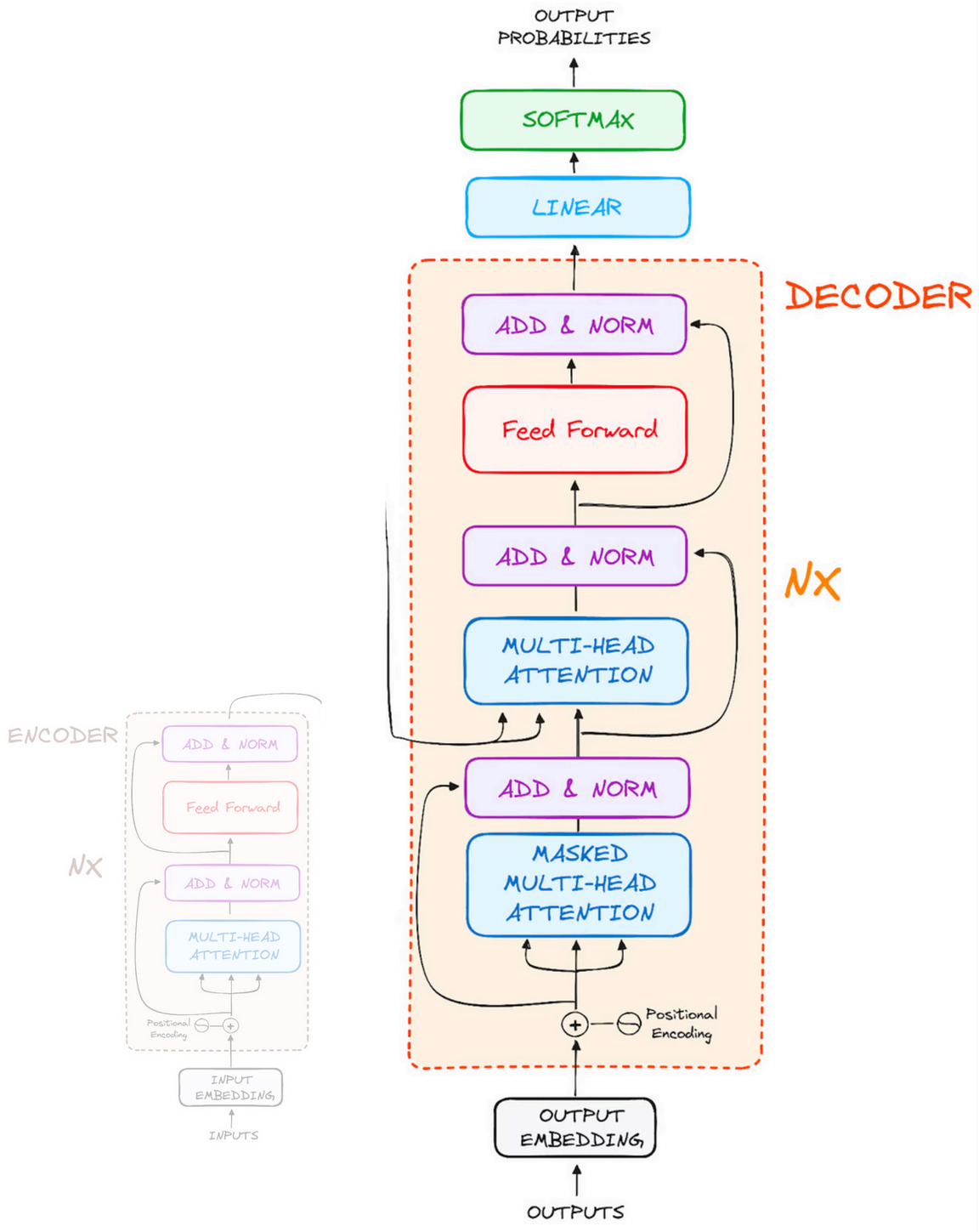


Decoder architecture



Decoder architecture

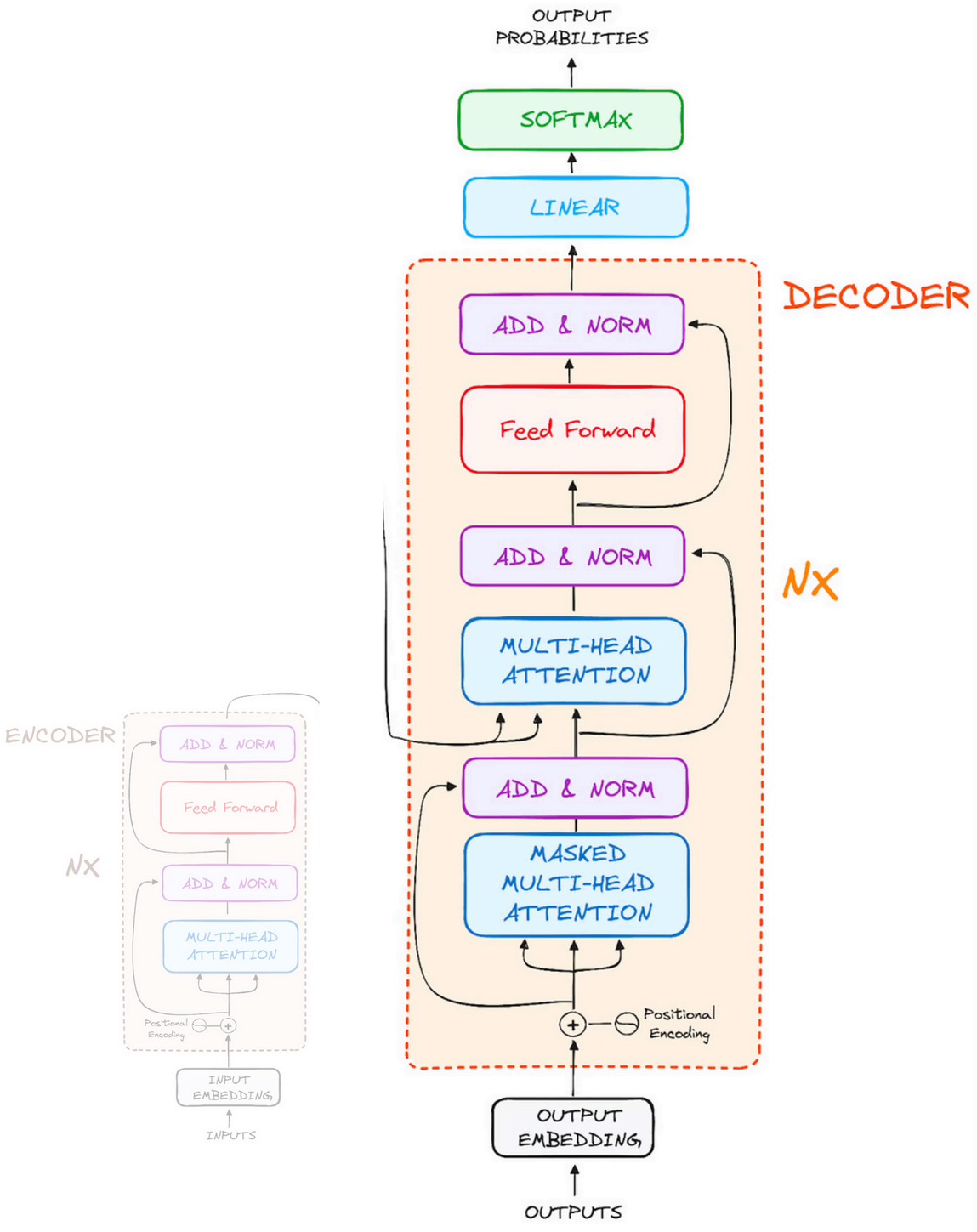
- The decoder mirros and replicates many components from the encoder like embedding, positional encoding ...etc



Decoder architecture

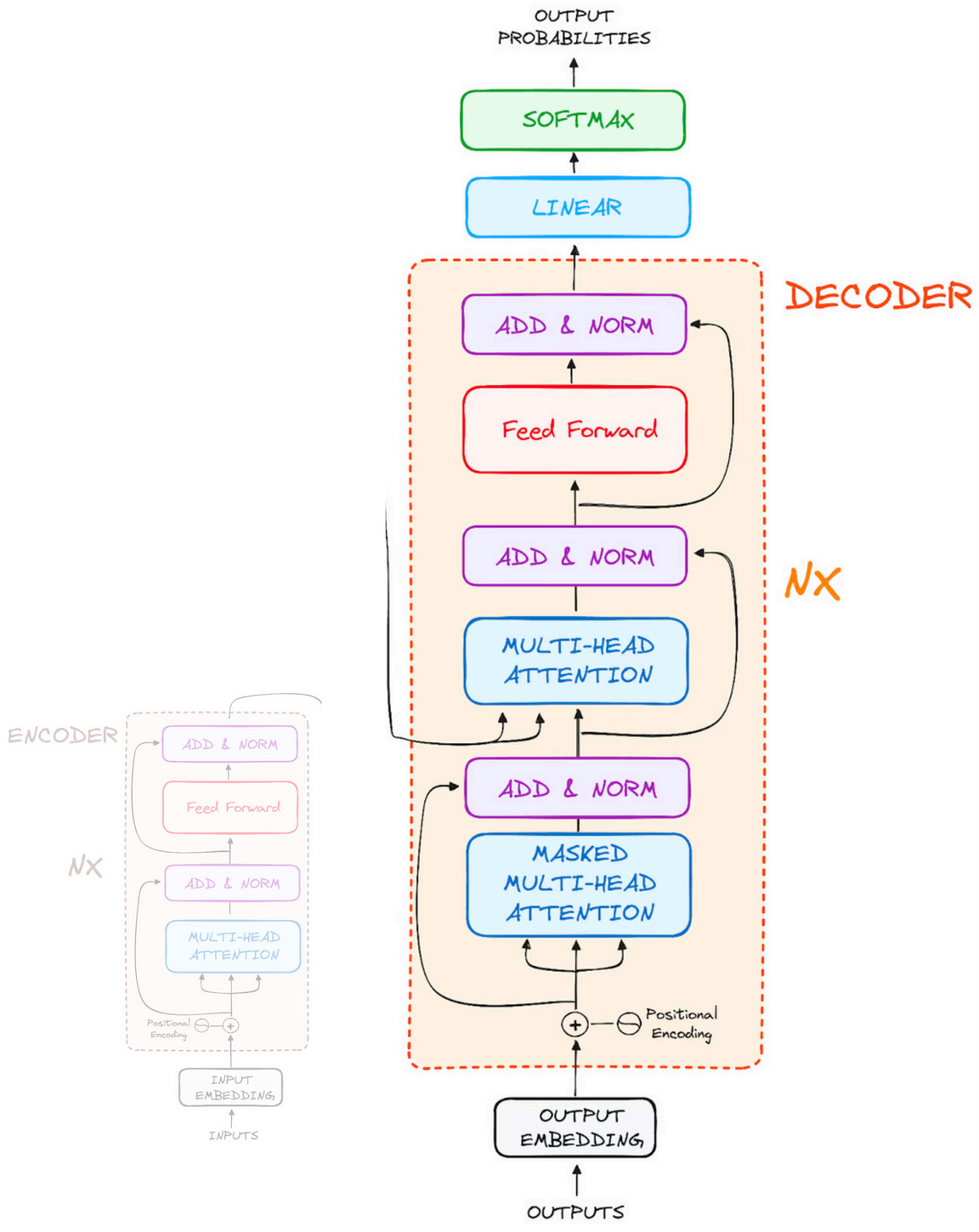
- The decoder mirros and replicates many components from the encoder like embedding, positional encoding …etc

- The **multi-head attention** modules have similar mechanism to the encoder but with twists to accomplish different task and achieve different goals



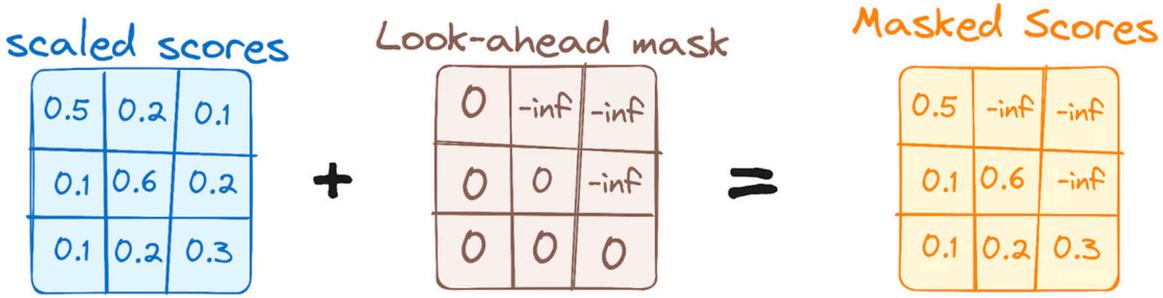
Decoder architecture

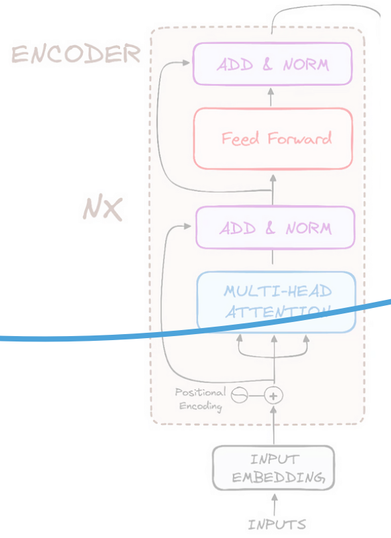
For instance, **masked multi-head attention** makes sure that tokens are not influenced by future ones in the prediction process (decoding)

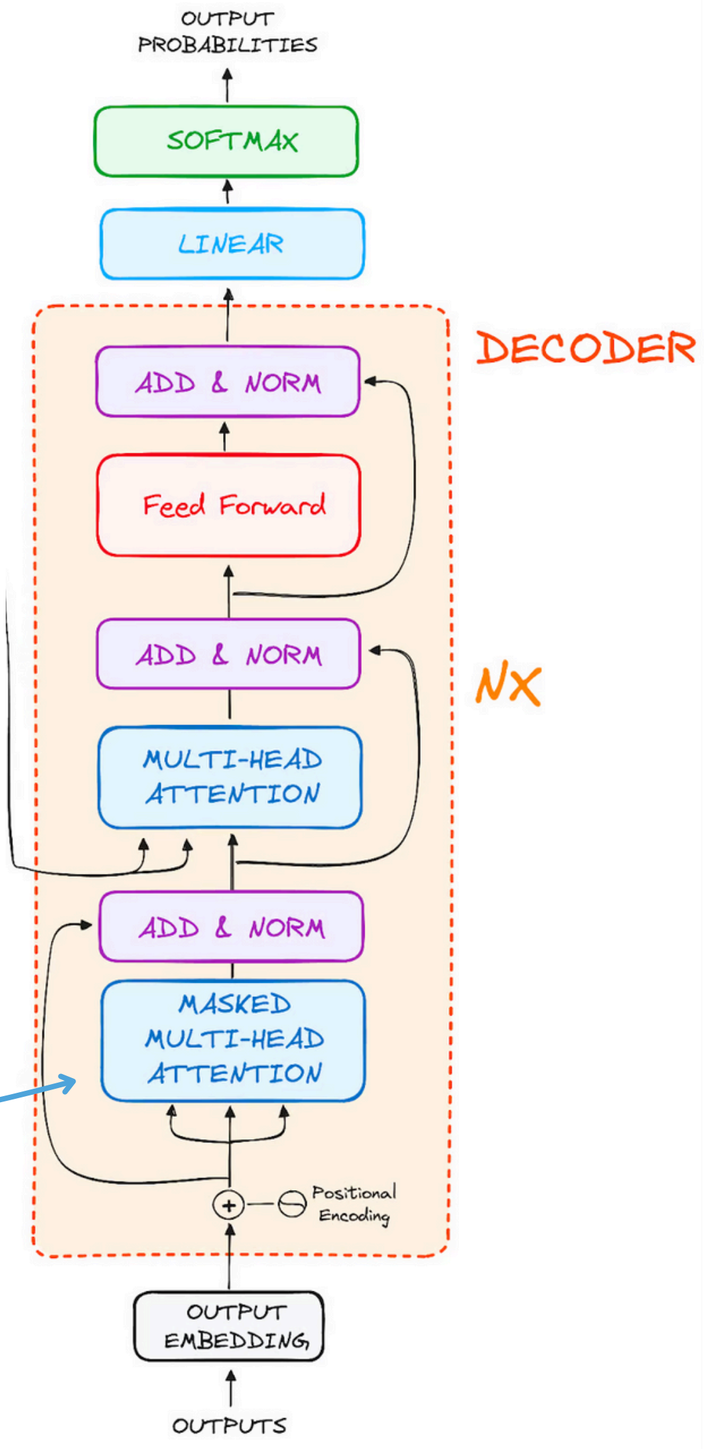


Decoder architecture

For instance, **masked multi-head attention** makes sure that tokens are not influenced by future ones in the prediction process (decoding)



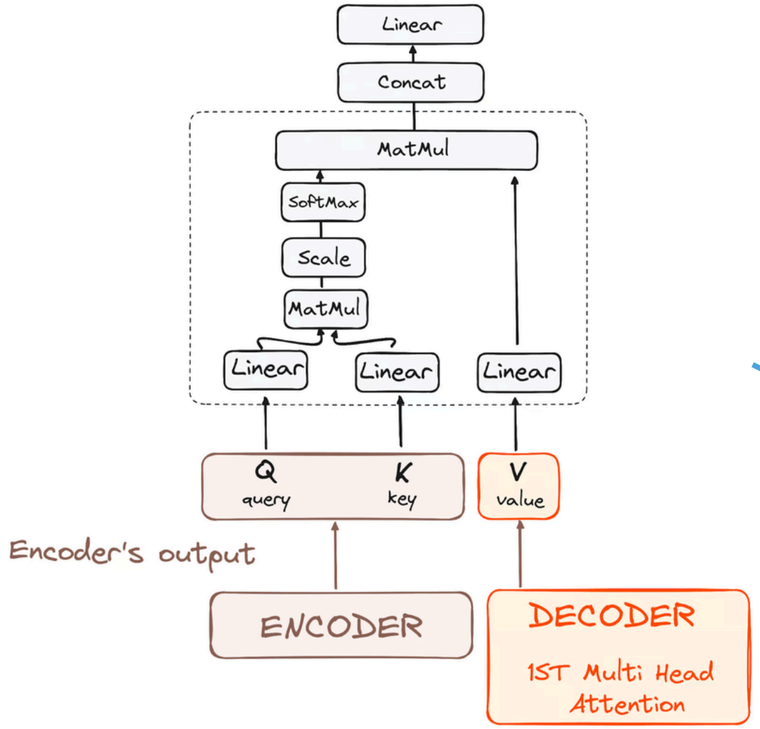


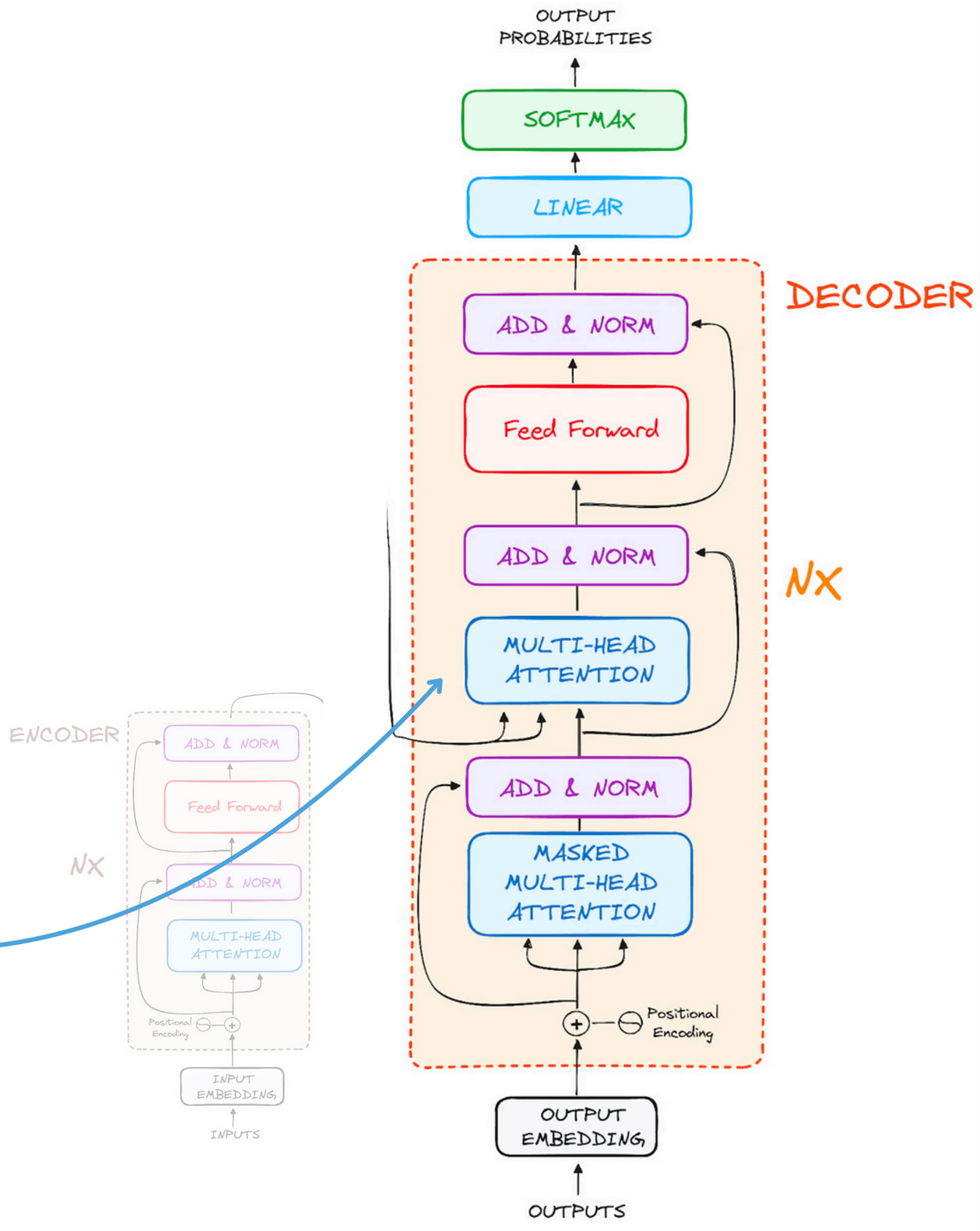


Decoder architecture

For instance, **masked multi-head attention** makes sure that tokens are not influenced by future ones in the prediction process (decoding)

in the decoder's second **multi-head attention (cross attention)** it uses encoder's outputs as **Keys** and **Queries** but uses its' **first masked multi-head attention** as **values (attention scores)**



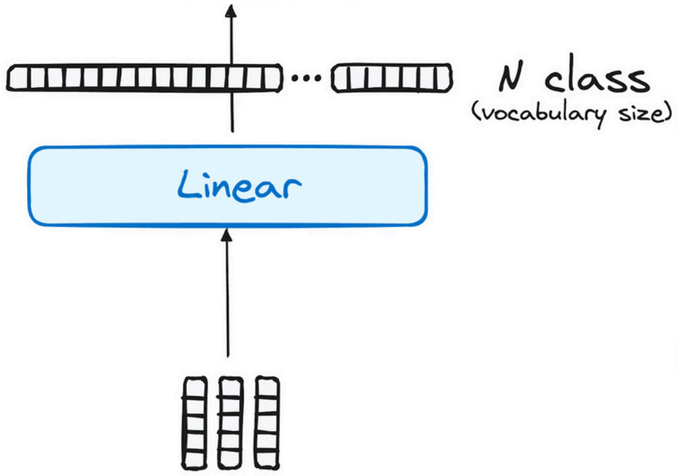


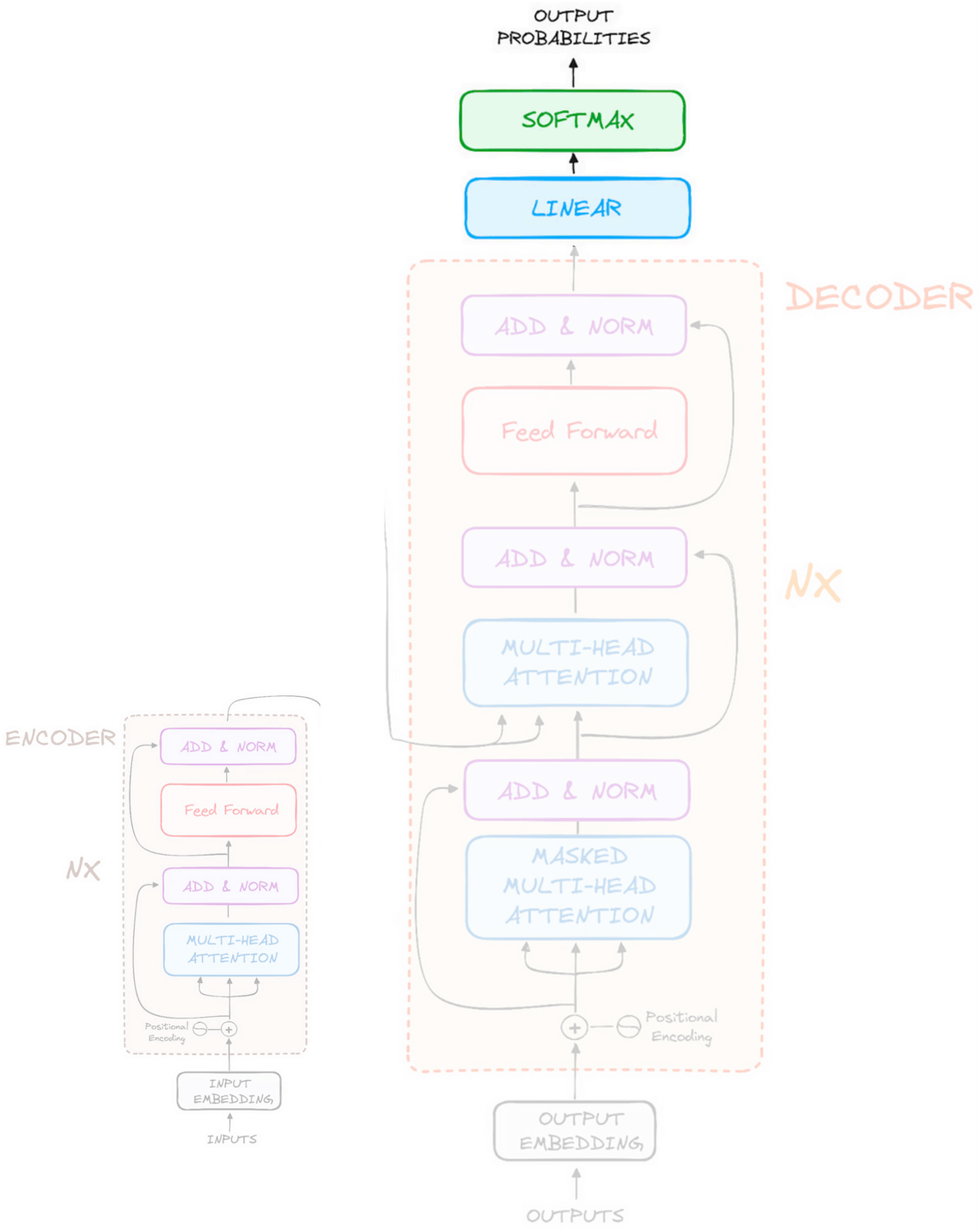
Decoder architecture

For instance, **masked multi-head attention** makes sure that tokens are not influenced by future ones in the prediction process (decoding)

in the decoder's second **multi-head attention (cross attention)** it uses encoder's outputs as **Keys** and **Queries** but uses its' **first masked multi-head attention** as **values (attention scores)**

The output of the decoder is a **vector of probabilities (classifier)** for each word in the **vocabulary** of the model



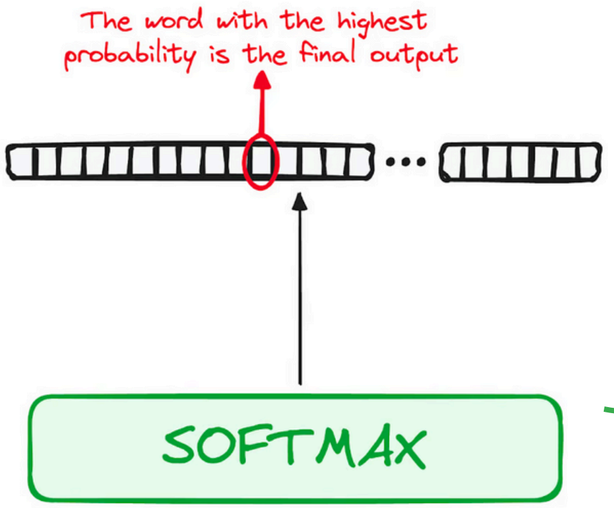


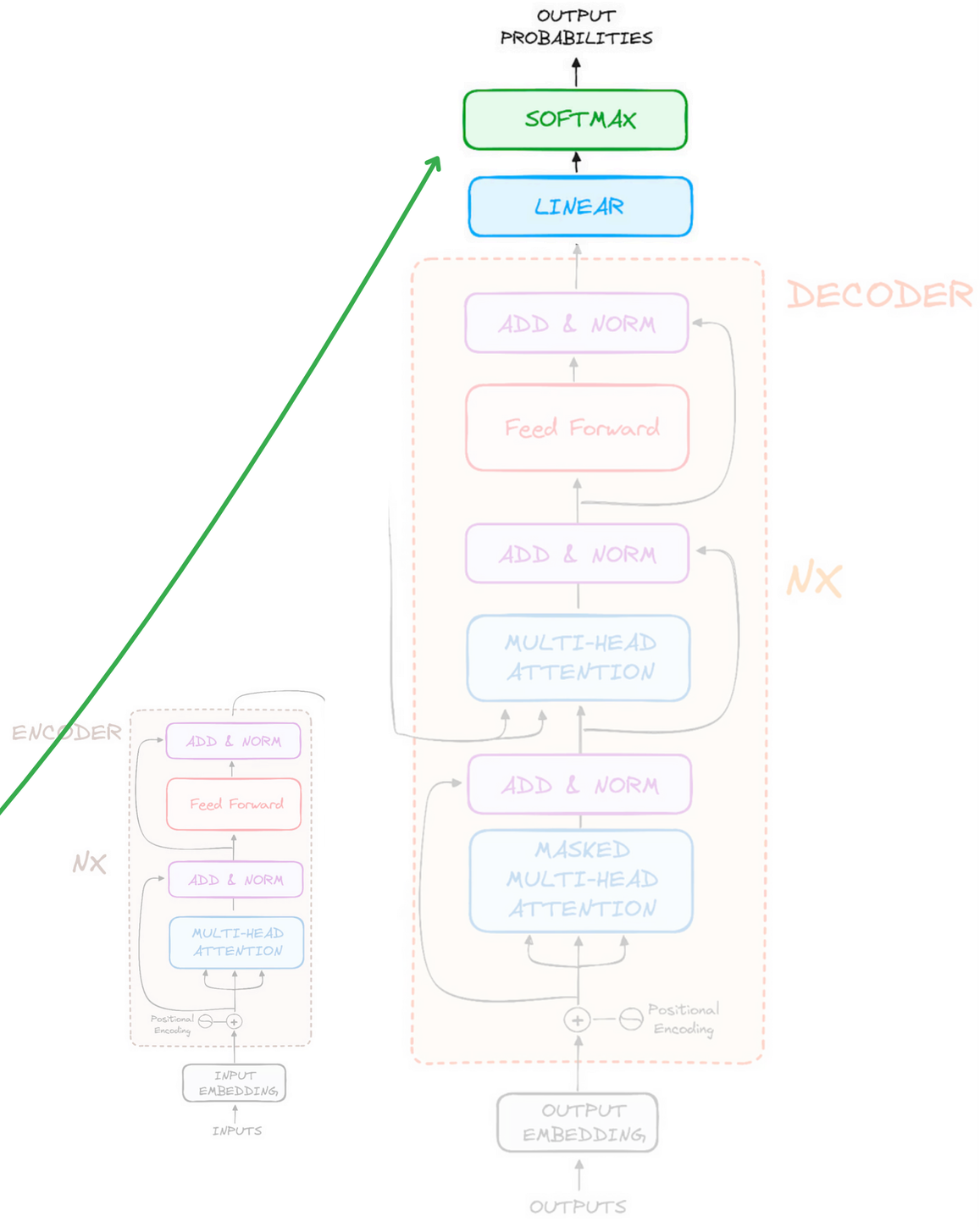
Decoder architecture

For instance, **masked multi-head attention** makes sure that tokens are not influenced by future ones in the prediction process (decoding)

in the decoder's second **multi-head attention (cross attention)** it uses encoder's outputs as **Keys** and **Queries** but uses its' **first masked multi-head attention** as **values (attention scores)**

The output of the decoder is a **vector of probabilities (classifier)** for each word in the **vocabulary** of the model





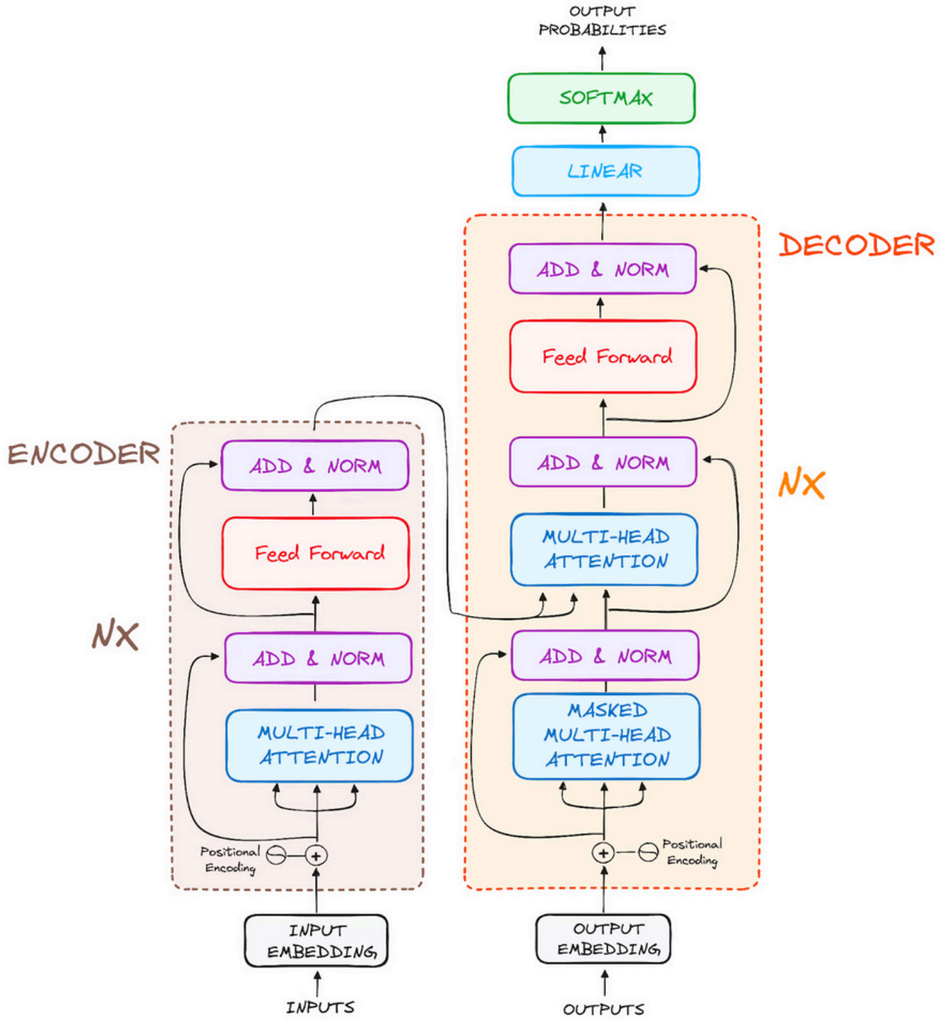
Content

- Overview on deep learning models
- LLM models are large and complex
- Smaller, lighter LLMs
- Challenges and perspectives
- Summary

Content

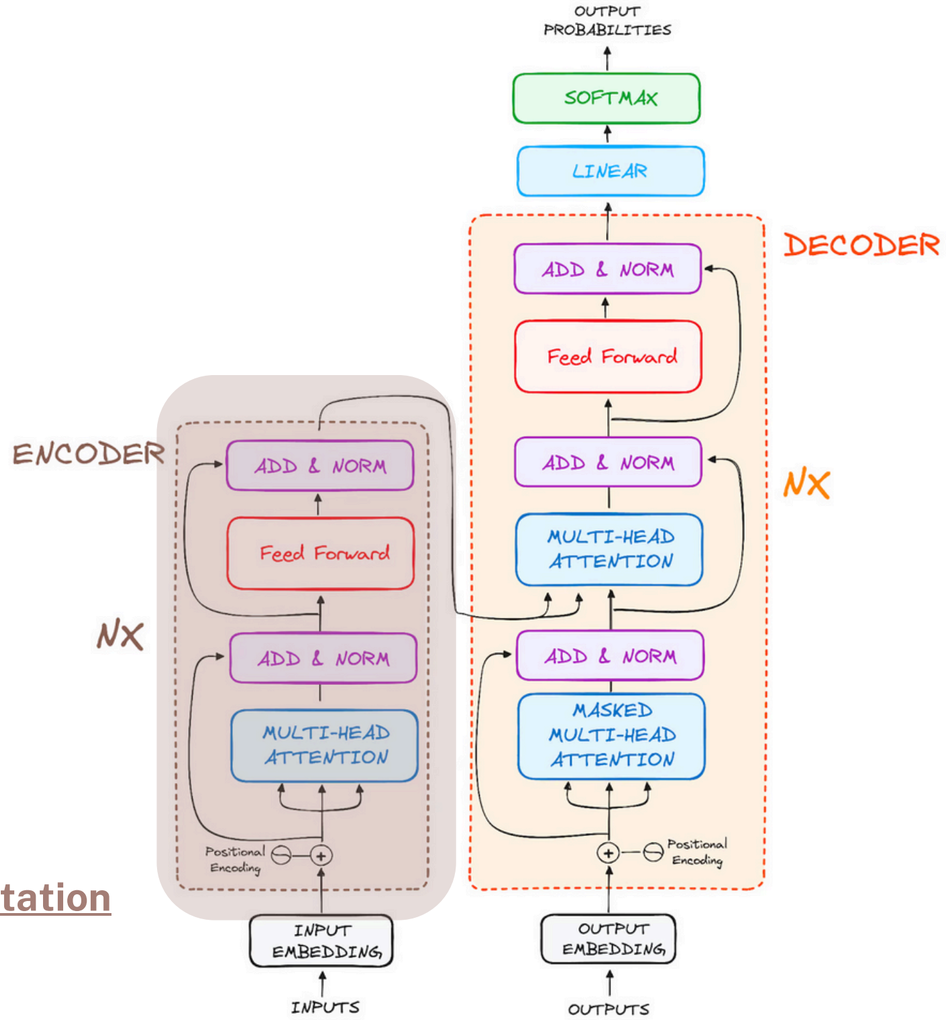
Evolution of transformer-based language model (LM)

Evolution of transformer-based language model (LM)



2017

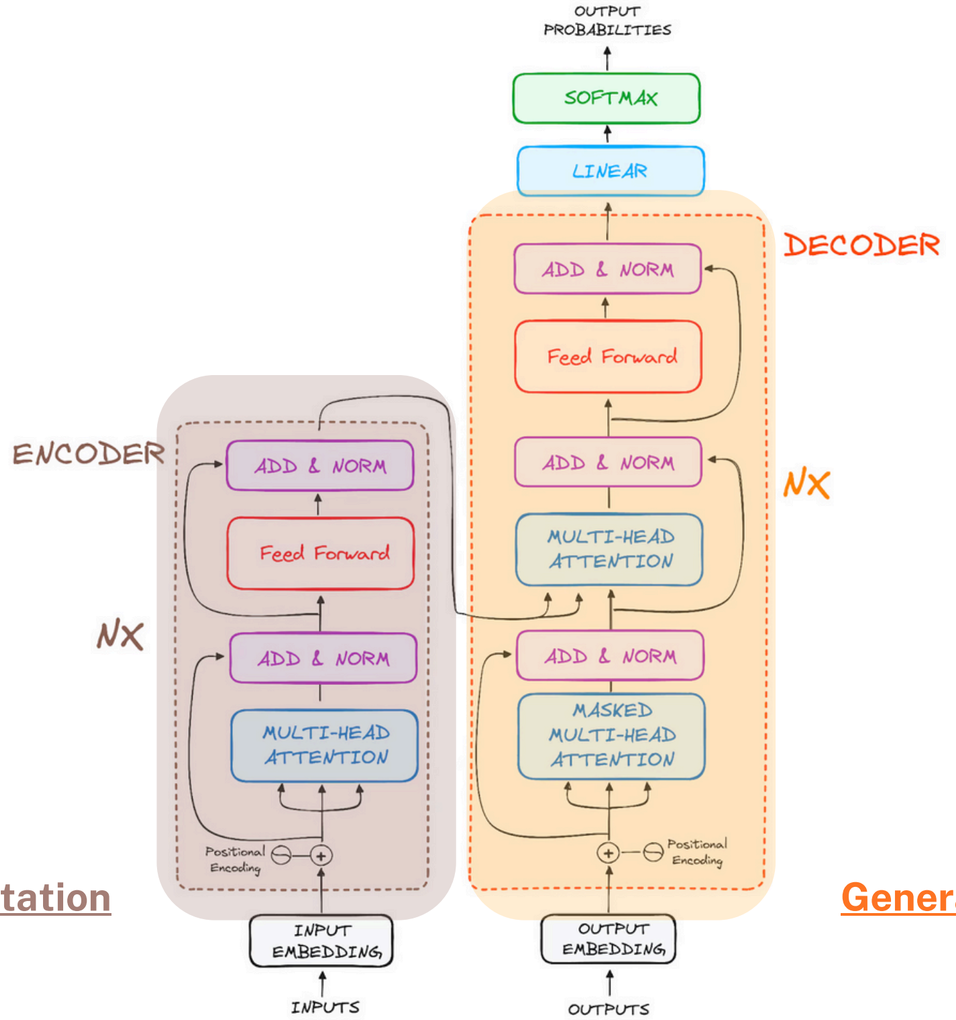
Evolution of transformer-based language model (LM)



Representation

2017

Evolution of transformer-based language model (LM)



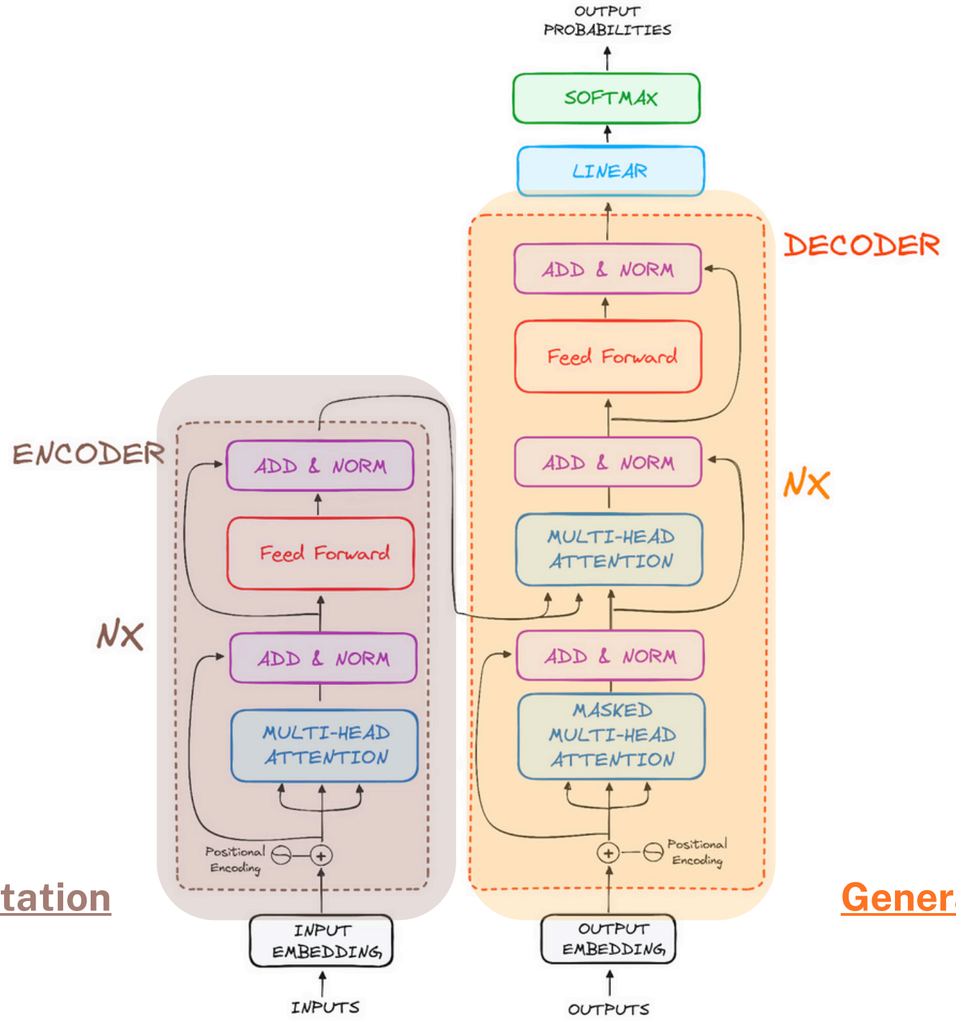
Representation

Generation

2017

Evolution of transformer-based language model (LM)

Can see all timesteps



Can see only previous timesteps

Representation

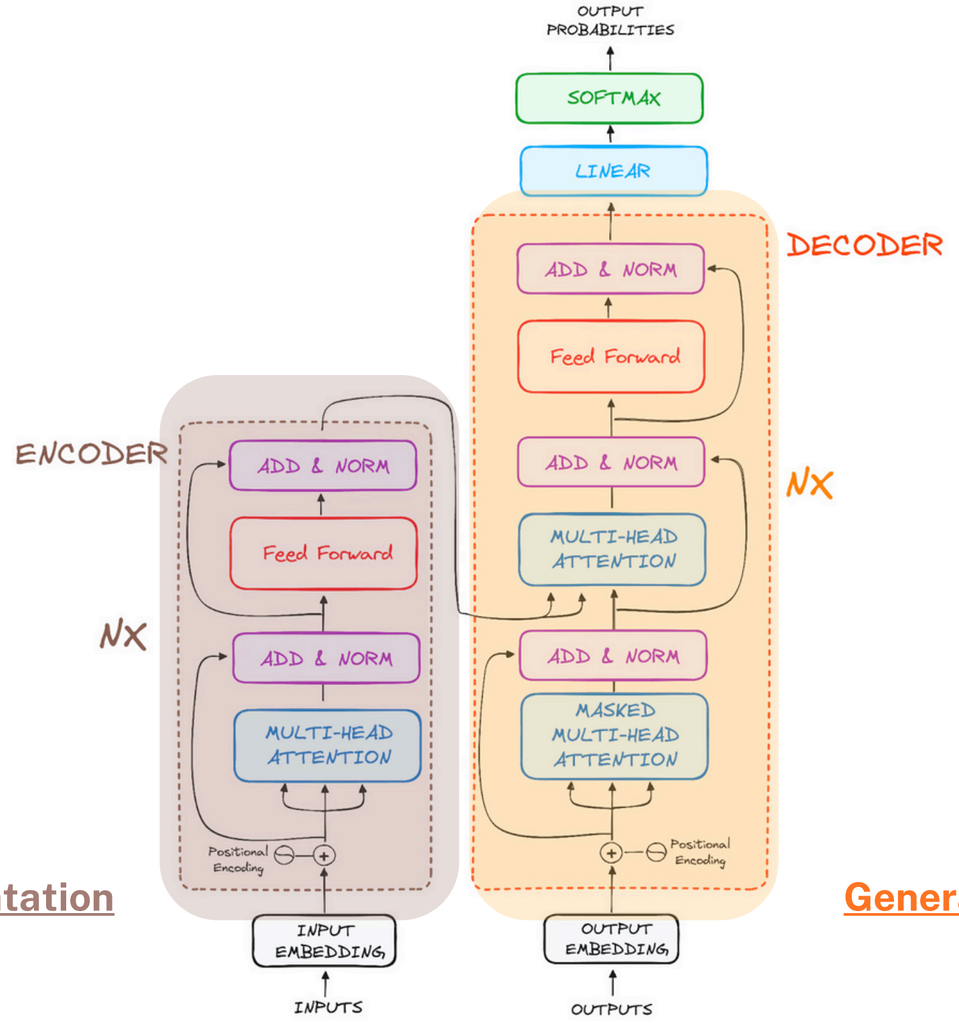
Generation

2017

Evolution of transformer-based language model (LM)

Can see all timesteps

Not auto-regressive



Can see only previous timesteps

auto-regressive (uses previous timesteps' outputs)

Representation

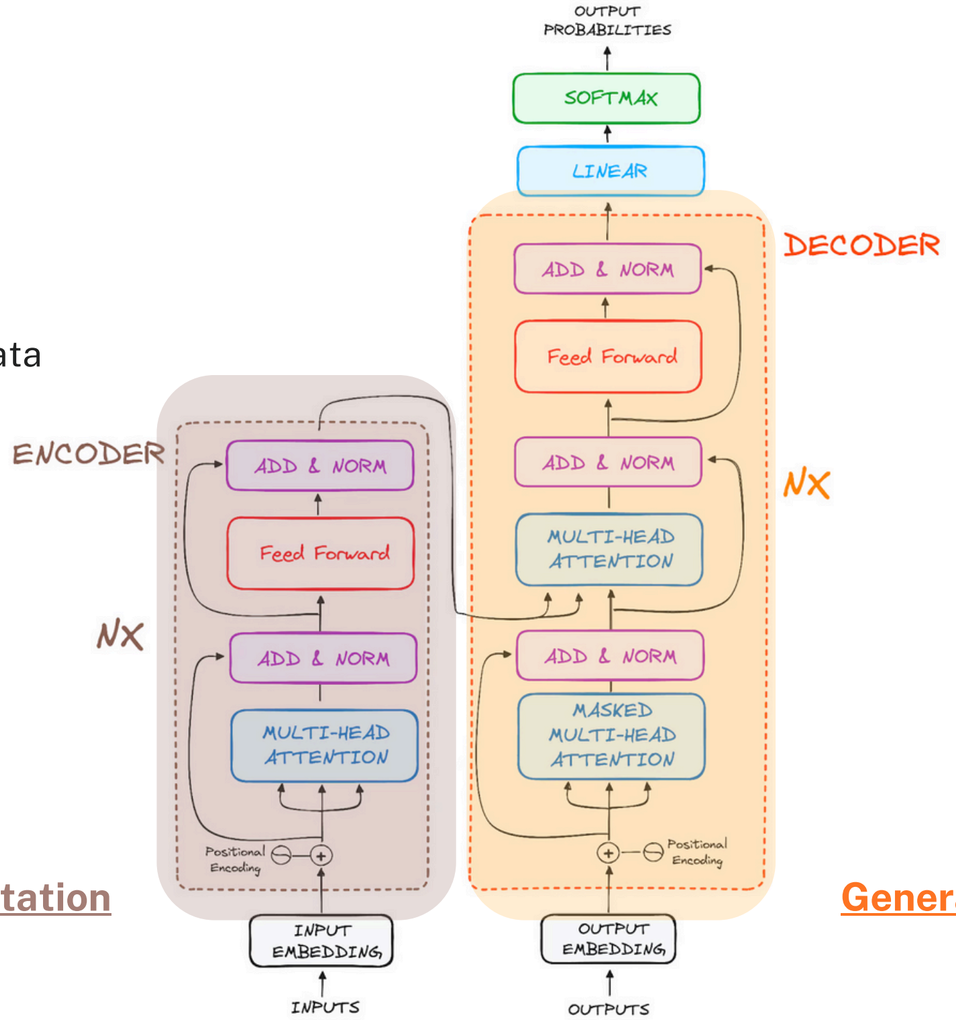
Generation

2017

Evolution of transformer-based language model (LM)

Challenges

- Lack of task-specific training data



Representation

Generation

2017

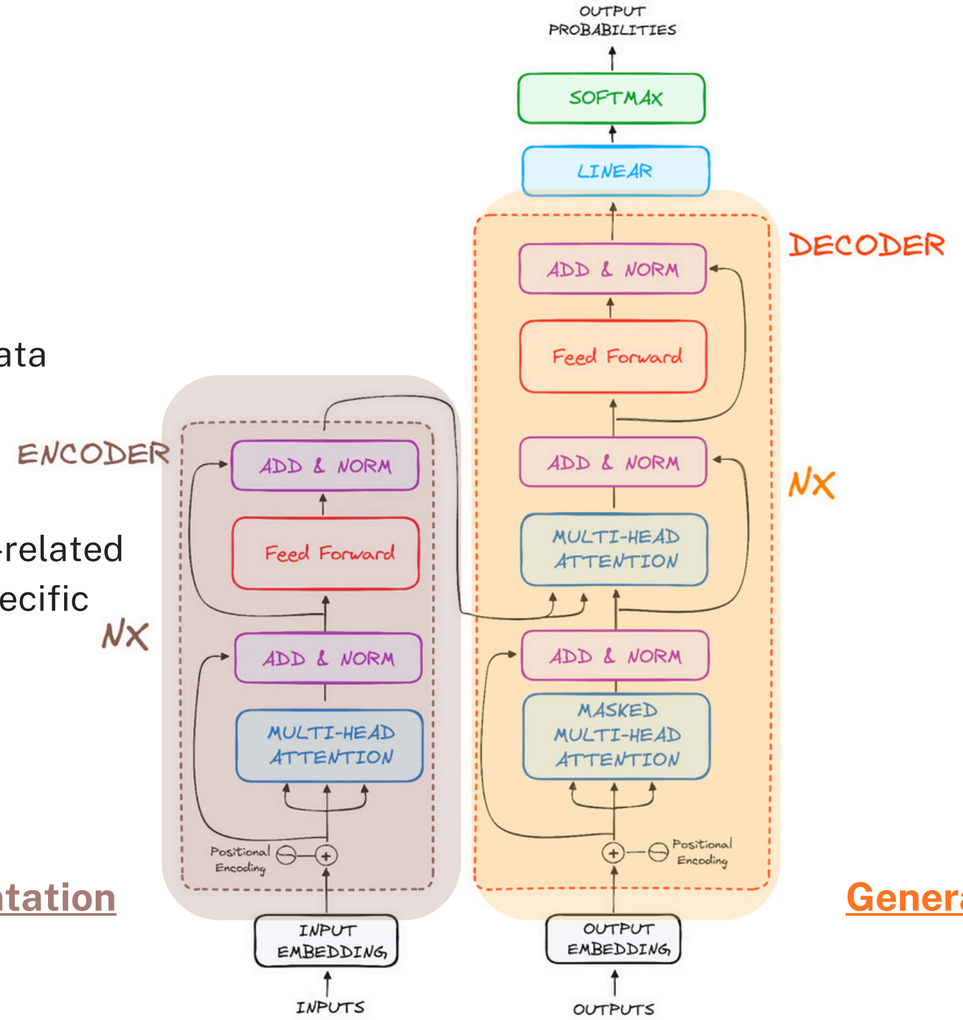
Evolution of transformer-based language model (LM)

Challenges

- Lack of task-specific training data

Solution

Train a baseline model for all NLP-related tasks, then fine-tune for task specific



Representation

Generation

2017

Evolution of transformer-based language model (LM)

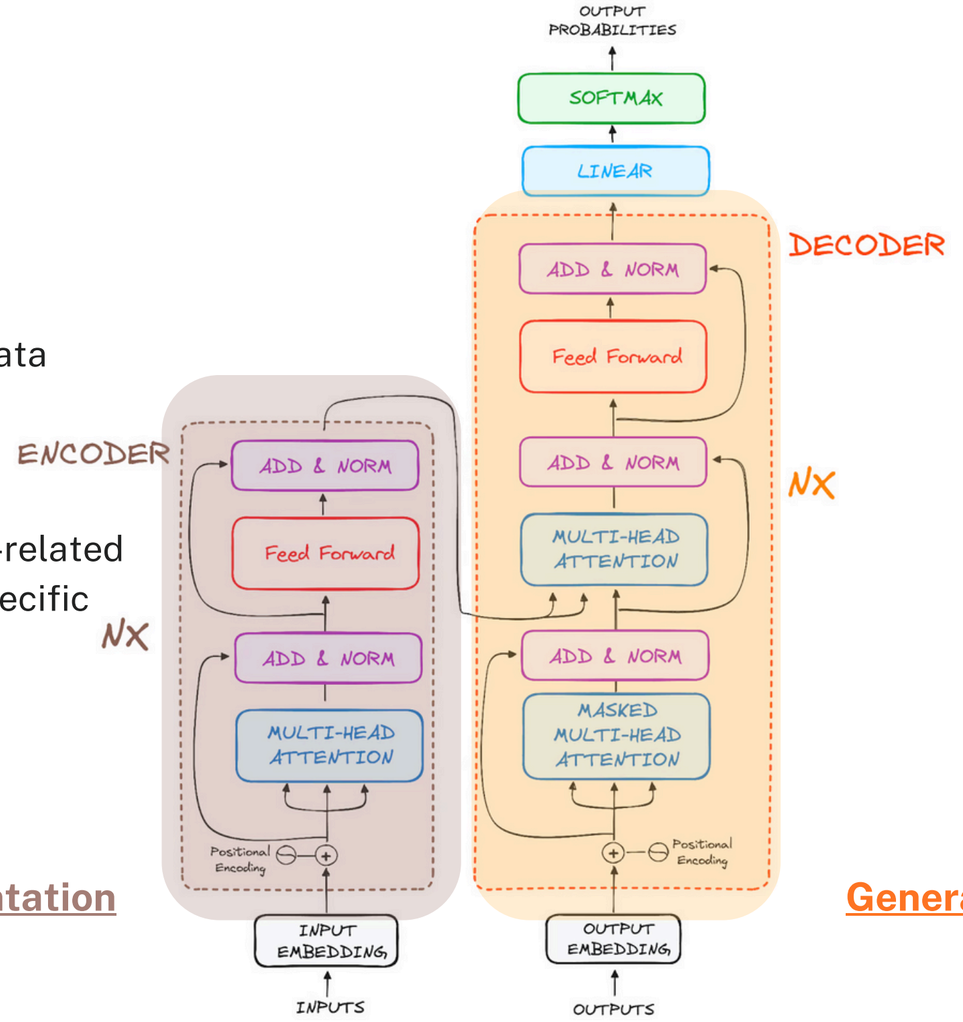
Challenges

- Lack of task-specific training data

Solution

Train a baseline model for all NLP-related tasks, then fine-tune for task specific

BERT (Bidirectional Encoder Representations)



Representation

Generation

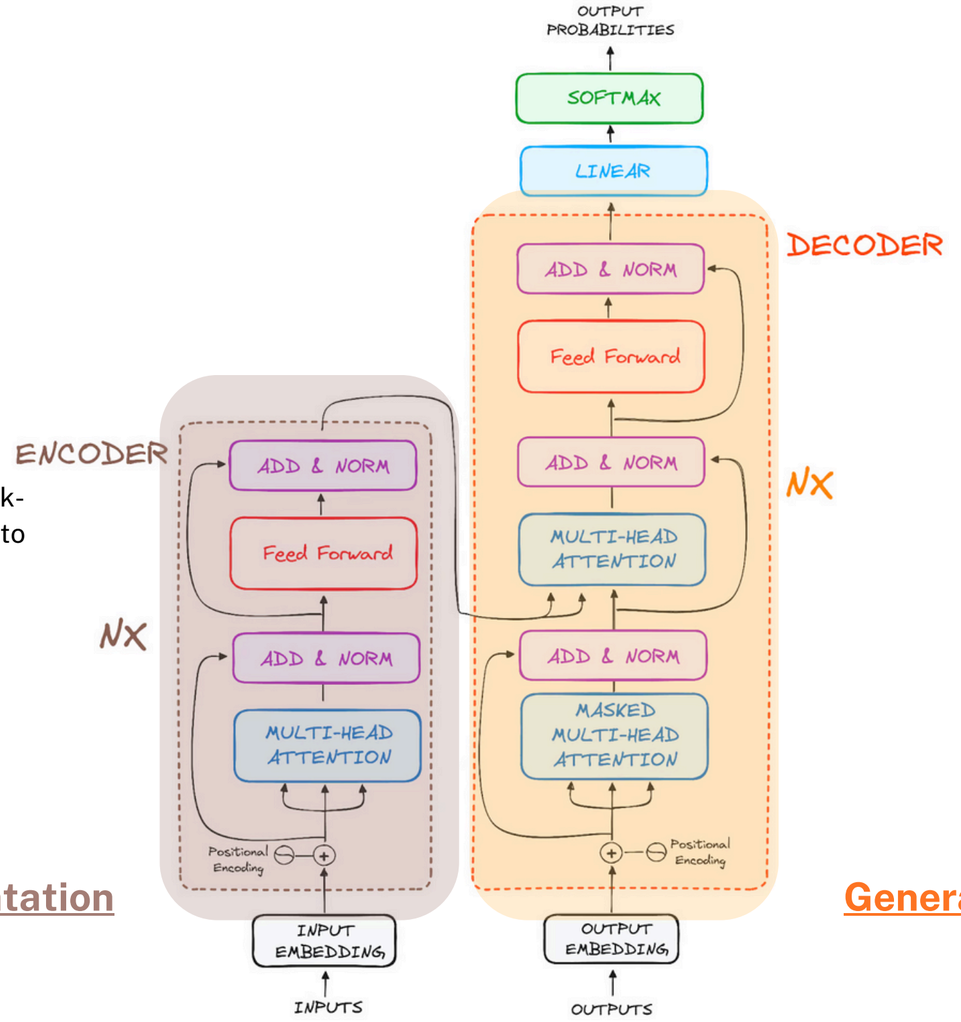
October 2018

Evolution of transformer-based language model (LM)

Challenges

BERT (Bidirectional Encoder Representations)

Use BERT pre-trainingprocess and add a task-specific module after the last encoder layer to map it to the desired dimension



Representation

Generation

October 2018

Evolution of transformer-based language model (LM)

Challenges

BERT (Bidirectional Encoder Representations)

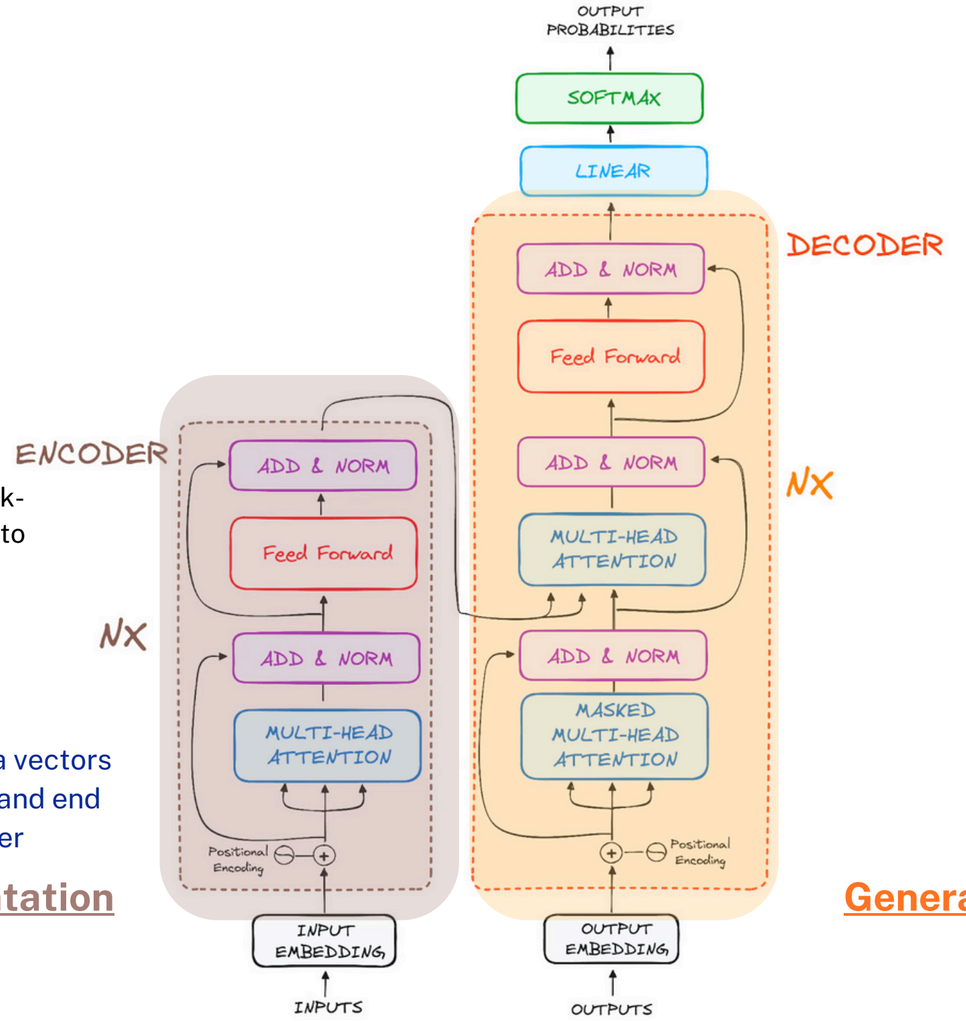
Use BERT pre-trainingprocess and add a task-specific module after the last encoder layer to map it to the desired dimension

Classification task

add a feed-forward layer after encoder output

Q&A task

Train two extra vectors to mark start and end of answer



Representation

Generation

October 2018

Evolution of transformer-based language model (LM)

Challenges

BERT (Bidirectional Encoder Representations)

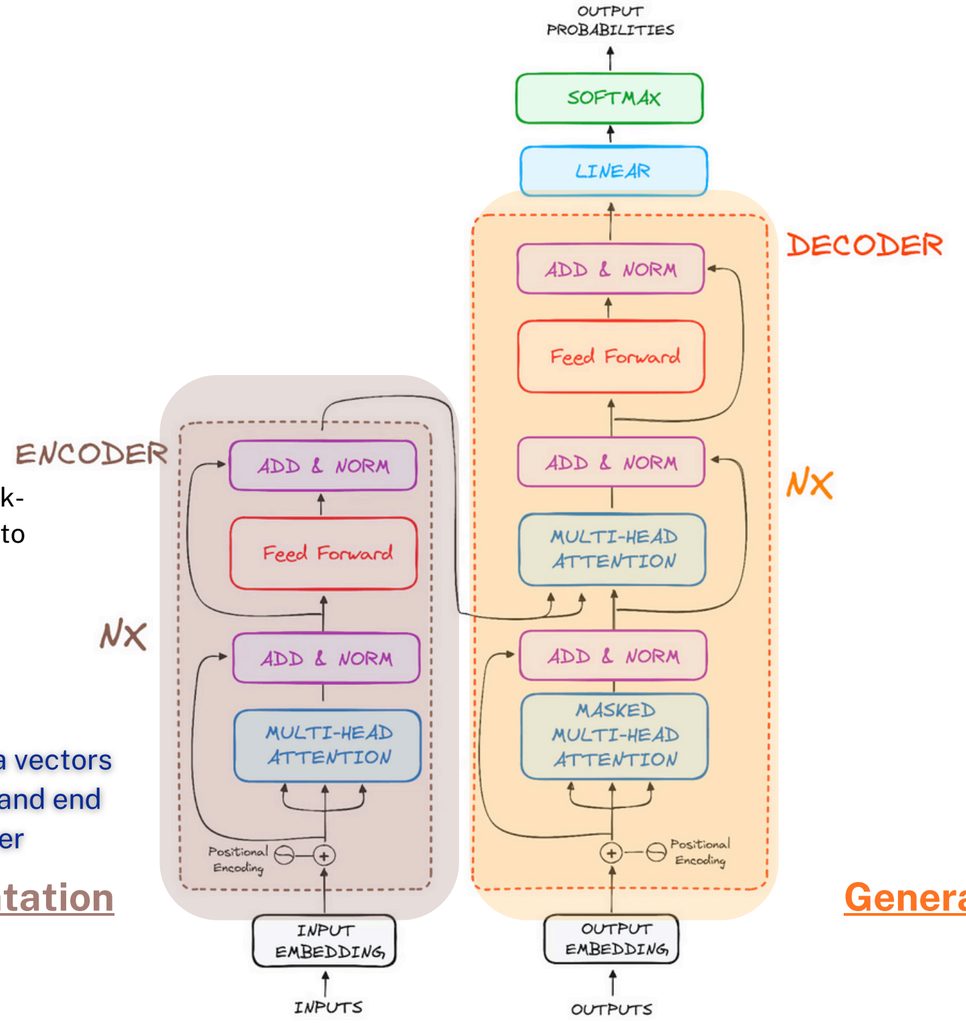
Use BERT pre-trainingprocess and add a task-specific module after the last encoder layer to map it to the desired dimension

Classification task

add a feed-forward layer after encoder output

Q&A task

Train two extra vectors to mark start and end of answer



Representation

Generation

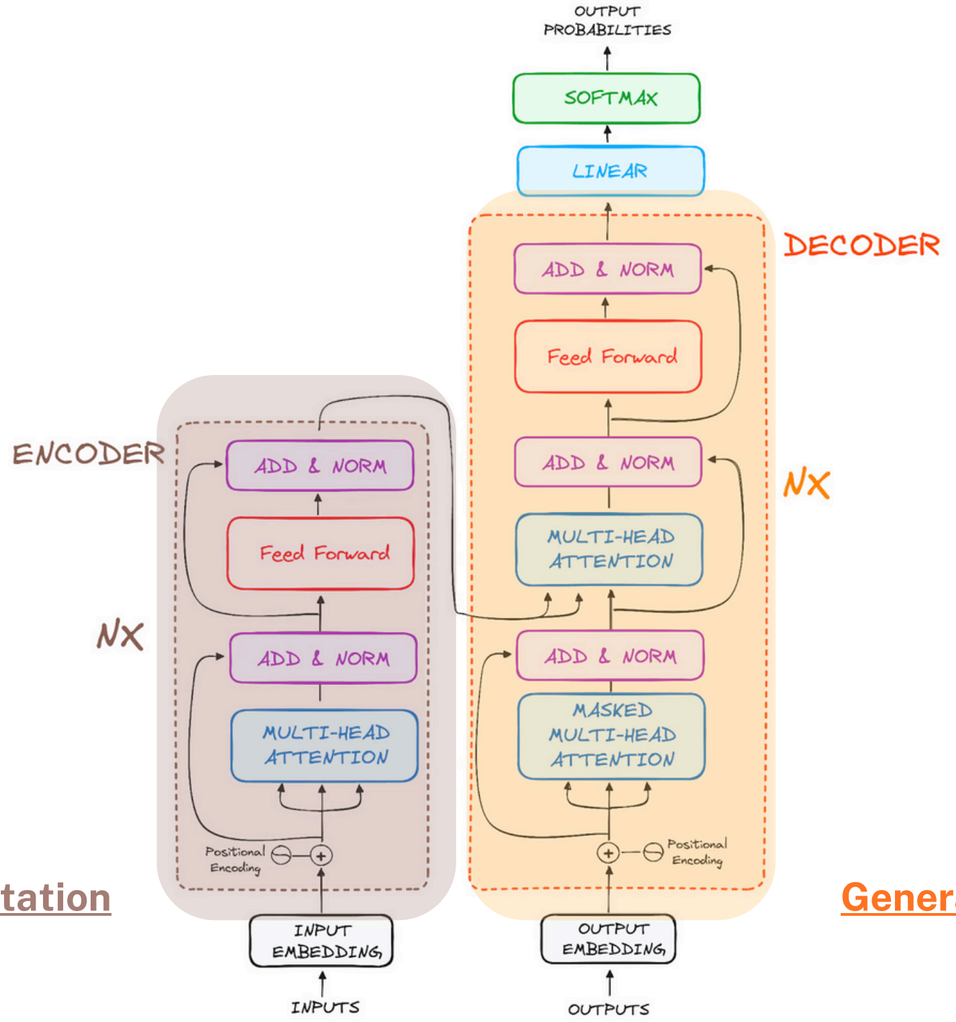
October 2018

Evolution of transformer-based language model (LM)

Challenges

BERT (Bidirectional Encoder Representations)

- Pre-training tasks can be **derived from base** process
- Effective large general representations can serve as backbone for specialized tasks **(highly transferable)**

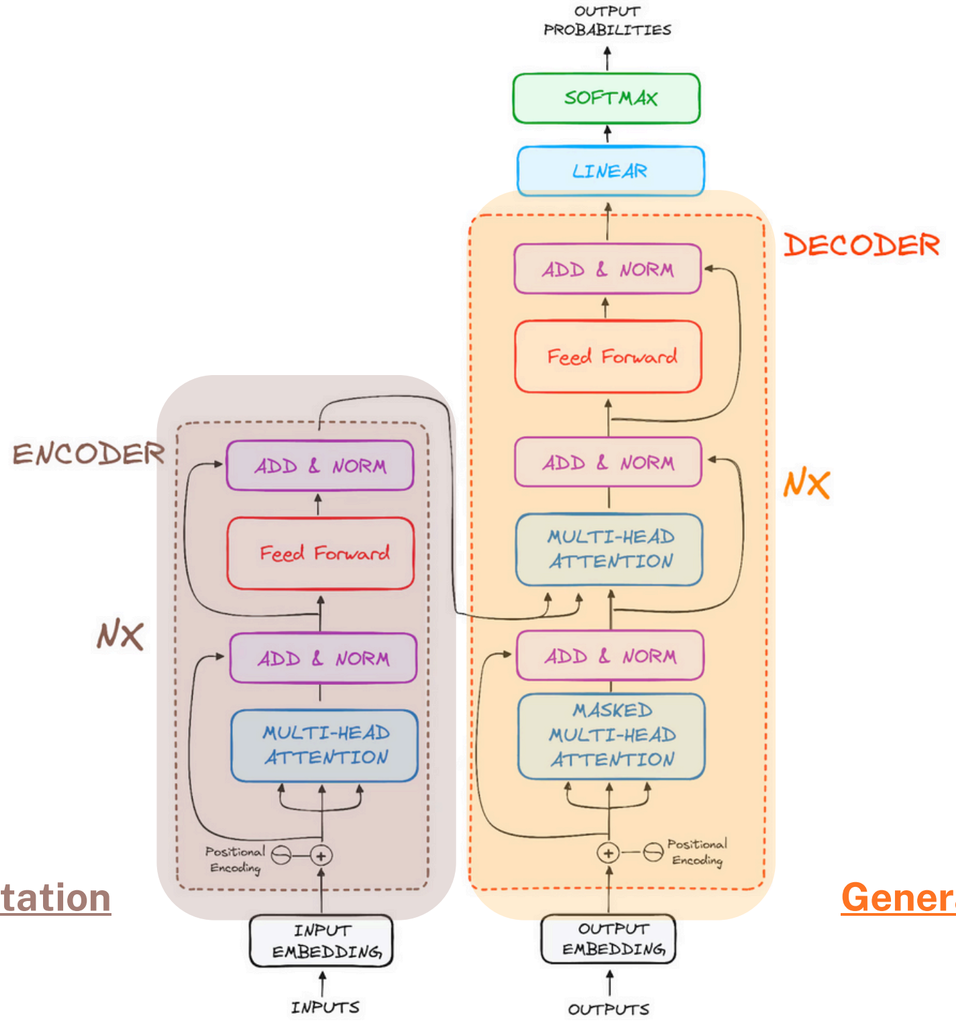


Representation

Generation

October 2018

Evolution of transformer-based language model (LM)



Representation

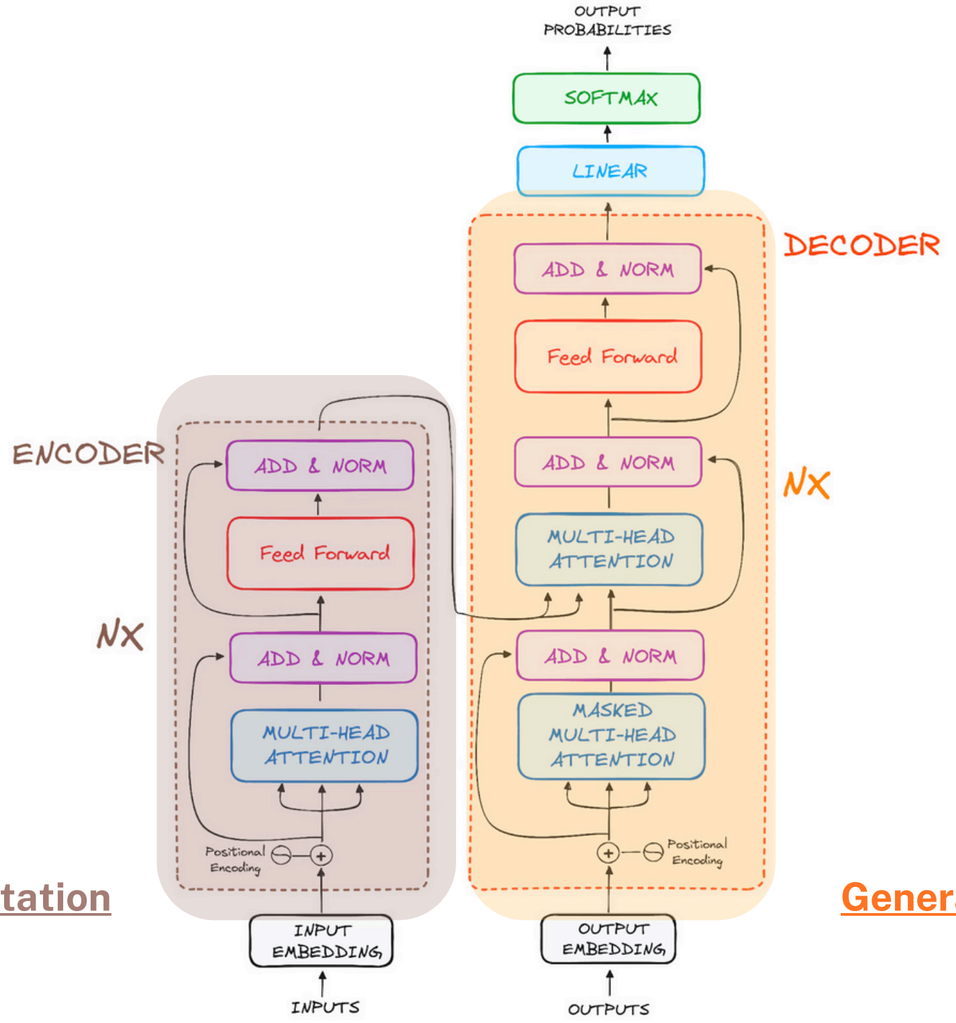
Generation

Challenges

- Lack of task-specific training data

October 2018

Evolution of transformer-based language model (LM)



Representation

Generation

Challenges

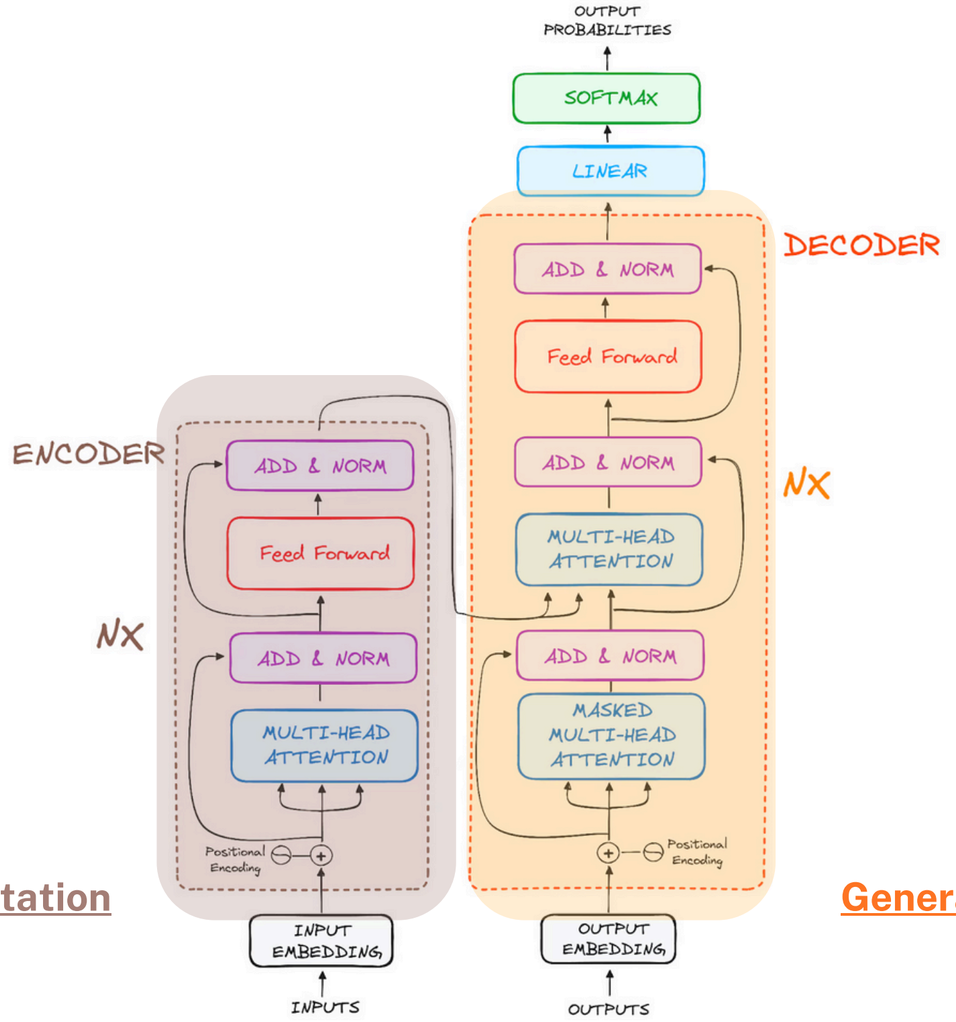
- Lack of task-specific training data

Solution

Leverage large amounts of unlabeled data to pretrain an LM to use general patterns

October 2018

Evolution of transformer-based language model (LM)



Representation

Generation

Challenges

- Lack of task-specific training data

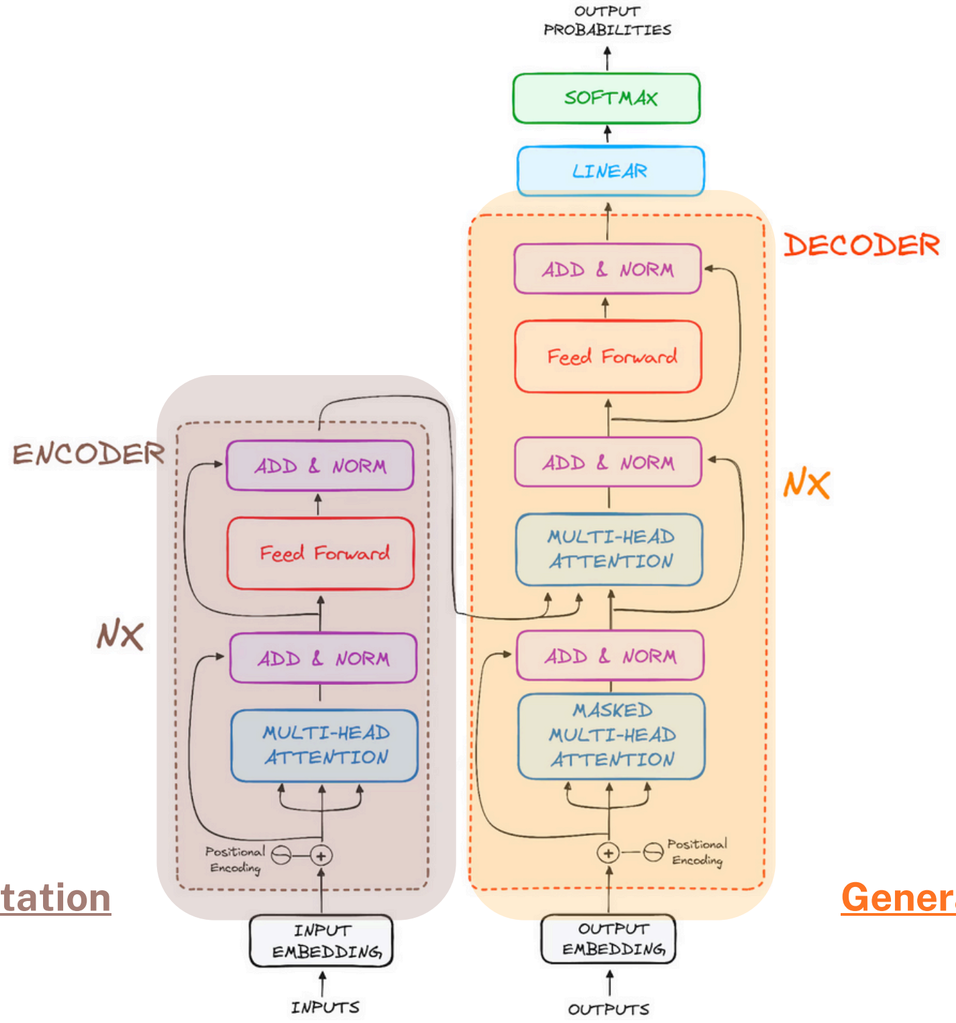
Solution

Leverage large amounts of unlabeled data to pretrain an LM to use general patterns

GPT (Generative Pretrained Transformers)

June 2018

Evolution of transformer-based language model (LM)



Representation

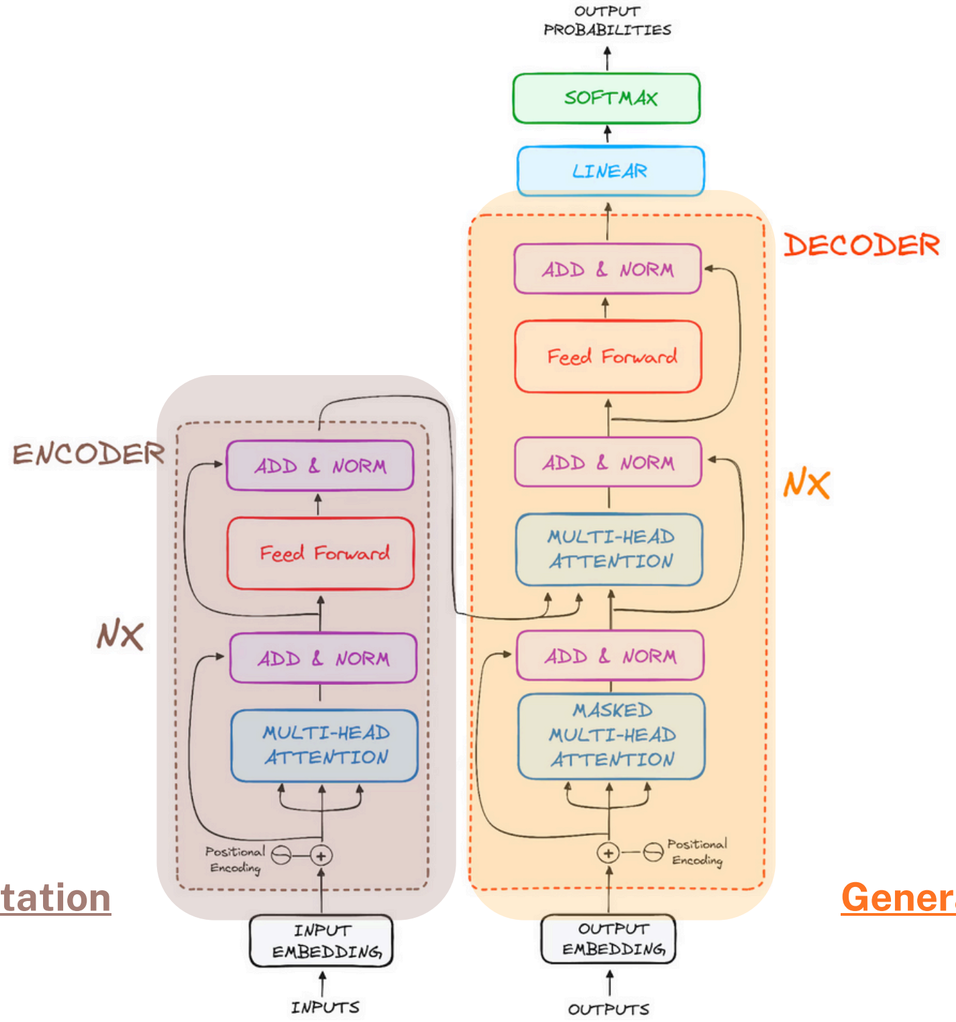
Generation

GPT (Generative Pretrained Transformers)

Fine-tuning baseline pretrained GPT with prompt (continuous stream) model fitting

June 2018

Evolution of transformer-based language model (LM)



Representation

Generation

GPT (Generative Pretrained Transformers)

- Self-supervised learning
- Zero-shot learning as the model itself serves as a knowledge base

June 2018