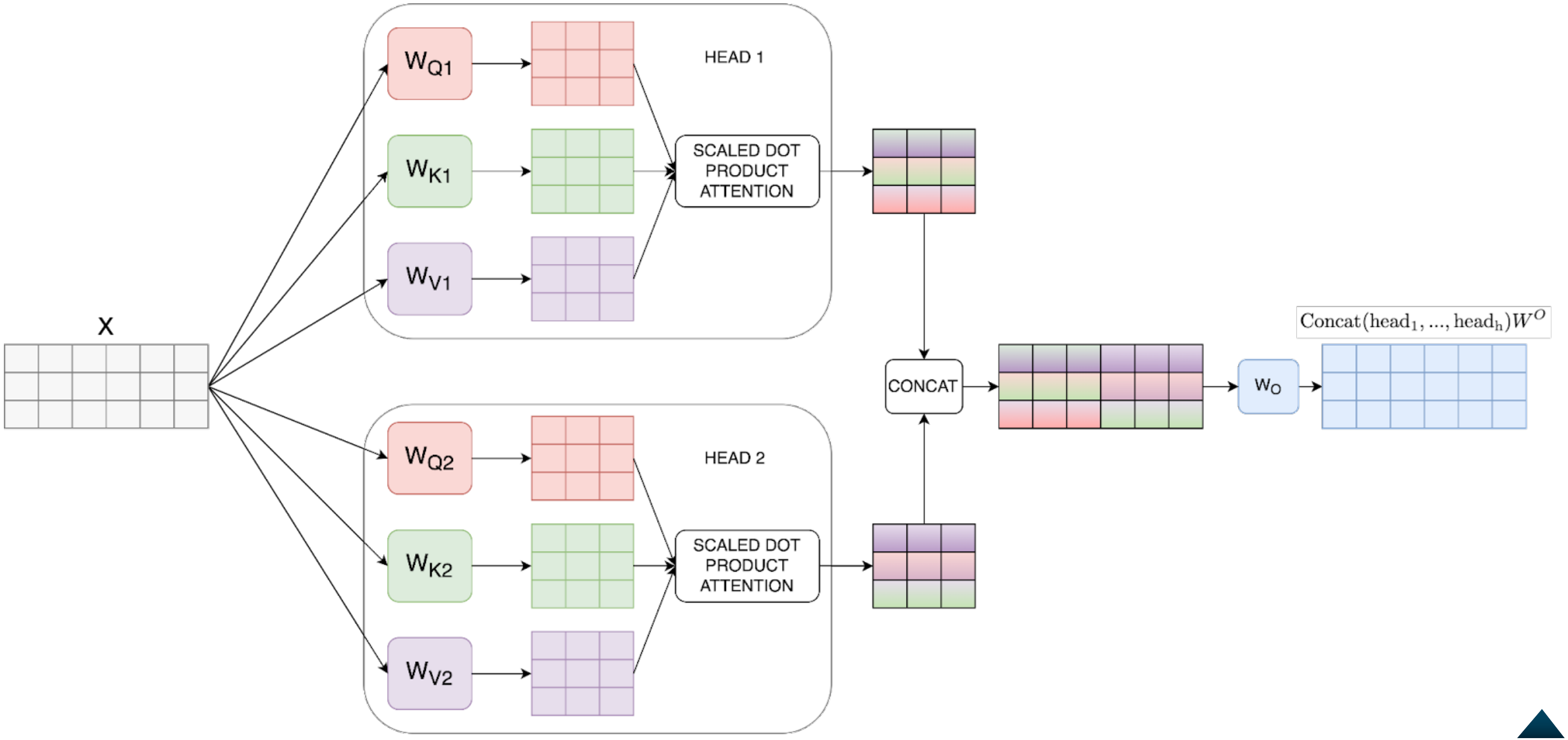
Hands on Generative AI

Transforming NLP

Data Trainers


X
W_{Q1}
W_{K1}
W_{V1}
HEAD 1
SCALED DOT PRODUCT ATTENTION
W_{Q2}
W_{K2}
W_{V2}
HEAD 2
SCALED DOT PRODUCT ATTENTION
CONCAT
W_O
$\text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$

In summary:

Multi-Head Attention is a vital component within Transformers.

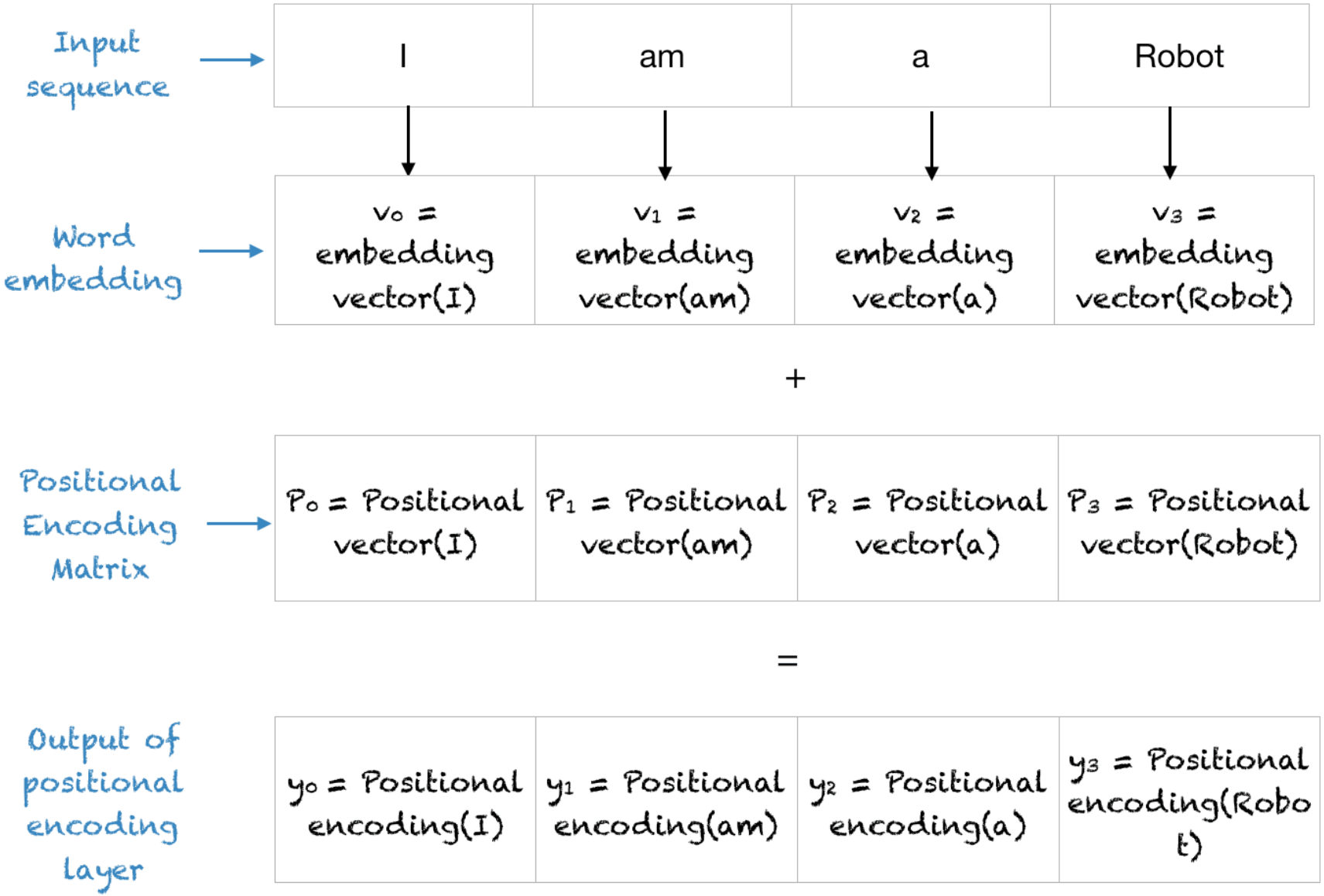
An example:

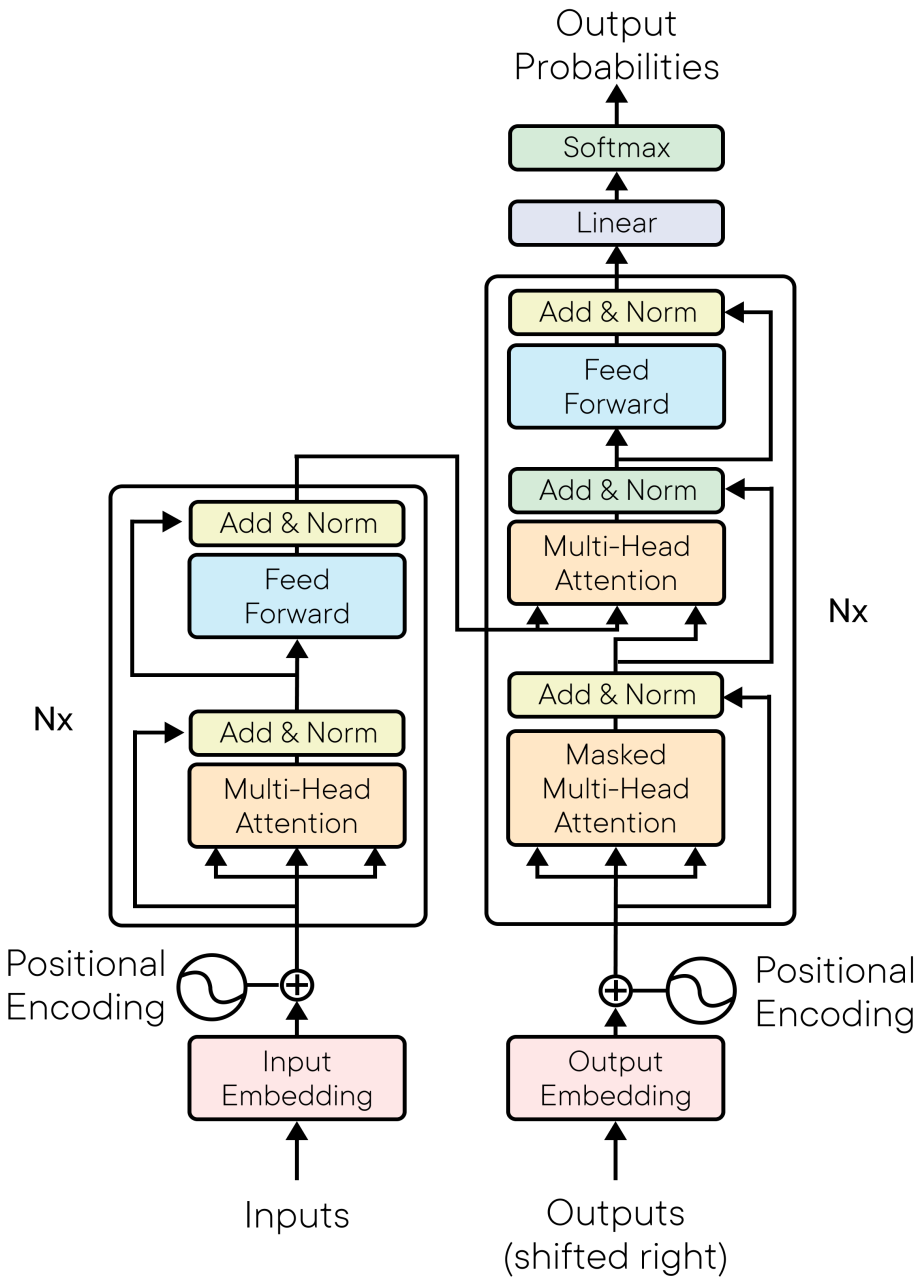
- **Positive:**

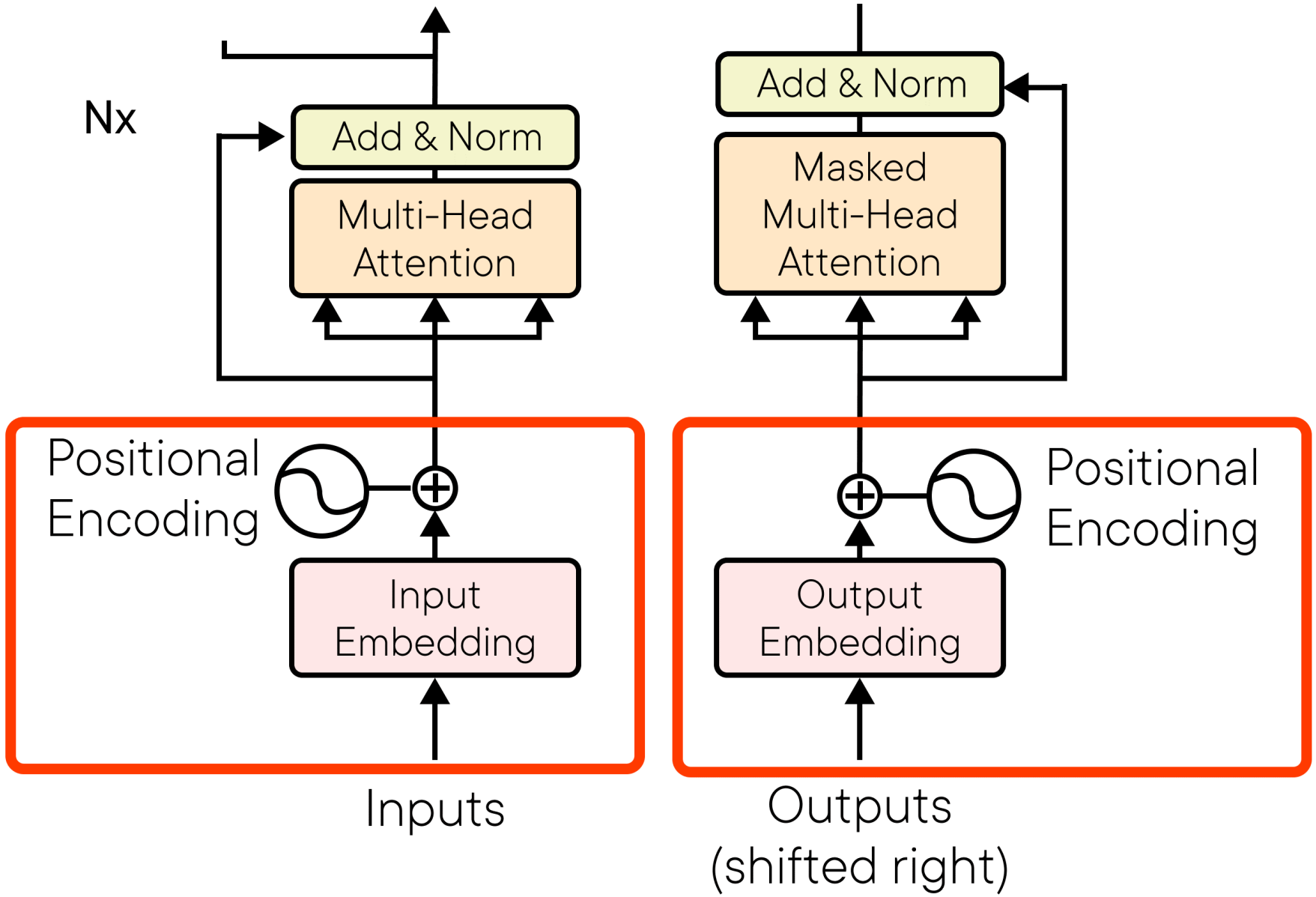
  I loved the restaurant even though the pasta was mediocre

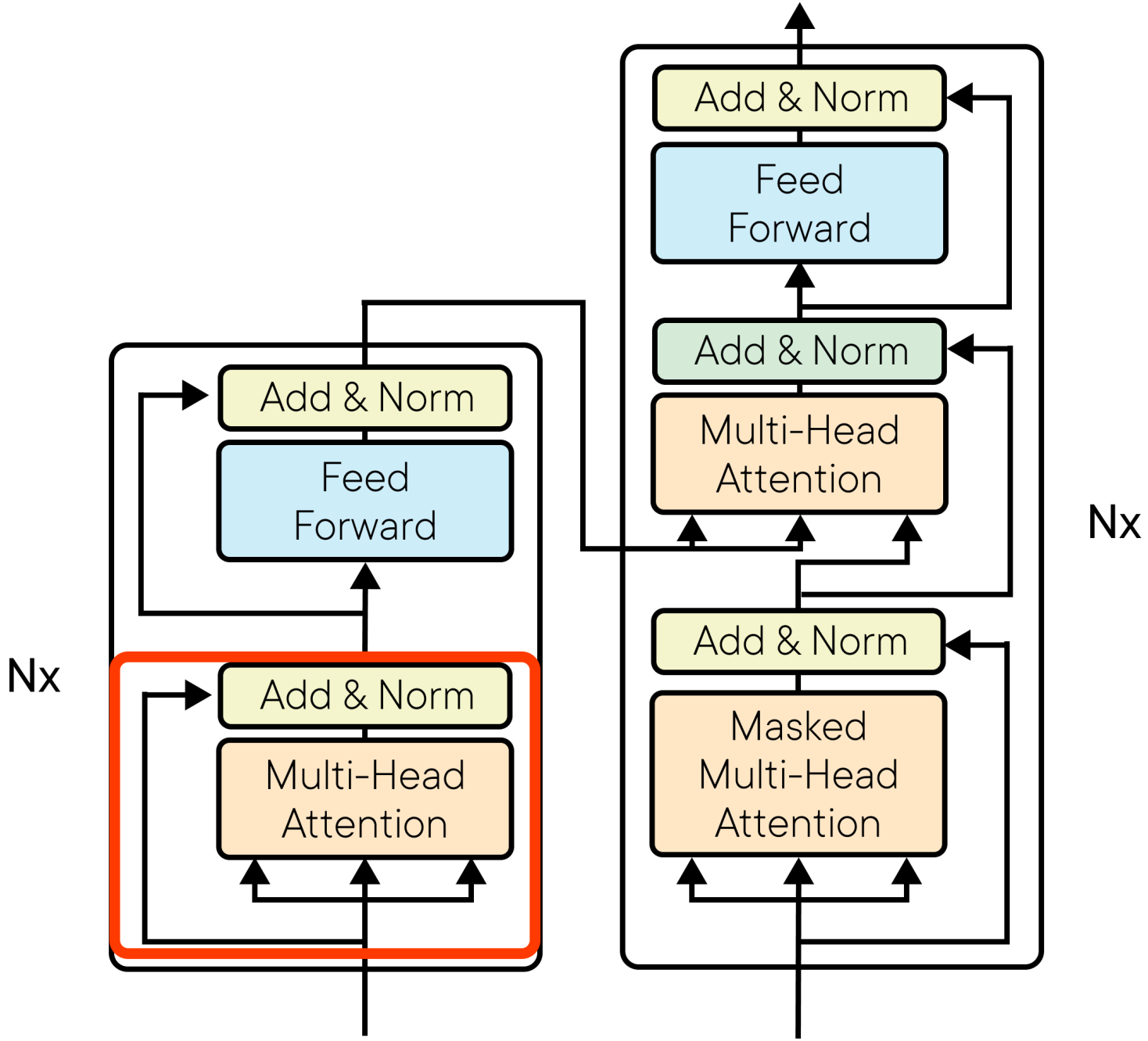
- **Negative:**

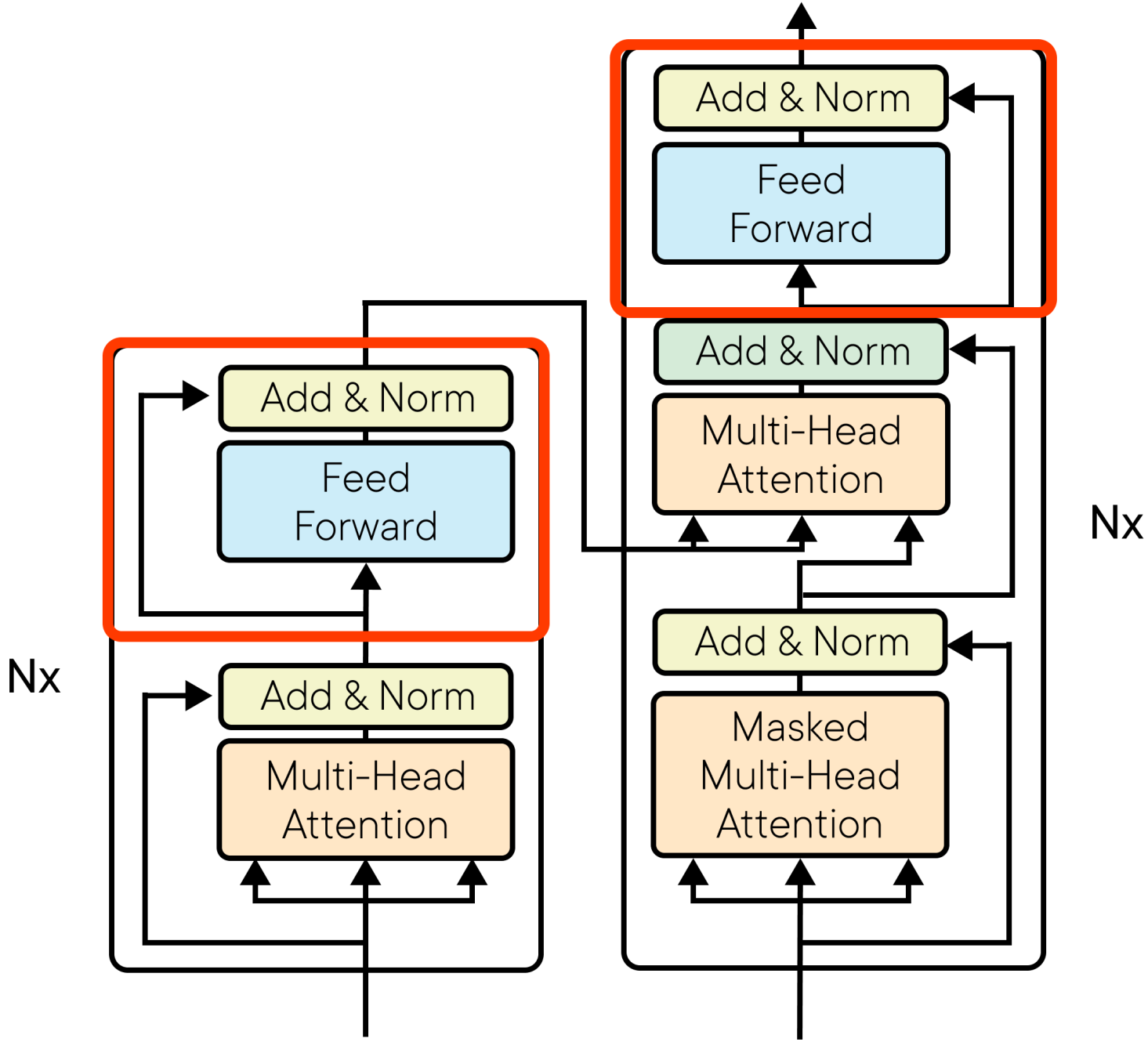
  I hate this place even though it has fantastic steak

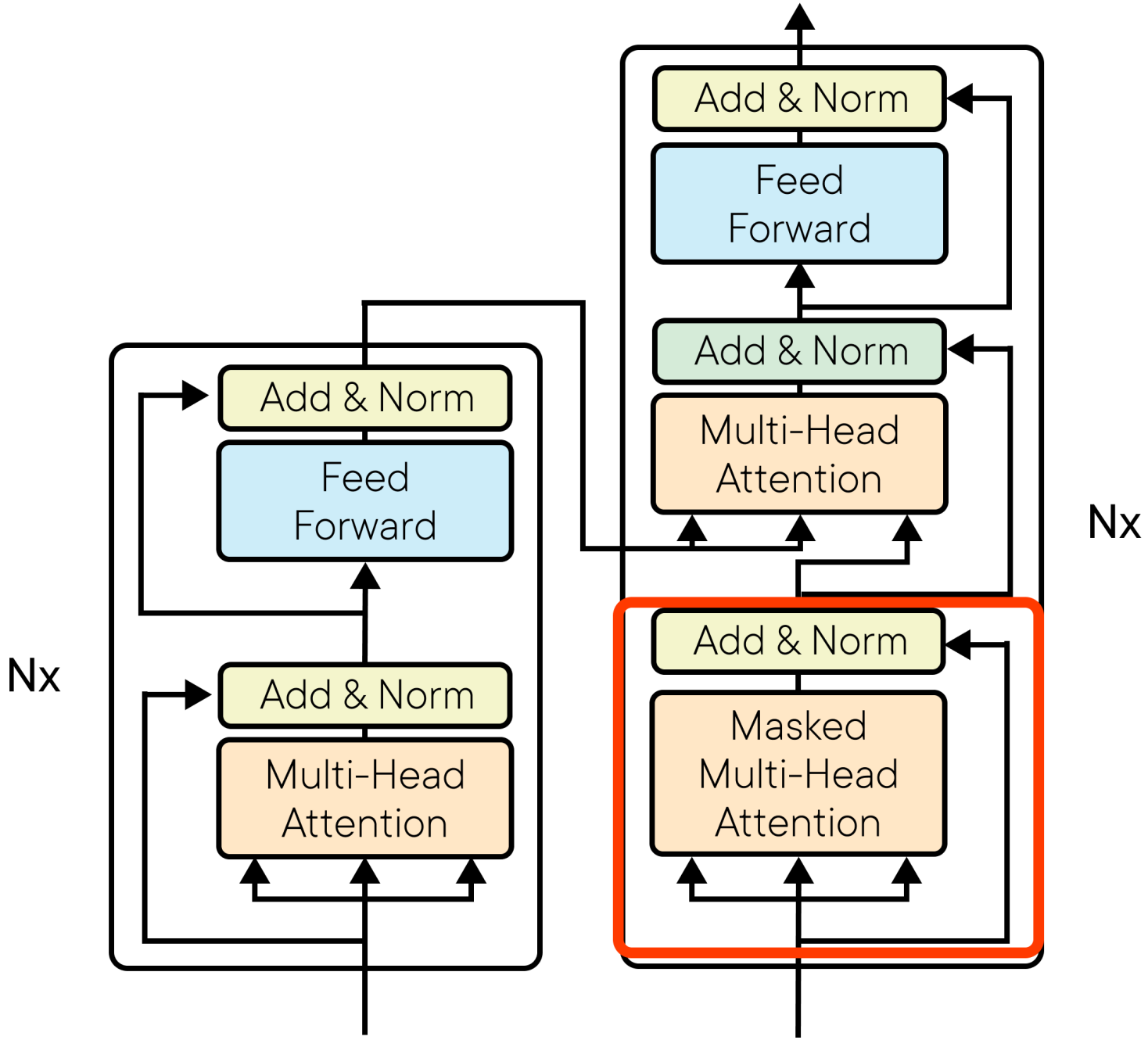

Input sequence
I
am
a
Robot
Word embedding
v0 = embedding vector(I)
v1 = embedding vector(am)
v2 = embedding vector(a)
v3 = embedding vector(Robot)
+
Positional Encoding Matrix
P0 = Positional vector(I)
P1 = Positional vector(am)
P2 = Positional vector(a)
P3 = Positional vector(Robot)
=
Output of positional encoding layer
y0 = Positional encoding(I)
y1 = Positional encoding(am)
y2 = Positional encoding(a)
y3 = Positional encoding(Robot)

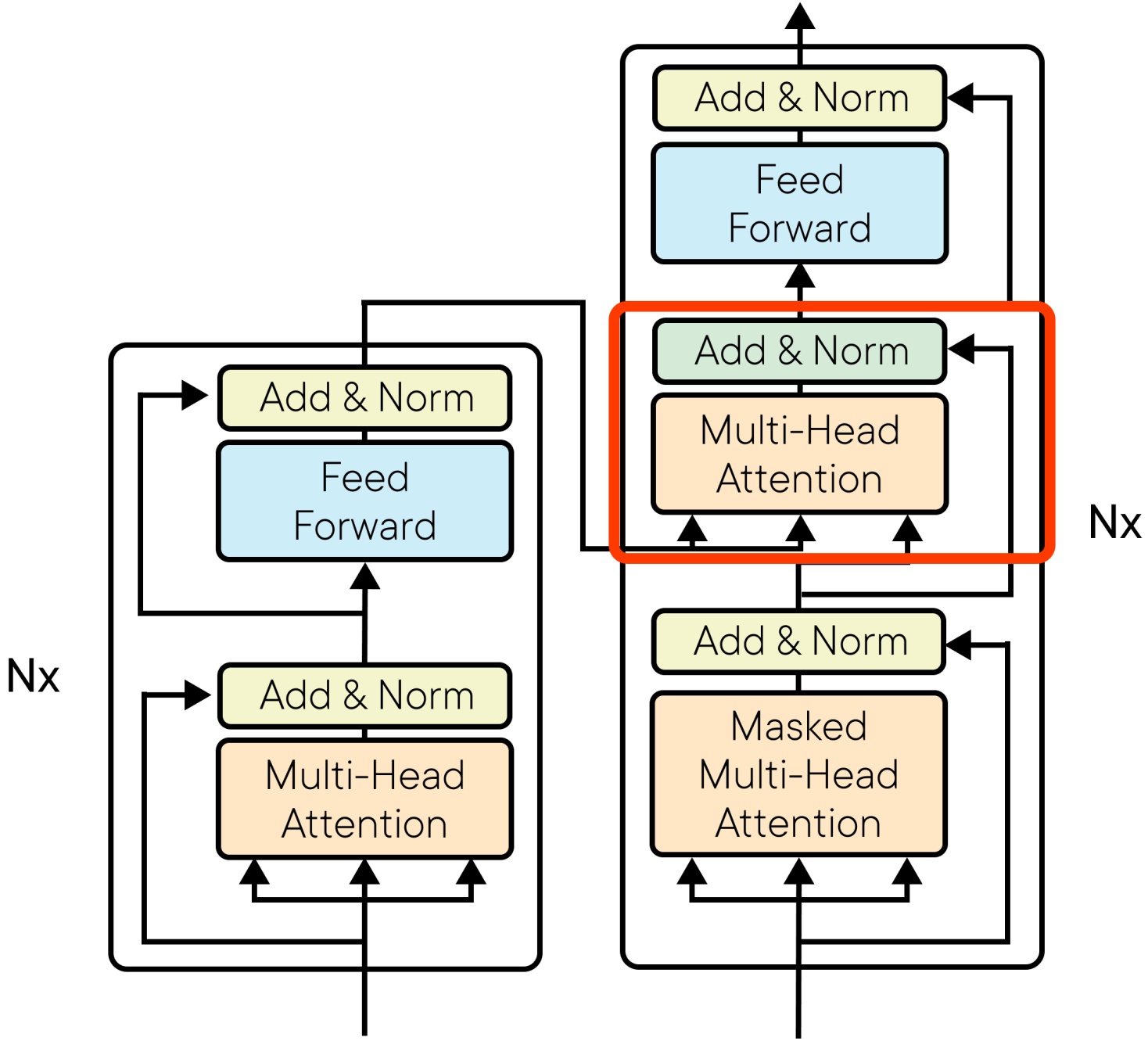

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

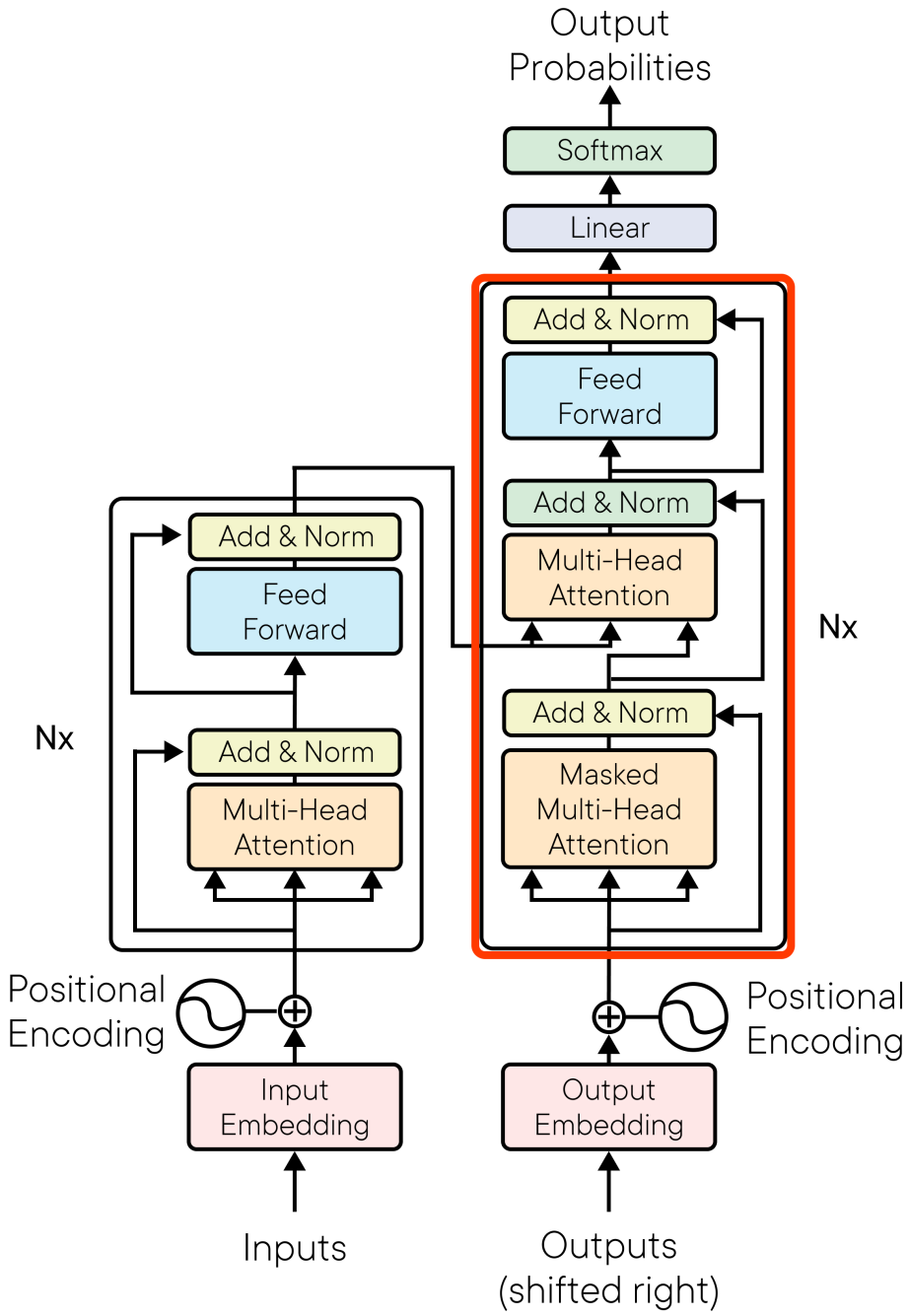

Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs
Add & Norm
Masked Multi-Head Attention
Positional Encoding
Output Embedding
Outputs (shifted right)







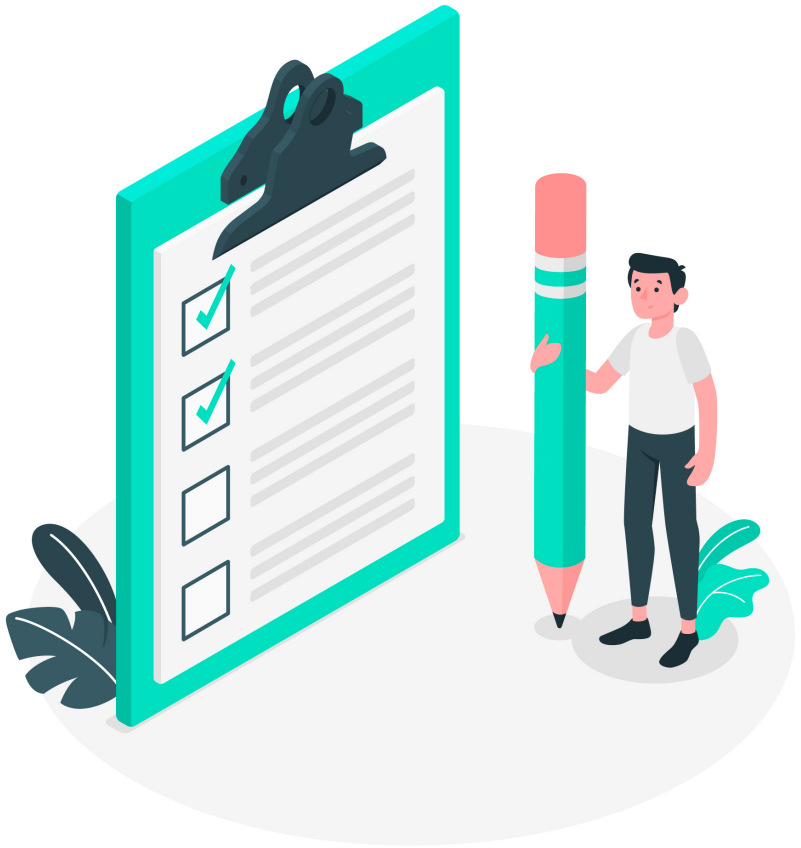



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Why do we need it?

- Transformers can parallelise computations across a GPU, which RNNs cannot
- Transformers do not have the informational bottleneck
- And Transformers have much fewer parameters for the same size of architecture than an RNN

- Our first transformer



Introducing Hugging Face

Hugging Face

- Hugging Face is an open-source library and plataforma that provides a wide range of tolos and models for NLP tasks.

Features

- **Pretrained Models:** Hugging Face provides a vast collection of pretrained models, including popular architectures like BERT, GPT and many more.
- **Transformers Library:** The Transformers Library, developed by Hugging Face, is a go-to resource for NLP practitioners.
- **Datasets library:** At your disposal, you have the datasets library to download tons of famous datasets along with their metrics to compare your models with the current state of the art.

In code

1. Download and load the tokenizer and model

index.py

Data Trainers
Transforming NLP

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'],
test_size=0.2, random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True,
padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
```

In code

2. Load the IMDB dataset from Hugging Face datasets library

index.py

Data Trainers
Transforming NLP

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'],
test_size=0.2, random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True,
padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
```

In code

2. Split the dataset into training and testing sets

index.py

Data Trainers
Transforming NLP

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'],
test_size=0.2, random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True,
padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True,
```

In code

4. Tokenize the datasets

index.py

Data Trainers
Transforming NLP

```
import tensorflow as tf
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Download and load the tokenizer and model
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = TFAutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

# Load the IMDb dataset from Hugging Face datasets library
dataset = load_dataset("imdb")

# Split the dataset into training and testing sets
train_dataset, test_dataset = train_test_split(dataset['train'],
test_size=0.2, random_state=42)

# Tokenize the training dataset
train_encodings = tokenizer(train_dataset['text'], truncation=True,
padding=True)
train_labels = train_dataset['label']

# Tokenize the testing dataset
test_encodings = tokenizer(test_dataset['text'], truncation=True,
```

In code

Create TensorFlow datasets

index.py

Data Trainers
Transforming NLP

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)


# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

In code

Configure training parameters

index.py

Data Trainers
Transforming NLP

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)

# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])

# Train the model
model.fit(train_dataset, epochs=3)

# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

In code

Train the model

index.py

Data Trainers
Transforming NLP

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)


# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

In code

Evaluate the model

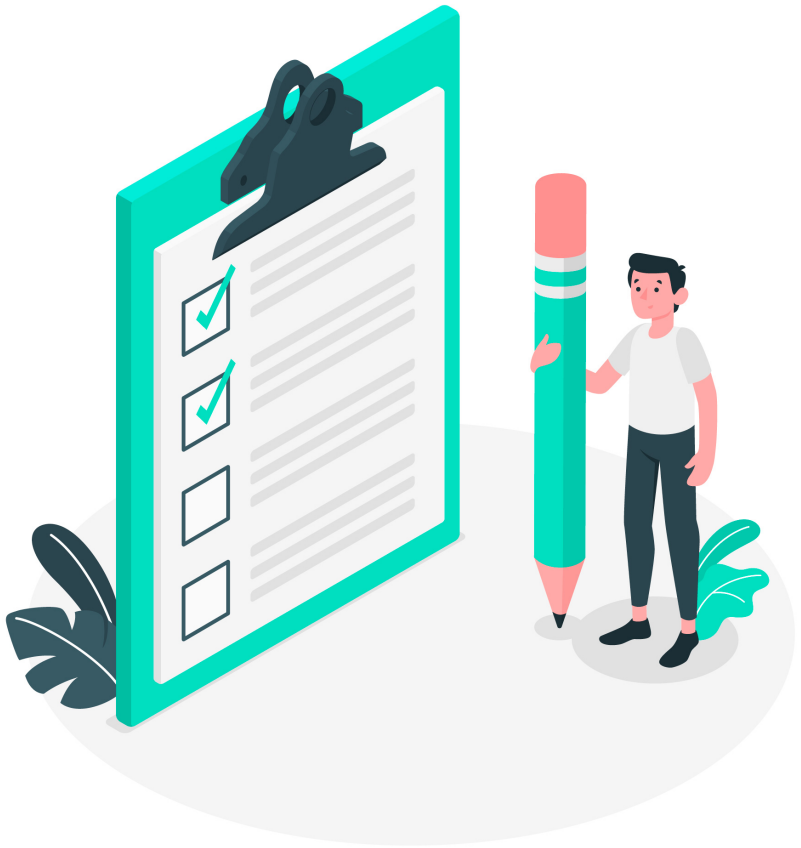
index.py

Data Trainers
Transforming NLP

```
# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings),
train_labels)).batch(16)
test_dataset = tf.data.Dataset.from_tensor_slices((dict(test_encodings),
test_labels)).batch(16)


# Configure training parameters
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])


# Train the model
model.fit(train_dataset, epochs=3)


# Evaluate the model
results = model.evaluate(test_dataset)
print("Test Loss:", results[0])
print("Test Accuracy:", results[1])
```

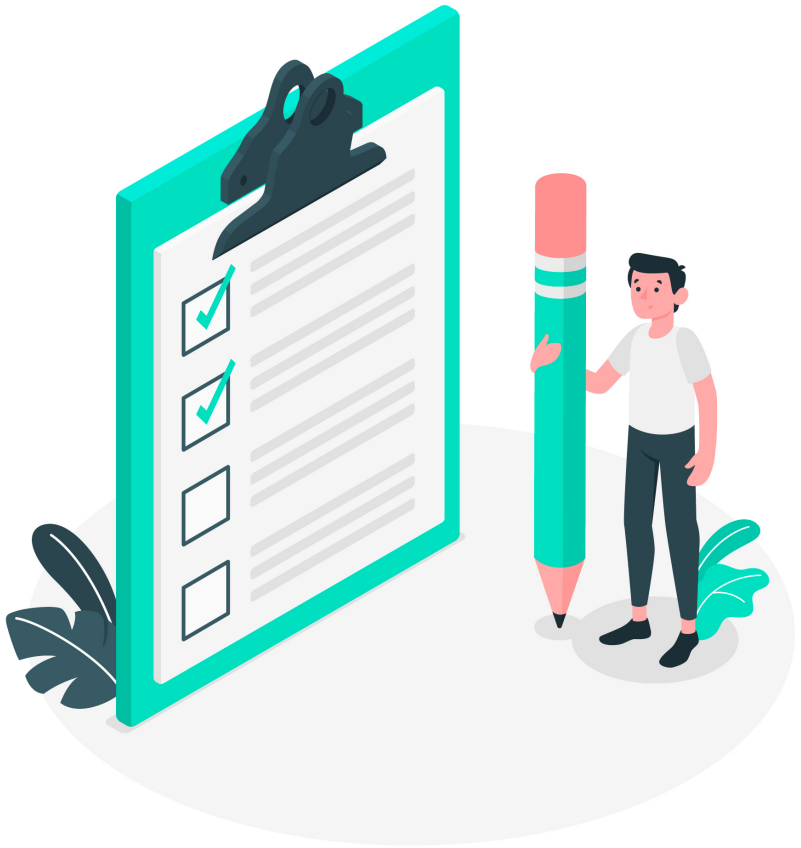
Demo

- Prompt Engineering with Flan-t5



Demo

- Transfer learning on Transformers with Hugging Face



Summary

- The Transformer architecture solved all of the significant challenges we had with RNNs for sequence models
- With Hugging Face, now it most straightforward tan ever just to graba checkpoint from someone and finetune it to your needs
- Some models like T5 can perform on multiple task because that's how they were trained and we can leverage it