# SYMBOLIC CALCULATION

## Fast Accurate Symbolic Empirical Representation Of Histograms

Name - Shivam Maheshwari

## Task 1. Dataset Preprocessing

#### Generating the dataset

In [1]:
```
import sympy as sp
import random
import pandas as pd

# Define the symbolic variable
x = sp.Symbol('x', real=True)

base_functions = [
    sp.sin(x), sp.cos(x), sp.exp(x), sp.log(1+x), sp.tan(x),
    sp.sinh(x), sp.cosh(x), sp.asin(x), sp.acos(x), sp.atan(x),
    x, x**2, x**3
]

def generate_random_expression(num_terms=3):

    expr = 0
    for _ in range(num_terms):
        f = random.choice(base_functions)
        coeff = random.randint(-5, 5)
        expr += coeff * f
    return expr

# Number of expressions to generate
num_expressions = 20000
```

```
dataset = []
for _ in range(num_expressions):
    terms = random.randint(1, 3)
    expr = generate_random_expression(num_terms=terms)

    # Skip if expression is 0
    if expr == 0:
        continue

    # Compute series expansion around x=0 up to x^4
    # Sympy's 'order=5' generates terms up to x^4, then we remove O(x^5)
    series_expr = sp.series(expr, x, 0, 5).removeO()

    dataset.append((str(expr), str(series_expr)))


df = pd.DataFrame(dataset, columns=["function", "taylor_expansion"])

df.drop_duplicates(inplace=True)

# Save to CSV
df.to_csv("taylor_dataset_extended.csv", index=False)

print(df.head(10))
print(f"Total unique expressions: {len(df)}")
```

```
                                 function  \
0                                 -4*x**2
1     -5*sin(x) + 2*sinh(x) - 2*atan(x)
2                  -cosh(x) + 3*asin(x)
3                             5*acos(x)
4               -log(x + 1) + 3*asin(x)
5        5*tan(x) - 5*acos(x) + asin(x)
6  -2*exp(x) - 3*log(x + 1) + 2*sinh(x)
7                   -x**3 + 5*atan(x)
8                     -4*x - 5*exp(x)
9                   5*exp(x) - tan(x)

                               taylor_expansion
0                                       -4*x**2
1                             11*x**3/6 - 5*x
2        -x**4/24 + x**3/2 - x**2/2 + 3*x - 1
3                    -5*x**3/6 - 5*x + 5*pi/2
4              x**4/4 + x**3/6 + x**2/2 + 2*x
5                   8*x**3/3 + 11*x - 5*pi/2
6          2*x**4/3 - x**3 + x**2/2 - 3*x - 2
7                             -8*x**3/3 + 5*x
8  -5*x**4/24 - 5*x**3/6 - 5*x**2/2 - 9*x - 5
9     5*x**4/24 + x**3/2 + 5*x**2/2 + 4*x + 5
Total unique expressions: 9181
```

In [ ]:
```
df_backup = df.copy()
```

### Tokenising the dataset

In [2]:
```
def tokenize_expression(expr):
    return list(expr)

def add_special_tokens(expr):
    # Insert single-character tokens for start/end
    return f"¶{expr}µ"

df["taylor_expansion"] = df["taylor_expansion"].apply(add_special_tokens)

# Create new columns in the DataFrame with token lists
df["function_tokens"] = df["function"].apply(tokenize_expression)
```

```
df["taylor_tokens"] = df["taylor_expansion"].apply(tokenize_expression)
print(df.head())
```

```
                              function                          taylor_expansion  \
0                              -4*x**2                                ¶-4*x**2µ
1  -5*sin(x) + 2*sinh(x) - 2*atan(x)                     ¶11*x**3/6 - 5*xµ
2                 -cosh(x) + 3*asin(x)  ¶-x**4/24 + x**3/2 - x**2/2 + 3*x - 1µ
3                            5*acos(x)                 ¶-5*x**3/6 - 5*x + 5*pi/2µ
4              -log(x + 1) + 3*asin(x)       ¶x**4/4 + x**3/6 + x**2/2 + 2*xµ

                                   function_tokens  \
0                          [-, 4, *, x, *, *, 2]
1  [-, 5, *, s, i, n, (, x, ),  , +,  , 2, *, s, ...
2  [-, c, o, s, h, (, x, ),  , +,  , 3, *, a, s, ...
3                    [5, *, a, c, o, s, (, x, )]
4  [-, l, o, g, (, x,  , +,  , 1, ),  , +,  , 3, ...

                                     taylor_tokens
0                    [¶, -, 4, *, x, *, *, 2, µ]
1  [¶, 1, 1, *, x, *, *, 3, /, 6,  , -,  , 5, *, ...
2  [¶, -, x, *, *, 4, /, 2, 4,  , +,  , x, *, *, ...
3  [¶, -, 5, *, x, *, *, 3, /, 6,  , -,  , 5, *, ...
4  [¶, x, *, *, 4, /, 4,  , +,  , x, *, *, 3, /, ...
```

# Task 2: LSTM Model

### Training the LSTM model

In [3]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import LSTM, Dense, Embedding, Input
from tensorflow.keras.models import Model

# 1) Tokenize and Pad
function_texts = df["function"].tolist()
taylor_texts = df["taylor_expansion"].tolist()

# Character-level tokenizer
```

```
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(function_texts + taylor_texts)


input_sequences = tokenizer.texts_to_sequences(function_texts)
output_sequences = tokenizer.texts_to_sequences(taylor_texts)

# debug
for i in range(3):
    print("Original expansion:", taylor_texts[i])
    print("Token IDs:", output_sequences[i][:15])


max_input_length = 50
max_output_length = 50
input_sequences = pad_sequences(input_sequences, maxlen=max_input_length, padding='post')
output_sequences = pad_sequences(output_sequences, maxlen=max_output_length, padding='post')

vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 50
latent_dim = 128

# right and left shift the decoder
decoder_input_data = np.zeros_like(output_sequences)
decoder_input_data[:, 1:] = output_sequences[:, :-1]

decoder_target_data = np.zeros_like(output_sequences)
decoder_target_data[:, :-1] = output_sequences[:, 1:]
decoder_target_data = np.expand_dims(decoder_target_data, -1)

# 4) Define the Encoder
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(vocab_size, embedding_dim)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
_, encoder_hidden_state, encoder_cell_state = encoder_lstm(enc_emb)
encoder_states = [encoder_hidden_state, encoder_cell_state]

# 5) Define the Decoder
decoder_inputs = Input(shape=(None,))
dec_emb = Embedding(vocab_size, embedding_dim)(decoder_inputs)
decoder_lstm_layer = LSTM(latent_dim, return_sequences=True, return_state=True)
```

```
decoder_outputs, _, _ = decoder_lstm_layer(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# 6) Build and Compile the Seq2seq Model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
model.summary()

# 7) Train the Model
history_lstm = model.fit(
    [input_sequences, decoder_input_data],
    decoder_target_data,
    batch_size=64,
    epochs=100,
    validation_split=0.2
)
```

```
Original expansion: ¶-4*x**2µ
Token IDs: [14, 4, 11, 1, 3, 1, 1, 5, 15]
Original expansion: ¶11*x**3/6 - 5*xµ
Token IDs: [14, 20, 20, 1, 3, 1, 1, 6, 10, 22, 2, 4, 2, 13, 1]
Original expansion: ¶-x**4/24 + x**3/2 - x**2/2 + 3*x - 1µ
Token IDs: [14, 4, 3, 1, 1, 11, 10, 5, 11, 2, 7, 2, 3, 1, 1]
```

**Model: "functional"**

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---:|---|
| input_layer (InputLayer) | (None, None) | 0 | - |
| input_layer_1 (InputLayer) | (None, None) | 0 | - |
| embedding (Embedding) | (None, None, 50) | 1,650 | input_layer[0][0] |
| embedding_1 (Embedding) | (None, None, 50) | 1,650 | input_layer_1[0][0] |
| lstm (LSTM) | [(None, 128), (None, 128), (None, 128)] | 91,648 | embedding[0][0] |
| lstm_1 (LSTM) | [(None, None, 128), (None, 128), (None, 128)] | 91,648 | embedding_1[0][0], lstm[0][1], lstm[0][2] |
| dense (Dense) | (None, None, 33) | 4,257 | lstm_1[0][0] |

**Total params:** 190,853 (745.52 KB)

**Trainable params:** 190,853 (745.52 KB)

**Non-trainable params:** 0 (0.00 B)

```
Epoch 1/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 1.9160 - val_loss: 0.7859
Epoch 2/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.6573 - val_loss: 0.4978
Epoch 3/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.4602 - val_loss: 0.4390
Epoch 4/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.4191 - val_loss: 0.4104
Epoch 5/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.3898 - val_loss: 0.3882
Epoch 6/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.3729 - val_loss: 0.3693
Epoch 7/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.3521 - val_loss: 0.3590
Epoch 8/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.3365 - val_loss: 0.3431
Epoch 9/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.3215 - val_loss: 0.3324
Epoch 10/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.3134 - val_loss: 0.3196
Epoch 11/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.3027 - val_loss: 0.3139
Epoch 12/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2968 - val_loss: 0.3062
Epoch 13/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2891 - val_loss: 0.2990
Epoch 14/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2812 - val_loss: 0.2927
Epoch 15/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2762 - val_loss: 0.2865
Epoch 16/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2718 - val_loss: 0.2799
Epoch 17/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2648 - val_loss: 0.2754
Epoch 18/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.2567 - val_loss: 0.2672
Epoch 19/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.2512 - val_loss: 0.2605
Epoch 20/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2415 - val_loss: 0.2571
Epoch 21/100
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2372 - val_loss: 0.2526
Epoch 22/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2325 - val_loss: 0.2491
Epoch 23/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2259 - val_loss: 0.2443
Epoch 24/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2210 - val_loss: 0.2346
Epoch 25/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.2128 - val_loss: 0.2286
Epoch 26/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.2084 - val_loss: 0.2278
Epoch 27/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.2048 - val_loss: 0.2235
Epoch 28/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.2028 - val_loss: 0.2211
Epoch 29/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1962 - val_loss: 0.2113
Epoch 30/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1906 - val_loss: 0.2067
Epoch 31/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1856 - val_loss: 0.2054
Epoch 32/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1813 - val_loss: 0.2034
Epoch 33/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1782 - val_loss: 0.2003
Epoch 34/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1748 - val_loss: 0.1948
Epoch 35/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.1688 - val_loss: 0.1945
Epoch 36/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.1690 - val_loss: 0.1924
Epoch 37/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1648 - val_loss: 0.1912
Epoch 38/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1632 - val_loss: 0.1880
Epoch 39/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1590 - val_loss: 0.1846
Epoch 40/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1568 - val_loss: 0.1859
Epoch 41/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1562 - val_loss: 0.1838
```

```
Epoch 42/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1529 - val_loss: 0.1826
Epoch 43/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1510 - val_loss: 0.1806
Epoch 44/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.1474 - val_loss: 0.1772
Epoch 45/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.1431 - val_loss: 0.1750
Epoch 46/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1412 - val_loss: 0.1747
Epoch 47/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1382 - val_loss: 0.1721
Epoch 48/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.1359 - val_loss: 0.1714
Epoch 49/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1314 - val_loss: 0.1694
Epoch 50/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1301 - val_loss: 0.1702
Epoch 51/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1266 - val_loss: 0.1659
Epoch 52/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1229 - val_loss: 0.1671
Epoch 53/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.1205 - val_loss: 0.1617
Epoch 54/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.1166 - val_loss: 0.1609
Epoch 55/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1146 - val_loss: 0.1611
Epoch 56/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.1133 - val_loss: 0.1579
Epoch 57/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1092 - val_loss: 0.1551
Epoch 58/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1066 - val_loss: 0.1550
Epoch 59/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1026 - val_loss: 0.1556
Epoch 60/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1021 - val_loss: 0.1516
Epoch 61/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.0993 - val_loss: 0.1499
Epoch 62/100
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0959 - val_loss: 0.1537
Epoch 63/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0937 - val_loss: 0.1535
Epoch 64/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0929 - val_loss: 0.1508
Epoch 65/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0903 - val_loss: 0.1490
Epoch 66/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0859 - val_loss: 0.1507
Epoch 67/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0840 - val_loss: 0.1509
Epoch 68/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0828 - val_loss: 0.1479
Epoch 69/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0826 - val_loss: 0.1462
Epoch 70/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.0780 - val_loss: 0.1489
Epoch 71/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.0773 - val_loss: 0.1448
Epoch 72/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0755 - val_loss: 0.1455
Epoch 73/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0718 - val_loss: 0.1489
Epoch 74/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0712 - val_loss: 0.1454
Epoch 75/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0669 - val_loss: 0.1487
Epoch 76/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0697 - val_loss: 0.1490
Epoch 77/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0678 - val_loss: 0.1453
Epoch 78/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0648 - val_loss: 0.1480
Epoch 79/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0649 - val_loss: 0.1478
Epoch 80/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.0622 - val_loss: 0.1480
Epoch 81/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0615 - val_loss: 0.1462
Epoch 82/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0608 - val_loss: 0.1456
```

```
Epoch 83/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0589 - val_loss: 0.1471
Epoch 84/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0566 - val_loss: 0.1486
Epoch 85/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0553 - val_loss: 0.1513
Epoch 86/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0558 - val_loss: 0.1461
Epoch 87/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0522 - val_loss: 0.1492
Epoch 88/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0514 - val_loss: 0.1553
Epoch 89/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0514 - val_loss: 0.1540
Epoch 90/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0503 - val_loss: 0.1489
Epoch 91/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0485 - val_loss: 0.1528
Epoch 92/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0476 - val_loss: 0.1566
Epoch 93/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0482 - val_loss: 0.1530
Epoch 94/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0466 - val_loss: 0.1551
Epoch 95/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0465 - val_loss: 0.1557
Epoch 96/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0460 - val_loss: 0.1562
Epoch 97/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.0438 - val_loss: 0.1525
Epoch 98/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0421 - val_loss: 0.1541
Epoch 99/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0421 - val_loss: 0.1550
Epoch 100/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.0388 - val_loss: 0.1584
```

## Task 3: Transformer Model

In [4]:
```
from tensorflow.keras.layers import LayerNormalization, Dropout, Input, Lambda

# Positional Encoding Function
def get_positional_encoding(max_len, d_model):
    pos_enc = np.zeros((max_len, d_model))
    for pos in range(max_len):
        for i in range(0, d_model, 2):
            pos_enc[pos, i] = np.sin(pos / (10000 ** ((2 * i) / d_model)))
            if i + 1 < d_model:
                pos_enc[pos, i + 1] = np.cos(pos / (10000 ** ((2 * (i+1)) / d_model)))
    return tf.cast(pos_enc, dtype=tf.float32)

# Transformer Encoder Block
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    attn_output = tf.keras.layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads
    )(inputs, inputs)
    attn_output = Dropout(dropout)(attn_output)
    out1 = LayerNormalization(epsilon=1e-6)(inputs + attn_output)

    ffn_output = Dense(ff_dim, activation="relu")(out1)
    ffn_output = Dense(inputs.shape[-1])(ffn_output)
    ffn_output = Dropout(dropout)(ffn_output)
    return LayerNormalization(epsilon=1e-6)(out1 + ffn_output)

# Transformer Decoder Block with causal masking
def transformer_decoder(inputs, enc_output, head_size, num_heads, ff_dim, dropout=0):
    causal_mask = Lambda(
        lambda x: tf.tile(
            tf.expand_dims(tf.linalg.band_part(tf.ones((tf.shape(x)[1], tf.shape(x)[1])), -1, 0), 0),
            [tf.shape(x)[0], 1, 1]
        )
    )(inputs)

    attn1 = tf.keras.layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads
    )(inputs, inputs, attention_mask=causal_mask)
    attn1 = Dropout(dropout)(attn1)
    out1 = LayerNormalization(epsilon=1e-6)(inputs + attn1)
```

```
    attn2 = tf.keras.layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads
    )(out1, enc_output)
    attn2 = Dropout(dropout)(attn2)
    out2 = LayerNormalization(epsilon=1e-6)(out1 + attn2)

    ffn_output = Dense(ff_dim, activation="relu")(out2)
    ffn_output = Dense(out2.shape[-1])(ffn_output)
    ffn_output = Dropout(dropout)(ffn_output)
    return LayerNormalization(epsilon=1e-6)(out2 + ffn_output)

# hyperparameters
embedding_dim = 64
head_size = 64
num_heads = 4
ff_dim = 128
num_encoder_layers = 2
num_decoder_layers = 2
dropout_rate = 0.1

# Build the Model
encoder_inputs = Input(shape=(max_input_length,))
decoder_inputs = Input(shape=(max_output_length,))

# Encoder embedding with positional encoding
enc_emb = Embedding(vocab_size, embedding_dim, mask_zero=True)(encoder_inputs)
pos_encoding_enc = get_positional_encoding(max_input_length, embedding_dim)
enc_emb = enc_emb + pos_encoding_enc

enc_output = enc_emb
for _ in range(num_encoder_layers):
    enc_output = transformer_encoder(enc_output, head_size, num_heads, ff_dim, dropout_rate)

# Decoder embedding with positional encoding
dec_emb = Embedding(vocab_size, embedding_dim, mask_zero=True)(decoder_inputs)
pos_encoding_dec = get_positional_encoding(max_output_length, embedding_dim)
dec_emb = dec_emb + pos_encoding_dec

dec_output = dec_emb
for _ in range(num_decoder_layers):
    dec_output = transformer_decoder(dec_output, enc_output, head_size, num_heads, ff_dim, dropout_rate)
```

```
# Final output dense layer
outputs = Dense(vocab_size, activation="softmax")(dec_output)

transformer_model = Model([encoder_inputs, decoder_inputs], outputs)
transformer_model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="sparse_categorical_crossentropy"
)

transformer_model.summary()

# 4) Train the Model
history_transformer = transformer_model.fit(
    [input_sequences, decoder_input_data],
    decoder_target_data,
    batch_size=64,
    epochs=100,
    validation_split=0.2
)
```

**Model: "functional_1"**

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---:|---|
| input_layer_2 (InputLayer) | (None, 50) | 0 | - |
| embedding_2 (Embedding) | (None, 50, 64) | 2,112 | input_layer_2[0][0] |
| add (Add) | (None, 50, 64) | 0 | embedding_2[0][0] |
| multi_head_attention (MultiHeadAttention) | (None, 50, 64) | 66,368 | add[0][0], add[0][0] |
| dropout_1 (Dropout) | (None, 50, 64) | 0 | multi_head_attention[… |
| add_1 (Add) | (None, 50, 64) | 0 | add[0][0], dropout_1[0][0] |
| layer_normalization (LayerNormalization) | (None, 50, 64) | 128 | add_1[0][0] |
| dense_1 (Dense) | (None, 50, 128) | 8,320 | layer_normalization[0… |
| dense_2 (Dense) | (None, 50, 64) | 8,256 | dense_1[0][0] |
| dropout_2 (Dropout) | (None, 50, 64) | 0 | dense_2[0][0] |
| add_2 (Add) | (None, 50, 64) | 0 | layer_normalization[0… dropout_2[0][0] |
| layer_normalization_1 (LayerNormalization) | (None, 50, 64) | 128 | add_2[0][0] |
| multi_head_attention_1 (MultiHeadAttention) | (None, 50, 64) | 66,368 | layer_normalization_1… layer_normalization_1… |
| input_layer_3 (InputLayer) | (None, 50) | 0 | - |
| dropout_4 (Dropout) | (None, 50, 64) | 0 | multi_head_attention_… |
| embedding_3 (Embedding) | (None, 50, 64) | 2,112 | input_layer_3[0][0] |

| | | | |
|---|---|---|---|
| add_3 (Add) | (None, 50, 64) | 0 | layer_normalization_1… dropout_4[0][0] |
| add_5 (Add) | (None, 50, 64) | 0 | embedding_3[0][0] |
| layer_normalization_2 (LayerNormalization) | (None, 50, 64) | 128 | add_3[0][0] |
| lambda (Lambda) | (None, 50, 50) | 0 | add_5[0][0] |
| dense_3 (Dense) | (None, 50, 128) | 8,320 | layer_normalization_2… |
| multi_head_attention_2 (MultiHeadAttention) | (None, 50, 64) | 66,368 | add_5[0][0], add_5[0][0], lambda[0][0] |
| dense_4 (Dense) | (None, 50, 64) | 8,256 | dense_3[0][0] |
| dropout_7 (Dropout) | (None, 50, 64) | 0 | multi_head_attention_… |
| dropout_5 (Dropout) | (None, 50, 64) | 0 | dense_4[0][0] |
| add_6 (Add) | (None, 50, 64) | 0 | add_5[0][0], dropout_7[0][0] |
| add_4 (Add) | (None, 50, 64) | 0 | layer_normalization_2… dropout_5[0][0] |
| layer_normalization_4 (LayerNormalization) | (None, 50, 64) | 128 | add_6[0][0] |
| layer_normalization_3 (LayerNormalization) | (None, 50, 64) | 128 | add_4[0][0] |
| multi_head_attention_3 (MultiHeadAttention) | (None, 50, 64) | 66,368 | layer_normalization_4… layer_normalization_3… |
| dropout_9 (Dropout) | (None, 50, 64) | 0 | multi_head_attention_… |
| add_7 (Add) | (None, 50, 64) | 0 | layer_normalization_4… dropout_9[0][0] |
| layer_normalization_5 | (None, 50, 64) | 128 | add_7[0][0] |

| | | | |
|---|---|---|---|
| (LayerNormalization) | | | |
| dense_5 (Dense) | (None, 50, 128) | 8,320 | layer_normalization_5… |
| dense_6 (Dense) | (None, 50, 64) | 8,256 | dense_5[0][0] |
| dropout_10 (Dropout) | (None, 50, 64) | 0 | dense_6[0][0] |
| add_8 (Add) | (None, 50, 64) | 0 | layer_normalization_5…<br>dropout_10[0][0] |
| layer_normalization_6<br>(LayerNormalization) | (None, 50, 64) | 128 | add_8[0][0] |
| lambda_1 (Lambda) | (None, 50, 50) | 0 | layer_normalization_6… |
| multi_head_attention_4<br>(MultiHeadAttention) | (None, 50, 64) | 66,368 | layer_normalization_6…<br>layer_normalization_6…<br>lambda_1[0][0] |
| dropout_12 (Dropout) | (None, 50, 64) | 0 | multi_head_attention_… |
| add_9 (Add) | (None, 50, 64) | 0 | layer_normalization_6…<br>dropout_12[0][0] |
| layer_normalization_7<br>(LayerNormalization) | (None, 50, 64) | 128 | add_9[0][0] |
| multi_head_attention_5<br>(MultiHeadAttention) | (None, 50, 64) | 66,368 | layer_normalization_7…<br>layer_normalization_3… |
| dropout_14 (Dropout) | (None, 50, 64) | 0 | multi_head_attention_… |
| add_10 (Add) | (None, 50, 64) | 0 | layer_normalization_7…<br>dropout_14[0][0] |
| layer_normalization_8<br>(LayerNormalization) | (None, 50, 64) | 128 | add_10[0][0] |
| dense_7 (Dense) | (None, 50, 128) | 8,320 | layer_normalization_8… |
| dense_8 (Dense) | (None, 50, 64) | 8,256 | dense_7[0][0] |

| | | | |
|---|---|---|---|
| dropout_15 (Dropout) | (None, 50, 64) | 0 | dense_8[0][0] |
| add_11 (Add) | (None, 50, 64) | 0 | layer_normalization_8… dropout_15[0][0] |
| layer_normalization_9 (LayerNormalization) | (None, 50, 64) | 128 | add_11[0][0] |
| dense_9 (Dense) | (None, 50, 33) | 2,145 | layer_normalization_9… |

**Total params:** 472,161 (1.80 MB)

**Trainable params:** 472,161 (1.80 MB)

**Non-trainable params:** 0 (0.00 B)

```
Epoch 1/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 48s 172ms/step - loss: 1.4741 - val_loss: 0.5845
Epoch 2/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.5242 - val_loss: 0.4268
Epoch 3/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.4163 - val_loss: 0.3818
Epoch 4/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.3741 - val_loss: 0.3472
Epoch 5/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.3343 - val_loss: 0.3087
Epoch 6/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - loss: 0.2983 - val_loss: 0.2731
Epoch 7/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.2657 - val_loss: 0.2505
Epoch 8/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.2456 - val_loss: 0.2313
Epoch 9/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.2249 - val_loss: 0.2170
Epoch 10/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 0.2089 - val_loss: 0.2076
Epoch 11/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1969 - val_loss: 0.1981
Epoch 12/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 0.1912 - val_loss: 0.1889
Epoch 13/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.1804 - val_loss: 0.1811
Epoch 14/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1763 - val_loss: 0.1774
Epoch 15/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.1690 - val_loss: 0.1727
Epoch 16/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.1640 - val_loss: 0.1649
Epoch 17/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.1563 - val_loss: 0.1628
Epoch 18/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1515 - val_loss: 0.1604
Epoch 19/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1476 - val_loss: 0.1551
Epoch 20/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1438 - val_loss: 0.1503
Epoch 21/100
```

```
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.1410 - val_loss: 0.1461
Epoch 22/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.1374 - val_loss: 0.1439
Epoch 23/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.1339 - val_loss: 0.1453
Epoch 24/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1282 - val_loss: 0.1331
Epoch 25/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1247 - val_loss: 0.1339
Epoch 26/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.1223 - val_loss: 0.1271
Epoch 27/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - loss: 0.1160 - val_loss: 0.1238
Epoch 28/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1132 - val_loss: 0.1199
Epoch 29/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.1087 - val_loss: 0.1206
Epoch 30/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.1070 - val_loss: 0.1136
Epoch 31/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.1013 - val_loss: 0.1109
Epoch 32/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.1000 - val_loss: 0.1103
Epoch 33/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0953 - val_loss: 0.1059
Epoch 34/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0909 - val_loss: 0.1024
Epoch 35/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0911 - val_loss: 0.1018
Epoch 36/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0871 - val_loss: 0.1004
Epoch 37/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - loss: 0.0868 - val_loss: 0.0950
Epoch 38/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0836 - val_loss: 0.0960
Epoch 39/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0827 - val_loss: 0.0928
Epoch 40/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0799 - val_loss: 0.0928
Epoch 41/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0907 - val_loss: 0.0910
```

```
Epoch 42/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - loss: 0.0753 - val_loss: 0.0891
Epoch 43/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0734 - val_loss: 0.0901
Epoch 44/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0726 - val_loss: 0.0866
Epoch 45/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - loss: 0.0749 - val_loss: 0.0850
Epoch 46/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0709 - val_loss: 0.0851
Epoch 47/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0698 - val_loss: 0.0824
Epoch 48/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0677 - val_loss: 0.0834
Epoch 49/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0663 - val_loss: 0.0823
Epoch 50/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0641 - val_loss: 0.0815
Epoch 51/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0636 - val_loss: 0.0809
Epoch 52/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - loss: 0.0616 - val_loss: 0.0795
Epoch 53/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0636 - val_loss: 0.0799
Epoch 54/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0603 - val_loss: 0.0787
Epoch 55/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0598 - val_loss: 0.0763
Epoch 56/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0590 - val_loss: 0.0787
Epoch 57/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - loss: 0.0573 - val_loss: 0.0749
Epoch 58/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0588 - val_loss: 0.0764
Epoch 59/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0550 - val_loss: 0.0749
Epoch 60/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0555 - val_loss: 0.0733
Epoch 61/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0534 - val_loss: 0.0733
Epoch 62/100
```

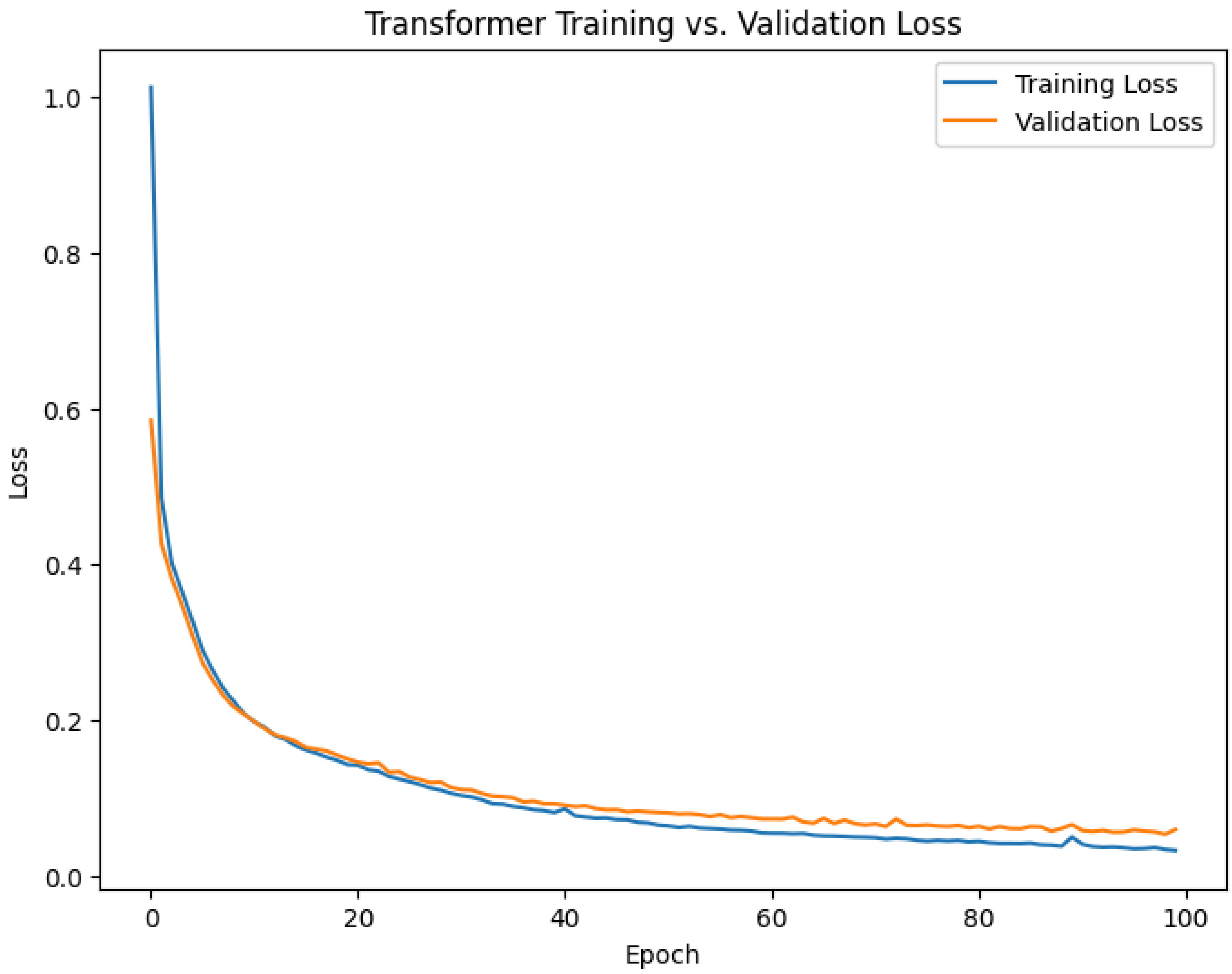
```
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - loss: 0.0544 - val_loss: 0.0731
Epoch 63/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.0528 - val_loss: 0.0755
Epoch 64/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0550 - val_loss: 0.0695
Epoch 65/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0512 - val_loss: 0.0677
Epoch 66/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - loss: 0.0504 - val_loss: 0.0740
Epoch 67/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0516 - val_loss: 0.0673
Epoch 68/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0495 - val_loss: 0.0718
Epoch 69/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0489 - val_loss: 0.0672
Epoch 70/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0484 - val_loss: 0.0654
Epoch 71/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0483 - val_loss: 0.0668
Epoch 72/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0460 - val_loss: 0.0634
Epoch 73/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0463 - val_loss: 0.0732
Epoch 74/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0478 - val_loss: 0.0651
Epoch 75/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0446 - val_loss: 0.0647
Epoch 76/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0438 - val_loss: 0.0656
Epoch 77/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0444 - val_loss: 0.0641
Epoch 78/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0444 - val_loss: 0.0636
Epoch 79/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0447 - val_loss: 0.0649
Epoch 80/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0427 - val_loss: 0.0620
Epoch 81/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - loss: 0.0412 - val_loss: 0.0638
Epoch 82/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0416 - val_loss: 0.0603
```

```
Epoch 83/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0410 - val_loss: 0.0634
Epoch 84/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0408 - val_loss: 0.0609
Epoch 85/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0405 - val_loss: 0.0606
Epoch 86/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - loss: 0.0407 - val_loss: 0.0635
Epoch 87/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0396 - val_loss: 0.0630
Epoch 88/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0387 - val_loss: 0.0575
Epoch 89/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0367 - val_loss: 0.0610
Epoch 90/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0410 - val_loss: 0.0659
Epoch 91/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0429 - val_loss: 0.0584
Epoch 92/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0369 - val_loss: 0.0571
Epoch 93/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0355 - val_loss: 0.0586
Epoch 94/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0353 - val_loss: 0.0561
Epoch 95/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0344 - val_loss: 0.0565
Epoch 96/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - loss: 0.0336 - val_loss: 0.0592
Epoch 97/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.0346 - val_loss: 0.0577
Epoch 98/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0344 - val_loss: 0.0566
Epoch 99/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 0.0336 - val_loss: 0.0535
Epoch 100/100
115/115 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0309 - val_loss: 0.0596
```

In [6]:
```
import matplotlib.pyplot as plt
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
```

```
plt.figure(figsize=(8, 6))
plt.plot(history_transformer.history['loss'], label='Training Loss')
plt.plot(history_transformer.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Transformer Training vs. Validation Loss')
plt.legend()
plt.show()

# predictions
predictions = transformer_model.predict([input_sequences, output_sequences])
predicted_sequences = np.argmax(predictions, axis=-1)


# decode sequence back to strings
def decode_sequence(seq, tokenizer):
    reverse_word_index = {v: k for k, v in tokenizer.word_index.items()}
    tokens = [reverse_word_index.get(num, '') for num in seq if num != 0]
    out_str = ''.join(tokens)
    # remove special tokens like '¶' or 'µ', remove them
    out_str = out_str.replace('¶', '').replace('µ', '')
    return out_str

# display some examples
num_examples = 5
for i in range(num_examples):
    input_str = decode_sequence(input_sequences[i], tokenizer)
    true_str = decode_sequence(output_sequences[i], tokenizer)
    pred_str = decode_sequence(predicted_sequences[i], tokenizer)
    print(f"Example {i+1}")
    print("Input Function:     ", input_str)
    print("True Taylor:        ", true_str)
    print("Predicted Taylor:   ", pred_str)
    print("-" * 30)


bleu_scores = []
# Smoothing helps avoid zero BLEU for short sequences
smooth_fn = SmoothingFunction().method1

for i in range(len(output_sequences)):
```

```
    # Prepare reference and candidate as lists of characters (for char-level)
    reference = [list(decode_sequence(output_sequences[i], tokenizer))]
    candidate = list(decode_sequence(predicted_sequences[i], tokenizer))
    score = sentence_bleu(reference, candidate, smoothing_function=smooth_fn)
    bleu_scores.append(score)

average_bleu = np.mean(bleu_scores)
print("Average BLEU Score on Test Set:", average_bleu)
```



```
287/287 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
Example 1
Input Function:      -4*x**2
True Taylor:         -4*x**2
Predicted Taylor:    4*x**2
------------------------------
Example 2
Input Function:      -5*sin(x) + 2*sinh(x) - 2*atan(x)
True Taylor:         11*x**3/6 - 5*x
Predicted Taylor:    *x**3/6 - 3*x
------------------------------
Example 3
Input Function:      -cosh(x) + 3*asin(x)
True Taylor:         -x**4/24 + x**3/2 - x**2/2 + 3*x - 1
Predicted Taylor:    **4/24 + x**3/2 - x**2/2 + 3*x - 1
------------------------------
Example 4
Input Function:      5*acos(x)
True Taylor:         -5*x**3/6 - 5*x + 5*pi/2
Predicted Taylor:    **x**3/6 - 5*x + 5*pi/2
------------------------------
Example 5
Input Function:      -log(x + 1) + 3*asin(x)
True Taylor:         x**4/4 + x**3/6 + x**2/2 + 2*x
Predicted Taylor:    **4/4 + x**3/6 + x**2/2 + 2*x
------------------------------
Average BLEU Score on Test Set: 0.8930689371586087
```

In [ ]: