COMPONENT 1 :- N-Gram Model

```
import random
import numpy as np

text = """
artificial intelligence is transforming modern society.
it is used in healthcare finance education and transportation.
machine learning allows systems to improve automatically with experience.
data plays a critical role in training intelligent systems.
large datasets help models learn complex patterns.
deep learning uses multi layer neural networks.
neural networks are inspired by biological neurons.
each neuron processes input and produces an output.
training a neural network requires optimization techniques.
gradient descent minimizes the loss function.

natural language processing helps computers understand human language.
text generation is a key task in nlp.
language models predict the next word or character.
recurrent neural networks handle sequential data.
lstm and gru models address long term dependency problems.
however rnn based models are slow for long sequences.
"""

text = text.lower().replace("\n", " ")
words = text.split()
```

```
from collections import defaultdict

n = 3  # Trigram
ngrams = defaultdict(list)

for i in range(len(words) - n + 1):
    key = (words[i], words[i+1])
    next_word = words[i+2]
    ngrams[key].append(next_word)
```

```
def generate_ngram(seed_text, num_words):
    seed_words = seed_text.lower().split()

    for _ in range(num_words):
        key = tuple(seed_words[-2:])
        if key in ngrams:
            next_word = random.choice(ngrams[key])
            seed_words.append(next_word)
        else:
            break

    return " ".join(seed_words)

print(generate_ngram("artificial intelligence", 10))
print(generate_ngram("neural networks", 10))
```

```
artificial intelligence is transforming modern society. it is used in healthcare finance
neural networks handle sequential data. lstm and gru models address long term
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1

input_sequences = []

for line in text.split('.'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_seq = token_list[:i+1]
```

```
        input_sequences.append(n_gram_seq)

max_seq_len = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_seq_len, padding='pre')

X = input_sequences[:, :-1]
y = input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

```
model = Sequential([
    Embedding(total_words, 100, input_length=max_seq_len-1),
    SimpleRNN(150),
    Dense(total_words, activation='softmax')
])

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
simple_rnn (SimpleRNN)	?	0 (unbuilt)
dense_4 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
model.fit(X, y, epochs=150, verbose=1)
```

```
Epoch 146/150
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0124
Epoch 147/150
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0128
Epoch 148/150
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0134
Epoch 149/150
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0130
Epoch 150/150
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.0128
<keras.src.callbacks.history.History at 0x79e8dc120320>
```

```
import numpy as np

def generate_rnn_text(seed_text, next_words):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_seq_len-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)

        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text

print(generate_rnn_text("artificial intelligence", 10))
print(generate_rnn_text("deep learning", 10))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
artificial intelligence is transforming modern society in education and transportation an output
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
deep learning uses multi layer neural networks handle sequential data biological neurons
```

COMPONENT 2 :- Transformer Based Text Generation

```
from tensorflow.keras.layers import LayerNormalization, MultiHeadAttention, Dropout
```

```
vocab_size = total_words
embedding_dim = 64
num_heads = 2
ff_dim = 128
```

```
def transformer_block(x):
    attention = MultiHeadAttention(num_heads=num_heads, key_dim=embedding_dim)(x, x)
    attention = Dropout(0.1)(attention)
    out1 = LayerNormalization(epsilon=1e-6)(x + attention)

    ffn = tf.keras.Sequential([
        Dense(ff_dim, activation='relu'),
        Dense(embedding_dim),
    ])

    ffn_output = ffn(out1)
    ffn_output = Dropout(0.1)(ffn_output)
    return LayerNormalization(epsilon=1e-6)(out1 + ffn_output)
```

```
inputs = tf.keras.Input(shape=(max_seq_len-1,))
embedding_layer = Embedding(vocab_size, embedding_dim)(inputs)

x = transformer_block(embedding_layer)
```

```
x = transformer_block(embedding_layer)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
outputs = Dense(vocab_size, activation="softmax")(x)

transformer_model = tf.keras.Model(inputs=inputs, outputs=outputs)

transformer_model.compile(loss="categorical_crossentropy",
                          optimizer="adam",
                          metrics=["accuracy"])

transformer_model.summary()
```

Model: "functional_2"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 9)	0	–
embedding_1 (Embedding)	(None, 9, 64)	12,480	input_layer_1[0]…
multi_head_attenti… (MultiHeadAttentio…	(None, 9, 64)	33,216	embedding_1[0][0]… embedding_1[0][0]
dropout_1 (Dropout)	(None, 9, 64)	0	multi_head_atten…
add (Add)	(None, 9, 64)	0	embedding_1[0][0]… dropout_1[0][0]
layer_normalization (LayerNormalizatio…	(None, 9, 64)	128	add[0][0]
sequential_1 (Sequential)	(None, 9, 64)	16,576	layer_normalizat…
dropout_2 (Dropout)	(None, 9, 64)	0	sequential_1[0][…
add_1 (Add)	(None, 9, 64)	0	layer_normalizat… dropout_2[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 9, 64)	128	add_1[0][0]
global_average_poo… (GlobalAveragePool…	(None, 64)	0	layer_normalizat…
dense_3 (Dense)	(None, 195)	12,675	global_average_p…

Total params: 75,203 (293.76 KB)
Trainable params: 75,203 (293.76 KB)
Non-trainable params: 0 (0.00 B)

```
transformer_model.fit(X, y, epochs=200)
```

```
Epoch 1/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.0044 - loss: 5.2977
Epoch 2/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0202 - loss: 5.0481
Epoch 3/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0473 - loss: 4.8567
Epoch 4/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0882 - loss: 4.6877
Epoch 5/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.1147 - loss: 4.6172
Epoch 6/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.1145 - loss: 4.4353
Epoch 7/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.1420 - loss: 4.3313
Epoch 8/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2115 - loss: 4.1877
Epoch 9/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.1719 - loss: 4.1551
Epoch 10/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2387 - loss: 3.9620
Epoch 11/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2519 - loss: 3.8096
Epoch 12/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.2302 - loss: 3.6631
Epoch 13/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2776 - loss: 3.4648
Epoch 14/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2939 - loss: 3.2742
Epoch 15/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3155 - loss: 3.2067
Epoch 16/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.3332 - loss: 2.9416
Epoch 17/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4069 - loss: 2.8080
```

```
Epoch 18/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4444 - loss: 2.5295
Epoch 19/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4354 - loss: 2.4308
Epoch 20/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5476 - loss: 2.2945
Epoch 21/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5267 - loss: 2.1350
Epoch 22/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6135 - loss: 1.9551
Epoch 23/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6264 - loss: 1.8779
Epoch 24/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6608 - loss: 1.7299
Epoch 25/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7165 - loss: 1.5824
Epoch 26/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7996 - loss: 1.4583
Epoch 27/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7980 - loss: 1.3778
Epoch 28/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8608 - loss: 1.2586
Epoch 29/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8658 - loss: 1.1946
```

```
def generate_text_transformer(seed_text, next_words):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_seq_len-1, padding='pre')
        predicted = np.argmax(transformer_model.predict(token_list), axis=-1)

        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text

print(generate_text_transformer("deep learning", 10))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 892ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
deep learning uses multi layer neural networks handle sequential data data data
```