An N-Gram model is a statistical language model that predicts the next word based on the previous N–1 words. bold text**bold text** For example:

Bigram (N=2) → uses 1 previous word

Trigram (N=3) → uses 2 previous words

It learns word patterns from a given dataset and generates new text by selecting the most likely next word.

It is simple, fast, and helps understand the basics of language modeling before moving to neural networks.

```
# ================================
# N-GRAM TEXT GENERATION MODEL
# ================================

import re
import random
from collections import defaultdict

# --------------------------------
# 1. Load Dataset
# --------------------------------

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.
"""

# --------------------------------
# 2. Preprocessing
# --------------------------------

text = text.lower()
text = re.sub(r'[^a-z\s]', '', text)
words = text.split()

print("Total Words:", len(words))

# --------------------------------
# 3. Build N-Gram Model (Trigram)
# --------------------------------

def build_ngram_model(words, n):
    model = defaultdict(list)

    for i in range(len(words) - n + 1):
        context = tuple(words[i:i+n-1])
        target = words[i+n-1]
        model[context].append(target)

    return model

n = 3  # Change to 2 for Bigram, 4 for 4-gram
ngram_model = build_ngram_model(words, n)

# --------------------------------
# 4. Text Generation Function
# --------------------------------

def generate_text(model, seed_text, n, num_words=20):
    seed_words = seed_text.lower().split()

    for _ in range(num_words):
        context = tuple(seed_words[-(n-1):])

        if context in model:
            next_word = random.choice(model[context])
            seed_words.append(next_word)
        else:
            break

    return " ".join(seed_words)
```

```
# ------------------------------
# 5. Generate Text
# ------------------------------

seed = "artificial intelligence"
generated_text = generate_text(ngram_model, seed, n, num_words=20)

print("\nGenerated Text:\n")
print(generated_text)
```

```
Total Words: 75

Generated Text:

artificial intelligence is transforming modern society it is used in healthcare finance education and transportation machine
```

🔹 Limitations

❌ Cannot capture long-term dependencies

❌ Suffers from data sparsity

❌ Memory requirement increases with larger N

❌ Cannot generalize well to unseen text

❌ Mostly memorizes patterns instead of understanding

Because of these limitations, we move to RNN/LSTM models next.