NAME : PAREENITA A.SHIRSATH PRN : 221101062 ROLL.NO : 57

B.E.A.I.&D.S.

NLP EXPERIMENT NO : 05

```
import re
from collections import defaultdict

def tokenize_text(text):
    """
    Tokenize the input text into a list of words.
    Lowercases and removes punctuation.
    """
    text = re.sub(r'[^\w\s]', '', text)
    return text.lower().split()

def build_ngram_model(sentences, n):
    """
    Build an N-Gram model from the input sentences.

    Args:
        sentences (list): List of sentences (strings).
        n (int): N-gram size (1=unigram, 2=bigram, 3=trigram, etc.).

    Returns:
        dict: Nested defaultdict representing the N-gram counts.
    """
    model = defaultdict(lambda: defaultdict(int))
    for sentence in sentences:
        tokens = tokenize_text(sentence)
        if n > 1:
            tokens = ["<START>"] * (n - 1) + tokens
        tokens = tokens + ["<END>"]
        for i in range(len(tokens) - n + 1):
            context = tuple(tokens[i:i + n - 1]) if n > 1 else ()
            next_word = tokens[i + n - 1]
            model[context][next_word] += 1
    return model

def predict_next_word(model, context):
    if context not in model or not model[context]:
        return "unknown"
    return max(model[context], key=model[context].get)

def bigram_probability(model, context, word):
    total_count = sum(model[context].values())
    word_count = model[context][word]
    return word_count / total_count if total_count > 0 else 0

# Sample sentences
sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "Artificial intelligence is transforming the world rapidly.",
    "Data science involves statistics, programming, and domain knowledge.",
    "Python is a popular programming language for developers.",
    "Machine learning models can predict outcomes based on data."
]

# Build bigram model
bigram_model = build_ngram_model(sentences, 2)

contexts_bigram = [
    ("<START>",),
    ("the",),
    ("artificial",),
    ("data",),
    ("python",),
    ("machine",),
    ("dog",)
]

print(f"{'Context':15} | {'Predicted Next Word':20} | {'Bigram Count':12} | {'Bigram Probability':18} | {'Most Probable Next Word':22}")
print("-"*100)

overall_counts = defaultdict(int)
```

```
for context in contexts_bigram:
    if context in bigram_model and bigram_model[context]:
        predicted_word = predict_next_word(bigram_model, context)
        count_bigram = bigram_model[context][predicted_word]
        prob_bigram = bigram_probability(bigram_model, context, predicted_word)
        most_probable_word = max(bigram_model[context], key=lambda w: bigram_probability(bigram_model, context, w))

        # Collect counts for overall most probable word later
        overall_counts[predicted_word] += count_bigram

        context_str = " ".join(context)
        print(f"{context_str:15} | {predicted_word:20} | {count_bigram:<12} | {prob_bigram:<18.4f} | {most_probable_word:22}")
    else:
        context_str = " ".join(context)
        print(f"{context_str:15} | {'No data':20} | {'-':12} | {'-':18} | {'-':22}")

print("-"*100)

# Find overall most probable next word across all contexts based on counts
if overall_counts:
    overall_most_probable_word = max(overall_counts, key=overall_counts.get)
    print(f"Overall most probable next word across contexts: {overall_most_probable_word}")
else:
    print("No overall most probable next word found.")
```

```
Context         | Predicted Next Word  | Bigram Count | Bigram Probability | Most Probable Next Word
----------------------------------------------------------------------------------------------------
<START>         | the                  | 1            | 0.2000             | the
the             | quick                | 1            | 0.3333             | quick
artificial      | intelligence         | 1            | 1.0000             | intelligence
data            | science              | 1            | 0.5000             | science
python          | is                   | 1            | 1.0000             | is
machine         | learning             | 1            | 1.0000             | learning
dog             | <END>                | 1            | 1.0000             | <END>
----------------------------------------------------------------------------------------------------
Overall most probable next word across contexts: the
```