```
# Install nltk (only once)
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```

```
# Import required libraries
import nltk
import re
import math
import random
import numpy as np
import pandas as pd
from collections import Counter, defaultdict
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
# Download required NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
# Sample text corpus (repeat to reach 1500+ words)
text = """
Natural Language Processing is a field of artificial intelligence.
It helps computers understand human language.
Language models predict the probability of words.
Unigram, Bigram and Trigram models are basic models.
They are widely used in NLP applications.
""" * 300   # repeats to exceed 1500 words
```

```
# Display sample text
print(text[:500])
```

```
Natural Language Processing is a field of artificial intelligence.
It helps computers understand human language.
Language models predict the probability of words.
Unigram, Bigram and Trigram models are basic models.
They are widely used in NLP applications.

Natural Language Processing is a field of artificial intelligence.
It helps computers understand human language.
Language models predict the probability of words.
Unigram, Bigram and Trigram models are basic models.
They are widely used in
```

```
def preprocess_text(text):
    text = text.lower()                      # lowercase
    text = re.sub(r'[^a-z\s]', '', text)     # remove punctuation & numbers
    sentences = sent_tokenize(text)          # sentence tokenization

    stop_words = set(stopwords.words('english'))
    processed_sentences = []

    for sent in sentences:
        words = word_tokenize(sent)
        words = [w for w in words if w not in stop_words]
        processed_sentences.append(['<s>'] + words + ['</s>'])

    return processed_sentences
```

```
nltk.download('punkt_tab')
sentences = preprocess_text(text)
print(sentences[:2])
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[['<s>', 'natural', 'language', 'processing', 'field', 'artificial', 'intelligence', 'helps', 'computers', 'understand', 'hu
```

```
random.shuffle(sentences)

split = int(0.8 * len(sentences))
train_sentences = sentences[:split]
test_sentences = sentences[split:]
```

```
def build_ngram(sentences, n):
    ngrams = []
    for sent in sentences:
        ngrams += list(zip(*[sent[i:] for i in range(n)]))
    return Counter(ngrams)
```

```
unigram_counts = build_ngram(train_sentences, 1)
bigram_counts = build_ngram(train_sentences, 2)
trigram_counts = build_ngram(train_sentences, 3)

vocab_size = len(unigram_counts)
```

```
def sentence_probability(sentence, ngram_counts, lower_counts, n):
    prob = 1
    sentence = ['<s>'] + sentence + ['</s>']

    for i in range(len(sentence) - n + 1):
        ngram = tuple(sentence[i:i+n])
        lower_ngram = tuple(sentence[i:i+n-1])

        numerator = ngram_counts[ngram] + 1
        denominator = lower_counts[lower_ngram] + vocab_size

        prob *= numerator / denominator
    return prob
```

```
# ---------- MINIMAL ONE CELL N-GRAM PROBABILITY CODE ----------

from collections import Counter
import math

# Small sample training data (already tokenized)
train_sentences = [
    ['<s>', 'natural', 'language', 'processing', '</s>'],
    ['<s>', 'language', 'models', 'are', 'useful', '</s>'],
    ['<s>', 'ngram', 'models', 'predict', 'words', '</s>'],
    ['<s>', 'probability', 'is', 'important', '</s>'],
    ['<s>', 'language', 'processing', 'uses', 'models', '</s>']
]

test_sentences = [
    ['<s>', 'language', 'models', 'predict', 'words', '</s>'],
    ['<s>', 'natural', 'language', 'models', '</s>']
]

# Build n-grams
def build_ngram(sentences, n):
    ngrams = []
    for sent in sentences:
        for i in range(len(sent) - n + 1):
            ngrams.append(tuple(sent[i:i+n]))
    return Counter(ngrams)

unigram_counts = build_ngram(train_sentences, 1)
bigram_counts = build_ngram(train_sentences, 2)
trigram_counts = build_ngram(train_sentences, 3)

vocab_size = len(unigram_counts)

# Sentence probability with Laplace smoothing
def sentence_probability(sentence, ngram_counts, lower_counts, n):
    prob = 1
    for i in range(len(sentence) - n + 1):
        ngram = tuple(sentence[i:i+n])
        lower_ngram = tuple(sentence[i:i+n-1])
```

```
        prob *= (ngram_counts.get(ngram, 0) + 1) / (lower_counts.get(lower_ngram, 0) + vocab_size)
    return prob

# Test sentences
test_sentences_sample = test_sentences[:5]

for sent in test_sentences_sample:
    words = sent[1:-1]
    print("Sentence:", " ".join(words))
    print("Unigram Prob:", sentence_probability(sent, unigram_counts, Counter(), 1))
    print("Bigram Prob :", sentence_probability(sent, bigram_counts, unigram_counts, 2))
    print("Trigram Prob:", sentence_probability(sent, trigram_counts, bigram_counts, 3))
    print("-" * 40)
```

```
Sentence: language models predict words
Unigram Prob: 0.00020227160493827164
Bigram Prob : 2.8935185185185183e-05
Trigram Prob: 0.00011488970588235294
----------------------------------------
Sentence: natural language models
Unigram Prob: 0.0015170370370370372
Bigram Prob : 0.00015432098765432098
Trigram Prob: 0.00048828125
----------------------------------------
```

```
def perplexity(sentence, ngram_counts, lower_counts, n):
    sentence = ['<s>'] + sentence + ['</s>']
    N = len(sentence)
    log_prob = 0

    for i in range(len(sentence) - n + 1):
        ngram = tuple(sentence[i:i+n])
        lower_ngram = tuple(sentence[i:i+n-1])

        prob = (ngram_counts[ngram] + 1) / (lower_counts[lower_ngram] + vocab_size)
        log_prob += math.log(prob)

    return math.exp(-log_prob / N)
```

```
for sent in test_sentences_sample:
    words = sent[1:-1]
    print("Sentence:", " ".join(words))
    print("Unigram PP:", perplexity(words, unigram_counts, Counter(), 1))
    print("Bigram PP :", perplexity(words, bigram_counts, unigram_counts, 2))
    print("Trigram PP:", perplexity(words, trigram_counts, bigram_counts, 3))
    print()
```

```
Sentence: language models predict words
Unigram PP: 4.127409061118283
Bigram PP : 5.707277056455109
Trigram PP: 4.535444049403089

Sentence: natural language models
Unigram PP: 3.662695064479402
Bigram PP : 5.785155024015764
Trigram PP: 4.594793419988139
```