```
import re
import math
from collections import Counter, defaultdict
import random
```

```
with open("/content/ngram_dataset.txt", "r", encoding="utf-8") as f:
    text = f.read()

print(text[:500])
```

```
Natural language processing is a field of artificial intelligence that focuses on the interaction between computers and human

A language model assigns probabilities to sequences of words. By learning patterns from large text corpora, it can predict th
```

```
def preprocess(text):
    text = text.lower()
    text = re.sub(r"[^a-z\s]", "", text)
    words = text.split()
    return words

tokens = preprocess(text)
print(tokens[:20])
```

```
['natural', 'language', 'processing', 'is', 'a', 'field', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on', 'the',
```

```
sentences = text.split(".")
processed_sentences = []

for s in sentences:
    s = preprocess(s)
    if len(s) > 0:
        processed_sentences.append(["<s>"] + s + ["</s>"])
```

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

```
def get_ngrams(tokens, n):
    return [tuple(tokens[i:i+n]) for i in range(len(tokens)-n+1)]

unigrams = Counter()
bigrams = Counter()
trigrams = Counter()

for sentence in train_sentences:
    unigrams.update(get_ngrams(sentence, 1))
    bigrams.update(get_ngrams(sentence, 2))
    trigrams.update(get_ngrams(sentence, 3))
```

```
V = len(unigrams)
print("Vocabulary size:", V)
```

```
Vocabulary size: 200
```

```
def unigram_prob(word):
    return (unigrams[(word,)] + 1) / (sum(unigrams.values()) + V)

def bigram_prob(w1, w2):
    return (bigrams[(w1, w2)] + 1) / (unigrams[(w1,)] + V)

def trigram_prob(w1, w2, w3):
    return (trigrams[(w1, w2, w3)] + 1) / (bigrams[(w1, w2)] + V)
```

```
test_samples = random.sample(test_sentences, 5)
```

```
def sentence_prob(sentence, model="unigram"):
    prob = 1.0
```

```
    if model == "unigram":
        for w in sentence:
            prob *= unigram_prob(w)

    elif model == "bigram":
        for i in range(len(sentence)-1):
            prob *= bigram_prob(sentence[i], sentence[i+1])

    elif model == "trigram":
        for i in range(len(sentence)-2):
            prob *= trigram_prob(sentence[i], sentence[i+1], sentence[i+2])

    return prob
```

```
for s in test_samples:
    print("Sentence:", " ".join(s))
    print("Unigram:", sentence_prob(s, "unigram"))
    print("Bigram:", sentence_prob(s, "bigram"))
    print("Trigram:", sentence_prob(s, "trigram"))
    print()
```

```
Sentence: <s> ngram models are simple yet powerful tools for modeling language </s>
Unigram: 2.3160577238505738e-27
Bigram: 2.9837813798767887e-25
Trigram: 9.668696319398035e-24

Sentence: <s> this knowledge can be applied in education communication and technology development </s>
Unigram: 3.3710177190169187e-31
Bigram: 3.9140566903393 53e-28
Trigram: 4.8585199004975125e-26

Sentence: <s> the study of language models also helps researchers understand linguistic patterns and structures </s>
Unigram: 1.1477867362387793e-32
Bigram: 5.507736950932714e-32
Trigram: 2.358504806066754e-30

Sentence: <s> learning how to implement and evaluate ngram models is an essential skill for anyone studying natural language p
Unigram: 2.3723944943821894e-45
Bigram: 4.570367545291382e-43
Trigram: 1.1107247818149765e-41

Sentence: <s> although they have limitations such as data sparsity and limited context they remain important in both academic
```

```
Unigram: 4.507185075396329e-55
Bigram: 6.76022412844989e-51
Trigram: 4.697903036483993e-49
```

```
def perplexity(sentence, model="unigram"):
    N = len(sentence)
    log_prob = 0

    if model == "unigram":
        for w in sentence:
            log_prob += math.log(unigram_prob(w))

    elif model == "bigram":
        for i in range(len(sentence)-1):
            log_prob += math.log(bigram_prob(sentence[i], sentence[i+1]))

    elif model == "trigram":
        for i in range(len(sentence)-2):
            log_prob += math.log(trigram_prob(sentence[i], sentence[i+1], sentence[i+2]))

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

```
for s in test_samples:
    print("Sentence:", " ".join(s))
    print("Unigram Perplexity:", perplexity(s, "unigram"))
    print("Bigram Perplexity:", perplexity(s, "bigram"))
    print("Trigram Perplexity:", perplexity(s, "trigram"))
    print()
```

```
Sentence: <s> ngram models are simple yet powerful tools for modeling language </s>
Unigram Perplexity: 165.80751802716475
Bigram Perplexity: 110.60363917085128
Trigram Perplexity: 82.77248745338215

Sentence: <s> this knowledge can be applied in education communication and technology development </s>
Unigram Perplexity: 220.8098190759452
Bigram Perplexity: 128.30968918601846
Trigram Perplexity: 88.55070049160796

Sentence: <s> the study of language models also helps researchers understand linguistic patterns and structures </s>
```

```
Unigram Perplexity: 134.69224342282493
Bigram Perplexity: 121.32075955790444
Trigram Perplexity: 94.44034152749904

Sentence: <s> learning how to implement and evaluate ngram models is an essential skill for anyone studying natural language p
Unigram Perplexity: 170.3101983042498
Bigram Perplexity: 130.91892893965726
Trigram Perplexity: 111.61425811753

Sentence: <s> although they have limitations such as data sparsity and limited context they remain important in both academic
Unigram Perplexity: 230.60763326320009
Bigram Perplexity: 151.81197817882625
Trigram Perplexity: 126.24714427061872
```

COMPARISION AND ANALYSIS : The trigram model generally produced the lowest perplexity, indicating better prediction because it captures more context. However, trigrams sometimes performed worse when training data was insufficient, showing the problem of data sparsity. Bigram models often performed better than unigram models because they consider word dependencies. Unigram models had the highest perplexity because they ignore context. When unseen words appeared, probabilities decreased significantly, but smoothing helped reduce the impact. Laplace smoothing improved stability by preventing zero probabilities. Overall, trigram models are more accurate but require larger datasets, while bigram models offer a good balance between accuracy and data requirements.