Step 1: Open Google Colab

STEP 2 — Import required libraries

```
# For text cleaning (removing punctuation, numbers)
import re
# For counting words and n-grams
from collections import Counter, defaultdict
# For math operations (log, power)
import math
```

STEP 3 — Load dataset

```
# Small text corpus (dataset)
corpus = """
I love natural language processing.
Natural language processing is fun.
I love learning new things.
Language models predict words.
"""

# Remove extra spaces and newlines
corpus = corpus.strip()

# Display sample text
print("Sample Text:\n", corpus)
```

```
Sample Text:
 I love natural language processing.
Natural language processing is fun.
I love learning new things.
Language models predict words.
```

STEP 4 — Preprocess Text

```
def preprocess_text(text):
    #raw text is cleaned and prepared is called preprocesed
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation and numbers
    text = re.sub(r'[^a-z\s]', '', text)

    # Split text into sentences
    sentences = text.split('\n')

    processed_sentences = []

    for sentence in sentences:
        # Tokenize sentence into words
        #text/words split into smaller units called tokens.
        words = sentence.split()

        # Add start and end tokens
        words = ['<s>'] + words + ['</s>']

        processed_sentences.append(words)

    return processed_sentences

# Apply preprocessing
tokenized_sentences = preprocess_text(corpus)

print("Tokenized Sentences:\n", tokenized_sentences)
```

```
Tokenized Sentences:
 [['<s>', 'i', 'love', 'natural', 'language', 'processing', '</s>'], ['<s>', 'natural', 'language', 'processing', 'is', 'fun', '
```

STEP 5 — Build N-Gram Models

```
# Flatten all words into one list
all_words = [word for sentence in tokenized_sentences for word in sentence]
# A unigram model considers only single words.
# Count unigrams
unigram_counts = Counter(all_words)

print("Unigram Counts:\n", unigram_counts)
```

```
Unigram Counts:
 Counter({'<s>': 4, '</s>': 4, 'language': 3, 'i': 2, 'love': 2, 'natural': 2, 'processing': 2, 'is': 1, 'fun': 1, 'learning': 1
```

Bigram Model

```
# Create bigrams
# Uses two-word sequences
bigrams = []
for sentence in tokenized_sentences:
    for i in range(len(sentence) - 1):
        bigrams.append((sentence[i], sentence[i+1]))

# Count bigrams
bigram_counts = Counter(bigrams)

print("Bigram Counts:\n", bigram_counts)
```

```
Bigram Counts:
 Counter({('<s>', 'i'): 2, ('i', 'love'): 2, ('natural', 'language'): 2, ('language', 'processing'): 2, ('love', 'natural'): 1,
```

Trigram Model

```
# Create trigrams
# Uses three-word sequences
trigrams = []
for sentence in tokenized_sentences:
    for i in range(len(sentence) - 2):
        trigrams.append((sentence[i], sentence[i+1], sentence[i+2]))

# Count trigrams
trigram_counts = Counter(trigrams)

print("Trigram Counts:\n", trigram_counts)
```

```
Trigram Counts:
 Counter({('<s>', 'i', 'love'): 2, ('natural', 'language', 'processing'): 2, ('i', 'love', 'natural'): 1, ('love', 'natural', 'l
```

STEP 6 — Apply Add-One (Laplace) Smoothing

```
# Smoothing is used to avoid zero probabilities in N-gram models
vocab_size = len(unigram_counts)

def laplace_smoothing(count, total, vocab_size):
    # Add-one smoothing formula
    return (count + 1) / (total + vocab_size)
```

STEP 7 — Sentence Probability Calculation

```
def unigram_probability(sentence):
    prob = 1
    total_words = sum(unigram_counts.values())

    for word in sentence:
        prob *= laplace_smoothing(
```

```
            unigram_counts[word], total_words, vocab_size
        )
    return prob
```

```
def bigram_probability(sentence):
    prob = 1
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1])
        prob *= laplace_smoothing(
            bigram_counts[bigram],
            unigram_counts[sentence[i]],
            vocab_size
        )
    return prob
```

```
def trigram_probability(sentence):
    prob = 1
    for i in range(len(sentence) - 2):
        trigram = (sentence[i], sentence[i+1], sentence[i+2])
        prob *= laplace_smoothing(
            trigram_counts[trigram],
            bigram_counts[(sentence[i], sentence[i+1])],
            vocab_size
        )
    return prob
```

```
test_sentences = [
    "i love language",
    "language processing is fun",
    "i love learning",
    "models predict words",
    "natural language models"
]

# Preprocess test sentences
processed_tests = preprocess_text("\n".join(test_sentences))
for sentence in processed_tests:
    print("\nSentence:", sentence)
    print("Unigram Probability:", unigram_probability(sentence))
    print("Bigram Probability:", bigram_probability(sentence))
    print("Trigram Probability:", trigram_probability(sentence))
```

```
Sentence: ['<s>', 'i', 'love', 'language', '</s>']
Unigram Probability: 6.886460447476689e-06
Bigram Probability: 9.105809506465124e-05
Trigram Probability: 0.0006920415224913496

Sentence: ['<s>', 'language', 'processing', 'is', 'fun', '</s>']
Unigram Probability: 2.1861779198338695e-07
Bigram Probability: 3.224974200206398e-05
Trigram Probability: 0.00011488970588235294

Sentence: ['<s>', 'i', 'love', 'learning', '</s>']
Unigram Probability: 3.4432302237383446e-06
Bigram Probability: 0.0002048807138954653
Trigram Probability: 0.0012975778546712804

Sentence: ['<s>', 'models', 'predict', 'words', '</s>']
Unigram Probability: 1.5303245438837084e-06
Bigram Probability: 0.00010279605263157894
Trigram Probability: 0.0010416666666666667

Sentence: ['<s>', 'natural', 'language', 'models', '</s>']
Unigram Probability: 4.5909736316511264e-06
Bigram Probability: 0.00012899896800825592
Trigram Probability: 0.0004955882352941176
```

STEP 8 — Perplexity Calculation

```
def perplexity(prob, N):
    # Perplexity formula
```

```
        #to guess the next word in a sentence
        return pow(1/prob, 1/N)
    for sentence in processed_tests:
        N = len(sentence)

        u_prob = unigram_probability(sentence)
        b_prob = bigram_probability(sentence)
        t_prob = trigram_probability(sentence)

        print("\nSentence:", sentence)
        print("Unigram Perplexity:", perplexity(u_prob, N))
        print("Bigram Perplexity:", perplexity(b_prob, N))
        print("Trigram Perplexity:", perplexity(t_prob, N))
```

```
Sentence: ['<s>', 'i', 'love', 'language', '</s>']
Unigram Perplexity: 10.774590877016513
Bigram Perplexity: 6.428894336448732
Trigram Perplexity: 4.285224403010953

Sentence: ['<s>', 'language', 'processing', 'is', 'fun', '</s>']
Unigram Perplexity: 12.884050430459103
Bigram Perplexity: 5.605043135821642
Trigram Perplexity: 4.535444049403088

Sentence: ['<s>', 'i', 'love', 'learning', '</s>']
Unigram Perplexity: 12.376754816194929
Bigram Perplexity: 5.466379565760732
Trigram Perplexity: 3.7789688757455884

Sentence: ['<s>', 'models', 'predict', 'words', '</s>']
Unigram Perplexity: 14.556041706258078
Bigram Perplexity: 6.274869706144199
Trigram Perplexity: 3.94870097166964

Sentence: ['<s>', 'natural', 'language', 'models', '</s>']
Unigram Perplexity: 11.68473965232866
Bigram Perplexity: 5.9962917153367075
Trigram Perplexity: 4.6508440608779935
```