**Project 1**

**Introduction:**

This Python program implements n-gram models for natural language processing using the Natural Language Toolkit (NLTK) package. Given the prior n-1 words, N-gram models predict the likelihood of the following word in a series of words. We employ bigram, trigram, and 4-gram models in this code to compute probabilities for each word given the previous n-1 words.

To begin, the code reads a text file and tokenizes it into words. The data is then divided into training and testing sets, with 80% used for training and 20% for testing.

The code then uses the training set to train the bigram model. It generates a bigram conditional frequency distribution (CFD) and computes the probability of each word given its preceding word. Bigram probs is a dictionary that stores the probabilities.Using the test set, the method then computes the bigram model's perplexity score. The model's perplexity is a measure of how well it predicts the test data. A lower perplexity score corresponds to better performance.

The similar procedure is used for the trigram and four-gram models. The code uses the training data to train these models, calculates the probability of each word given the previous n-1 words, and stores the probabilities in dictionaries labeled trigram probs and fourgram probs, respectively.Lastly, the algorithm computes and outputs the perplexity score for each model using the test set.

**Implementation:**

First, I created a virtual environment in jupyter called venv. Then I took the text in 5 blockchain papers document in a text file. Then in an ipynb file I imported the respective text file and the natural language toolkit library. Then I started with bigram model, and found the conditional probability distribution. Then I generated a random starting word and continued generating the rest of the words using the bigram model. The same way I also generated the conditional probability distribution for trigram and four gram models. For trigram I used 2 random words in the staring and generated the rest using the model I created. Similarly, for the 4 gram model I used 3 random words and generated the rest of 7 words using the model I created.

**Discussion:**

The first part of the code reads in the text file, tokenizes it into words, and splits the data into training and testing sets. The training set is used to build the n-gram models, while the testing set is used to calculate perplexity scores.

The code then trains the bigram model using the nltk.ConditionalFreqDist() function. This creates a frequency distribution of bigrams in the training data, where the conditions are the first words in the bigram, and the tokens are the second words. The code then calculates the probability of each token given its condition in the bigram model, and stores these probabilities in a dictionary.

The perplexity score for the bigram model is then calculated by iterating through the testing tokens and multiplying the inverse of the probability of each token given its condition in the bigram model. The product of these values is then raised to the power of 1 divided by the length of the testing set. The resulting value represents the perplexity score for the bigram model.

The code then repeats this process for the trigram and 4-gram models. The trigram model is trained using the nltk.ConditionalFreqDist() function, but with tuples of three words as the conditions instead of bigrams. The 4-gram model is trained using the nltk.ConditionalFreqDist() function, but with tuples of four words as the conditions instead of bigrams.

The perplexity scores for the trigram and 4-gram models are calculated in the same way as the bigram model, but with the appropriate probabilities and conditions for each model.

Finally, the code prints the perplexity scores for each model. A lower perplexity score indicates a better fit of the language model to the data.Overall, this code provides a simple implementation of n-gram language models and can be used to compare the performance of different n-gram models on a given text dataset.