NLP architectures use various methods for data preprocessing, feature extraction, and modelling. Some of these processes are:

1) **Data preprocessing:** These techniques aid in transforming raw text into a structured and processed format that NLP models can understand and work with effectively. Libraries like spaCy and NLTK offer tools to execute these preprocessing steps efficiently.

    1. Stemming and Lemmatization:
    Stemming: This technique aims to reduce words to their root form by removing prefixes or suffixes, using simple rules. For instance, "running" and "ran" might both be stemmed to "run."
    Lemmatization: More sophisticated, it uses linguistic knowledge and context to obtain the base or dictionary form of a word, taking into account grammar and part-of-speech. For example, "ran" could be lemmatized to "run."

    2. Sentence Segmentation:
    Breaking text into sentences, especially important in languages like English where periods denote sentence boundaries.

    3. Stop Word Removal:
    Commonly occurring words like "the," "and," "is," etc., which don't carry significant meaning in the context of the task, are removed to reduce noise and focus on more informative words.

    4. Tokenization:
    Splitting text into meaningful tokens, usually words or subwords, to create a structured format for processing. Each token might be represented by a numerical value for model input.

2) **Feature Extraction:** Most machine learning algorithms take in input as numerical data. These are generally numerical data which represent the relation a document holds in respect to the entire corpus. I have tested the following 2 methods:

    1. TF-IDF (Term Frequency-Inverse Document Frequency):
    Term Frequency (TF): Calculates the frequency of a word in a document relative to the total number of words in that document. It highlights the importance of a word within a document.

Inverse Document Frequency (IDF): Measures how important a word is across the entire corpus by penalising common words and highlighting rare ones.
TF-IDF Score: It's the product of TF and IDF. Words with high TF-IDF scores are those that occur frequently in a document but are rare across the corpus.

2. Word2Vec:
   A neural network-based technique to create word embeddings from raw text.
   Two models: Skip-Gram predicts the context given a target word, and Continuous Bag-of-Words (CBOW) predicts the target word given its context.
   These models learn high-dimensional vector representations (embeddings) for words based on the surrounding context in which they appear.

   I observed that the model performs vastly better when the features are extracted with Tf-Idf than Word2Vec. Eg. accuracy scores for SVM with tf-idf and word2vec are 0.988 and 0.314 respectively.

3) **Modelling and Fine-Tuning:** I have tried the following 5 models and calculated the accuracy scores for all:

1. **Decision Tree:** Implement Decision Tree Classifier for text classification. Decision trees are a class of supervised classification models that split the dataset based on different features to maximise information gain in those splits.
   Accuracy = 83.53%

2. **Logistic Regression:** Utilise Logistic Regression model for multiclass text classification. Logistic regression is a supervised classification algorithm that aims to predict the probability that an event will occur based on some input.
   Accuracy = 97.75%

3. **Naive Bayes:** Implement Multinomial Naive Bayes Classifier for text classification. Naive Bayes is a supervised classification algorithm that finds the conditional probability distribution P(label | text) using the following Bayes formula:
   P(label | text) = P(label) x P(text|label) / P(text) and predicts based on which joint distribution has the highest probability.
   Accuracy = 96.56%

4. **Linear SVM:** Implement Support Vector Machine Classifier with a linear kernel. Support Vector Machine (SVM) with a linear kernel is a binary classification model extended to multi-class classification using techniques like One-vs-Rest (OvR).
   Linear SVM aims to find the best hyperplane that separates classes by maximising the margin between them.
   Accuracy = 98.80% (after fine tuning the hyperparameters)

5. **Deep Learning (LSTMs):** Utilise LSTM (Long Short-Term Memory) neural network for text classification. Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) known for capturing long-term dependencies in sequential data, making it suitable for text sequences.
   The architecture typically consists of an Embedding layer, LSTM layer(s), Dense layer(s), and Dropout layer(s).
   Accuracy = <90% (varies but upper bound is fixed)

Of all these architectures, I found that the SVM with linear kernel has the highest accuracy after fine tuning (98.8%). **So, my final trained model is SVM with linear kernel and Tf-Idf feature extraction**.

The **evaluation metrics** for SVM with Tf-Idf feature extraction on **training data** are:
SVM Metrics (Tf-Idf):
Accuracy: 98.80239520958084%
Precision: 0.9881159949253597
Recall: 0.9880239520958084
F1-score: 0.9880320996705151

Instructions to run the test dataset:

In the final code, make sure to name the testing data as test.csv and to place it in the same directory as that of code and training data.
On running the final code, my code will print the evaluation metrics(accuracy, precision, recall, F1-score) for the test data. Also it saves the final trained model as “final_model.pkl”.

Steps to test the trained model on testing data are as follows:
1) Prepare Test Data:

Prepare your test data or text on which you want to perform predictions.
Ensure that the test data is preprocessed in the same way as the training data (e.g., tokenized, cleaned, vectorized, encoded, etc.).

2) Load the Saved Model:
Loading the previously saved model using joblib.load from joblib library:

```
# Load the saved SVM model from file
loaded_SVM = joblib.load('svm_model.pkl')
```

3) Perform Predictions:
Using the loaded model to make predictions on the test data:

```
predictions = loaded_SVM.predict(test_data)
```

4) Evaluate Predictions :
From the ground truth labels for our test data, we can evaluate the predictions' accuracy or any other relevant metrics.