Project Statement: Neural Models for Token-Level Python Code Completion

The objective of this project is to construct, assess, and analyze neural network models for token-level Python code completion, utilizing the Python subset of the CodeXGLUE Code Completion (Token-level) dataset. The work will follow a structured methodology, progressing from initial data preparation through the implementation of baseline recurrent architectures, culminating in the development of an enhanced residual model.

Task 1: Data Preparation

The initial phase requires preparing the dataset for subsequent model training and evaluation.

1. **Vocabulary Construction:** A vocabulary comprising the 10,000 most frequently occurring tokens from the training split shall be constructed. All other tokens must be mapped to the `<UNK>` token.
2. **Sequence Normalization:** Each code sequence will be uniformly processed to a fixed length of 1,000 tokens through either padding or truncation.
3. **Validation Split Generation:** The training data must be partitioned into an 80% training set and a 20% validation set, utilizing a fixed random seed of 42 to ensure reproducibility.
4. **DataLoader Creation:** PyTorch `DataLoaders` are required to be built for the finalized training, validation, and test sets.

Task 2: Baseline Models – Implementation, Training, and Comparative Analysis

Two distinct baseline models for next-token prediction must be implemented:

1. **Recurrent Neural Network (RNN):** Implementation using `torch.nn.RNN`.
2. **Long Short-Term Memory (LSTM):** Implementation using `torch.nn.LSTM`.

Both baseline models must adhere to the following architectural specifications:

- **Embedding Layer:** An embedding dimension of 256 is required (the layer may be trainable or pre-trained).
- **Recurrent Layers:** Two recurrent layers, each containing 512 hidden units.
- **Activation Function:** A justified choice of activation function is mandatory.
- **Regularization:** Dropout must be applied between recurrent layers and preceding the final output layer.

- **Output Layer:** A final Linear layer must project the output to the size of the constructed vocabulary.

Training Protocol:

- **Optimizer:** The Adam optimizer must be employed.
- **Loss Function:** `CrossEntropyLoss` is required, configured with `ignore_index` for `<PAD>` tokens.
- **Epoch Management:** Training shall proceed for a determined number of epochs or until validation loss exhibits no further improvement.
- **Evaluation Metrics:** Top-5 Accuracy and Perplexity (PPL) must be calculated and recorded at the conclusion of each epoch.

Task 3: Residual Recurrent Model

The best-performing model (either the RNN or LSTM from Task 2) must be enhanced through the integration of a residual skip connection.

Residual Architecture:

- **Residual Connection:** The output of the first recurrent layer (H_1) must be added to the output of the second recurrent layer (H_2):
  $$H_{final} = H_1 + H_2$$
- **Output Flow:** The resulting H_{final} must be passed into the final output linear layer.

Training and Evaluation:

- The identical training protocol established in Task 2 must be utilized.
- The final Top-5 Accuracy, Perplexity, and total number of trainable parameters must be compared against the performance of the chosen baseline model.
- A formal discussion evaluating the impact of the residual connection on model performance is required.

Overall Project Objective:

Upon successful completion of this project, the student will have executed comprehensive dataset processing, implemented and benchmarked fundamental RNN and LSTM architectures, performed quantitative performance analysis using critical metrics, and successfully implemented and evaluated an enhanced recurrent architecture utilizing residual connections.