Programming Assignment: SMS Spam Detection

Objective:

The goal of this assignment is to build a model that accurately detects spam messages in SMS data (e.g., https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset). You are allowed to use any model or approach, as long as it complies with the requirements. You are expected to experiment with various tokenization and preprocessing techniques to optimize your model's performance.

Reference Dataset:

The dataset can be found here: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset. It consists of a collection of 5,572 SMS messages labeled as "ham" (not spam) or "spam." You are free to preprocess this dataset in any way you deem suitable for improving your model's accuracy.

For testing, an **unseen dataset** will be provided by the grader. This means your model should be trained and validated on the given dataset, but final testing and evaluation will be done on a separate, unseen dataset during the grading process.

Requirements:

1. **Model Selection:**
    - The final model must be saved in a **.pt** or **.pth** format (PyTorch compatible) and should be ready for immediate inference without any retraining.
    - You are free to choose any architecture, but it must be implemented in a way that allows direct loading and inference in PyTorch.
2. **Model Evaluation:**
    - Your model will be evaluated based on its F1-score using an unseen test set. The results must be generated for a test set provided in a file called

test.csv. The predictions should be saved in a file called result.txt, where each line corresponds to a classification of "ham" (0) or "spam" (1) for each message in the test set.

- Output Format: The result.txt file should contain one line per message from test.csv, with 0 for "ham" and 1 for "spam."

3. **Inference Time:**
    - **No retraining should occur.** The submitted model must be ready for inference immediately after loading.
    - Ensure the inference process completes within **0.05 seconds per SMS message** (i.e., processing 1,000 messages should take no more than 50 seconds).
4. **Model Size Limit:**
    - The total size of your submitted model file must be **100MB or smaller** to ensure efficient loading and inference within the time constraints.
5. **Libraries:**
    - **You are only allowed** to use the following machine learning libraries: **scikit-learn, PyTorch, Keras, and Hugging Face.** Other machine learning libraries are not permitted.
6. **Server Environment:**
    - Memory: 32GB
    - GPU: V100 (1 GPU)
    - CUDA version: 11.7
    - Python version: 3.9

Submission Items:

1. **run.py**: This script will load the pre-trained or fine-tuned model and perform inference on `test.csv`. The results should be saved to a file called `result.txt`, where each line corresponds to a classification of "ham" (0) or "spam" (1) for each message in the `test.csv`.

2. **Report**: A PDF report that details your approach, including preprocessing, tokenization, model selection, training process, and analysis of the model's performance. Include comparisons between different tokenization/preprocessing methods and their impact on the F1-score.

3. **requirements.txt**: A list of Python libraries used in the project. This allows the grader to recreate your environment for running the code.

4. **Trained Mode:** The model must be saved in a PyTorch-compatible format (`.pt` or `.pth`).

Evaluation Criteria:

1. Code Functionality: Does the `run.py` code load the model and perform inference correctly?

2. Model Performance: Your model will be evaluated based on its F1-score on a test set within the time limit.

3. Documentation: Your report should clearly explain your preprocessing methods, tokenization, model architecture, training process, and analysis of model performance.