# Assignment #5: NLP, LLM, and Speech

Total score: 100

This assignment can be completed individually or by a team of a maximum of three members.

Due date: See the course page

## Directory Layout:

In the assignment files you'll see a directory structure with 7 python files (see the entire document below for specifics) and then you'll also see three directories labeled "audio", "data" and "text". The "audio" directory contains WAV files, the "data" directory contains PDF, CSV and TXT files and the "text" directory only contains TXT files. Each of these folders contain two sub-directories called "input" and "output". The "input" sub-folder contains all of the initial input data that'll be needed for all of your programs (you don't need to download any dataset – all data are included in the initial assignment files), whereas your programs should output any files they computed to their relative "output" directory. Simply put, the "input" sub-directories hold data that's inputted into your scripts (you shouldn't write to this directory) and the "output" sub-directory contains the files you create/write to. This is to help modularize your program layout and assist with running unit tests cleanly.

## 1. Text Classification for Spam filtering (Total: 15 points)

In this assignment, you will build a classifier for Spam filtering using the email dataset "**L06_NLP_emails.csv**" (this is located in your "./**data/input/L06_NLP_emails.csv**" directory). This dataset contains a collection of email messages, each labeled as spam (1) or non-spam (0). Each row in the file (email message) is considered as a "document" and the entire file considered as a corpus. Fill in the skeleton code from "**_1_spam_filter.py**" that

(a) **Preprocesses the text data**. This should include removing special characters, stop words, whitespaces, lower casing, tokenization, stemming, lemmatization, etc. You can use the **sklearn.feature_extraction.text** package or other NLP libraries. In particular, the **CountVectorizer** class can be used to create a Bag of Words (document-term matrix, DTM). In the DTM, each row represents a document, each column represents a unique word in the corpus, and each cell represents the frequency of the word in that document.

(b) **Train and test the MLP classifier**, and compute the performance in terms of % accuracy (the unit tests require an accuracy of at least 90% to pass).

**(Report Question) Why would we need to remove special characters, stop words, whitespaces, enforce lower casing, implement stemming and lemmatization? What happens if we don't do this and what would change in the DTM if we didn't do these steps (5 points)**

## 2. Fine-tuning a Large Language Model and RAG for the LLM (Total: 60 points)

In this exercise, you will fine-tune a small, lightweight LLM and apply Retrieval-Augmented Generation (**RAG**) using the model:

"**TinyLlama/TinyLlama-1.1B-Chat-v1.0**"

You will modify three separate Python programs to handle fine-tuning, RAG-tuning, and testing. Each program has a specific role to keep the workflow modular and efficient.

Refer to the sample programs that use the TinyLlama LLM.

(a) **Fine-tune the model (10 pts)**

Modify the skeleton code from the script "**2a_tiny_llama_fine.py**" to fine-tune the model on the course syllabus (CPSC_254_SYL.pdf – located at "./data/input/CPSC_254_SYL.pdf"). You will need the following packages (and any dependencies):

- transformers (Hugging Face Transformers library for models, tokenizers, training)
- PyMuPDF (for extracting text from PDF files)
- Datasets (for handling text data and applying transformations)

Obtain a Hugging Face token (personal API key) to access private models and APIs.
Save the fine-tuned model to disk so it can be loaded later by the testing program.

Notes: Ensure the output model directory is clearly named (e.g., tiny_llama_fine_model/) so it can be loaded later.

(b) **RAG-tune the model (10 pts)**

Modify the "**_2b_tiny_llama_rag.py**" skeleton code to apply RAG on the syllabus and save the RAG-tuned model to disk for later testing.

Keep RAG-tuning separate from fine-tuning to avoid unnecessary retraining.

(c) **Test the models**. **(10 pts)**
Modify the "**_2c_tiny_llama_test.py**" skeleton code to evaluate both the fine-tuned and RAG-tuned models.

Load both models, ask at least **three course specific questions** based on the syllabus, and compare the responses from both models and discuss:

- Which model produced more accurate or relevant answers
- Possible reasons for the difference in performance

**(Report Question) Would you expect the fine-tuned model or RAG to underperform on the data provided? What made you come to this conclusion and was it supported by the results you obtained in parts (a)-(c)? Discuss your thought process as you were coding each script. Which challenges were you facing and was there anything odd you noticed about your results? How did you address any problems you encountered? (15 pts)**

**Note: I provided you with the "shopping_summary.csv" file you should've obtained from your Tesseract OCT model in Assignment #4 in order to do part (d) of this question. The file is located at "./data/input/shopping_summary.csv".**

(d) **OCR-based RAG application (10 pts)**. Reference the "true_receipt_contents.txt" file (which contains all of the receipt text data from the prior assignment – I manually transcribed all 7 receipt images for you so you wouldn't have to) and fill out the "**_2d_tiny_llama_ocr.py**" skeleton script to RAG-tune the model on the "true_receipt_contents.txt" file.

Next, use the extracted text data (i.e. the "shopping_summary.csv" file) I provided from Assignment #4 and convert the extracted data into a text file "extracted_receipt_contents.txt" (this workflow should be completed within the "**_2d_tiny_llama_ocr.py**" script as well – you won't need any other scripts to do this). Feed this extracted text to the RAG-tuned model to identify and list all purchased items.

Include screen snapshots showing the interaction between your queries and the model's responses.

**(Report Question) Discuss whether the RAG-tuned model improved the accuracy of text extraction from the receipt images or if there was no improvement. State your original hypothesis on whether the RAG-tuned model would increase or decrease text extraction accuracy and why and then compare it to the results you obtained. (5 pts)**

## 3. Speech Recognition and Voice Synthesis (25 points)

In this exercise, you will integrate STT and TTS functionalities with a RAG-tuned LLM. Refer to the sample programs for STT and TTS examples. **Note:** *I included a "**driver_stt_tts_rag.py**" script that integrates the "stt_rag.py" and "tts_rag.py" into a single workflow. Feel free to experiment with this script, but I won't need it in your submission for this assignment. I also included a "**helper_your_voice_to_wav.py**" if you want to record your voice to a WAV file and input it into your model for fun, but you don't need to submit any audio files. All audio files should be in the "audio" directory, whereas the text translations are in the "text" directory.*

### (a) Speech-To-Text (STT) (5 pts)

Write "**stt_rag.py**" that allows users to ask questions from the syllabus by voice. The program should convert the spoken input into text and use it to query your RAG-tuned model.

### (b) Text-To-Speech (TTS) (5 pts)

Write "**tts_rag.py**" that converts the responses from your RAG-tuned model into speech and plays them aloud.

**(Report Question) Describe the entire workflow architecture of how "driver_stt_tts_rag.py" pieces everything together from the beginning of the assignment to the end of it. Were your results highly accurate/expected? If not, which architectural changes would you suggest to implement in order to improve the overall workflow from start to finish and obtain better results? (15 pts)**

## What to submit

- A report file in word or PDF format that includes the name or names of the team members, and the % contribution of each member. If there is no agreement on individual contributions, briefly describe the tasks assigned and completed by each member. Different scores may be awarded based on individual contributions. If all members contributed equally, simply state "equal contribution." The report should also include descriptions of the answers (if questions are asked) and a portion of the screenshot demonstrating the functionality of each program.
- Each program file individually as well as dependencies, if any. DO NOT submit any zip file as Canvas cannot open them. Here's the files you need to submit **(Note that you do NOT need to submit any unittest files [i.e. files beginning with "test_<script name>.py"] or data [i.e. anything from the "audio", "data" or "text" directories] – only the following ".py" files)**:
  - _1_spam_filter.py
  - _2a_tiny_llama_fine.py
  - _2b_tiny_llama_rag.py
  - _2c_tiny_llama_test.py
  - _2d_tiny_llama_ocr.py
  - _3a_stt_rag.py
  - _3b_tts_rag.py
- Submit only one report file and the necessary file(s) for the entire team.

## Grading criteria

- Does each program successfully perform the required tasks and generate the correct results?
- Does the report demonstrate that the team has a clear understanding of the relevant concepts, programming techniques, functional requirements?
- Does it show how the team applied this knowledge to successfully complete the tasks?
- Is the effort put into the project clearly reflected in both the report and the program files?
- Make sure you answer all of the report questions and upload them in your finalized report file