INSTRUCTIONS

- We are floating **five problem statements** for the course project of AIL861/ELL881/ELL8299.
- **Please get in touch with the consulting TA corresponding to the project you choose for any help or assistance**.
- **Be honest in your work**.
- You will be evaluated based on the rigor of your experiments and analysis; thus, analyze the cause even if you get negative results and report your findings.
- Remember that this course project holds the highest weightage (25%) in the grading of this course - please devote appropriate time to it. If you start the project towards the end of the semester and seek help from us, sorry, we will not be able to help you.
- Please do not copy any part of your report from online sources. If you refer to any source/paper, cite it appropriately. **Any sort of plagiarism will be heavily penalized.**
- **Do not use any GenAI tool (like, ChatGPT) to write your report. You will be penalized for it.**
- **You can, however, adapt or use code from any public GitHub repository. If you do, please mention and cite them in your report.**
- **The report must be written in the ACL format** (Overleaf Template). Submit the non-anonymous version as a PDF with the names of group members.
- **The GitHub Repository with all the source code must be shared with the corresponding consulting TA before the final submission deadline**. The following are the GitHub usernames of the TAs:
    - Shashank Agarwal (shanks733)
    - Prottay Kumar Adhikary (proadhikary)
    - Aswini Kumar Padhi (AsswiniNLP)
    - Anwoy Chatterjee (C-anwoy)
- **If you are interested in doing a self-proposed project, please discuss your proposal with Prof. Tanmoy after the class or meet him in his office or get formal approval by mailing him.**
- **Each project can be taken up by a maximum of 4 groups (each group can have 2 members at max.), allotted based on FCFS**. So select your project quickly, and fill it up in this sheet.
- Last but not least, **enjoy the exploration in your project**. Experiment with any idea you get. Think about the problem. All problems are of an exploratory nature - choose the one that aligns best with your interests.

Project 01: "How Much Can You Trust Your Friend, Who Knows Everything?" Analyzing the Vulnerabilities of Large Language Models in Jailbreaking by Evaluating Their Capacity to Detect Hate Speech and Generate Counterspeech

Consulting TA: Aswini Kumar Padhi (eez238359@ee.iitd.ac.in)

Objective: The goal of this mini-project is to investigate the vulnerabilities of Large Language Models (LLMs) in resisting jailbreaking attempts. Specifically, this study will focus on the LLMs' capabilities in analyzing hate speech and generating counterspeech. The project is divided into two key hypotheses that will guide the experimental procedures.

Research Background: The working principles for this project should be followed as in the methodologies presented in paper 1 and paper 2. These references will provide the foundational frameworks for training and evaluating the LLMs.

Hypotheses and Methodology:

Hypothesis 1: Consistency of Toxic Vector Generation by LLMs

This hypothesis explores whether LLMs produce consistent sets of toxic vectors when exposed to hate speech across multiple instances.

1. **Train a Toxic Comment Classification Model:**
    - Utilize a toxic comment classification dataset to train the LLM.
2. **Identify and Analyze Toxic Vectors:**
    - Determine the span and nature of toxic vectors generated, referencing the methodologies described in paper 1.
3. **Train a Counterspeech Generation Model:**
    - Independently train a separate model for counterspeech generation.
4. **Compare Toxic Vectors:**
    - Analyze whether the toxic vectors identified by the hate speech classification model are consistent with those influencing the counterspeech generation model.

Hypothesis 2: Role of MLP Value Vectors in Generating Non-Toxic Outputs

This hypothesis investigates whether all value vectors in the Multilayer Perceptron (MLP) are involved in producing non-toxic outputs. It aims to determine if these vectors genuinely contribute to non-toxic output generation or if transformations of key vectors play a pivotal role.

5. **Train a Counterspeech Generation Model:**
    - Train a model for counterspeech generation using the CONAN dataset.
6. **Analyze the Role of Toxic Vectors:**

- Investigate whether the value vectors in the MLP are only responsible for generating non-toxic outputs. If inconsistencies are observed, analyze the influence of toxic vectors in this process.

Tools and Resources: For all experiments, consider using a GPT-2 medium model or any comparable model that can be effectively run in Google Colab. Contact us if you face any problems with resources.

Deliverables:

- **Code Repository:** A GitHub repository containing your code for the experiments.
- **Final Report:** A comprehensive analysis report covering the following:
  - The consistency of toxic vectors across models.
  - The role of value vectors in generating non-toxic outputs.
  - Comparative analysis between hate speech classification and counterspeech generation models.

Project 02: Can You Trust the Facts? Analyzing Factuality in Counterspeech Generation Using Large Language Models

Consulting TA: Aswini Kumar Padhi (eez238359@ee.iitd.ac.in)

Objective: This mini-project aims to analyze the factuality of informative counterspeech generated by Large Language Models (LLMs) in response to hate speech. This study will focus on identifying the paths leading to factual inaccuracies and exploring intervention techniques to correct these inaccuracies, thereby improving the reliability and trustworthiness of LLM-generated counterspeech.

Research Background: This project builds on the methodologies presented in the articles "Locating and Editing Factual Associations in GPT" and "Inference-Time Intervention: Eliciting Truthful Answers from a Language Model." These references will provide the foundational frameworks for identifying factual inaccuracies and implementing corrective interventions during counterspeech generation.

Hypotheses and Methodology:

Hypothesis 1: Pathways Leading to Factual Inaccuracies in Counterspeech

This hypothesis explores whether specific internal mechanisms or "knowledge neurons" in LLMs contribute to the generation of factually incorrect counterspeech when responding to hate speech.

- **Step 1: Generation of Counterspeech**

    - Utilize a pre-trained LLM to generate counterspeech for various instances of hate speech within a designated CONAN dataset.
    - Focus on ensuring that the generated counterspeech is informative and attempts to counter or correct the hate speech.
- **Step 2: Identification of Inaccuracies**
    - Analyze the generated counterspeech to identify instances where factual inaccuracies are present.
    - Trace the model's processing paths to determine which internal mechanisms or knowledge neurons contribute to these inaccuracies, referencing the methods described in "Locating and Editing Factual Associations in GPT."

Hypothesis 2: Effectiveness of Inference-Time Intervention in Correcting Inaccuracies

This hypothesis investigates whether Inference-Time Intervention (ITI) can effectively correct factual inaccuracies in real-time during the counterspeech generation process, thereby enhancing the model's reliability.

- **Step 1: Application of Inference-Time Intervention**
    - Implement ITI techniques to intervene during the generation of counterspeech, using external knowledge sources and consistency checks to correct inaccuracies as they occur.
    - Experiment with various ITI methods to determine their effectiveness in aligning the model's output with verified facts.
- **Step 2: Evaluation of Corrected Outputs**
    - Evaluate the factual accuracy of the counterspeech post-intervention.
    - Compare the accuracy of the original versus the corrected outputs, analyzing the success of ITI in improving the reliability of the LLM's responses.

Tools and Resources: For all experiments, consider using a pre-trained LLM, such as GPT-2 medium or large, that can be effectively run on platforms like Google Colab. Access the dataset provided through the given link to conduct your experiments. Contact us if you face any problems with resources.

Deliverables:

- **Code Repository:** A GitHub repository containing your code, experiments, and any scripts used for generating and analyzing counterspeech.
- **Final Report:** A comprehensive analysis report covering the following:
    - The pathways leading to factual inaccuracies in counterspeech generation.
    - The effectiveness of Inference-Time Intervention in correcting these inaccuracies.
    - A comparison between the original and corrected outputs, with a focus on improving the factual reliability of LLMs.

Project 03: From Layers to Latents: Pruning and Aligning LLMs for Efficiency and Safety

Consulting TA: Shashank Agarwal (aiz247543@iitd.ac.in)

Objective: The goal of this project is to explore pruning strategies in large language models and to understand the importance of layers and attention heads. You will experiment with both static and dynamic pruning. In part 2 of the project, you will identify and remove bias and toxic behavior by aligning the model towards safety, and also analyze which parts of the model contribute most to harmful outputs.

Research Background: Please read the papers given below to understand the project. You are free to try different methods other than the ones used in the papers, which you think may produce better results. Papers: **Task 1**, **Task 2**

Tasks and Methodology:

Task 1: Investigating the Importance of Layers and Attention Heads

In this task, you will experiment with different pruning strategies. Please refer to the tools and resources for the model to be used and the datasets for downstream tasks.

1. **Static Pruning:** Experiment by removing entire layers from the model. Perform this for every layer and record the resulting performance changes. Plot accuracy versus the layer removed to visualize which layers contribute most to the model's performance. As a second experiment, design a class that takes a single hyperparameter, p, representing the percentage of model weights to be pruned. Implement the pruning logic inside this class so that when called, it actually zeroes out approximately p% of the model's weights. Plot accuracy versus p and perplexity versus p for your datasets to analyze the effect of pruning.
2. **Dynamic Pruning:** Instead of performing static pruning of MLP layers, you will be pruning attention heads in a dynamic way. Refer to the paper for Task 1 for guidance. Learn a gating network that selects which attention head to focus on for each token. Train the router using a load-balancing loss as described for any Mixture-of-Experts framework. Visualize the distribution of attention heads and compare the model's performance with the baseline.

Task 2: Aligning a Model Towards Safety

The objective of this task is to identify and reduce biases present in a language model and align it towards safer behavior.

1. **Dataset Creation:-** Construct a dataset containing a mix of neutral and negative prompts. Negative prompts can be drawn from any category of undesired

content, such as biased, toxic, or harmful text. Set aside a portion of the dataset for validation.

2. **Identifying the target layer and safety alignment:-** Implement a small neural network at each layer of the model and train it on your dataset to classify whether a given prompt is negative or neutral. During training, plot the loss versus epoch for each layer to understand which layers are learning the most about harmful content. After training is complete, evaluate the classifiers on a test dataset to determine which layer's classifier performs best, as the layer with the highest accuracy is considered to store the most relevant information regarding harmful behavior. Report the accuracy of the classifier at every layer. As a final task, Construct the harmful vector at the chosen layer, representing the model's harmful direction in the activation space. Please refer the paper on how to get the harmful vector using the model's internal representation. Use this vector to remove harmful behavior from the model. You may explore different approaches, including training-based or non-training-based methods. Evaluate the model's performance on the test set to demonstrate the effectiveness of the safety alignment.

Tools and Resources: For Task 1, you will conduct your experiments using two datasets: MMLU and GSM8K. (10% of test data for both datasets should suffice). You may use any small language model that can run on Kaggle GPUs, such as LLaMA 1B or Qwen3-0.6B. For Task 2, you can use any non-instruction-tuned model, for example GPT-2 XL or GPT-2 Large.

Fundamental Concepts for Further Reading:

- **Low-Rank Adaptation (LoRA):** A parameter-efficient fine-tuning (PEFT) technique.
- **Model Merging:** Techniques for combining the weights of two or more models.
- **AI Safety and Security:** The broader field concerned with preventing unintended and malicious behavior from AI systems.

Deliverables:

- **Code Repository:** A GitHub repository containing your code for the experiments and visualization of results.
- **Final Report:** A comprehensive analysis report presenting and analyzing your findings in the project.

Project 04: "How Secure is Your Specialized Knowledge?" An Investigation into Weight-Stealing Attacks and Defenses in Multi-Agent LLM Systems

Consulting TA: Prottay (proadhikary@ee.iitd.ac.in)

Objective: The goal of this project is to simulate and analyze a sophisticated security threat in multi-agent systems: knowledge stealing. We will create two Large Language Model (LLM) agents, an **Attacker** and a **Victim**. The Victim model will be fine-tuned on distinct, specialized domains. The Attacker's objective is to extract this specialized knowledge, not through conversational probing, but by directly stealing the Victim's fine-tuned LoRA (Low-Rank Adaptation) adapter weights from its file system.

Research Background: The idea for this project is inspired by foundational work in knowledge stealing by Tramèr, Florian, et al. "Stealing Machine Learning Models via Prediction APIs." Link to paper. This paper establishes the precedent for extracting model functionality and architecture through black-box queries, a principle we extend to a more direct file-based attack.

Methodology:

1. **Model Preparation and Fine-Tuning:**
    - **Base Models:** Instantiate two `meta-llama/Llama-3.2-1B` models. One will serve as the Attacker, the other as the Victim.
    - **Victim Specialization:** Fine-tune the Victim model using LoRA on two books to imbue it with specialized knowledge. **Book 01**, **Book 02**
    - The resulting LoRA adapters, which contain the specialized knowledge, will be saved to a simulated file system.
2. **Agent and Environment Setup:**
    - **LangChain Agents:** Develop both the Attacker and Victim as autonomous agents using the **LangChain framework.**
    - **Victim's Persona:** The Victim agent will be instructed to be helpful and answer questions, but to avoid revealing specifics about its training or internal architecture. This model should be finetuned on Book 02, which is rich in information.
    - **Attacker's Toolkit:** Equip the Attacker agent with tools to interact with a simulated environment. Crucially, this includes a Python interpreter tool, **PythonREPLTool, or a Shell tool** that allows it to execute code, list files, read file contents, and copy files within the shared directory. This model only knows about personality from Book 01, but to know more, it will try to steal the knowledge from the victim.
    - **The Heist:** The Attacker's primary goal is to use its tools to locate and copy the Victim's LoRA adapter files (`adapter_model.bin`) and other necessary files. It

will do this by systematically exploring the file system and identifying files associated with the Victim model.

3. **Knowledge Integration and Evaluation:**
    - **Model Merging:** Once the Attacker successfully acquires the LoRA weights, it will use a merging technique (e.g., TIES-Merging) to integrate the stolen adapters into its own base model.
    - **Verification:** The success of the attack will be quantified by evaluating the newly merged Attacker model(test set will be released). A significant performance increase compared to its baseline will confirm the successful knowledge transfer.

Deliverables:

- **Code Repository:** A GitHub repository containing the complete code for setting up the agents, performing the fine-tuning, executing the attack, and evaluating the final merged model, and the model weights pushed on Huggingface.
- **Final Report:** A detailed report analyzing:
    - The strategies and tools the Attacker used to successfully locate and steal the weights.
    - A quantitative analysis of the merged model's performance on the target domains.
    - A discussion on the security implications of such attacks and potential defense mechanisms (e.g., file system permissions, agent sandboxing).

Project 05: Building a Low-Latency Mixture-of-Experts Orchestrator for Domain-specific data

Consulting TA: Prottay (proadhikary@ee.iitd.ac.in)

Objective: This project aims to design and implement a lightweight, CPU-friendly, and fast **Mixture-of-Experts (MoE)-like architecture** from scratch. The system will feature a central **Orchestrator** that intelligently routes user queries to one of several specialized, small LLMs. Each "expert" LLM will be fine-tuned on a specific mental health domain (e.g., depression, anxiety). The core focus is on achieving high performance and relevance with minimal computational resources and latency, making it suitable for real-world, resource-constrained applications.

Research Background: This project's architecture is grounded in the principles of sparse model activation, as detailed in the following paper, which introduced the modern MoE concept.

Shazeer, Noam, et al. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer." *arXiv preprint arXiv:1701.06538*. 2017. Link to paper. This seminal paper provides the theoretical foundation for training vast models where only a fraction of the parameters are used for any given input.

Methodology:

1. **Expert Model Development:**
    - **Base Model:** Use the `meta-llama/Llama-3.2-1B` model as the foundation for all experts due to its strong performance and ability to run on a CPU. Check out Google AI Edge Gallery for inspiration.
    - **Specialized Fine-Tuning:** Fine-tune five separate instances of the base model on specific mental health datasets from Hugging Face(will be provided)
2. **Orchestrator (Router) Design:**
    - **The "Gating Network":** The Orchestrator's role is to act as a fast and accurate router. Instead of a complex neural network, we will implement a classic, computationally inexpensive classifier, such as a **Multinomial Naive Bayes** or a **Logistic Regression** model using `scikit-learn`.
    - **Training the Orchestrator:** Create a balanced training dataset for the orchestrator by sampling prompts from each of the five fine-tuning datasets and labeling them with the corresponding category (0 for Depression, 1 for Anxiety, etc.). The text will be converted to numerical features using a `TfidfVectorizer`.
    - **Routing Logic:** When a new user query arrives, it will be vectorized and fed into the trained Orchestrator, which will instantly predict the most relevant expert. The query is then passed to that specific fine-tuned model for a response.
3. **System Integration and Evaluation:**
    - **Full Pipeline:** Build the end-to-end system where an input query is received, routed by the Orchestrator, and processed by the selected expert LLM.
    - **Performance Metrics:** The system's success will be measured by:
        - **Orchestrator Accuracy:** How accurately does the router classify incoming prompts to the correct expert domain?
        - **End-to-End Latency:** Measure the total time from receiving a query to generating a complete response on a standard CPU.
        - **Response Quality:** A qualitative analysis of the expert models' responses to ensure they are relevant, coherent, and specialized.

Fundamental Concepts for Further Reading:

- **Mixture of Experts (MoE):** An architecture that enables models to learn specialized sub-networks (experts) and use a gating network to route inputs to the most relevant ones.
- **Model Compression & Quantization:** Techniques to reduce the size and computational cost of models, making them faster and more efficient.

- **Efficient Decoding Strategies:** Methods like speculative decoding that speed up the text generation process in LLMs.

Deliverables:

- **Code Repository:** A GitHub repository with the code for fine-tuning the expert models, training the orchestrator, and the final integrated MoE-like application, and the model weights pushed on Huggingface.
- **Final Report:** A comprehensive report that includes:
    - An analysis of the Orchestrator's classification performance (accuracy, precision, recall).
    - Detailed latency benchmarks for the entire system under various loads.
    - A comparative analysis of the quality of responses from the specialized experts versus a general-purpose base model.
    - Discussion on the trade-offs between this lightweight approach and more complex, GPU-intensive MoE architectures.