# Evaluating data Generation methodologies for fine-tuning LLM

Summary of Project: The primary goal of the project is to explore and evaluate different data generation methodologies for fine-tuning a large language model (LLM). The aim is to determine which methodology yields the best performance based on specific evaluation metrics.

## Objectives

* Explore various data generation methodologies for the fine-tuning of LLMs.
* Evaluate the fine-tuned LLMs using specific performance metrics.
* Determine the best data generation methodologies for fine-tuning an LLM.

## Scope

* Research available open-source datasets and LLMs.
* Develop code to run and fine-tune LLMs.
* Generate data using different methodologies for fine-tuning.
* Fine-tune the LLM using generated datasets.
* Evaluate the performance of the fine-tuned LLMs.

## Expected Outcomes

* Working code for automatic data generation for fine-tuning an LLM.
* Working code for running and fine-tuning an LLM.
* Analysis and comparison of different data generation methodologies.
* Fine-tuned LLM models with performance evaluations.
* Documentation and reports summarizing the findings and methodologies.

## Data

* Use public datasets related to the nuclear industry, such as books, research papers, articles, journals, etc.
* Ensure the datasets are free for commercial use.

## Methodologies/ plan of attack

* 1. Perform research on Data, and LLMs (1 weeks)

Tasks:

* + - Identify available open-source datasets related to nuclear industry.
    - Investigate methods to collect the data efficiently.
    - Identify latest available open-source LLMs such as NuclearN.ai, LLAMA, Grok, etc.
    - Access the performances evaluation criteria of LLMs such as MMLU, TruthfulQA, etc while selecting an LLM.
    - Assess the data format and requirements for fine-tuning LLMs.
    - Access the computational resources required for inferencing, fine-tuning, etc.
    - Ensure data and models are free for commercial use.

Outcomes:

* + - Quick progress update and feedback session.
    - Selected dataset resources.
    - Selected a few suitable LLMs for the project.
    - Identified the specific fine-tuning requirements for the chosen model.
    - A written summary of findings for every step.
  1. Gather data (0.5 week)

Tasks:

* + - Collect the data using the identified methods.

Outcomes

* + - Collected data with a good directory structure.
  1. Develop code to run Open-Source non-fine tuned LLM (0.5 week)

Tasks:

* + - Obtain the model files from a trusted source or repository (e.g., Hugging Face, GitHub).
    - Set-up virtual environment.
    - Install necessary libraries and frameworks.
    - Develop inference logic to generate responses from the model.

Outcomes:

* + - An inference ready LLM integrated into your environment, capable of processing and responding to user inputs.
  1. Data Generation for fine-tuning LLM (3 weeks)

Tasks:

* + - Research and identify different methodologies for generating fine-tuning data from the selected dataset.
    - Ensure that the data generation methodologies align with the requirements of the chosen LLM.
    - Develop code to implement data generation methodologies.

Outcomes:

* + - Script(s) that can be used to generate data for fine tuning LLM utilizing the selected methodologies.
    - Multiple directories, one for every selected data generation methodology, containing data in proper format for fine-tuning LLM.
  1. Fine-tuning LLM (2 weeks)

Tasks:

* + - Research and identify different techniques to fine-tune an LLM.
    - Fine-tune the chosen LLM model on each of the different datasets.
    - Document the process, challenges, and solutions encountered during fine-tuning.

Outcomes:

* + - Fine-tuned models, one for every selected data methodology.
    - Detailed documentation of the fine-tuning process and challenges.
  1. Model Evaluation (1 week)

Tasks:

* + - Identify and select appropriate metrics for evaluating the performance of the fine-tuned LLM models.
    - Evaluate each fine-tuned model based on the selected metrics.
    - Perform a comparative analysis of the models fine-tuned with different data generation methodologies.
    - Analyze the evaluation results to determine which data generation methodology produced the best performance.
    - Offer a detailed explanation and justification for the chosen methodology based on the evaluation metrics.
    - Suggest potential improvements or future work.

Outcomes:

* + - Evaluation reports for each fine-tuned model.
    - Comparative analysis report highlighting the strengths and weaknesses of each data generation methodology.
    - Recommendations for the best data generation methodology based on the evaluation metrics.

# General Guidance

## Project Management

Effective project management requires using a task management dashboard, such as GitHub project or Azure DevOps, to track tasks and milestones.

Roles and responsibilities must be clearly defined and assigned for effective problem tackling. A person should be identified as a project manager lead, to recommend backlog items and assign to each person. As well as updating tasks onto a PM tool.

Regular update meetings should be held weekly to review progress, address issues, and adjust plans as necessary. The meeting will be led from the identified PM of the group, and the supervisor will provide further guidance and confirmation to the team.

## Coding Practices

Code should be organized into clear, functional modules to enhance readability and maintainability. Following a consistent coding style and naming conventions is crucial for a uniform codebase.

Some general considerations:

1. Make modular code by having classes and functions: Utilize classes and functions to encapsulate functionality, promote reusability, and organize code logically.
2. Create a virtual environment for your project, keep track of the list of libraries and keep the versions consistent. Consider use tools like “pipenv” to manage the libraries and collaborate with team.
3. Include docstrings: Provide comprehensive docstrings for all functions, methods, and classes to describe their purpose, parameters, and return types.

* Docstrings should include a quick **description** of the function, **args**, and **output**.

1. Include type definitions: Define types for function parameters and return values to improve code clarity and help catch errors early.
2. Adopt script writing: Structure scripts to separate data processing, analysis, and utility functions into distinct modules, making the code easier to navigate and maintain.

## Version Control

Using **GitHub** for version control is essential for managing code changes and collaboration.

Initially, a public repository needs to be created and can be used for access and collaboration, but it should be migrated to a private repository for production to ensure security and control.

Adopting a clear branching strategy; For team members contributing to the code, either fork the repository or make a separate branch to avoid conflicts. Meaningful commit messages that describe the changes made should be written to aid in understanding the project history.

Pull requests should be used to review and discuss code changes before merging them into the main branch.

## Testing and Deployment

Robust testing and deployment practices are critical for ensuring the reliability and scalability of the application.

Unit tests should be written for individual functions and modules using frameworks such as pytest. Efficient testing/unit-tests is a integral part of the final product.

tests should ensure that different parts of the application work together as expected.

If time permits, setting up continuous integration (CI) pipelines to automatically run tests and checks on code commits helps maintain code quality. Continuous deployment (CD) pipelines should automate the deployment process to various environments, streamlining the release process.

## Documentation

### Code Documentation

Start with a comprehensive **README** file at the root of the project repository, providing an overview of the project, its purpose, main features, and instructions for setting up the development environment. Include a list of prerequisites, such as software dependencies and system requirements.

### Methodology Documentation

Create a **word document** at the beginning, serving as a dynamic tracker, and documentation for the project. As the project progresses, continue to update the documentation with findings from experiments, analyses, and testing. Include detailed records of methodologies used, such as data collection processes, analysis techniques, and tools employed. Document any challenges encountered and how they were addressed, along with any changes in strategy or scope.

This document will ensure that lead, and future teams can quickly get up to speed and understand the project's evolution.