Team 3: Generative AI for Advanced Text Summarization

Objective

- Develop a model that generates concise summaries of long documents or articles. Enhance these summaries with contextually relevant information generated by the AI, such as background information, definitions, or explanations that may not be explicitly detailed in the text but are inferred by the model. This could serve well in academic, corporate, or legal environments where detailed summaries can aid decision-making or learning.
- Public Datasets:
    - CNN/DailyMail dataset, which is a large and widely used dataset for training summarization models. It contains news articles paired with multi-sentence summaries.
    - News API Dataset, is a comprehensive collection of real-time news articles from global sources.

Performance Metrics

- Effectiveness (Predictive and Generative Performance):
    - Rouge Score: Measures the overlap of n-grams between the generated summaries and reference summaries.
    - Consistency and Coherence: Evaluate how logically sound and consistent the generated expansions are with the base content.
- Efficiency (Runtime):
    - Generation Time: Measures the time required to generate a summary.
    - Computational Resources: Assesses the model's demand on computational resources, which is crucial for deployment scenarios.

Minimum Performance Required

- TBD: Set specific targets for Rouge scores and consistency evaluations after preliminary model iterations and based on what is realistic given the model design and dataset capabilities.

Metrics Presented to Key Stakeholders

- Technical Stakeholders:
    - Detailed metrics on the quality of the summaries (Rouge scores, consistency, coherence).
    - Insights into the generative aspects of the model and how it enhances basic summarization.
- Non-Technical Stakeholders:
    - Visualizations of before and after enhancements to summaries to show the added value.

- Examples of practical applications of the summaries in different scenarios (e.g., summarizing earnings reports for stakeholders, briefing documents for legal cases).

---

1. Initial Setup and Tools

- Objective: To set up all necessary infrastructure and tools required for the project to ensure a smooth development process, which supports collaboration, version control, and reproducibility across Team 3's members.

- Tasks:
    - Version Control System Setup:
        - Create a GitHub repository named generative-ai-text-summarization.
        - Initialize with a comprehensive README.md that includes the project's description, setup instructions, and contribution guidelines.
    - Folder Structure Organization:
        - Establish well-defined directory structures:
            - data/ for datasets and data-related utilities.
            - notebooks/ for Jupyter notebooks to perform exploratory data analysis and initial model development.
            - src/ for all source code including utility scripts and feature engineering code.
            - models/ for storing trained model binaries and serialization files.
            - api/ for the FastAPI application files for deploying the model.
            - docs/ for all documentation needs, including project reports and API documentation.
    - Environment Setup:
        - Use Python virtual environments (via venv or conda) for managing dependencies.
        - Install critical libraries such as TensorFlow or PyTorch for deep learning, spaCy or NLTK for NLP tasks, and Transformers from Hugging Face for utilizing state-of-the-art summarization models.
        - Document the exact environment setup in an environment.yml or requirements.txt file for easy replication.
    - Development Tools Installation:
        - Set up MLflow within the local development environment for tracking experiments and model versions.
        - Configure Hyperopt for efficient hyperparameter tuning.
        - Install Sphinx for generating comprehensive documentation of the codebase and usage.

        - Prepare initial Docker configurations, focusing on a basic Dockerfile that encapsulates the Python environment and dependencies.
    - Collaboration Practices Setup:
        - Establish coding standards based on PEP 8 for Python.
        - Set up branch policies on GitHub for feature branches and pull requests to encourage best practices in code reviews and version control.
        - Implement issue tracking using GitHub Issues to manage tasks, bugs, and feature requests effectively.
- Estimated Completion Time: 2-3 hours
- Hints:
    - Version Control: Regularly commit even small changes to GitHub to avoid loss of work and maintain a backup. Use descriptive commit messages to help team members understand changes.
    - Library Installation: When installing libraries, start with installing the most critical ones first and test their installation. For compatibility, sometimes it is necessary to install a specific version of a library.
    - Docker Use: Familiarize yourself with basic Docker commands. Initially, build your Docker image with just the essential tools to ensure everything is working before adding more complex setups.
    - Documentation Early: Start writing documentation early in the process. Use Sphinx's autodoc feature to automatically generate documentation from docstrings in the code.
    - Environment Replication: Make sure all team members can replicate the development environment by providing a detailed setup guide in the README.md and using the requirements.txt or environment.yml files.

---

2. Project Planning

- Objective: Efficiently outline the project's scope, objectives, and deliverables to ensure a clear understanding and effective execution within the constraints of a small team and limited time.
- Tasks:
    - Define Project Scope and Objectives:
        - Clearly articulate the goals, expected outcomes, and functionalities of the summarization model. Keep this concise but comprehensive to ensure both team members are aligned.
    - Outline Deliverables:
        - Identify and list the key deliverables for each phase:
            - Data dictionary and data sources.
            - EDA notebooks.

        - Model development scripts.
        - API code and documentation.
        - Final presentation materials and project report.
    - Develop a Simplified Timeline:
        - Assign a rough timeline for each major task, ensuring that the total does not exceed the 36-hour project limit. This should include buffer time for unexpected issues.
    - Assign Roles and Responsibilities:
        - Divide the tasks between the two team members, balancing the workload and playing to each member's strengths or learning goals. Roles might overlap, but clear responsibilities will help avoid duplication of effort.
    - Risk Assessment and Mitigation Strategies:
        - Quickly identify any potential risks that could derail the project, such as technical challenges with the model or data issues. Decide on straightforward solutions or alternatives.
    - Set Up Communication and Documentation Standards:
        - Decide on an informal yet effective method for ongoing communication, such as regular check-ins via chat or phone.
        - Agree on a simple documentation process using Markdown files in the GitHub repository to track changes and decisions.

- Estimated Completion Time: 1-2 hours

- Hints:
    - Direct Communication: Since the team is small, maintain open and continuous communication through direct messaging or calls to resolve issues swiftly.
    - Prioritize Tasks: Focus on prioritizing tasks that are critical to the project's success and can be realistically achieved within the allotted time.
    - Streamlined Documentation: Utilize templates for documentation to save time; this can be as simple as predefined headers in your Markdown files.
    - Continuous Integration: Make use of GitHub's features like issues and milestones to keep track of progress and ensure nothing is overlooked.

---

3. Data Collection and Preparation

- Objective: Collect and prepare datasets that will be used to train and retrain transformer and GPT models for text summarization. Ensure the data is clean, well-documented, and stored in an organized manner.

- Tasks:

- Identify Data Sources:
    - Use the CNN/DailyMail dataset, accessible via an API from the Hugging Face Datasets library.
    - Scrape new.org, via their API.
- Data Access via Hugging Face API:
    - Install the Hugging Face datasets library if not already installed: pip install datasets.
    - Use the library to download the CNN/DailyMail dataset:
    - Save the downloaded data in a CSV or JSON file under the data/ directory.
- Data Access via newsapi:
    - Install the NewsAPI Python client library to interact with the NewsAPI
    - Use the NewsAPI client to fetch 100 news articles
    - Save the fetched data in a CSV or JSON file under the data/ directory.
- Data Cleaning and Preprocessing:
    - Create scripts in the src/ directory to clean and preprocess the data. This includes removing HTML tags, handling special characters, and splitting the data into training and validation sets.
    - Ensure that the data is in a suitable format for training the models, such as a list of text-summary pairs.
- Document Data Sources and Preprocessing Steps:
    - Update the data dictionary in the docs/ directory, detailing the sources, format, and any preprocessing steps applied to the datasets.
    - Ensure that the README in the data/ directory explains how to access and use the datasets.

- Estimated Completion Time: 3-4 hours

- Hints:
    - Check API and Website Changes: Regularly check if there are any changes to the API or website layout from where you are scraping data to ensure your scripts remain functional.
    - Automate Data Updates: Consider automating the scraping script to run at regular intervals if the project requires the latest data, ensuring that your model can be retrained on the most recent data.
    - Use Version Control: Commit the preprocessing scripts and documentation updates to your GitHub repository to track changes and collaborate efficiently.

---

4. Exploratory Data Analysis (EDA)

- Objective: Conduct a thorough exploratory analysis of the data to identify key characteristics, detect anomalies, and prepare the data for effective model training.
- Tasks:
    - Setup Analysis Environment:
        - Open a new Jupyter Notebook in the notebooks/ directory.
        - Import necessary libraries
    - Load and Review Data:
        - Load the CNN/DailyMail dataset and the web-scraped dataset from the data/ directory
        - Execute basic data checks such as viewing the first few rows, checking for null values, and understanding data types:
    - Text-Specific Analysis:
        - Analyze the distribution of document and summary lengths. Calculate statistics such as average length, standard deviation, and range:
        - Plot histograms for document and summary lengths:
        - Use NLTK to analyze the frequency of words in the documents and summaries:
    - Visualize Data Insights:
        - Create visualizations to understand the commonalities and differences between the datasets:
        - Identify outliers in text lengths and visualize word frequency distributions to detect skewness or anomalies.
    - Document Insights and Observations:
        - Summarize the findings from the EDA in a Markdown document or directly in the Jupyter Notebook. Include observations about data quality, anomalies, and any preprocessing needed.
        - Update the docs/ directory with a detailed EDA report that includes visualizations and insights.
    - Prepare Data for Modeling:
        - Based on the EDA findings, preprocess the data by cleaning text, handling outliers, and normalizing lengths where necessary. Document these steps in the Jupyter Notebook and save the scripts in the src/ directory.

- Estimated Completion Time: 2-3 hours

- Hints:
    - Ensure Data Quality: Pay special attention to any inconsistencies in data formatting, especially when combining datasets from different sources.
    - Leverage Visual Analysis: Use the power of visual tools extensively to get a better grasp of the data structure and to share these insights effectively within your reports.

- Continuously Save Work: Regularly save your Jupyter Notebooks to avoid losing any work and commit changes to GitHub to maintain a record of your analysis progression.

---

5. Model Development

- Objective: Develop a custom transformer model using the Keras framework and then enhance the project by retraining a pre-trained model for comparison.
- Tasks:
    - Develop Custom Transformer Model with Keras:
        - Build a simple transformer architecture suitable for text summarization tasks. Implement this using Keras and TensorFlow
        - This model will be used for initial experiments. Configure it to handle sequences of tokenized text data suitable for summarization tasks.
    - Select and Retrain Pre-trained Model:
        - Utilize a pre-trained model such as BART from the Hugging Face Transformers library optimized for TensorFlow. Adapt it for the summarization task by retraining on the dataset.
    - Setup MLflow for Experiment Tracking:
        - Configure MLflow to log the training processes for both the custom model and the retrained BART model
    - Training and Evaluation Setup:
        - Prepare the data loaders and define the training routines for both models. For the custom model, ensure data is appropriately preprocessed and tokenized
        - For BART, adapt the input to match the tokenizer's format and utilize the custom training loop provided by Hugging Face if needed.
- Estimated Completion Time: 4-5 hours
- Hints:
    - Modular Code: Keep your code modular to facilitate easy switching between the custom model and the BART model for different experiments.
    - Documentation: Document each step in your model development process, including parameter settings and the rationale behind chosen architectures.
    - Version Control: Commit all changes, especially new scripts and configurations, to GitHub to maintain a robust version history.
    - Continuous Monitoring: Regularly monitor training progress using TensorBoard integrated with Keras to visualize performance metrics.

---

6. Model Training and Validation

- Objective: Train and validate the custom and pre-trained transformer models to ensure they meet the defined performance criteria, focusing on both predictive and generative effectiveness and computational efficiency.
- Tasks:
    - Configure Computing Resources:
        - Enable GPU acceleration in Google Colab to ensure efficient model training
    - Train Models:
        - Define training procedures for both the custom transformer model and the pre-trained BART model. Adjust learning rates, epochs, and batch sizes based on initial tests to optimize training
    - Set Up MLflow for Tracking Experiments:
        - Configure MLflow within Google Colab to track experiments, log parameters, and metrics. Since MLflow UI might require local hosting, consider logging the results to a public cloud or a setup that allows remote access
    - Validation and Performance Metrics:
        - Implement validation procedures using the ROUGE metric to assess summary quality and other custom scripts to evaluate runtime efficiency and resource utilization
    - Document Training and Validation Processes:
        - Record configurations, results, and insights comprehensively. Use the docs/ directory for detailed documentation and the notebooks/ directory for interactive Jupyter notebooks showcasing the training and validation process.
        - Store trained models and checkpoints in the models/ directory, with clear naming conventions to distinguish between different training sessions and model configurations.

- Estimated Completion Time: 6-8 hours

- Hints:
    - Monitor GPU Usage: Regularly check GPU usage in Colab to ensure that training is proceeding efficiently. Adjust batch sizes if necessary to optimize GPU utilization.
    - Data Handling: Manage data loading and batching efficiently to avoid memory overflows, especially when working with large datasets in Colab.

---

7. Model Auditing

- Objective: Ensure the model's performance is ethical, fair, robust, and secure from adversarial threats.
- Tasks:
    - Bias and Fairness Testing:
        - Utilize the AI Fairness 360 toolkit to conduct fairness assessments. Measure disparate impact and statistical parity differences to identify any potential bias:
        - Address significant disparities by adjusting model training processes or revisiting feature engineering.
    - Robustness Testing:
        - Test the model against adversarial attacks using the Adversarial Robustness Toolbox. Implement the Fast Gradient Sign Method to evaluate the model's defense mechanisms:
    - Performance Consistency Check:
        - Conduct cross-validation to assess the model's performance consistency across different data splits:
    - Document Auditing Processes and Outcomes:
        - Document the auditing methods, findings, and any remedial actions taken in detailed reports stored in the docs/ directory.
        - Ensure the auditing documentation includes specific recommendations for enhancing model security and fairness based on the tests conducted.
- Estimated Completion Time: 3-4 hours
- Hints:
    - Regularly update the model's training dataset and re-evaluate fairness and robustness metrics to ensure ongoing compliance with ethical standards.
    - Involve domain experts in reviewing the fairness assessments and interpreting the outcomes to align remediation strategies with industry best practices.

---

8. Model Serving and Containerization

- Objective: Prepare the model for real-world accessibility by containerizing the full model and data preprocessing pipeline and developing an API for interaction.
- Tasks:
    - Containerize Model and Preprocessing Scripts:
        - Develop Docker containers to encapsulate different parts of the application, ensuring each component can be managed and scaled independently.

        - Create Dockerfiles:
            - Model Serving Container
            - Model Training Container
            - Data Preprocessing Container
            - Data Collection Container
            - Place Dockerfiles in the project root directory. Use the src/ directory to store the source code for serving and preprocessing scripts.
    - Develop an API Using FastAPI:
        - Write an API in FastAPI to handle model inference requests. This API should be containerized in its Docker container for isolation and scalability
            - Include this script in the api/ directory and ensure it's added to the corresponding Docker container for the API service.
    - Container Management and Orchestration:
        - Use Docker Compose to manage multi-container setups, ensuring each part of the application communicates effectively
            - Place docker-compose.yml in the project root directory to coordinate between the various services.
    - Document Containerization and API Details:
        - Update the docs/ directory with detailed instructions and architecture diagrams explaining how the containers are set up and how they interact.
        - Ensure the API documentation is clear, providing examples of how to send requests and what responses to expect. This information should be included in the api/ directory.
- Estimated Completion Time: 3-4 hours
- Hints:
    - Test Locally: Before deploying, test the Docker containers locally to ensure everything works as expected, particularly the inter-container communication.
    - Security Practices: Implement security best practices in your Dockerfiles, such as using non-root users and specifying exact base images.

---

9. Project Documentation and Demo Preparation

- Objective: Compile comprehensive documentation of the entire project and prepare a detailed presentation and live demonstration to showcase the capabilities and potential impact of the developed models.
- Tasks:
    - Compile Project Documentation:
        - Document each phase of the project comprehensively. This includes the initial setup, data collection, model development, training/validation, auditing, and deployment:

        - Create a master document in the docs/ directory that links to detailed documents for each phase. Ensure each document includes objectives, methods, results, and conclusions.
        - Include code snippets, configuration details, and usage instructions to ensure the documentation is practical for both technical and non-technical stakeholders.
        - Prepare troubleshooting guides and FAQs based on common issues encountered during the project.
    - Prepare the Final Presentation:
        - Develop a PowerPoint presentation or a similar format to summarize the project's scope, process, key findings, and conclusions:
            - Use clear, concise slides to outline the problem statement, methodologies, key data insights, model performance, and the benefits of the solution.
            - Include visual aids such as charts, graphs, and diagrams to illustrate technical concepts and model performance effectively.
            - Ensure the presentation narrative is compelling and logically structured to maintain audience engagement.
    - Organize a Live Demonstration:
        - Set up a live demonstration of the model to illustrate its real-world application and effectiveness:
            - Prepare a demo script that guides the presentation, showing both the input and output of the model, and highlighting its features and benefits.
            - Use the FastAPI interface to show how the model can be interacted with in real-time during the demo.
            - Test the demonstration setup thoroughly to ensure smooth execution on the day of the presentation.
    - Final Review and Rehearsal:
        - Conduct a final review of all documentation and the presentation materials with the team to ensure accuracy and completeness.
        - Rehearse the presentation and demo multiple times, ideally in front of an audience from within the organization for feedback, to refine delivery and content clarity.
- Estimated Completion Time: 4 hours
- Hints:
    - Feedback Loop: Incorporate feedback from rehearsals to refine the presentation and demonstration, focusing on clarity, pacing, and technical accuracy.
    - Backup Plans: Prepare backup materials and technical contingencies for the demonstration to handle unexpected issues such as connectivity problems or technical glitches.
    - Engage the Audience: Plan for a Q&A session post-presentation to address stakeholder questions and further engage them with the project outcomes.