**Deliverables**

Notebook link:

<https://drive.google.com/file/d/1NELvqA4y-gLJd_RVLrg4-FlaHcW53TI4/view?usp=sharing>

Sample data link (snapshot at the end):

https://drive.google.com/file/d/1tK6Pr54fYAVlqt7TPjUD9s-Gdkr46wg5/view?usp=sharing

**Assignment Report**

**Dataset Sources and Total Size**

The raw dataset for this assignment was a subset of the **wikimedia/wikipedia** dataset, specifically the **20231101.en** configuration. This public dataset, hosted on the Hugging Face Hub, was chosen as the primary source due to its encyclopedic nature and its suitability for pre-training large language models.

To avoid downloading the entire corpus (which is tens of gigabytes), a memory-efficient approach using Hugging Face's **streaming mode** was employed.

A computer code on a black background

Description automatically generated

To get the required **1GB** of raw text, we collected and processed approximately **150,000 articles** into an in-memory Dataset object.

A screen shot of a computer

Description automatically generated

**Cleaning Strategies and Reasoning**

The preprocessing pipeline was designed to transform the raw text into a high-quality, normalized corpus for pre-training.

Two custom functions were written.

1. clean\_and\_normalize(). It leverages regular expression to remove HTML tags and Wikipedia internal links, converts all letters to lower case, removes irrelevant symbols and white spaces.
2. filter\_short\_documents(). It removes all documents shorter than 50 words.

Than we apply map() function on the raw dataset, and filter() function on the cleaned and normalized dataset, as this separation would maximize efficiency of those two functions.

**A screen shot of a computer program

Description automatically generated**

**Tokenization Choices**

We chose to use a **WordPiece** tokenizer, specifically the **bert-base-uncased** tokenizer from Hugging Face. This is a standard choice for transformer-based models and is well-suited for English text.

A key decision was the **chunking strategy** to handle the very long documents found in the Wikipedia dataset. We set a **max\_length of 512** tokens, which is a common block size for models like BERT. To avoid data loss and preserve context, we used an **overlap stride of 128 tokens** between consecutive chunks. This ensured that no information was lost at the boundaries of the chunks.

A screenshot of a computer program

Description automatically generated

**Data Loader Implementation Details**

The data loader was implemented using the standard PyTorch DataLoader and was designed to work efficiently with the tokenized, streaming data.

* **Data Structure:** The tokenized data was maintained in an efficient, in-memory Hugging Face Dataset object.
* **Batching and Padding:** We used a **DataCollatorWithPadding** to dynamically pad each batch to the length of the longest sequence within that batch. This is more memory-efficient than padding all sequences to the maximum sequence length (512) and significantly speeds up training.
* **Iterable Streaming:** The DataLoader was built on a streaming dataset, meaning it processed data on-the-fly without needing to load the entire tokenized corpus into memory, which was a critical detail given the dataset's size.

A computer screen with text and images

Description automatically generated

**Verifying saved file**

We load the sample\_processed\_data.py file and check the first batch. It should have two keys “input\_ids” and “attention\_mask”, and their values should be vectors. The snapshot confirms it’s correct.

**A screenshot of a computer screen

Description automatically generated**

**Challenges Encountered**

* **Memory Overflow on Colab:** The most significant challenge was the sheer size of the raw Wikipedia data. This was mitigated by using Hugging Face's streaming=True feature, which allowed us to process a large dataset iteratively without requiring a massive amount of RAM or disk space. Also, we shrank to 150 documents instead of the full 1GB raw data, so that we could get the code correct.
* **Data shape expected by from\_list() method.** This method expects a list of dictionaries, but when we retrieved ‘text’ key from Wikipedia, it’s a list of strings.

**Reflections on Preprocessing Impact**

The preprocessing pipeline is a foundational step that directly impacts the quality and efficiency of model pre-training. By carefully cleaning and normalizing the data, we provide the model with a consistent, high-quality signal, reducing the noise it must learn to ignore. The chunking and padding strategies, while complex to implement, were essential for making the training process tractable and memory-efficient. Ultimately, a well-preprocessed dataset is the bedrock upon which a robust and performant foundation model is built.