Assignment 1: Data Collection and Preprocessing for Foundation Model Pre-Training

1 Background and Motivation

Pretraining foundation models requires **large-scale, diverse, and high-quality datasets**. Raw text data often contains duplicates, noise, or formatting issues that can negatively impact model learning. Effective preprocessing—including cleaning, normalization, tokenization, and batching—is crucial for training efficient, reliable models.

This assignment provides hands-on experience with collecting large-scale textual datasets and preparing them for transformer-based model pretraining using modern libraries such as Hugging Face Transformers, PyTorch, or TensorFlow.

2 Learning Objectives

By completing this assignment, students will be able to:

1. Identify and access **publicly available large-scale text datasets** suitable for pretraining.
2. Implement **data cleaning and normalization pipelines**, including deduplication and low-quality content removal.
3. Apply **tokenization strategies** suitable for transformer-based foundation models.
4. Develop **custom data loaders** in PyTorch or TensorFlow for efficient batch training.
5. Document **challenges, trade-offs, and decisions** during preprocessing for large-scale datasets.
6. Demonstrate understanding of **data quality, diversity, and preprocessing impact** on model performance.

3 Dataset Requirements

- Total size: **at least 1GB** of raw text.
- Sources: public datasets such as Wikipedia dumps, Common Crawl subsets, OpenWebText.
- Diversity: include multiple domains (encyclopedic, news, general web text).
- Cleanliness: raw datasets may contain noise; cleaning steps are required.

4 Preprocessing Requirements

The preprocessing pipeline must include:

4.1 Cleaning

- Remove duplicate documents.
- Normalize text: lowercase, remove extra whitespace, strip irrelevant symbols.
- Remove low-quality or very short documents (e.g., fewer than 50 words).
- Optionally remove HTML tags, markdown, or reference markers.

4.2 Tokenization

- Use a transformer-compatible tokenizer (Hugging Face AutoTokenizer or similar).
- Support BPE, WordPiece, or GPT-style tokenization.
- Handle sequences longer than the model's maximum block size via chunking.
- Maintain tokenized sequences in an efficient data structure (list of lists, arrays, or tensors).

4.3 Custom Data Loader

- Implement a custom data loader in PyTorch or TensorFlow:
  - PyTorch: `torch.utils.data.Dataset` and `DataLoader`
  - TensorFlow: `tf.data.Dataset` pipeline
- Support batching and shuffling.
- Handle variable-length sequences with padding or truncation if needed.
- Enable iterable streaming for large datasets to avoid memory bottlenecks.

5 Submission Requirements

5.1 Code

- Well-commented scripts or notebooks performing:
  1. Dataset download/collection
  2. Cleaning and normalization
  3. Tokenization
  4. Custom data loader implementation
- Code should run without modification on a standard Python 3 environment.

5.2 Sample Processed Data

- Provide sample batches of processed/tokenized data (e.g., first 5–10 blocks).
- Save as `.pt` (PyTorch) or `.tf` (TensorFlow) files.

5.3 Report

Include a 2–4 page PDF documenting:

- Dataset sources and total size
- Cleaning strategies and reasoning
- Tokenization choices (e.g., tokenizer type, vocabulary size, block size)
- Data loader implementation details
- Challenges encountered (memory, deduplication, noise, batching)
- Reflections on preprocessing impact on model pretraining quality

6 Deliverables

1. Python scripts/notebook: `data_collection_preprocessing.py` or `.ipynb`
2. Sample tokenized data file: `sample_dataset.pt` or `sample_dataset.tf`
3. Report (PDF): `Assignment1_Report.pdf`

7 Evaluation Criteria

Criterion	**Weight**	**Description**
Dataset Quality	20%	Diversity, size $\geq$1GB, low noise, deduplicated
Preprocessing Pipeline	25%	Cleaning, normalization, handling long sequences, tokenization
Custom Data Loader	20%	Functionality, batching, memory efficiency
Code Quality	15%	Readability, documentation, reproducibility
Sample Data	10%	Correct tokenization, batch structure, completeness
Report	10%	Clarity, completeness, discussion of challenges and decisions

8 Optional Extensions

- Integrate streaming datasets to handle very large corpora.
- Perform data quality analysis, e.g., token length distributions or domain coverage.
- Implement mixed-language preprocessing if multilingual datasets are used.

9 Tools and Libraries

- Python 3.x
- Hugging Face `datasets` and `transformers`
- PyTorch or TensorFlow
- Optional: `tqdm`, `regex`, `numpy`, `pandas`