Assignment 1 Report

Data Collection and Preprocessing for Foundation Model Pre-Training

1. Dataset Sources and Total Size

For this assignment, I collected publicly available large-scale text datasets from multiple domains to ensure diversity:

- **Wikipedia Dump (2023-09 snapshot)**: Encyclopedic, factual writing.
- **Common Crawl Subset (C4 filtered)**: Broad general web text.
- **OpenWebText**: Curated Reddit-linked articles to capture conversational/news style text.

After combining and cleaning, the raw dataset size exceeded **1.2 GB of plain text**, meeting the ≥1 GB requirement.

2. Cleaning Strategies and Reasoning

The preprocessing pipeline focused on improving dataset quality while preserving semantic richness.

- **Deduplication**: Used hashing (MD5) at the document level to remove exact duplicates.
- **Normalization**:
  - Converted all text to lowercase.
  - Removed excessive whitespace and non-textual symbols.
- **Low-Quality Filtering**:
  - Discarded documents with fewer than **50 words**.
  - Filtered out boilerplate HTML tags, markdown syntax, and reference markers.
- **Noise Handling**: Regex-based removal of scripts, tables, or malformed encodings.

These steps improved data consistency and reduced redundancy, leading to a leaner and more meaningful dataset.

3. Tokenization Choices

For tokenization, I used **Hugging Face AutoTokenizer** with a GPT-2 style Byte Pair Encoding (BPE):

- **Tokenizer Type**: GPT2TokenizerFast (efficient and widely compatible).
- **Vocabulary Size**: ~50,000 tokens (default GPT-2 vocab).
- **Block Size**: 1,024 tokens.
- **Long Sequences**: Longer texts were chunked into fixed-size blocks with overlap handling.

The resulting tokenized data was stored in memory-efficient **PyTorch tensors** for downstream model training.

4. Data Loader Implementation

The custom data loader was implemented in **PyTorch** using `torch.utils.data.Dataset` and `DataLoader`.

Key features:

- **Batching & Shuffling**: Enabled randomized access during training.
- **Variable Length Handling**: Applied padding/truncation for uniform sequence lengths.
- **Iterable Streaming**: Implemented lazy loading for large files to avoid memory overload.
- **Scalability**: Loader design supports GPU-based training pipelines.

Sample batches of tokenized data were generated and saved as `.pt` files.

5. Challenges Encountered

Several issues arose during preprocessing:

1. **Memory Bottlenecks**: Initial attempts to load entire Common Crawl data into memory failed; resolved with streaming.
2. **Deduplication Trade-offs**: Hash-based deduplication removed exact duplicates but not near-duplicates (paraphrased or truncated content).
3. **Token Length Distribution**: Found a heavy-tailed distribution, requiring careful block-size selection to avoid wasting padding.
4. **Cleaning Heuristics**: Aggressive cleaning risked discarding valuable data; tuned regex filters to strike a balance.

6. Reflections on Preprocessing Impact

Effective preprocessing significantly influences model quality:

- **Deduplication** reduced memorization risk and improved dataset variety.
- **Normalization** ensured consistent input representation, aiding convergence.
- **Tokenization strategy** impacted efficiency—BPE balanced vocabulary compactness and flexibility.
- **Loader design** determined training scalability; streaming enabled handling >1GB datasets smoothly.

Overall, the preprocessing pipeline ensured **high-quality, diverse, and model-compatible training data**.