Data Collection and Preprocessing for Foundation Model Pre-Training

1. Dataset Sources and Total Size

To meet the assignment requirement of at least 1GB of raw text, I collected data from three diverse public sources using the Hugging Face *datasets* library. The selection prioritizes domain diversity, covering encyclopedic knowledge, general web content, and news journalism.

Data Sources

- **WikiText-103** (wikitext-103-raw-v1): A large-scale collection of Good and Featured articles from Wikipedia, providing encyclopedic domain coverage.
- **OpenWebText:** An open-source recreation of the WebText dataset, representing diverse general web content including blogs, forums, and articles.
- **CNN/DailyMail:** A collection of full-length news articles from CNN and the Daily Mail, adding a journalistic domain to the corpus. This replaced our initial choice of AG News, which contained mostly short headlines that were eliminated during preprocessing (see Section 5.4).

Dataset Statistics

The total raw text collected exceeded the 1GB requirement:

Dataset	Raw Size (MB)	Document Count
WikiText-103	235.75 MB	500,000
OpenWebText	718.47 MB	150,000
CNN/DailyMail	584.16 MB	150,000
Total	**1,538.38 MB (1.502 GB)**	**800,000**

2. Cleaning Strategies and Reasoning

Raw text data often contains noise, duplicates, and formatting issues that can negatively impact model learning. I implemented a three-stage preprocessing pipeline to ensure data quality.

2.1 Deduplication

Strategy: Exact deduplication using hash-based matching. Each document is hashed using Python's built-in hash() function, and only the first occurrence of each unique hash is retained.

Result: Removed 47,173 duplicate documents total. WikiText-103 had the most duplicates (44,085 out of 323,818 non-empty documents), which is expected given that Wikipedia articles share boilerplate sections and repeated headers. CNN/DailyMail had 3,088 duplicates, while OpenWebText had none.

Reasoning: Duplicate documents artificially inflate training metrics and can cause the model to memorize specific phrases rather than learning generalized language patterns. Hash-based deduplication is fast and memory-efficient since only hashes are stored in the set, not full document strings.

2.2 Text Normalization

Strategy: A normalization pipeline using Python's *re* library to remove HTML tags and Markdown formatting (links, headers, bold markers), convert text to lowercase, and collapse extra whitespace and irrelevant symbols.

Reasoning: Removing markup artifacts ensures the model learns natural language rather than formatting syntax. Lowercasing standardizes the input and reduces the effective vocabulary size, helping the model generalize better across different capitalizations of the same word.

2.3 Filtering Low-Quality Documents

Strategy: Filtering out documents containing fewer than 50 words after normalization.

Result: Removed 95,599 short documents (16.6% of the deduplicated corpus). The vast majority came from WikiText-103 (95,568 removed), which contains many short section headings and stubs. OpenWebText lost only 6 documents, and CNN/DailyMail lost only 25, confirming that both sources consist of full-length articles.

Reasoning: Extremely short texts often lack sufficient context for learning long-range dependencies. Removing them improves the average information density of each training batch.

Pipeline Summary:

Stage	Input	Removed	Output
Deduplication	623,818	-47,173	576,645
Normalization + Filtering	576,645	-95,599	481,046

3. Tokenization Choices

I utilized the GPT-2 tokenizer via Hugging Face's AutoTokenizer.

- **Tokenizer Type: Byte-Pair Encoding (BPE).** BPE was chosen because it effectively handles out-of-vocabulary words through subword units, balancing vocabulary size and representation capability. The GPT-2 tokenizer has a vocabulary of 50,257 tokens.
- **Block Size: 512 tokens.** This balances context length with memory efficiency. A 512-token window provides sufficient context for language modeling while fitting within typical GPU memory constraints, compared to smaller windows like 128 or 256 tokens.
- **Chunking Strategy:** A custom tokenize_and_chunk function splits each document into non-overlapping 512-token blocks. Only complete blocks are retained; partial chunks at the end of documents are discarded to maintain uniform block sizes and eliminate the need for padding tokens.

Tokenization Results by Source

With CNN/DailyMail replacing AG News, the final tokenized block distribution shows meaningful contribution from both the web and news domains:

Source	Documents	Tokenized Blocks	Share
WikiText-103	184,165	395	0.09%
OpenWebText	149,994	234,187	56.3%
CNN/DailyMail	146,887	181,676	43.6%
Total	**481,046**	**416,258**	**100%**

CNN/DailyMail articles are long enough to produce multiple 512-token blocks per document, contributing 43.6% of the final training data. This is a major improvement over the original AG News approach, which yielded zero tokenized blocks. WikiText-103 remains underrepresented (395 blocks) because most Wikipedia entries, even after passing the 50-word filter, are still shorter than 512 tokens.

4. Custom Data Loader Implementation

I implemented a custom data loader using PyTorch (torch.utils.data).

- **Dataset Class:** The TextDataset class flattens the dictionary of tokenized blocks from all sources into a single list of 416,258 blocks for seamless indexing. It returns both input_ids and labels in each sample. For causal language modeling, labels are set equal to input_ids, enabling the model to learn next-token prediction in a self-supervised manner.
- **DataLoader:** Configured with batch_size=8, shuffle=True, and num_workers=2 to ensure efficient, parallel data loading during training. Shuffling ensures each batch contains a mix of domains rather than sequential blocks from the same source. This produces 52,033 total batches per epoch.
- **Verification:** A test batch was loaded to confirm correct tensor shapes (batch_size × block_size = 8 × 512). Ten sample batches were saved as sample_dataset.pt for the assignment deliverable.

5. Challenges Encountered

5.1 Dataset Size Estimation

Challenge: Initial sample sizes were underestimated, resulting in a total raw size below 1GB. **Solution:** Iteratively adjusted sample sizes for each source and recalculated the total size until exceeding the target. The final configuration (500k WikiText, 150k OpenWebText, 150k CNN/DailyMail) yields 1.502 GB of raw text.

5.2 Empty Documents in WikiText

Challenge: WikiText-103 contained approximately 35% empty or whitespace-only rows. Despite requesting 500,000 samples, only 323,818 non-empty documents were collected. **Solution:** Filtered empty documents during the collection phase and documented the natural data loss. Increasing the sample count to 500k helped compensate for this attrition.

5.3 Processing Time

Challenge: Tokenizing 150,000 OpenWebText documents took approximately 13 minutes, and 147,000 CNN/DailyMail articles took approximately 10 minutes on CPU. **Solution:** Implemented per-source checkpointing by saving .pkl files after each source's tokenization. This prevents data loss in case of Colab runtime disconnections and avoids re-running already completed tokenization steps.

5.4 AG News Replacement

Challenge: The original news domain source, AG News, contained primarily short headlines and summaries. Over 90% of its documents were removed by the 50-word filter, and the remaining documents were too short to form complete 512-token blocks, resulting in zero tokenized blocks from the entire dataset. **Solution:** Replaced AG News with CNN/DailyMail, which contains full-length news articles averaging 500–800 words. This swap yielded 181,676 tokenized blocks (43.6% of the total), resolving the domain coverage gap while requiring minimal code changes.

5.5 Memory Management

Challenge: Loading over 1.5GB of raw text data and processing it simultaneously creates significant memory pressure in Google Colab. **Solution:** Utilized streaming mode (streaming=True) when loading all three datasets. This allows the script to iterate through samples one at a time without loading the entire dataset into RAM, preventing environment crashes.

6. Reflections on Preprocessing Impact

The most important takeaway from this assignment is that every preprocessing decision compounds. A block size choice, a minimum word threshold, and a tokenizer's subword granularity each seem reasonable in isolation, but together they shape what the model actually sees during training. In the final dataset, WikiText-103 contributes only 395 blocks despite starting with 500,000 raw samples, because its documents are individually shorter than the 512-token block size. The data that survives preprocessing is not necessarily representative of the data that went in.

This has a direct implication for model quality. A model trained on this corpus will be disproportionately exposed to web-style prose and news articles, with very little encyclopedic content. If downstream tasks require factual or encyclopedic knowledge, the model would likely underperform compared to one trained on a more balanced distribution. Preprocessing is not just a technical step before training—it is effectively a decision about what the model will and will not learn.

Looking forward, techniques like document concatenation (merging short texts into full-length blocks), domain-aware sampling (adjusting representation at the batch level rather than the document level), and variable block sizes could help preserve domain diversity through the pipeline. The broader lesson is that preprocessing pipelines need to be evaluated not just on cleanliness metrics, but on the distribution of the final training data they produce.