

# Data Collection and Preprocessing Report

## Foundation Model Pretraining Pipeline

This document describes the dataset sources, cleaning strategies, tokenization choices, data loader implementation, challenges, and reflections on preprocessing impact for the pretraining data pipeline implemented in this project.

---

### 1. Dataset Sources and Total Size

#### Sources

The pipeline is designed to support **multiple configurable data sources** via Hugging Face Datasets. The current configuration uses:

Source Identifier	Text Field	Notes
CC News	cc_news	text Common Crawl news subset; primary source.

The codebase also supports **streaming** for large corpora. Data sources are defined in `config.py` under `Config.DATA_SOURCES`; each entry specifies `name`, `text_field`, and optional `sample_frac` for downsampling.

#### Collection Modes

- **Non-streaming (default):** Full dataset is loaded into memory with `load_dataset(..., streaming=False)`. Text columns are extracted, optionally renamed to "text", and concatenated across sources into a single DataFrame.
- **Streaming:** When `USE_STREAMING=True`, data is consumed with `streaming=True` until total byte size reaches `MIN_DATA_SIZE_GB` (default **1.0 GB**), reducing memory use for very large corpora.

#### Size Requirements and Output

- **Minimum size:** Raw data must reach at least **1.0 GB** (configurable via `MIN_DATA_SIZE_GB`) after collection; otherwise the pipeline raises an error. Size is checked by writing a temporary CSV and measuring file size.
  - **Deliverables:** Raw data is persisted as `raw_data.csv`. After cleaning and tokenization, the pipeline produces **860,891 tokenized chunks** (from the included quality report), implying a substantial cleaned corpus (e.g. on the order of hundreds of thousands of documents after deduplication and quality filtering).
- 

### 2. Cleaning Strategies and Reasoning

Cleaning is implemented in `data_cleaner.py` and applied in a fixed order: deduplication → normalization → HTML removal → quality filtering.

## 2.1 Deduplication

- **Method:** `drop_duplicates(subset="text", keep="first")` on the raw `text` column.
- **Reasoning:** Exact duplicate documents add no information for pretraining and can bias the model and waste compute. Keeping the first occurrence preserves document order and is deterministic.

## 2.2 Text Normalization

- **Steps:**
  - Cast to string and **lowercase**.
  - Collapse runs of whitespace to a single space: `re.sub(r"\s+", " ", text)`.
  - Remove characters that are not word characters, spaces, or the punctuation . , ! ?: `re.sub(r"[^\w\s\.\,\!\?]", "", text)`.
  - Strip leading/trailing whitespace.
- **Reasoning:** Lowercasing and normalizing whitespace reduce vocabulary surface forms and improve consistency. Retaining basic punctuation preserves sentence structure for language modeling while stripping noisy symbols (e.g. HTML remnants, special Unicode).

## 2.3 HTML Tag Removal

- **Method:** `re.sub(r"<.*?>", "", text)` on the normalized text.
- **Reasoning:** CC News and similar web-sourced data can contain HTML tags; removing them avoids teaching the model markup as content and reduces tokenizer noise.

## 2.4 Low-Quality Filtering

- **Criterion:** Documents with fewer than **50 words** (after cleaning) are dropped (`MIN_WORDS_PER_DOC=50` in config).
- **Reasoning:** Very short documents (headlines, fragments, boilerplate) provide little context for pretraining and can dominate the distribution if not filtered. A 50-word minimum keeps meaningful paragraphs and articles while removing debris.

The cleaned output is saved as `cleaned_data.csv` with a single `text` column (renamed from the internal `cleaned_text`).

---

## 3. Tokenization Choices

### 3.1 Tokenizer Type and Vocabulary

- **Primary:** GPT-2 BPE tokenizer (`gpt-2` via `AutoTokenizer.from_pretrained("gpt2")`). Vocabulary size is **50,257** (standard GPT-2).
- **Optional:** For multilingual data, the pipeline can use **BERT-base-multilingual-uncased** (`bert-base-multilingual-uncased`), selected when `ENABLE_MULTILINGUAL=True`.

**Reasoning:** GPT-2’s BPE tokenizer is widely used for causal language modeling and aligns with many open pretrained LMs. The multilingual option supports mixed-language corpora without changing the rest of the pipeline.

## 3.2 Padding Token

- GPT-2 has no dedicated pad token; the code sets `tokenizer.pad_token = tokenizer.eos_token` so that batching and attention masking can use a consistent pad ID (e.g. 50256 for GPT-2).

## 3.3 Block Size and Chunking

- **Max sequence length (block size):** 512 tokens (`MAX_SEQUENCE_LENGTH=512`). This matches common Transformer limits and balances context length with memory and compute.
- **Chunking:** Long documents are split into non-overlapping blocks of 512 tokens. No truncation is applied at tokenizer level (`truncation=False, padding=False`); chunking is done after tokenization in `_tokenize_text()`.
- **Minimum chunk length:** Chunks shorter than 10 tokens (`MIN_CHUNK_LENGTH=10`) are discarded to avoid tiny fragments that add little signal and can skew length statistics.

## 3.4 Tokenization Flow

- Each document is tokenized to `input_ids`; the 1D tensor is sliced into consecutive segments of length `max_block_size`. Only segments of length  $\geq \text{min\_chunk\_length}$  are kept. All chunks are collected into a single list and saved as `tokenized_data.pt` (list of tensors).
- 

# 4. Data Loader Implementation Details

## 4.1 Dataset Class: `PretrainDataset`

- **Input:** A list of tokenized chunks (1D tensors of token IDs) and config-derived `max_block_size` (512) and `pad_token_id`.
- **`__getitem__(idx)`:**
  - If chunk length  $< 512$ : **right-pad** with `pad_token_id` to 512.
  - If chunk length  $> 512$ : **truncate** to the first 512 tokens.
- **Output:** Each sample is a dict `{"input_ids": tensor}` of shape `(512,)`. No separate `attention_mask` is returned; a training script can derive it from `input_ids != pad_token_id` if needed.

## 4.2 DataLoader Configuration

- **Batch size: 16** (`BATCH_SIZE=16`).
- **Shuffle:** `shuffle=True` by default in `create_dataloader()` for training.
- **Workers: 4** (`NUM_WORKERS=4`) for parallel data loading.
- **Pin memory:** `pin_memory=True` to speed up CPU→GPU transfer when using CUDA.

No custom `collate_fn` is used; batching relies on default stacking of the `{"input_ids": chunk}` dicts, which works because every chunk is normalized to length 512 in `__getitem__`.

## 4.3 Sample Data

The pipeline saves the first **8** batches (`SAMPLE_BATCHES=8`) from this `DataLoader` as `sample_dataset.pt` for inspection and submission. Each batch has `input_ids` of shape `(16, 512)`.

---

# 5. Challenges Encountered

## 5.1 Memory

- **Issue:** Loading full CC News (or multiple large datasets) into memory can exceed RAM.
- **Mitigation:** Streaming mode (`USE_STREAMING=True`) fetches records one-by-one until the target byte size (e.g. 1 GB) is reached, avoiding full load. For non-streaming, using a single source or `sample_frac` in config limits size. Tokenization is done document-by-document and chunks are accumulated in a list, so peak memory is dominated by the cleaned DataFrame and the list of chunk tensors.

## 5.2 Deduplication

- **Issue:** Web and news data often contain many near-duplicates or exact duplicates.
- **Approach:** Exact deduplication on the raw `text` field is simple and effective; the pipeline reports how many duplicates were removed. Fuzzy or semantic deduplication is not implemented but could be added for stronger dedup at higher cost.

## 5.3 Noise

- **Issue:** Raw text contains casing variation, extra symbols, and HTML.
- **Approach:** Normalization (lowercase, whitespace, allowed punctuation only) and HTML stripping reduce noise. The 50-word minimum removes very short, often noisy snippets. The pipeline does not perform language detection or profanity filtering; those could be added for domain-specific needs.

## 5.4 Batching

- **Issue:** Chunks have variable length before padding; batching requires fixed-length sequences for the model.
- **Approach:** All sequences are normalized to exactly 512 tokens in `PretrainDataset.__getitem__` via padding or truncation, so the `DataLoader` can batch with the default collate. No dynamic padding or packing is used, which keeps the implementation simple and ensures uniform batch shape `(B, 512)`.

## 5.5 Tokenization Throughput

- Tokenization runs in a single process over a list of texts; for very large corpora this can be slow. The code includes a `batch_tokenize()` helper that still processes texts in batches but tokenizes each text individually within the batch (no batched tokenizer API). Using the tokenizer's native batch API (e.g. `tokenizer(texts,`

---

`padding="max_length", truncation=True, max_length=512)`) could improve throughput where full-sequence truncation is acceptable.

---

## 6. Reflections on Preprocessing Impact on Model Pretraining Quality

### 6.1 Data Diversity and Domain

- The pipeline is built to combine multiple sources (e.g. CC News, Wikipedia) to increase domain diversity. In the current config, a single source (CC News) still provides news-style text; adding more sources (as in the guide) would improve topical and stylistic variety and likely improve generalization.

### 6.2 Cleaning and Noise

- Deduplication and length filtering directly improve the effective information per token and reduce overfitting to repeated or trivial content. Aggressive normalization (e.g. stripping all but a few punctuation marks) can remove useful signal (e.g. quotes, structure) and may not be ideal for tasks that rely on exact formatting; for generic causal LM pretraining it is a reasonable trade-off.

### 6.3 Tokenization and Block Size

- GPT-2 BPE yields a fixed vocabulary and subword granularity that matches many existing models. Block size 512 limits long-range dependencies to 512 tokens; documents are split without overlap, so cross-block context is only learned implicitly through repeated exposure across batches. Minimum chunk length 10 avoids training on very short fragments that could dilute learning.

### 6.4 Data Loader and Training

- Fixed-length 512-token sequences and shuffling give stable, uniform batches that are easy to train on. The lack of explicit `attention_mask` in the dataset is acceptable if the training loop infers padding from `pad_token_id`; adding an explicit mask would improve clarity and correctness for loss masking. Quality analysis (chunk count, length distribution, percentiles) helps verify that the tokenized corpus has sensible statistics (e.g. 860,891 chunks with median length 399) before pretraining.

### 6.5 Summary

- Preprocessing choices in this pipeline prioritize reproducibility, simplicity, and compatibility with standard GPT-2-style pretraining. The main levers for improving pretraining quality from here are: expanding data sources and diversity, optionally refining cleaning (e.g. less aggressive normalization or language filtering), and ensuring padding is correctly masked in the loss during training.
-