Assignment_01

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

Student Name: [Chen Ting Kuo]

Date: February 4, 2026

Course: [CSYE7374]

---

1. Dataset Sources and Total Size

To ensure domain diversity for foundation model pre-training, we collected data from three distinct high-quality sources: **News (Current Events), Encyclopedic (Factual Knowledge), and Web Text (General Internet Usage)**. The data collection was implemented using the Hugging Face `datasets` library with streaming enabled to handle memory constraints efficiently.

Data Composition

Domain	Dataset Source	Specific Version / Subset	Samples Collected
News	`cc_news`	Original (Jan 2017 – Dec 2019)	100,000 documents
Encyclopedic	`wikimedia/wikipedia`	20231101.en	100,000 documents
General Web	`Skylion007/openwebtext`	Reddit-sourced WebText	100,000 documents

- **Total Raw Data Volume: 1.13 GB** (approx. 300,000 documents)
- **Format:** The raw data was aggregated and streamed into a unified `raw_dataset.jsonl` file to ensure persistence and minimize RAM usage during collection.

---

2. Cleaning Strategies and Reasoning

Raw web data is inherently noisy and redundant. We implemented a strict preprocessing pipeline focusing on deduplication and quality filtering.

2.1 Deduplication (MD5 Hashing)

Since the `cc_news` dataset contains raw crawls, duplicate articles are common.

- **Strategy:** We utilized **MD5 hashing** to generate a 32-character "fingerprint" for each normalized document.
- **Implementation:** We maintained a `seen_hashes` set in memory. If a document's hash already existed in the set, it was discarded.
- **Reasoning:** Hashing reduces memory footprint significantly and allows for $O(1)$ lookups, enabling fast exact deduplication.

2.2 Normalization and Filtering

- **Normalization:** All text was converted to lowercase, and excessive whitespace (tabs, newlines, multiple spaces) was collapsed into single spaces. This reduces the vocabulary size required and standardizes the input.

- **Length Filtering:** We removed documents with fewer than **50 words**. Short texts often represent navigation menus, error messages, or low-quality snippets that do not provide sufficient context for language modeling.

2.3 Cleaning Results

- **Original Documents:** 300,000
- **Duplicates Removed:** 16,358
- **Short Docs Removed:** 9,921
- **Final Cleaned Volume: 1.10 GB** (273,721 documents)

```
Starting preprocessing pipeline on /Users/zhenting/7374_LLM/raw_dataset.jsonl...
Cleaning: 100%|██████████████████████████████| 300000/300000 [00:36<00:00, 8127.49it/s]

=== Preprocessing Report ===
Original Docs: 300000
Duplicates Removed: 16358
Short Docs Removed (<50 words): 9921
Final Docs Kept: 273721
Final Dataset Size: 1.10 GB
Cleaned data saved to: /Users/zhenting/7374_LLM/clean_dataset.jsonl
```

---

3. Tokenization Choices

We selected a tokenizer compatible with the **GPT-2** architecture to support autoregressive language modeling tasks.

3.1 Tokenizer Configuration

- **Model:** `gpt2` (Hugging Face AutoTokenizer).
- **Algorithm: Byte-Level Byte-Pair Encoding (BPE)**. This allows the model to handle out-of-vocabulary words by falling back to byte-level representations, making it robust for diverse web text.
- **Vocabulary Size:** 50,257 tokens.
- **Special Tokens:** Since GPT-2 does not have a native padding token, we mapped `pad_token` to `eos_token` (`<|endoftext|>`) to enable compatibility with batch processing.

3.2 Chunking and Block Size

- **Block Size: 1024 tokens**.
- **Strategy:** We employed a **Concatenation and Slicing** strategy.
    1. All documents were tokenized and appended with an `EOS` token.
    2. The tokens were concatenated into a continuous stream.
    3. The stream was sliced into fixed-size blocks of 1024 tokens.
    4. The final incomplete block was discarded to ensure consistent tensor shapes.
- **Reasoning:** This approach maximizes training efficiency by eliminating padding inside training blocks and allowing the model to learn context across document boundaries.

3.3 Storage (Sharding)

To prevent memory overflows during the conversion of 1GB+ text into integer tensors, we implemented **Sharding**:

- Processed tensors were saved into multiple `.pt` files (`shard_0.pt`, `shard_1.pt`, ...), each containing approximately 50,000 blocks.
- **Final Token Count:** ~266 Million tokens.

```
Loading tokenizer: gpt2...
Tokenizing with gpt2 (Block size: 1024)...
Tokenizing:   0%|                                          | 0/273721 [00:00<?, ?it/s]Token indices s
specified maximum sequence length for this model (1123 > 1024). Running this sequence through the model w
Tokenizing:  29%|██████████                             | 79668/273721 [00:44<13:21, 242.13it/s]
50004
tensor([ 8117,   338,   257,  ..., 16667, 28841,   365])
tensor([[ 8117,   338,   257,  ...,    13,  2008, 12537],
        [  299,    88, 21101,  ...,    85,  4763, 47735],
        [21421,    13,   299,  ...,   717,  5545,  3652],
        ...,
        [40138,   410, 40138,  ..., 45630,   272, 14549],
        [19322,   784, 44873,  ...,  3104,   784,   556],
        [ 2516,   286,   277,  ..., 16667, 28841,   365]])
Saved shard 0 to /Users/zhenting/7374_LLM/tokenized_data/shard_0.pt: shape torch.Size([50004, 1024])
Tokenizing:  45%|███████████████                  | 122192/273721 [01:34<02:39, 949.94it/s]
```

4. Data Loader Implementation Details

We implemented a custom PyTorch `IterableDataset` (`GPTDataset`) to handle the large-scale data efficiently without loading the entire dataset into RAM.

4.1 Streaming Architecture

The data loader does not preload data. Instead, it iterates through the `.pt` shard files one by one. Only one shard (approx. 50MB–100MB) resides in memory at any given time.

4.2 Double Shuffling Strategy

1. **Shard-Level Shuffle:** The order of `.pt` files is randomized at the start of each epoch.
2. **Sample-Level Shuffle:** Once a shard is loaded into memory, its internal indices (rows) are shuffled before yielding.

```
Loading shard: shard_1.pt
Batch 0 shape: torch.Size([8, 1024])
Batch 1 shape: torch.Size([8, 1024])
Batch 2 shape: torch.Size([8, 1024])
Data Loader test passed!
```

---

5. Challenges Encountered

Memory Management (RAM)

- **Issue:** Initially, appending all 300,000 raw text strings to a Python list caused the kernel to crash due to high memory overhead.
- **Solution:** We switched to a **streaming approach**, writing each document directly to a `.jsonl` file immediately after fetching (`stream_to_file`). Similarly, during tokenization, we periodically flushed processed tokens to disk (`shard_X.pt`) to clear the buffer.

---

6. Reflections on Preprocessing Impact

This assignment highlighted the critical trade-offs between **data quality** and **computational cost**.

1. **Impact of Deduplication:** While MD5 deduplication added a processing overhead, removing ~16,000 duplicates ensures the model doesn't memorize repeated text, which promotes better generalization.
2. **Normalization Trade-off:** Aggressive normalization (lowercasing) simplifies the vocabulary but loses semantic nuances (e.g., "Apple" the company vs. "apple" the fruit). For a larger-scale model, a cased tokenizer might be preferable.
3. **Efficiency of Binary Formats:** Storing data as `.pt` tensors rather than raw text significantly accelerated the Data Loader. The CPU no longer needs to tokenize text on-the-fly during training, preventing the data loading step from becoming a bottleneck for the GPU.