Pipeline Overview

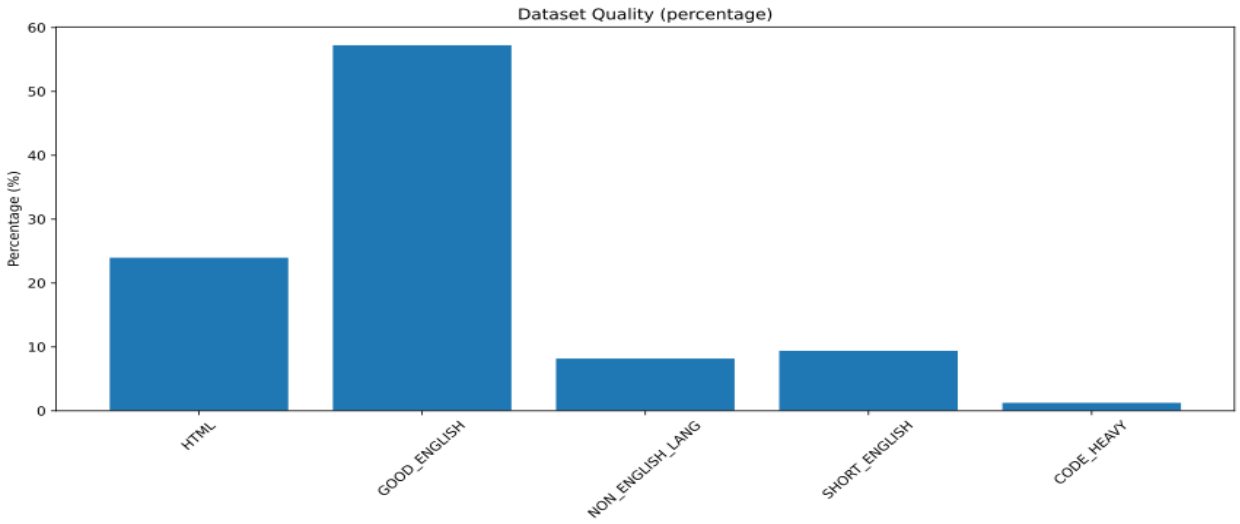
The MainpipeNS pipeline performs an end-to-end transformation of a raw, noisy web-scraped corpus into a clean, English-only, tokenized, and fully sharded training dataset suitable for large-scale LLM pretraining. It includes raw dataset inspection, exact deduplication, HTML and boilerplate removal, language identification, code-heavy filtering, text normalization, GPT-2–style extended tokenization (with BOS/EOS/PAD/UNK), packing into fixed 2048-token blocks, and splitting into train/val/test shards.

It also has additional automated reporting components such as category distributions, cleaning summaries, token-length statistics, and a full quality report (PII, toxicity, perplexity, linguistic coverage), ensuring transparency and reproducibility across every stage of corpus preparation.

Full details of the pipeline (including all modules, processing stages, and usage instructions) are provided in the project README. The complete codebase is available on GitHub at: https://github.com/nsahuastro/MainpipeNS.git.
The repository also includes three interactive Jupyter notebooks that mirror the major stages of the workflow (raw data inspection, cleaning, and tokenization), enabling users to explore the dataset, debug intermediate steps, and experiment with components in an incremental and interpretable manner.

Raw Data Statistics

The raw dataset (≈269k JSONL documents) shows strong internal variability in structure, length, and content quality. Quick-stats reveal a **broad text-length distribution**, with many multi-paragraph documents but also a noticeable fraction of very short entries and HTML-heavy pages.



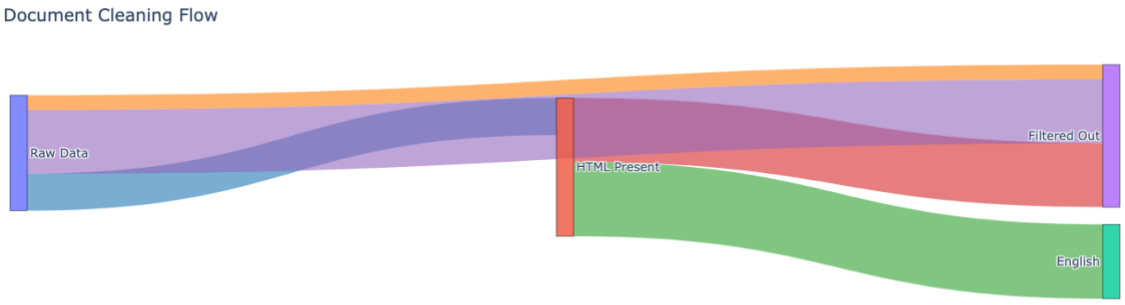
After applying the exclusive document classifier, the histogram shows that **GOOD_ENGLISH** text forms the largest portion of the corpus (~57%), indicating a substantial amount of clean, natural English suitable for LLM training. Meanwhile, **HTML** documents account for ~24%, suggesting significant web-scraped boilerplate and markup contamination. **Non-English** content (~8%) and **Short English** fragments (~10%) reflect typical noise in mixed-quality web sources. **Code-heavy** samples represent only ~1%, suggesting that programmatic content is present but not dominant; however, the current code-detector script is relatively simple and may miss certain code formats or embedded snippets, so future refinement is needed.

This categorization histogram above provides a clear snapshot of the corpus composition and highlights the importance of the cleaning pipeline before tokenization.

Dataset Statistics

After exact **deduplication** and **multi-stage cleaning**, around 39–40% of the corpus is retained for model training.

The full cleaning pipeline removes HTML-heavy entries, non-English texts, code-dominant pages, extremely short/long documents, and noisy or malformed samples. Intermediate reports include document-length histograms, HTML/code distribution breakdowns, and exclusive category percentages for both raw and cleaned data.



The resulting cleaned dataset (~107k documents) represents a well-structured, consistent, and linguistically coherent English corpus suitable for downstream tokenization and packing.

Tokenization, Packing, and Sharding

Following cleaning, each document is tokenized using an extended GPT-2 BPE tokenizer with added special tokens (such as <|bos|>, <|eos|>, <|pad|> etc.). Every cleaned sample is converted

into a sequence of token IDs and wrapped with BOS/EOS markers, with sequences truncated at 2048 tokens when necessary.

To make the dataset training-ready, the pipeline packs variable-length tokenized documents into **fixed 2048-token blocks** by concatenating them sequentially and padding any final space using <|pad|> tokens. This ensures that every block fits the transformer architecture cleanly and consistently.

After packing, the cleaned corpus yields **24,207 fixed-length 2048-token blocks**. These blocks are then sharded into **train/validation/test** splits using a **98/1/1 ratio**, with each shard configured to hold **up to 50,000 blocks**. Because the dataset size is smaller than the shard capacity, this results in **one shard per split** (train/val/test). Although overprovisioned for the current scale, this layout is advantageous as the dataset grows toward millions of blocks. This structure follows common LLM pretraining formats, optimizes data-loader throughput, and ensures fully reproducible subsets for downstream experiments.

Key Quality Metrics (Sample Outputs)

Quality evaluation on a sample of cleaned documents includes lightweight PII detection (emails, phone numbers, credit-card–like patterns), toxicity scoring using Detoxify, approximate perplexity estimation via GPT-2-small, and language distribution checks.

For example, a 1500-document sample produced:

PII hits: 27 emails, 57 phone numbers, 11 credit-card patterns

Toxicity: avg = 0.0085, max = 0.8266

Perplexity: avg = 44.8, median = 34.2

Languages: 100% EN

These metrics confirm that the cleaned dataset is predominantly English, low-toxicity, and free of substantial PII contamination, while providing a transparent snapshot of dataset quality.

Further Improvements

If I had more time to extend this project, there are several areas I would focus on to further improve the quality, robustness, and generalizability of the pipeline:

Improve Inline Code Detection: I would develop more accurate detectors for inline code fragments—potentially using AST-based parsing or lightweight classifier models—to better separate natural-language text from mixed code–text documents.

Segment Very Long Documents: Instead of discarding extremely long samples, I would break them into coherent, semantically consistent segments so that more usable text contributes to pretraining.

Merge Short, Related Documents: For very short documents, I would explore clustering or semantic similarity–based grouping, allowing me to merge related fragments into richer pretraining samples.

Optimize Time Complexity: Many operations could be optimized by reducing repeated computations, caching intermediate results, and parallelizing expensive tasks such as language detection or HTML stripping. I would refactor the pipeline to be more computationally efficient.

Make the Pipeline More Generalizable: I would extend the pipeline to support a wider set of tokenizers and architectures and allow flexible packed block sizes and formats.

Add Advanced Quality and Bias Evaluations: With additional time, I would integrate deeper quality assessments (fluency, coherence) and perform bias/fairness analyses, such as demographic toxicity breakdowns or stereotype benchmarks.