Mid Evaluation Report

Name: Ansh Chablani
Roll No.: 2022111031

Information

Language 1: English
Language 2: Hindi
Language 3: Sanskrit

Approach

Getting the Data

The data collection strategy was designed to meet the project's requirement of a 3 billion token corpus with a specified language distribution.

- **English and Hindi:** Given the availability of large web corpora, the FineWeb dataset was chosen as the primary source. Its focus on quality filtering makes it an excellent starting point for building a high-quality pretraining dataset.
- **Sanskrit:** To acquire a substantial and verified Sanskrit corpus, data was sourced from the **ai4bharat/sangraha** dataset available on the Hugging Face Hub. The verified texts from https://huggingface.co/datasets/ai4bharat/sangraha/tree/main/verified/san provided a diverse collection of sources, satisfying the requirement for the Indian language component of the dataset.

This collection strategy ensures a robust foundation for the model, balancing a large, high-quality web corpus with a specialized, verified corpus for the less-resourced language.

Preprocessing the Data

The raw text data was processed using a comprehensive pipeline (tasks/01_preprocess_data.py) to ensure quality and consistency. The key steps were:

1. **Text Cleaning:** Each line of text from all language files was individually cleaned. This involved:
    - Using the ftfy library to fix potential Unicode errors and inconsistencies (mojibake).
    - Normalizing all whitespace (tabs, newlines, multiple spaces) into a single space to create uniform sentences.
2. **Parallel Processing:** To handle the large volume of data efficiently, the cleaning process was parallelized to utilize all available CPU cores, significantly reducing the processing time.
3. **Global Deduplication:** To prevent the model from overfitting on repeated sentences, a two-step deduplication was performed. First, duplicates were removed within each language file. Then, all unique lines were combined into a single master set, which removed any duplicates that existed *between* the language files.

4. **Data Splitting:** The final, clean, and globally unique dataset was shuffled thoroughly to ensure a random distribution of languages. It was then split into three sets for the model lifecycle:
    - **Training Set:** 98%
    - **Validation Set:** 1%
    - **Test Set:** 1%

This process resulted in clean, non-redundant, and well-structured datasets saved in the `data/processed/` directory, ready for model training.

Tokeniser

A single, unified tokenizer was trained on the combined raw text of all three languages using the script `02_train_tokeniser.py`.

- **Choice of Tool: SentencePiece** was selected due to its language-agnostic nature. It operates directly on Unicode characters, making it ideal for handling datasets with different scripts (Latin for English, Devanagari for Hindi and Sanskrit) without requiring complex, language-specific pre-tokenization rules.
- **Algorithm:** The **Byte-Pair Encoding (BPE)** algorithm was used. BPE is effective at creating a subword vocabulary that can handle complex morphology and out-of-vocabulary words by breaking them down into smaller, known pieces.
- **Training Configuration and Method:** The training process was carefully configured for efficiency and multilingual effectiveness:
    - **Unified Corpus:** All raw text files (`lang_english.txt`, `lang_hindi.txt`, `lang_sanskrit.txt`) were passed as a single input stream. This allows SentencePiece to build a vocabulary that is representative of all three languages.
    - **Vocabulary Size:** A `vocab_size` of **32,000** was chosen as a standard size that provides a good balance between vocabulary richness and model efficiency.
    - **Full Character Coverage:** `character_coverage` was set to **1.0**, ensuring that every unique character from all three languages is included in the initial vocabulary, preventing any character from being immediately marked as unknown.
    - **Memory-Efficient Sampling:** To handle the massive corpus without running out of RAM, the training was configured to use a large, random sample of the data. `input_sentence_size` was set to **10,000,000** with `shuffle_input_sentence` enabled, ensuring the tokenizer learns from a diverse and representative subset of the total ~47 million sentences.
    - **Parallel Processing:** The training was configured to utilize all available CPU cores (`num_threads`), significantly accelerating the BPE merge process.
    - **Special Tokens:** Four special tokens were explicitly defined in the vocabulary: `pad_id=0` (padding), `unk_id=1` (unknown), `bos_id=2` (beginning-of-sentence), and `eos_id=3` (end-of-sentence). These are essential for the subsequent pretraining phase.

The resulting tokenizer files (`multilingual_spm.model` and `multilingual_spm.vocab`) are saved in the `model/tokenizer/` directory.

Evaluating the Tokeniser

The tokenizer was evaluated to ensure it supports all three languages adequately, a critical requirement for a fair multilingual model. The evaluation was two-fold (`evaluate/02_evaluate_tokenizer.py`):

1. **Quantitative Analysis (Token Coverage):** The primary evaluation metric was token coverage, calculated as `100% - (percentage of unknown '<unk>' tokens)`. This was measured **independently for each language** by tokenizing a large sample (500,000 lines) of each raw text file.

   Results:

   - **English Coverage:** `99.9993`%
   - **Hindi Coverage:** `99.9998`%
   - **Sanskrit Coverage:** `100.0`%

   The high and comparable coverage rates across all three languages confirm that the vocabulary is well-balanced and does not favor one language over the others.

2. **Qualitative Analysis (Vocabulary Inspection):** A manual inspection of the `multilingual_spm.vocab` file was performed. Samples from the beginning, middle, and end of the vocabulary list showed a healthy mix of English subwords (e.g., " and", "ation"), Devanagari characters and subwords (e.g., " ार", " कि"), and transliterated tokens. This provides strong anecdotal evidence that the tokenizer successfully learned a shared representation for the multilingual corpus.

Future Timeline and Workflow

The following timeline outlines the plan for completing the remaining project phases leading up to the final deadline of September 15, 2025.

Phase	Week	Primary Tasks	Deliverables
Phase 3: Model Pretraining	**Week 1** (Aug 25 - Aug 31)	1. **Model Architecture:** Implement the autoregressive language model architecture (e.g., a Transformer decoder) in PyTorch, ensuring the parameter count is between 100M-150M. 2. **Data Pipeline:** Develop an efficient data loader to feed the processed text data to the model. 3. **Training Setup:** Set up the training environment on Google Colab or Kaggle. Implement the training loop, optimizer, and loss function.	- Python script for model architecture. - Python script for the data loader.
	Week 2 (Sep 1 - Sep 7)	4. **Initial Training & Debugging:** Start the pretraining process on a small subset of the data to ensure the pipeline is working correctly. 5. **Full-Scale Pretraining:** Launch the full pretraining run on the entire 3-billion-token dataset. Implement robust checkpointing to save progress in batches and manage resource limits. 6. **Monitor Training:** Continuously monitor training logs for metrics like perplexity and loss to ensure the model is learning effectively.	- Pretrained model checkpoints (saved periodically). - Training logs.

Phase	Week	Primary Tasks	Deliverables
Phase 4: Finetuning	**Week 2** (Sep 8 - Sep 10)	1. **Data Preparation:** Write scripts to format the two specified reasoning tasks using chat-style templates. 2. **Finetuning Script:** Adapt the pretraining script for the fine-tuning process, loading the pretrained model weights. 3. **Launch Finetuning:** Run the fine-tuning process for both tasks.	- Scripts for preparing fine-tuning data. - Fine-tuning scripts. - Fine-tuned model checkpoints for both tasks.
Phase 5: Evaluation & Reporting	**Week 3** (Sep 11 - Sep 14)	1. **Evaluation Scripts:** Develop scripts to evaluate the pretrained and fine-tuned models on the test set using metrics like perplexity and task-specific accuracy. 2. **Error Analysis:** Perform a detailed error analysis to identify model strengths and weaknesses. 3. **Final Report:** Write the final report detailing the complete approach, methodology, experiments conducted, and a thorough analysis of the results. 4. **Code & Model Upload:** Clean up all code, add comments, and upload the final scripts, model checkpoints, and tokenizer files to GitHub/Hugging Face Hub.	- Evaluation scripts. - Final PDF report. - Link to the public GitHub / Hugging Face repository.
Submission	**Sep 15**	**Final Submission:** Submit the GitHub Classroom repository link containing all deliverables.	- Completed Project Submission.