MINI PROJECT FINAL REPORT

Phase 1 & 2 : DATA & TOKENIZER

Per-language statistics:

Language	Sentences	Tokens	Share of total	Types(unique ids)
English	**85,560,609**	**1,867,232,562**	**51.58%**	**37,999**
Telugu	**94,877,782**	**1,312,757,994**	**36.26%**	**22,716**
Sindhi	**11,600,789**	**439,998,236**	**12.15%**	**30,864**

Overall statistics:

- **Total sentences : 192,039,180**
- **Total tokens : 3,619,988,792**
- **Total no of tokens : > 3.6 billion**

- The corpus was composed of three languages: English (51% of tokens), Telugu (36%), and Sindhi (12%).
- Preprocessing techniques - cleaning, deduplication, and sentence segmentation were applied to the data.
- The data was stored in memory-mapped (`.mmap`) files for efficient loading during training.

Tokenizer Training

- Byte-Pair Encoding (BPE) tokenizer was trained using the Hugging Face `tokenizer` library.
- Vocabulary size fixed at 64,000 subword units.
- Added special tokens `[PAD]`, `[UNK]`, `<s>`, `</s>` for sequence handling.

Phase 3: MODEL PRETRAINING

- Qwen-3 model was built with a parameter size of approx 110M

- Implemented a Qwen-3 with:
  - **12 Transformer layers**,
  - **8 attention heads**,
  - **Rotary Positional Embeddings (RoPE)** for handling long sequences,

    - **SwiGLU feed-forward networks** for efficient non-linearity,
    - **RMSNorm** for stable training.

- Model Configuration:
    - **`d_model`**: 512
    - **`n_layers`**: 12
    - **`n_heads`**: 8
    - **`d_ff`**: 2048
    - **`vocab_size`**: 64000
    - **`max_seq_len`**: 2048

- The model includes:
    - nn.Embedding layer for token embeddings
    - a series of `TransformerBlock` layers
    - `nn.Linear` layer (`lm_head`) for output logits.

- `TransformerBlock` consists of : `Qwen3Attention` module ,`nSwiGLUFeedForward` module, both followed by `RMSNorm` and dropout layers.

Training

- Training was done by following below methods for better handling large dataset
    - Memory-mapped dataset (`MemoryMappedDataset`): This approach allows the model to access the data directly from disk, reducing memory usage and enabling training on datasets larger than available RAM.
    - Optimizer: **AdamW** with cosine learning rate schedule and warmup.
    - Gradient clipping and accumulation were applied to stabilize training.
    - **Automatic Mixed Precision (AMP)** was used to reduce GPU memory usage and speed up training.
    - Checkpointing: Checkpoints were saved at regular intervals (`eval_every` = 1000 steps) to save progress and only last 10 checkpoints were stored everytime due to space issues(so when 11th checkpoint comes 1st checkpoint will be removed)

Phase 4 : Finetuning

Finetuning was performed to adapt the model to two specific reasoning tasks:

1. Sentiment Analysis – identifying whether a statement is positive, negative, or neutral.
2. Clause Prediction – extracting the main clause from sentences with multiple clauses.

Method

- Used **LoRA (Low-Rank Adaptation)** adapters for (PEFT) parameter-efficient finetuning.

- This allowed training on limited resources without updating the entire model, while still achieving strong task specialization.

- Finetuning data was prepared in **chat-style JSON format,** with separate files for:

    - English sentiment analysis,
    - Telugu sentiment analysis,
    - English clause prediction,
    - Telugu clause prediction.

- From each dataset, 500 samples were reserved for testing, while the rest were used for training.

LoRA Configuration:

- `lora_rank`: 8
- `lora_alpha`: 16
- `lora_dropout`: 0.05
- `target_modules`: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `down_proj`, `up_proj`
- The fine-tuning data was prepared in a chat-style template. A new
- `FinetuneDataset` class was implemented to handle the JSON data, and a `collate_fn` was created to correctly pad the sequences and generate labels for causal language modeling. The fine-tuning was run for 5 epochs.

4. Results

- **Pretraining:**

Loss	**4.7642**
Accuracy	**0.2600**
Perplexity	**117.24**

Analysis :

- The model's high perplexity shows that it has picked up the basics of language structure but still struggles with making confident predictions.

- An accuracy of 26% means the model is starting to recognize meaningful patterns, but it's not strong enough for practical use yet.

- This is expected — pretraining is about giving the model a broad understanding of language, not about excelling at specific tasks. The real improvements come after fine-tuning, where the model learns to apply this foundation to tasks like sentiment analysis and clause prediction.

- **Fine-tuning Performance:**

Loss	**0.8139**
Perplexity	**2.26**

Analysis :

- During finetuning, both training loss and perplexity dropped steadily across epochs, showing that the model was successfully adapting to the tasks.
- On the test set, the finetuned model achieved significantly lower loss and perplexity compared to pretraining
- Significant reduction in perplexity (down to ~2–3 from 117).
- Improved validation accuracy and reasoning quality in both sentiment classification and clause identification tasks.

- **Key Observations:**

  - LoRA fine-tuning was much efficient

  - Clause prediction was more challenging than sentiment analysis due to the syntactic nature of the task.

  - Errors mostly occurred in complex sentences with nested clauses and ambiguous sentiments giving neutral whenever it couldnt decide the sentiment.

Other related files in drive :

https://drive.google.com/drive/folders/1IvX67T7TrO8GsvrCeHfNDJOVgjpg2_84?usp=sharing