2. **Byte-Pair Encoding (BPE) Tokenizer**

    1. **Understanding Unicode**

        a. `chr(0)` returns '`b'\x00`"

        b. `__repr__()` shows an escaped, unambiguous form whereas its printed representation outputs the character itself which is actual NULL character(invisible).

        c. When we add it to text, it simply adds an invisible character at that position which may not be visually noticeable.

    2. **Unicode Encodings**

        a. UTF-8 is preferred over UTF-16 or UTF-32 due to its superior compression, universal applicability without out of vocabulary errors, and native compability with byte-stream data.

        b. If we give the input 牛, it breaks and gives an error "UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe7 in position 0: unexpected end of data". This is due to the fact that the decoding is done byte by byte and when the encoded character is multi-byte, it fails to decode it properly.

        c. `b'\xC3\x28`' is invalid in UTF-8 because 0xC3 indicates the start of a 2-byte character, but 0x28 is not a valid continuation byte.

    3. **BPE Training on TinyStories**

        a. It seems that the longest token is 'accomplishment' with a length of 15 bytes. The memory usage is around 158.3 MB and the training took approximately 61.72 seconds. It makes sense that longer tokens are formed from frequently occurring sequences of characters in the dataset, and 'accomplishment' might be a common word in the TinyStories dataset.

        b. Pre-tokenization part took the most time (40 seconds). Finding chunks and their word count seems to be the most time-consuming part of the process. Second highest is merging the tokens based on the pairs found (20 seconds).

    4. **Experiments with tokenizers**

        a. The tokenizer's compression ratio (bytes/token) is around 2.8609 in a sample of 10 documents from TinyStories dataset.

        b. The througput of the tokenizer is approximately 3000 bytes/second. To tokenize the Pile dataset (about 825 GB), it would take around 76,388 hours.

        c. uint16 is appropriate choice for encoding the token IDs since the vocabulary size is 10,000 which fits well within the range of uint16 (0 to $2^{16} - 1 = 65535$).

3. **Transformer Language Model Architecture**

    1. Transformer LM resource accounting

        a. With the current implementation, GPT-2 XL has nearly 2,127,057,600 (≈2.13B) trainable parameters. At 4 bytes (32 bit floating point) per parameter, this amounts to approximately 8.5 GB of memory just for model parameters.

        b. Calculation is as follows:

        - **Per Transformer Block (L = 48 in total)**

- **Attention Projections (W_Q, W_K, W_V)**: 3 matrix multiplications,

$$= 3 * 2 * seq\_len * d\_model * d\_model$$
$$= 3 * (2 * 1024 * 1600 * 1600)$$
$$= \textbf{15,728,640,000 FLOPs}$$

- **Attention Scores (QK^T)**: 1 matrix multiplication,

$$= 2 * seq\_len^2 * num\_heads * (d\_model/num\_heads)$$
$$= 2 * 1024^2 * 25 * (1600/25)$$
$$= \textbf{3,355,443,200 FLOPs}$$

- **Attention Output($ScoresV$)**: 1 matrix multiplication,

$$= 2 * seq\_len^2 * num\_heads * (d\_model/num\_heads)$$
$$= 2 * 1024^2 * 25 * (1600/25)$$
$$= \textbf{3,355,443,200 FLOPs}$$

- **Attention Output Projection(W_O)**: 1 matrix multiplication,

$$= 2 * seq\_len * d\_model^2$$
$$= 2 * 1024 * 1600^2$$
$$= \textbf{5,242,880,000 FLOPs}$$

- **FFN(W_1, W_3)**: 2 matrix multiplications,

$$= 2 * (2 * seq\_len * d\_model * d\_ff)$$
$$= 2 * 2 * 1024 * 1600 * 6400$$
$$= \textbf{41,943,040,000 FLOPs}$$

- **FFN(W_2)**: 1 matrix multiplication,

$$= 2 * seq\_len * d\_ff * d\_model$$
$$= 2 * 1024 * 6400 * 1600$$
$$= \textbf{20,971,520,000 FLOPs}$$

So for all blocks combined, the total FLOPs is:

$$= 48 * (15,728,640,000 + 3,355,443,200 + 3,355,443,200 + 5,242,880,000$$
$$+ 41,943,040,000 + 20,971,520,000)$$
$$= 48 * 90,596,486,400$$
$$= \textbf{4,348,631,347,200 FLOPs}$$

- **Final LayerNorm and Output Projection**: 1 matrix multiplication,

$$= 2 * seq\_len * d\_model * vocab\_size$$
$$= 2 * 1024 * 1600 * 50257$$
$$= \textbf{164,682,137,600 FLOPs}$$

So the total FLOPs are:

$$= 4,348,631,347,200 \text{ (Blocks)} + 164,682,137,600 \text{ (Head)}$$
$$= \textbf{4,513,313,484,800 FLOPs}$$
$$\boxed{\approx \textbf{4.51 TFLOPs}}$$

c. Most of the FLOPs occur within the **Transformer blocks** (about 96.3% of total FLOPs) and within each block, the **FFN layers** contribute about 70% of the block's FLOPs (62.9 GFLOPs) compared to the attention mechanism (27.7 GFLOPs).

d. The breakdown of the FLOPs for each model is as follows:

- **GPT-2 Small (L = 12, d_model = 768, num_heads = 12)**
  - Blocks: $\mathbf{2.7 * 10^{11}}$ **FLOPs** (77.7% of total)
  - Final LayerNorm and Output Projection: $\mathbf{0.79 * 10^{11}}$ **FLOPs** (22.3% of total)
  - Total: Approximately **0.32 TFLOPs** per forward pass.
- **GPT-2 Medium (L = 24, d_model = 1024, num_heads = 16)**
  - Blocks: $\mathbf{9.27 * 10^{11}}$ **FLOPs** (90% of total)
  - Final LayerNorm and Output Projection: $\mathbf{1.05 * 10^{11}}$ **FLOPs** (10% of total)
  - Total: Approximately **1.03 TFLOPs** per forward pass.
- **GPT-2 Large (L = 36, d_model = 1280, num_heads = 20)**
  - Blocks: $\mathbf{2.1 * 10^{12}}$ **FLOPs** (94% of total)
  - Final LayerNorm and Output Projection: $\mathbf{1.32 * 10^{11}}$ **FLOPs** (6% of total)
  - Total: Approximately **2.23 TFLOPs** per forward pass.

As the model size increases, the proportional FLOPs cost changes toward the **Transformer blocks**, especially the FFN layers, which dominate the computational cost in larger models. The final output projection becomes relatively less expensive as the model size increases.

e. Increasing the context_length from 1024 to 16,384 (a 16x increase) results in a significant rise in total FLOPS, approximately **33.2x** (from 4.51 TFLOPs to around 149.8 TFLOPs). As the context length grows, the contribution to the FLOPS shifts from the FFN layers to the attention score calculation. This shift becomes more pronounced at longer context lengths, as **attention score calculations** become the dominant cost due to their quadratic scaling with sequence length. Consequently, attention score computations grow from **7%** to **55%** of total FLOPS, becoming the new bottleneck.

4. **Training a Transformer LM**

1. **Tuning the learning rate**

With lr = 1e1, the loss keeps decaying properly. Inital iterations see a sharp drop in loss, and then it continues to decrease steadily and slowly.

With lr = 1e2, the loss keeps fluctuating and diverging. It starts with a sharp drop but then it increaes and decreases erratically, indicating that the model is not converging properly.

With lr = 1e3, the loss diverges immediately and keeps increasing without any sign of convergence.

2. **Resource accounting for training with AdamW**

   a.

   b.

   c.

   d.

6 **Generating text**