简单大模型推理系统实现

由于考研复试在即，本人时间过于仓促，故完成了基本功能，达到通过要求:

- 文本生成
- AI对话

该项目只需cargo run就可直接运行，下面讲述这两个功能的具体实现方法

文本生成

完成模型的推理功能就可以实现文本生成，推理过程被封装在 `forward` 前向传播函数中，该函数在初始化参数后，迭代每个layer，在一个layer中进行一次注意力计算，这里要完成的是自注意力得分的计算。

由于Q的头数是KV头数的倍数，因此要分组进行计算，每个Q头共享一个，因此循环每一个KV头，再循环每一个Q头，计算一次注意力得分，写到attention矩阵中，这样将整个矩阵运算过程拆分成了多次点积运算。

计算完点积后，还要和V矩阵中的向量再进行一次点积，之后就完成了整个注意力的计算。

代码实现：

Code block

```
fn self_attention(
    hidden_states: &mut Tensor<f32>, // (seq, n_kv_h * n_groups * dqkv)
    att_scores: &mut Tensor<f32>,    // (n_kv_h, n_groups, seq, total_seq)
    q: &Tensor<f32>,                 // (seq, n_kv_h * n_groups * dqkv)
    k: &Tensor<f32>,                 // (total_seq, n_kv_h * dqkv)
    v: &Tensor<f32>,                 // (total_seq, n_kv_h * dqkv)
    n_kv_h: usize,
    n_groups: usize,
    seq_len: usize,
    total_seq_len: usize,
    dqkv: usize,
) {
    // 计算注意力分数
    for kv_head in 0..n_kv_h {
        for group in 0..n_groups {
            let mut attn_data = unsafe { att_scores.data_mut() };
            let q_head = kv_head * n_groups + group; // 当前 Q 头索引
            let q_stride = n_kv_h * n_groups * dqkv; // Q 每个 seq 位置的总维度
            let k_stride = n_kv_h * dqkv; // K 每个 seq 位置的总维度
```

```
20
21              // 点积
22              for q_pos in 0..seq_len {
23                  for k_pos in 0..total_seq_len {
24                      // Q[q_pos, q_head * dqkv + d] & K[k_pos, kv_head * dqkv
    + d]
25                      let score = (0..dqkv)
26                          .map(|d| {
27                              q.data()[q_pos * q_stride + q_head * dqkv + d]
28                                  * k.data()[k_pos * k_stride + kv_head * dqkv
    + d]
29                          })
30                          .sum::<f32>()
31                          * (1.0 / (dqkv as f32).sqrt());
32
33                      // 存入 att_scores: [kv_head][group][q_pos][k_pos]
34                      let attn_idx = kv_head * n_groups * seq_len *
    total_seq_len
35                          + group * seq_len * total_seq_len
36                          + q_pos * total_seq_len
37                          + k_pos;
38                      attn_data[attn_idx] = score;
39                  }
40              }
41          }
42      }
43
44      masked_softmax(att_scores);
45
46      // 加权求和 (Attn @ V)
47      for kv_head in 0..n_kv_h {
48          for group in 0..n_groups {
49              let q_head = kv_head * n_groups + group;
50              let v_stride = n_kv_h * dqkv; // V 每个 seq 位置的总维度
51              let h_stride = n_kv_h * n_groups * dqkv; // hidden_states 的总维度
52              let mut hidden_data = unsafe { hidden_states.data_mut() };
53
54              for q_pos in 0..seq_len {
55                  for d in 0..dqkv {
56                      // 维度索引，非 K 位置
57                      // 注意力权重: att_scores[kv_head][group][q_pos][k_pos]
58                      // V 的值: V[k_pos][kv_head * dqkv + d]
59                      let value = (0..total_seq_len)
60                          .map(|k_pos| {
61                              let attn_idx = kv_head * n_groups * seq_len *
    total_seq_len
62                                  + group * seq_len * total_seq_len
```

```
                                + q_pos * total_seq_len
                                + k_pos;
                            att_scores.data()[attn_idx]
                                * v.data()[k_pos * v_stride + kv_head * dqkv
    + d]
                        })
                        .sum::<f32>();

                    // 存入 hidden_states: [q_pos][q_head * dqkv + d]
                    let h_idx = q_pos * h_stride + q_head * dqkv + d;
                    hidden_data[h_idx] = value;
                }
            }
        }
    }
}
```

计算出一次得分矩阵后，将结果乘以权重再和 `residual` 相加，通过FFN后作为下一layer的输入。

接着完善generate函数，初始化好cache，和input向量，逐个token生成内容，将input更新为next_token以供下一轮输出

Code block

```
pub fn generate(
    &self,
    token _ids: &[u32],
    max_len: usize,
    top_p: f32,
    top_k: u32,
    temperature: f32,
) -> Vec<u32> {
    let mut result = Vec::<u32>::from(token_ids);
    result.push(self.bos_token_id);
    let mut cache: KVCache<f32> =
        KVCache::new(self.n_layers, self.max_seq_len, self.n_kv_h *
self.dqkv, 0);
    let mut input = Tensor::<u32>::new(token_ids.to_vec(), &vec!
[token_ids.len()]);

    // 按照最大长度生成结果
    for _ in 0..max_len {
        // 前向传播，获取 logits
        let logits = self.forward(&input, &mut cache);

        let next_token = OP::random_sample(&logits, top_p, top_k,
temperature);
```

```
21
22          if next_token == self.eos_token_id {
23              break;
24          }
25          result.push(next_token);
26
27          input = Tensor::<u32>::new(vec![next_token], &vec![1]);
28      }
29
30      result
31  }
```

最后cargo run运行代码：

```
Once upon a time, a little boy named Tim wanted to find a new screen in the park. He saw a big tree with many acorner on it. Tim wanted to enter the branch with the branch.
In the tree, Tim saw a big tree with many animals. Tim thought it was a magic leave. He wanted the bird to open the tree. So, he picked up the bird away.
Tim took the bird away, but he still felt dizzy. He took the automobile the fruits branches. Tim felt good and said, "I'm sorry, little bird. I will make it for you."
The bird flew away from the tree, and the birds fell down. Tim felt brave and agreed. They played together and had fun. Tim learned a lesson that day.<|end_story|>
```

AI对话

由于时间仓促，在本功能中我直接把prompt写死在对话中，而没有使用模板，故回答质量一般，但也算是完成了任务。

代码实现：

封装了一个对话Manager，存储历史记录和cache，在终端接收用户的输入并作出回答

Code block

```
1   impl ChatManager {
2       fn new(llama: Llama<f32>, tk: Tokenizer) -> Self {
3           ChatManager {
4               messages: vec![],
5               kv_cache: llama.new_cache(),
6               history: HashMap::new(),
7               llama,
8               tokenizer: tk,
9           }
10      }
11
12      fn run(&mut self) {
13          loop {
14              self.messages
15                  .push("user: hello. AI is chatting with user\n".to_string());
16
17              self.messages.push("AI: nice to meet you\n".to_string());
18
19              self.messages
```

```
                .push("user: please chat wite me\n".to_string());

            self.messages.push("AI: OK\n".to_string());


            print!("user: ");
            io::stdout().flush().unwrap();
            let mut input = String::new();
            io::stdin().read_line(&mut input).unwrap();
            let input = input.trim();

            let prompt = self.messages.join("") + "AI: ";
            let binding = self.tokenizer.encode(prompt, false).unwrap();
            let input_ids = binding.get_ids();

            let output_ids = self.llama.answer(input_ids, 100, 0.8, 30, 1., &mut self.kv_cache);
            let resp = self.tokenizer.decode(&output_ids, true).unwrap();

            self.messages
                .push(format!("AI: {}\n", resp));
            println!("AI: {}", resp);
        }
    }
}
```

实现了一个answer函数，用于回答问题

Code block

```
pub fn answer(
    &self,
    token_ids: &[u32],
    max_len: usize,
    top_p: f32,
    top_k: u32,
    temperature: f32,
    kv_cache: &mut KVCache<f32>,
) -> Vec<u32> {
    let mut result = Vec::<u32>::from(token_ids);
    result.push(self.bos_token_id);
    let mut input = Tensor::<u32>::new(token_ids.to_vec(), &vec![token_ids.len()]);

    // 按照最大长度生成结果
    for _ in 0..max_len {
```

```
16            // 前向传播，获取 logits
17            let logits = self.forward(&input, kv_cache);
18
19            let next_token = OP::random_sample(&logits, top_p, top_k,
    temperature);
20
21            if next_token == self.eos_token_id {
22                break;
23            }
24
25            result.push(next_token);
26            input = Tensor::<u32>::new(vec![next_token], &vec![1]);
27        }
28
29        result
30    }
```

运行代码：

```
user: hahaha
AI: user: hello. AI is chatting with user
AI: nice to meet you
user: please chat wite me
AI: OK
AI: "Why could you make my cut?"
```

结语

在训练营学习的这段时光非常可贵，收益良多。在此之前，我对人工智能的了解甚少，更别说大模型了。但通过这个训练营，我对人工智能的基础理论和应用有了新的认知，我深感AI的博大精深，有太多我要学习的了。

在该训练营中，我在有限的时间里尽力完成每一个作业和任务，奈何本人能力和时间有限，远远没有达到完美，但我觉得我只要保持兴趣，一定能不断地接近完美。

在此向训练营的所有老师表达由衷的感谢！