项目报告

项目地址 https://github.com/451846939/learning-lm-rs

1. 算子实现

swiglu

```
pub fn swiglu<T: Default + Copy + Sum + Float>(y: &mut Tensor<T>, x: &Tensor<T>) {
    assert!(
        y.shape() == x.shape(),
        "Input and output tensors must have the same shape"
    );

    let length = y.size();
    let y_data = unsafe { y.data_mut() };
    let x_data = x.data();

    for i in 0..length {
        let x_sigmoid = T::one() / (T::one() + (-x_data[i]).exp());
        y_data[i] = y_data[i] * x_sigmoid * x_data[i];
    }
}
```

matmul_transb

```
pub fn matmul_transb<T: Default + Copy + Sum + Float>(
    c: &mut Tensor<T>, // 输出矩阵 C
    beta: T,           // 缩放因子，用于累加已有的输出
    a: &Tensor<T>,     // 输入矩阵 A
    b: &Tensor<T>,     // 输入矩阵 B，需要转置
    alpha: T,          // 缩放因子，用于输出累积
) {
    // 获取形状信息
    let (m, k_a) = (a.shape()[0], a.shape()[1]);
    let (k_b, n) = (b.shape()[1], b.shape()[0]);
    let (m_c, n_c) = (c.shape()[0], c.shape()[1]);

    // 检查形状合法性
    assert_eq!(
        k_a, k_b,
        "A.cols must match B.cols for C = A x B^T! (A: [m,k], B: [n,k])"
    );
    assert_eq!(m, m_c, "C rows must match A rows!");
    assert_eq!(n, n_c, "C cols must match B rows (B is transposed)!");

    let a_data = a.data();
    let b_data = b.data();
    let c_data = unsafe { c.data_mut() };

    // 计算 C = alpha * (A x B^T) + beta * C
    for i in 0..m {
        for j in 0..n {
            let mut sum = T::zero();

            // 计算 A 的第 i 行和 B 转置的第 j 行的点积
            for k in 0..k_a {
                let a_idx = i * k_a + k; // A 的索引
                let b_idx = j * k_b + k; // B^T 的索引 (B 的转置)

                // 检查索引是否越界
                assert!(
                    a_idx < a_data.len(),
                    "Index out of bounds for A: a_idx={}, len={}",
                    a_idx,
                    a_data.len()
                );
                assert!(
                    b_idx < b_data.len(),
                    "Index out of bounds for B: b_idx={}, len={}",
                    b_idx,
                    b_data.len()
                );

                sum = sum + a_data[a_idx] * b_data[b_idx];
            }

            // 计算 C[i, j]
            let c_idx = i * n + j;
            assert!(
                c_idx < c_data.len(),
                "Index out of bounds for C: c_idx={}, len={}",
                c_idx,
                c_data.len()
            );
            c_data[c_idx] = alpha * sum + beta * c_data[c_idx];
        }
    }
}
```

rms_norm

这里使用了broadcast

```
pub fn rms_norm<T>(y: &mut Tensor<T>, x: &Tensor<T>, w: &Tensor<T>, epsilon: T)
where
    T: Copy + Default + std::ops::Mul<Output = T> + std::ops::Div<Output = T> + Float
+ std::iter::Sum,
{
    // 提前获取只读信息，避免后续对 y 的不可变借用
    let y_shape = y.shape().clone();
    let y_size = y.size();

    // 获取输入和输出张量的形状
    let x_shape = x.shape();
    let w_shape = w.shape();

    // 检查权重张量 w 必须是 1D
    assert_eq!(
        w_shape.len(),
        1,
        "Weight tensor w must be 1D, but got shape: {:?}",
        w_shape
    );

    // 检查权重张量的大小是否与最后一个维度匹配
    assert_eq!(
        w.size(),
        *y_shape.last().unwrap(),
        "Size of weight tensor must match the last dimension of output tensor"
    );

    // 如果输入形状与输出形状不一致，尝试广播输入张量
    let x_broadcasted = if x_shape != &y_shape {
        broadcast_tensor(x, &y_shape)
    } else {
        x.clone()
    };

    // 广播权重张量到输出张量的形状
    let w_broadcasted = broadcast_tensor(w, &vec![*y_shape.last().unwrap()]);

    // 获取张量数据
    let x_data = x_broadcasted.data();
    let w_data = w_broadcasted.data();

    // 提取 y 的可变数据
    let y_data = unsafe { y.data_mut() };

    let last_dim = *y_shape.last().unwrap();
    let batch_size = y_size / last_dim;

    for b in 0..batch_size {
        let base = b * last_dim;
        // 计算每个向量的均值平方根 (RMS)
        let mean_square: T = (0..last_dim).map(|i| x_data[base + i].powi(2)).sum::<T>
() / T::from(last_dim).unwrap();
        let rms = mean_square.sqrt().max(epsilon.into());

        // 归一化并乘以权重
        for i in 0..last_dim {
            let x_val:  T= x_data[base + i].into();
            let w_val: T = w_data[i].into();
            y_data[base + i] = T::from((x_val / rms) * w_val).unwrap();
        }
    }
}

pub fn broadcast_shapes(shape1: &Vec<usize>, shape2: &Vec<usize>) -> Vec<usize> {
    let mut result = vec![];
    let len1 = shape1.len();
    let len2 = shape2.len();

    for i in 0..len1.max(len2) {
        let dim1 = if i >= len1 { 1 } else { shape1[len1 - 1 - i] };
        let dim2 = if i >= len2 { 1 } else { shape2[len2 - 1 - i] };

        if dim1 != dim2 && dim1 != 1 && dim2 != 1 {
            panic!("Cannot broadcast shapes {:?} and {:?}", shape1, shape2);
        }
        result.push(dim1.max(dim2));
    }

    result.reverse();
    result
}

pub fn broadcast_tensor<T: Copy + Default>(
    tensor: &Tensor<T>,
    target_shape: &Vec<usize>,
) -> Tensor<T> {
    let source_shape = tensor.shape();
    let source_data = tensor.data();

    // 1) 先求目标形状的总元素数
    let broadcasted_size: usize = target_shape.iter().product();
    let mut broadcasted_data = vec![T::default(); broadcasted_size];

    // 2) 预计算源张量每个维度的 strides (从左到右，最后一个维度 stride=1)
    //    比如 source_shape = [2, 3, 4], strides = [3*4, 4, 1] = [12, 4, 1]
    let mut strides = vec![1; source_shape.len()];
    for i in (0..source_shape.len() - 1).rev() {
        strides[i] = strides[i + 1] * source_shape[i + 1];
    }

    // 3) 额外检查: 逐个维度 (从右到左) 判断是否可广播
    {
        let mut i_s = source_shape.len() as isize - 1;
        let mut i_t = target_shape.len() as isize - 1;
        while i_s >= 0 && i_t >= 0 {
            let sdim = source_shape[i_s as usize];
            let tdim = target_shape[i_t as usize];
            if sdim != 1 && tdim != 1 && sdim != tdim {
                panic!(
                    "无法将形状 {:?} 广播到 {:?}, 维度 {} 与 {} 不兼容",
                    source_shape, target_shape, sdim, tdim
                );
            }
            i_s -= 1;
            i_t -= 1;
        }
        // 注意: 如果 source 维数比 target 维数更少，是允许的；那些更“高”位会被当作复制。
        // 若 target 更短，一般在外层逻辑就不会调用此函数去做“反向”广播，这里不再处理。
    }

    // 4) 遍历目标张量的每个索引 i (相当于 linear index) ，逐步解码到各维度，然后映射回源张量。
    //    解码顺序: 先 / target_shape[last] 求出“在最后一维的坐标”，再 / target_shape[last-
1]...
    for i in 0..broadcasted_size {
        let mut tmp = i;
        let mut source_index = 0;

        // 从右到左遍历 target_shape
        for dim_t in (0..target_shape.len()).rev() {
            let coord_t = tmp % target_shape[dim_t]; // 目标张量该维度的坐标
            tmp /= target_shape[dim_t];

            // 计算对应的源张量的维度下标 dim_s
            let offset = target_shape.len() - 1 - dim_t; // 与最右维的距离
            let dim_s = (source_shape.len() as isize - 1) - (offset as isize);
            if dim_s < 0 {
                // 源张量在该维度上相当于自动视为 1，所有坐标都映射到源张量的0
                continue;
            }

            let dim_s = dim_s as usize;
            let size_s = source_shape[dim_s];
            let stride_s = strides[dim_s];

            // 如果源张量在该维度 = 1，就只能用 index=0 做复制
            // 否则就用 coord_t
            let src_coord = if size_s == 1 { 0 } else { coord_t };

            source_index += src_coord * stride_s;
        }

        // 根据算出的 source_index 去源张量 data() 中取值
        broadcasted_data[i] = source_data[source_index];
    }

    // 5) 构造新的广播后 Tensor 返回
    Tensor::new(broadcasted_data, target_shape)
}

pub fn add_broadcasted<T: Copy + Default + std::ops::Add<Output = T>>(
    a: &Tensor<T>,
    b: &Tensor<T>,
) -> Tensor<T> {
    let target_shape = broadcast_shapes(a.shape(), b.shape());
    let a_broadcasted = broadcast_tensor(a, &target_shape);
    let b_broadcasted = broadcast_tensor(b, &target_shape);

    let result_data: Vec<T> = a_broadcasted
        .data()
        .iter()
        .zip(b_broadcasted.data())
        .map(|(&x, &y)| x + y)
        .collect();

    Tensor::new(result_data, &target_shape)
}
```

mlp

```
fn mlp<T: Default + Copy + Sum + Float + FromPrimitive>(
    residual: &mut Tensor<T>,
    hidden_states: &mut Tensor<T>,
    gate: &mut Tensor<T>,
    up: &mut Tensor<T>,
    w_up: &Tensor<T>,
    w_down: &Tensor<T>,
    w_gate: &Tensor<T>,
    rms_w: &Tensor<T>,
    eps: T,
) {
    OP::rms_norm(hidden_states, &residual, &rms_w, eps);
    OP::matmul_transb(gate, T::zero(), &hidden_states, &w_gate, T::one());
    OP::matmul_transb(up, T::zero(), &hidden_states, &w_up, T::one());
    OP::swiglu(up, &gate);
    OP::matmul_transb(residual, T::one(), &up, &w_down, T::one());
}
```

llma 模型加载

```rust
pub fn from_safetensors(safetensor: &SafeTensors, config: &LlamaConfigJson) -> Self {
    // 打印 safetensors 中所有可用的张量名称
    println!("Available tensors: {:?}", safetensor.names());

    let get_tensor = |name: &str| -> Tensor<T> {
        let tensor_view = safetensor
            .tensor(name)
            .unwrap_or_else(|_| panic!("Tensor `{}` not found in safetensors", name));
        let tensor_dtype = tensor_view.dtype();
        let element_size = tensor_dtype.size();
        let data = tensor_view
            .data()
            .chunks_exact(element_size)
            .map(|chunk| T::from_le_bytes(chunk.try_into().unwrap()))
            .collect::<Vec<T>>();
        let shape = tensor_view.shape().to_vec();
        Tensor::<T>::new(data, &shape)
    };

    let n_layers = config.num_hidden_layers;

    let embedding_table = if config.tie_word_embeddings {
        get_tensor("lm_head.weight")
    } else {
        get_tensor("model.embed_tokens.weight")
    };

    LLamaParams {
        embedding_table: embedding_table,
        rms_att_w: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.input_layernorm.weight")))
            .collect(),
        wq: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.self_attn.q_proj.weight")))
            .collect(),
        wk: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.self_attn.k_proj.weight")))
            .collect(),
        wv: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.self_attn.v_proj.weight")))
            .collect(),
        wo: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.self_attn.o_proj.weight")))
            .collect(),
        rms_ffn_w: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.
{f}.post_attention_layernorm.weight")))
            .collect(),
        w_up: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.mlp.up_proj.weight")))
            .collect(),
        w_gate: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.mlp.gate_proj.weight")))
            .collect(),
        w_down: (0..n_layers)
            .map(|f| get_tensor(&format!("model.layers.{f}.mlp.down_proj.weight")))
            .collect(),
        rms_out_w: get_tensor("model.norm.weight"),
        lm_head: get_tensor("lm_head.weight"),
    }
}
```

完整的forward

```rust
pub fn forward(&self, input: &Tensor<u32>, cache: &mut KVCache<T>) -> Tensor<T> {
    let seq_len = input.size();
    let past_seq_len = cache.len();
    cache.increment(seq_len);
    let total_seq_len = past_seq_len + seq_len;
    let n_groups = self.n_q_h / self.n_kv_h;

    // Some pre-allocated buffers that will be reused
    let mut residual = Tensor::<T>::default(&vec![seq_len, self.d]);
    let mut hidden_states = Tensor::<T>::default(&vec![seq_len, self.d]);
    let mut q_buf = Tensor::<T>::default(&vec![seq_len, self.n_q_h * self.dqkv]);
    let mut att_scores =
        Tensor::<T>::default(&vec![self.n_kv_h, n_groups, seq_len, total_seq_len]);
    let mut gate_buf = Tensor::<T>::default(&vec![seq_len, self.di]);
    let mut up_buf = Tensor::<T>::default(&vec![seq_len, self.di]);

    // Computation Starts Here
    // Embedding lookup
    OP::gather(&mut residual, input, &self.params.embedding_table);

    for layer in 0..self.n_layers {
        OP::rms_norm(
            &mut hidden_states,
            &residual,
            &self.params.rms_att_w[layer],
            T::from_f32(self.eps).unwrap(),
        );

        let q = (&mut q_buf).reshape(&vec![seq_len, self.n_q_h * self.dqkv]); // (seq,
n_h * dqkv)
        let k = &mut cache.k_cache(layer, past_seq_len); // (seq, n_kv_h * dqkv)
        let v = &mut cache.v_cache(layer, past_seq_len); // (seq, n_kv_h * dqkv)
        OP::matmul_transb(
            q,
            T::zero(),
            &hidden_states,
            &self.params.wq[layer],
            T::one(),
        );
        OP::matmul_transb(
            k,
            T::zero(),
            &hidden_states,
            &self.params.wk[layer],
            T::one(),
        );
        OP::matmul_transb(
            v,
            T::zero(),
            &hidden_states,
            &self.params.wv[layer],
            T::one(),
        );
        OP::rope(
            q.reshape(&vec![seq_len, self.n_q_h, self.dqkv]),
            past_seq_len,
            T::from_f32(self.rope_theta).unwrap(),
        );
        OP::rope(
            k.reshape(&vec![seq_len, self.n_kv_h, self.dqkv]),
            past_seq_len,
            T::from_f32(self.rope_theta).unwrap(),
        );

        let full_k = &mut cache.k_cache(layer, 0); // (total_seq, n_kv_h * dqkv)
        let full_v = &mut cache.v_cache(layer, 0); // (total_seq, n_kv_h * dqkv)

        self_attention_multihead(
            &mut hidden_states,
            &mut att_scores,
            q,
            full_k,
            full_v,
            self.n_kv_h,
            n_groups,
            seq_len,
            total_seq_len,
            self.dqkv,
        );

        OP::matmul_transb(
            &mut residual,
            T::one(),
            &hidden_states,
            &self.params.wo[layer],
            T::one(),
        );

        mlp(
            &mut residual,
            &mut hidden_states,
            &mut gate_buf,
            &mut up_buf,
            &self.params.w_up[layer],
            &self.params.w_down[layer],
            &self.params.w_gate[layer],
            &self.params.rms_ffn_w[layer],
            T::from_f32(self.eps).unwrap(),
        );
    }

    // No matter what seq_len, the output is always a 1D vector of length vocab,
    // which contains the probabilities for the next token.
    let mut logits = Tensor::<T>::default(&vec![1, self.vocab]);
    let mut hidden_states = hidden_states.slice((seq_len - 1) * self.d, &vec![1,
self.d]);
    let residual = residual.slice((seq_len - 1) * self.d, &vec![ self.d]);

    OP::rms_norm(
        &mut hidden_states,
        &residual,
        &self.params.rms_out_w,
        T::from_f32(self.eps).unwrap(),
    );

    OP::matmul_transb(
        &mut logits,
        T::zero(),
        &hidden_states,
        &self.params.lm_head,
        T::one(),
    );

    logits
}
```

2. 项目阶段

Self-Attention 多头

```rust
pub fn self_attention_multihead<T: Default + Copy + Sum + Float + FromPrimitive>(
    hidden_states: &mut crate::tensor::Tensor<T>,
    att_scores: &mut crate::tensor::Tensor<T>,
    q: &crate::tensor::Tensor<T>,
    k: &crate::tensor::Tensor<T>,
    v: &crate::tensor::Tensor<T>,
    n_kv_h: usize,
    n_groups: usize,
    seq_len: usize,
    total_seq_len: usize,
    dqkv: usize,
) {
    // ========== Step 1: 先填 0 ==========
    {
        let att_data = unsafe { att_scores.data_mut() };
        att_data.fill(T::zero());
    }

    // ========== Step 2: 计算 Q×K^T => 填进 att_scores ==========
    // 这里再用 "att_data" 但记得别留到 masked_softmax 之后
    let q_data = q.data();
    let k_data = k.data();
    let inv_scale = (T::one()/ (T::from(dqkv).unwrap() ).sqrt());

    let num_key_value_heads = n_kv_h;
    let num_query_heads_per_kv_group = n_groups;
    let num_attention_heads = n_kv_h * n_groups;
    let d_head = dqkv;

    let total_d_q = num_attention_heads * d_head;
    let total_d_kv = num_key_value_heads * d_head;

    let total_d_atts_3 = num_query_heads_per_kv_group * seq_len * total_seq_len;
    let total_d_atts_2 = seq_len * total_seq_len;
    let total_d_atts_1 = total_seq_len;

    {
        // 因为还没调用 masked_softmax，这里可以暂时可变借用
        let att_data = unsafe { att_scores.data_mut() };

        for curr_k_head in 0..num_key_value_heads {
            let offset_k = curr_k_head * d_head;
            for curr_q_in_group in 0..num_query_heads_per_kv_group {
                let curr_att_head = curr_k_head*num_query_heads_per_kv_group +
curr_q_in_group;
                let offset_q = curr_att_head*d_head;
                for i_seq in 0..seq_len {
                    let begin_vec_q = i_seq*total_d_q + offset_q;
                    for i_tseq in 0..total_seq_len {
                        let begin_vec_k = i_tseq*total_d_kv + offset_k;
                        let mut dot = T::zero();
                        for dd in 0..d_head {
                            dot = dot+ q_data[begin_vec_q+dd]*k_data[begin_vec_k+dd];
                        }
                        dot = dot * inv_scale;

                        let att_idx = curr_k_head*total_d_atts_3
                            + curr_q_in_group*total_d_atts_2
                            + i_seq*total_d_atts_1
                            + i_tseq;
                        att_data[att_idx] = dot;
                    }
                }
            }
        }
    }

    // ========== Step 3: 调 masked_softmax(att_scores) ==========
    crate::operators::masked_softmax(att_scores);
    // 这里需要 &mut att_scores

    // ========== Step 4: hidden_states = att_scores × V ==========
    // 这里重新用 read-only 的 att_data
    let att_data = att_scores.data(); // <-- 只读
    let v_data = v.data();
    {
        let hs_data = unsafe { hidden_states.data_mut() };
        hs_data.fill(T::zero());

        for curr_v_head in 0..num_key_value_heads {
            let offset_matrix_v_g = curr_v_head*d_head;
            for curr_q_in_group in 0..num_query_heads_per_kv_group {
                let offset_matrix_a_h = curr_q_in_group*total_d_atts_2
                    + curr_v_head*total_d_atts_3;
                for curr_idx_seq in 0..seq_len {
                    let begin_vec_a = offset_matrix_a_h + curr_idx_seq*total_d_atts_1;
                    for curr_idx_dhead in 0..d_head {
                        let begin_vec_v = curr_idx_dhead + offset_matrix_v_g;
                        let mut sum_ = T::zero();
                        for curr_idx_tseq in 0..total_seq_len {
                            let idx_a = begin_vec_a + curr_idx_tseq;
                            let idx_v = begin_vec_v + curr_idx_tseq*total_d_kv;
                            sum_ = sum_ + att_data[idx_a] * v_data[idx_v];
                        }

                        let curr_att_head = curr_v_head*num_query_heads_per_kv_group +
curr_q_in_group;
                        let hs_offset = curr_idx_seq*(num_attention_heads*d_head)
                            + curr_att_head*d_head
                            + curr_idx_dhead;
                        hs_data[hs_offset] = sum_;
                        }
                    }
                }
            }
        }
    }
```

文本生成结果

```
Once upon a timeOnce upon a time, a little boy named Tim found a bitter box. It was
unknown under the box and started to play with it. Tim was so excited to play with it.
When Tim opened the box, he saw a big, round red key. He was so happy to play with it.
Tim took the key home and started to play with it.
Tim took the key home and showed it to his mom. "Look!" he called the key xnew. Tim
smiled and played catch with the key and had lots of fun. Then, he saw his friend,
Sam, in the house.
Tim wanted to play the key and play. He played with the key on Tim's bed in his toy
car. Soon, as he played, he felt much better another toy. He said, "Thank you, Mom!"
They played together all day with the key back, and Sue loved Tim because of the
kitten.<|end_story|>
```

ai对话

使用Message format来进行模版指定

```
#[derive(Clone)]
pub struct Message {
    pub(crate) role: String,
    pub(crate) content: String,
}

impl Message {
    pub(crate) fn format(&self) -> String {
        format!("<|im_start|>{}\n{}<|im_end|>\n", self.role, self.content)
    }
}
```

结果:

```
You: hi
Assistant: I am not capable of creating visual content . However , based on your
personal experience with the internet , here are some general tips on how to create
visual content for my website :

 1 . Use relevant keywords : Use relevant keywords to make the site more relevant to
your audience . For example , " The best way to create content is to use relevant
keywords to show case the website ' s content ."

 2 . Create a clear and conc ise design : use clear and conc ise language that is easy
to read . For example , " The website should be clean and modern , with clear

You:
```

网络服务和api

网络服务 server 使用http chunk 返回 server使用每个http请求会话一个kvcache

```
/// **多用户 KVCache 管理**
pub struct KVCacheManager<T> {
    cache_map: RwLock<HashMap<String, Arc<RwLock<KVCache<T>>>>>,  // ✅ 允许可变访问
    n_layers: usize,
    max_seq_len: usize,
    dim: usize,
}

impl<T: Default + Copy + Send + Sync + 'static> KVCacheManager<T> {
    pub fn new(n_layers: usize, max_seq_len: usize, dim: usize) -> Arc<Self> {
        Arc::new(KVCacheManager {
            cache_map: RwLock::new(HashMap::new()),
            n_layers,
            max_seq_len,
            dim,
        })
    }

    /// **获取用户的 KVCache**
    pub async fn get_cache_for_user(&self, user_id: &str) -> Arc<RwLock<KVCache<T>>> {
        let mut cache = self.cache_map.write().await;
        cache
            .entry(user_id.to_string())
            .or_insert_with(|| Arc::new(RwLock::new(KVCache::new(self.n_layers,
self.max_seq_len, self.dim, 0))))
            .clone()  // ✅ 现在 `.clone()` 没问题
    }

    /// **存储用户 KVCache** (可以直接更新 `RwLock<KVCache<T>>`, 不需要存回 `HashMap`)
    pub async fn store_cache_for_user(&self, _user_id: &str, _cache:
Arc<RwLock<KVCache<T>>>) {
        // 这里不需要额外存储, 因为 `Arc<RwLock<KVCache<T>>>` 内部已经更新
    }
}
```

cargo 引入

```
futures-util = { version = "0.3", default-features = false, features = ["sink", "std"]
}
tokio-tungstenite = { version = "0.21" }

tokio = { version = "1.6.0", features = ["full"] }
tokio-util = { version = "0.6.7", features = ["full"] }
tokio-stream = { version = "0.1.6"}
axum = { version = "0.7.5"}
axum-extra = { version = "0.9.3", features = ["typed-header"] }
log = "0.4.22"
```

```
2025-01-29T14:17:49.739092Z  WARN tokenizers::tokenizer::serialization: Warning: Token
'<|im_end|>' was expected to have ID '32000' but was given ID 'None'
2025-01-29T14:17:49.739117Z  WARN tokenizers::tokenizer::serialization: Warning: Token
'<|im_start|>' was expected to have ID '32001' but was given ID 'None'
2025-01-29T14:17:49.741357Z  INFO learning_lm_rust::server: 🚀 Server running at
127.0.0.1:8000
```

curl 请求

```
curl -N -X POST http://127.0.0.1:8000/chat \
     -H "Content-Type: application/json" \
     -d '{"user_id": "user1", "user_input": "Hello"}'
Hello ! My name is Maria . She is a music lover and is known for her hard rock - based
lyrics and traditional instruments . She is an ind ie pop singer who special izes in
folk music and dance . She is a great jazz and jazz drum mer who has always been an
independent and rock musician . %
```

混合精度

这里cargo 引入

```
num-traits = "0.2.19"
half = { version = "2.4.1", features = ["num-traits"] }
```

读取模型的config配置根据`torch_dtype` 来加载支持不同的精度这里使用范型来指定加载的mode，chat和server 同理

```rust
fn story_mode() {
    let project_dir = env!("CARGO_MANIFEST_DIR");
    let model_dir = PathBuf::from(project_dir).join("models").join("story");
    let config = crate::model::read_config(&model_dir);

    // 选择正确的 Llama<T> 类型
    match config.torch_dtype {
        TorchDType::Float32 => story_mode_inner::<f32>(model_dir, config),
        TorchDType::Float16 => story_mode_inner::<f16>(model_dir, config),
        TorchDType::BFloat16 => story_mode_inner::<bf16>(model_dir, config),
    }
}

// 泛型函数，减少重复代码
fn story_mode_inner<T>(model_dir: PathBuf, config: LlamaConfigJson)
where
    T: Float + Default + Copy + Sum + FromPrimitive + FromLeBytes,
{
    let llama: model::Llama<T> = model::Llama::<T>::from_safetensors(&model_dir,
config);

    let tokenizer = Tokenizer::from_file(model_dir.join("tokenizer.json")).unwrap();
    let input = "Once upon a time";
    let binding = tokenizer.encode(input, true).unwrap();
    let input_ids = binding.get_ids();

    print!("\n{}", input);
    let output_ids = llama.generate(
        input_ids,
        500,
        T::from_f32(0.8).unwrap(),  // ✅ 使用 `from_f32()`
        30,
        T::from_f32(1.0).unwrap(),  // ✅ 使用 `from_f32()`
    );
    println!("{}", tokenizer.decode(&output_ids, true).unwrap());
}
```