LSM-Tree phase2 预热

1 程序逻辑

测试文件包含 basic 和 large 两个测试。

1.1 basic 测试逻辑

在 basic 测试中，首先调用 embedding 函数，嵌入所有字符串，并获取每个字符串对应的向量。具体来说，对每个词语，函数使用 common_tokenize 将文本转化为 tokens，然后计算生成的向量数量，最后将 tokens 分批添加到处理对象中并调用 batch_decode 计算向量。

然后通过 common_embd_similarity_cos 函数，计算每两个字符串间的关联程度。

最后是三组测试用例。如果模型计算出的前两个词语的关联性大于后两个词语的关联性，即通过测试。

1.2 large 测试逻辑

large 测试首先读取了 trimmed_text.txt 文本。

然后，同样调用 embedding 函数，为每一行生成嵌入向量，然后为问题（chat_prompt）也生成嵌入向量。

接下来，遍历文件中每个句子的向量，找出与 chat_prompt 关联度最大的句子。如果与预期结果相同即通过测试。

1.3 输出含义

输出包含模型的加载日志。

输出了模型元数据、配置、上下文配置、内存分配、图配置、性能统计等。

2 代码片段逻辑

2.1 basic 部分

```
if (sim_matrix["Apple"]["Banana"] > sim_matrix["Apple"]["Man"]) {
  passed_count++;
}
if (sim_matrix["Apple"]["Orange"] > sim_matrix["Apple"]["Chicken"]) {
  passed_count++;
}
```

```
if (sim_matrix["Banana"]["Orange"] > sim_matrix["Banana"]["Man"]) {
  passed_count++;
}
```

这段代码包含三个测试案例；其中，每个测试案例中，前两个词的相似性应该大于后两个词的相似性。例如，在前两行中，“Apple” 和 “Banana” 的相关联程度应该高于 “Apple” 和 “Man” 的相关联程度。

2.2 large 部分

```
if (max_sim_sentence != supposed_sentence) {
  std::cerr << "Failed to find the correct sentence" << std::endl;
  return passed_count;
}
passed_count++;
```

如果模型给出的与问题关联度最大的句子等于预期结果，则通过测试。