

LSM-Tree Phase4 报告

2025 年 5 月 11 日

1 背景介绍

本阶段承接上一阶段，为 `search_knn_hnsw` 函数添加了删除和替换功能，并添加了向量和 HNSW 的持久化。

向量按更新顺序存在内存中，在需要删除、替换时，从最后追加即可。HNSW 的删除采用 lazy 模式，将被删除的点标记，搜索相似向量时跳过那些被标记删除的点。

向量和 HNSW 的持久化分别由两个模块负责。

代码实现完成后，运行对应测试检验结果的正确性。

2 代码设计

2.1 向量持久化

向量持久化的设计与文档中相同。

所有向量存放在 `embedding_data` 文件夹的某个文件内。文件开头存储每个向量的维度，然后按照顺序依次存储每个键和对应向量。

2.2 HNSW 删除

HNSW 的删除采用 lazy 模式。删除时，只需将原节点加入删除列表；修改时，将原节点加入删除列表，并新增节点。

删除列表不能简单地由向量构成。如果某个值被删除后又新增，因为值没有改变，向量也没有改变，所以向量仍然被存放在删除列表中，新增的节点也被标记为删除。只记录删除的键值对也会导致类似的问题。因此，需要为每个节点分配一个独立的 id，采用自增形式，可以唯一标记每个节点。

增加了 id 后，可以通过记录 id 记录删除的节点。但是，不能只记录 id，还要记录删除 id 对应的向量值，原因与持久化有关。某个点被删除后，它还存在于 HNSW 的数据结构中，所以

仍然需要保留它的向量、它与其它节点的关系等等；也就是说被删除点的向量需要持久化。对于未被删除的点，可以从持久化的向量中查找到所对应的向量。但对于被删除的键值对，我们只知道它原来的向量“位于键对应的最后一个向量之前”；例如，插入 1、string1，再经过两次修改，将其分别改为 1、string2，1、string3，此时无法直接通过 id 查找到对应向量值。如果一定要通过这种方式，就要遍历向量列表和 HNSW 中的节点列表，找到在 id 之前 id 对应 key 值出现的次数，再在向量列表中找到 key 等于 id->key，且出现次数与 id 相同的向量。这种方式逻辑较复杂，时间复杂度也较高，所以采用了记录被删除向量的方式。

2.3 HNSW 持久化

HNSW 持久化的设计也基本与文档中相同。需要记录的内容可分为三个部分：基本参数信息、删除节点信息、每个节点的具体信息（位置、向量值等）。

基本参数信息中，记录了节点总数、向量维度等信息；删除节点信息中，与上一节所说一致，记录了被删除节点的 id 和向量值；对于每个节点，记录了层级、键（用于获取向量）、邻居的数量和 id。

2.4 思考题

与文档不同的另一种设计方式：

将代码抽象分类为两个部分，分别用于数据记录和实现相应功能，以下简称记录类和实现类。

记录类为每次操作分配一个自增 id。在添加、删除、修改点时，在记录类中增加 id，追加新的向量，记录删除的向量，并为实现类提供接口。除了具体的 HNSW 树放在实现类里，其余记录数据都放在记录类中。持久化时，HNSW 只负责基本参数和邻居信息等。

向量、被删除的 id 及向量的持久化设置与文档相同。在 HNSW 中，只需记录 id 和邻居关系，不需要记录键值（向量）。

删除操作时，首先根据键找到要被删除的向量，将其 id 和向量添加到被删除的哈希表中，再从正常哈希表中删除，不需要对 HNSW 进行额外处理。修改操作等于删除操作加插入操作。

3 测试

3.1 实验设置

代码运行在 Linux 环境下，采用 Vector_Persistent_Test、HNSW_Delete_Test、原有的 HNSW_Persistent_Test 和张同学提供的 HNSW_Persistent_Test（以下称为 HNSW_Persistent_Complex_Test）进行测试。

3.2 预期结果

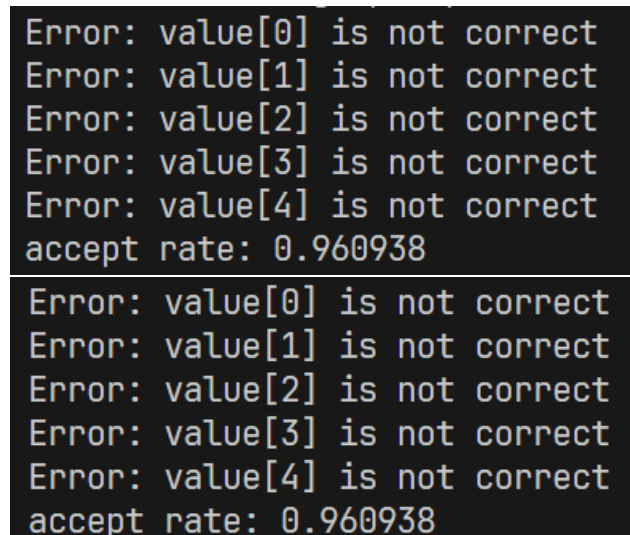
Vector_Persistent_Test、HNSW_Delete_Test、HNSW_Persistent_Complex_Test 通过;HNSW_Persistent_Test 正确率较高。

此外，HNSW_Persistent_Test 和 HNSW_Persistent_Complex_Test 在关闭程序、重启程序后进行了数次查询操作。如果将这些查询操作复制到关闭程序之前，两次查询给出的结果应该相同。

3.3 实验结果与分析

实验结果与预期一致。Vector_Persistent_Test 和 HNSW_Delete_Test 只有 Test Passed 输出，所以不单独配图。

HNSW_Persistent_Test 结果如图 1，HNSW_Persistent_Complex_Test 结果如图 2，各自上图为 phase1 结果，下图为 phase2 结果。可以看到 phase1、2 结果一致。



```
Error: value[0] is not correct
Error: value[1] is not correct
Error: value[2] is not correct
Error: value[3] is not correct
Error: value[4] is not correct
accept rate: 0.960938

Error: value[0] is not correct
Error: value[1] is not correct
Error: value[2] is not correct
Error: value[3] is not correct
Error: value[4] is not correct
accept rate: 0.960938
```

图 1: HNSW_Persistent_Test 测试结果

```
Replace Test Error: value[17] is not deleted
Replace Test Error: value[28] is not deleted
Replace Test Error: value[31] is not deleted
Test passed
Replace Test Error: value[17] is not deleted
Replace Test Error: value[28] is not deleted
Replace Test Error: value[31] is not deleted
Test passed
```

图 2: HNSW_Persistent_Complex_Test

持久化及删除、修改操作实现正确。

4 结论

本实验正确完成了向量、HNSW 的持久化实现，并支持了删改操作。

5 致谢

感谢知乎、维基百科等博客、网站提供的参考；感谢 deepseek、kimi 等大模型提供的思路与帮助。

感谢所有提问、帮助丰富问题文档的同学；感谢
感谢提供支持的朋友们。

6 其他和建议