<2025.9.10>

# 数据盘

AutoDL 已经给你挂好了数据盘，挂载点就是 `/root/autodl-tmp`，直接搬过去再软链接

一键命令

```
# 1. 搬数据（rsync 可续传，/root/autodl-tmp 就是数据盘）
rsync -aHAX --progress /root/ann-benchmarks/data/ /root/autodl-tmp/data/

# 2. 原目录改名备份（防止手抖）
mv /root/ann-benchmarks/data /root/ann-benchmarks/data.bak

# 3. 软链接回去，程序无感知
ln -s /root/autodl-tmp/data /root/ann-benchmarks/data

# 4. 验证
ls -l /root/ann-benchmarks/data          # 应指向 /root/autodl-tmp/data
python -c "import os; print(os.listdir('/root/ann-benchmarks/data'))"

# 5. 确认没问题后删掉备份，立即腾空间
rm -rf /root/ann-benchmarks/data.bak
```

---

再查一眼空间

```
df -h /
```

系统盘剩余空间立刻变大，后续下载的新数据集也直接丢 `/root/autodl-tmp/data` 即可。

# TEXT1M-200-angular

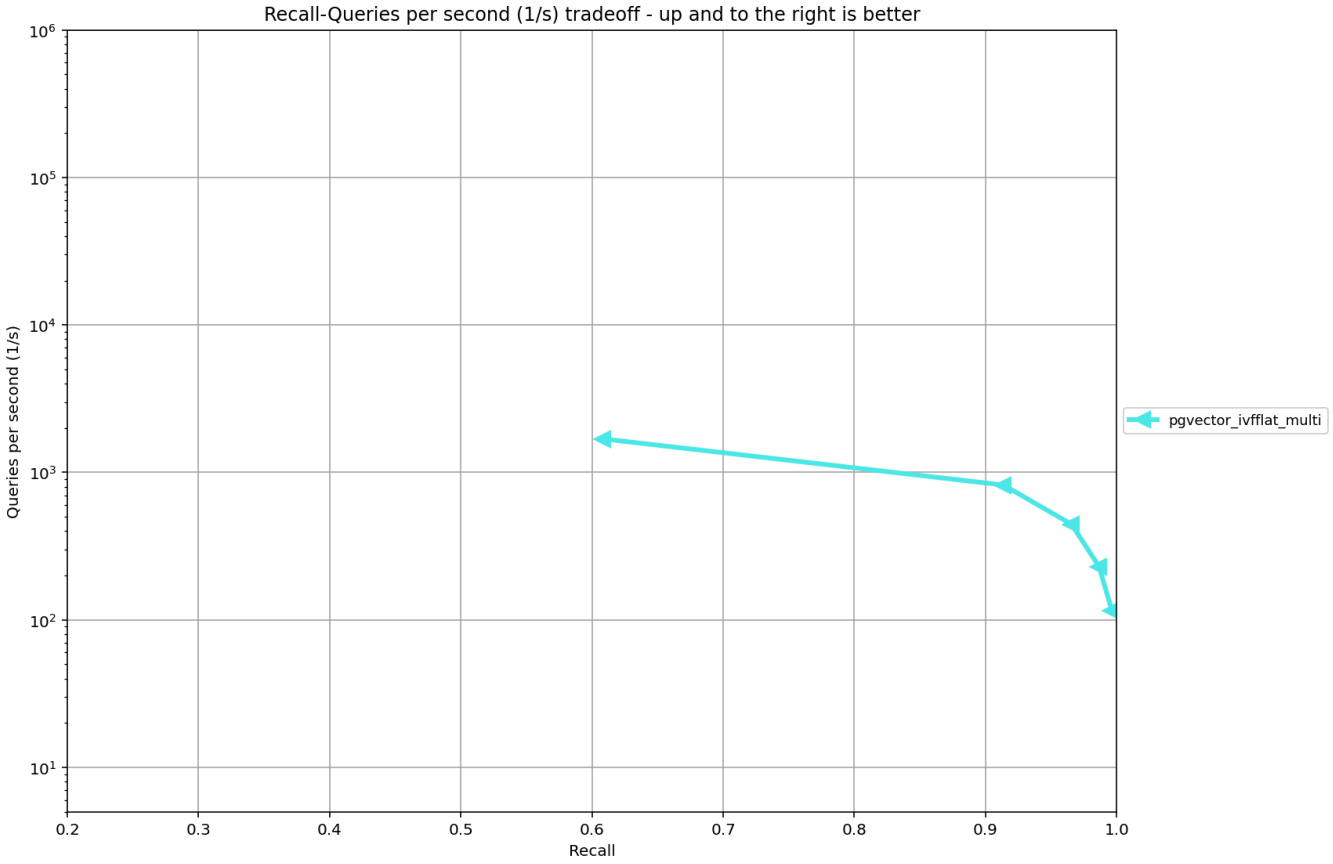
支持自动从网站上下载数据集

## Ours

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=999, probes=1) | 1.000 | 38073.943 | 26.265 |
| 1 | PGVector ivfflat Ours (lists=999, probes=5) | 1.000 | 37843.670 | 26.424 |
| 2 | PGVector ivfflat Ours (lists=999, probes=10) | 1.000 | 37915.426 | 26.374 |
| 3 | PGVector ivfflat Ours (lists=999, probes=20) | 1.000 | 37956.241 | 26.346 |
| 4 | PGVector ivfflat Ours (lists=999, probes=40) | 1.000 | 38553.306 | 25.938 |

## Origin

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=999, probes=1, workers=20) | 0.609 | 1687.259 | 592.677 |
| 1 | PGVector ivfflat Ours (lists=999, probes=5, workers=20) | 0.914 | 821.231 | 1217.684 |
| 2 | PGVector ivfflat Ours (lists=999, probes=10, workers=20) | 0.966 | 444.609 | 2249.165 |
| 3 | PGVector ivfflat Ours (lists=999, probes=20, workers=20) | 0.987 | 229.676 | 4353.962 |
| 4 | PGVector ivfflat Ours (lists=999, probes=40, workers=20) | 0.996 | 115.197 | 8680.769 |



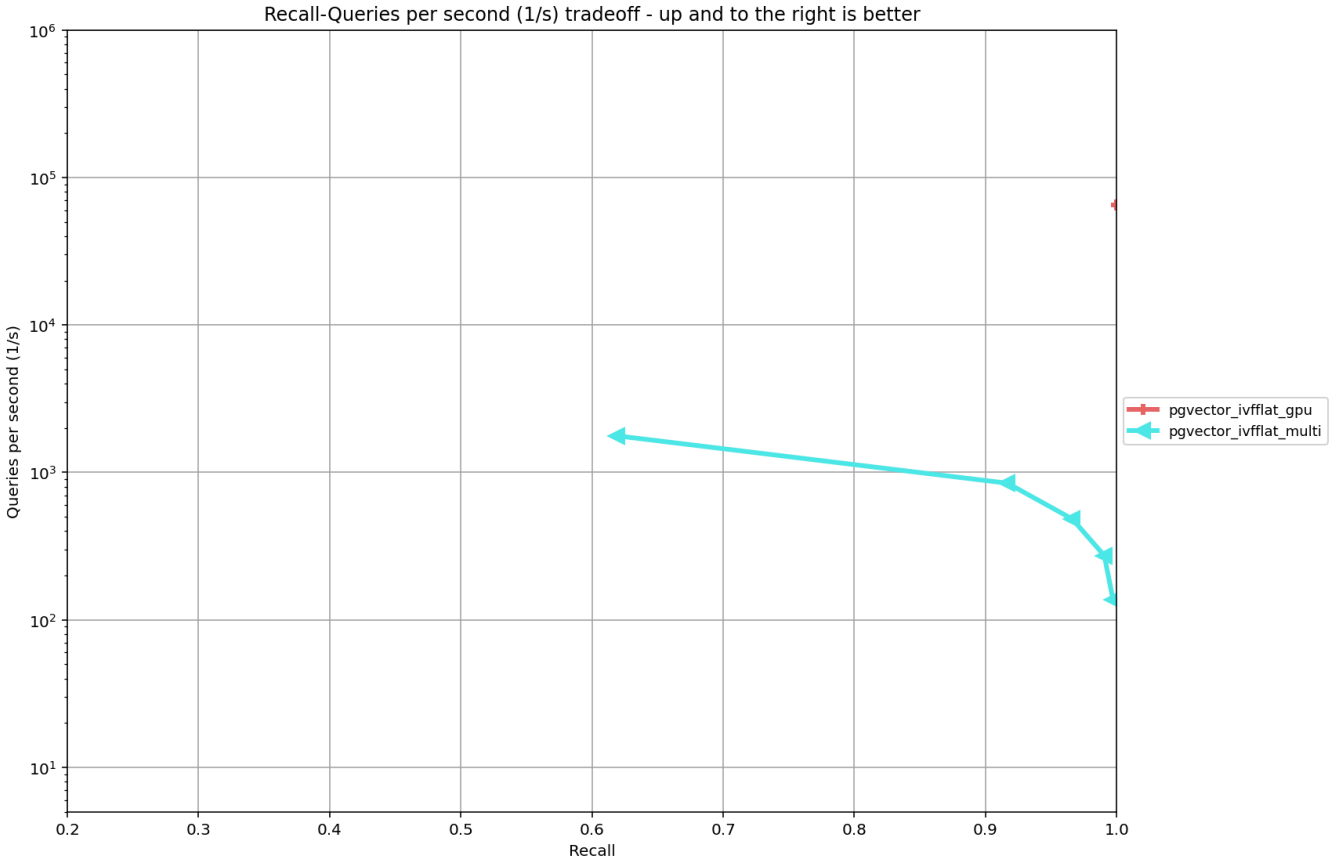

# TEXT500k-200-angular

## Ours

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=499, probes=1) | 1.000 | 64304.915 | 15.551 |
| 1 | PGVector ivfflat Ours (lists=499, probes=5) | 1.000 | 64255.337 | 15.563 |
| 2 | PGVector ivfflat Ours (lists=499, probes=10) | 1.000 | 64580.290 | 15.485 |
| 3 | PGVector ivfflat Ours (lists=499, probes=20) | 1.000 | 64662.516 | 15.465 |
| 4 | PGVector ivfflat Ours (lists=499, probes=40) | 1.000 | 64518.351 | 15.499 |

# Origin

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=499, probes=1, workers=20) | 0.619 | 1769.380 | 565.170 |
| 1 | PGVector ivfflat Ours (lists=499, probes=5, workers=20) | 0.917 | 844.587 | 1184.011 |
| 2 | PGVector ivfflat Ours (lists=499, probes=10, workers=20) | 0.966 | 482.779 | 2071.342 |
| 3 | PGVector ivfflat Ours (lists=499, probes=20, workers=20) | 0.991 | 272.388 | 3671.236 |
| 4 | PGVector ivfflat Ours (lists=499, probes=40, workers=20) | 0.997 | 137.092 | 7294.380 |



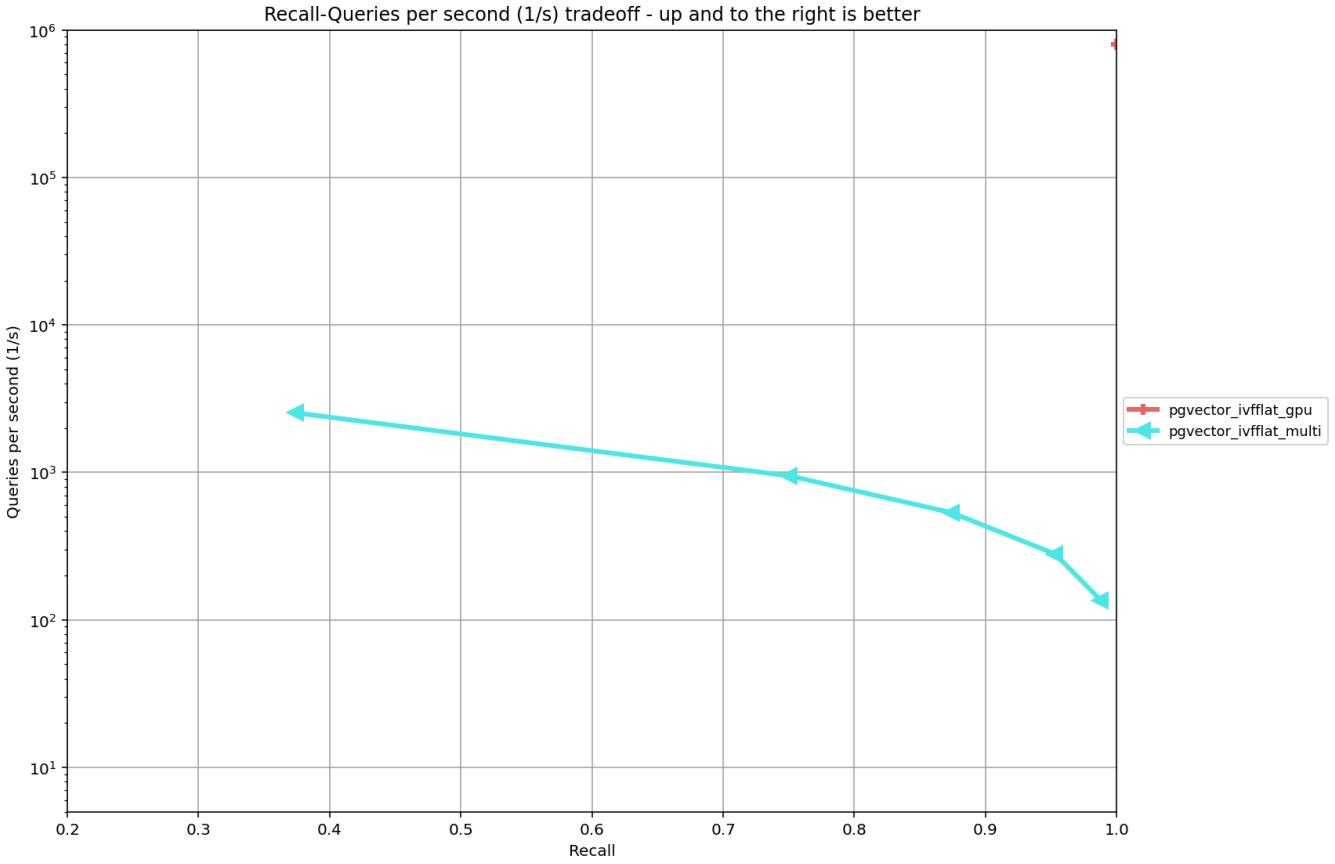
# sift-128-euclidean

## Ours

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=1000, probes=1) | 1.000 | 788733.807 | 1.268 |
| 1 | PGVector ivfflat Ours (lists=1000, probes=5) | 1.000 | 696991.488 | 1.435 |
| 2 | PGVector ivfflat Ours (lists=1000, probes=10) | 1.000 | 790450.905 | 1.265 |
| 3 | PGVector ivfflat Ours (lists=1000, probes=20) | 1.000 | 791105.170 | 1.264 |
| 4 | PGVector ivfflat Ours (lists=1000, probes=40) | 1.000 | 794673.507 | 1.258 |

## Origin

| 实验编号 | 方法描述 | Recall | QPS | DPV |
|---|---|---|---|---|
| 0 | PGVector ivfflat Ours (lists=1000, probes=1, workers=20) | 0.375 | 2531.536 | 395.017 |
| 1 | PGVector ivfflat Ours (lists=1000, probes=5, workers=20) | 0.751 | 946.861 | 1056.122 |
| 2 | PGVector ivfflat Ours (lists=1000, probes=10, workers=20) | 0.874 | 532.795 | 1876.895 |
| 3 | PGVector ivfflat Ours (lists=1000, probes=20, workers=20) | 0.953 | 279.858 | 3573.247 |
| 4 | PGVector ivfflat Ours (lists=1000, probes=40, workers=20) | 0.988 | 135.571 | 7376.235 |




# deep-image-96-angular（to-do)

## Ours


```
psycopg.errors.DiskFull: could not extend file "base/16388/83862": No space left on device
HINT:  Check free disk space.
```

pgvector_parallel

```
psycopg.ProgrammingError: cannot adapt type 'ndarray' using placeholder
'%s' (format: AUTO)
```

## Origin

pgvector_multi

```
psycopg.ProgrammingError: cannot adapt type 'ndarray' using placeholder
'%s' (format: AUTO)
```

# 内存泄露

## 1.什么是内存泄露?

内存泄漏(memory leak)是指由于疏忽或错误造成了**程序未能释放掉不再使用的内存的情况。**内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，失去了对该段内存的控制，因而造成了内存的浪费。
可以使用Valgrind, mtrace进行内存泄漏检查。

## 2.什么原因?

**程序逻辑错误，资源管理失控**

1）new/delete，new[]/delete[]没有配对使用

2）异常安全漏洞；异常改变了正常的执行流程，导致释放代码被跳过

3）当多个部分共享同一内存时，释放责任不明确

4）容器存储指针（如vector存储指针，容器清空时，指针所指向的资源没清空）

5）智能指针循环引用问题

# 函数调用分析和性能

## gperftools

### 1. 准备环境

```bash
# 1. 安装 pgvector 依赖（可选，真正落库才需要）
sudo apt install postgresql-14-pgvector

# 2. 安装 gperftools
sudo apt install -y google-perftools libgoogle-perftools-dev graphviz

# 3. 克隆 FlameGraph（后续画火焰图）
git clone https://github.com/brendangregg/FlameGraph ~/FlameGraph
```

## 2. 源码 `pgvector_demo.cpp`

```cpp
#include <gperftools/profiler.h>
#include <vector>
#include <random>
#include <algorithm>
#include <numeric>
#include <iostream>

constexpr int DIM = 1536;           // 与 OpenAI embedding 同维
constexpr int N = 200'000;          // 20 万条向量
constexpr int TOPK = 100;           // Top-100 召回

using Vec = std::vector<float>;

// L2 距离平方
float l2sq(const Vec& a, const Vec& b) {
    float sum = 0;
    for (int i = 0; i < DIM; ++i) {
        float d = a[i] - b[i];
        sum += d * d;
    }
    return sum;
}

// 模拟 pgvector 的暴力 <-> 排序 → LIMIT k
void brute_force_search(const std::vector<Vec>& base, const Vec& query)
{
    std::vector<std::pair<float, int>> buf;
    buf.reserve(base.size());
    for (size_t i = 0; i < base.size(); ++i)
        buf.emplace_back(l2sq(base[i], query), i);
```

```cpp
    std::partial_sort(buf.begin(), buf.begin() + TOPK, buf.end());
    volatile int sum = 0;   // 防止被优化掉
    for (int i = 0; i < TOPK; ++i) sum += buf[i].second;
}

int main() {
    std::mt19937 rng(42);
    std::uniform_real_distribution<float> dist(0, 1);

    std::cout << "generating " << N << " x " << DIM << "-D
vectors...\n";
    std::vector<Vec> base(N, Vec(DIM));
    for (auto& v : base)
        for (float& x : v) x = dist(rng);
    Vec query(DIM);
    for (float& x : query) x = dist(rng);

    ProfilerStart("cpu.prof");          // ← 开始采样
    brute_force_search(base, query);    // ← 热点在这里
    ProfilerStop();                     // ← 结束采样
    std::cout << "done → cpu.prof\n";
    return 0;
}
```

## 3. 编译 & 运行

```
g++ pgvector_demo.cpp -o pgvector_demo \
    -lprofiler -fno-omit-frame-pointer -O2
./pgvector_demo
```

运行结束会在当前目录得到 `cpu.prof`。

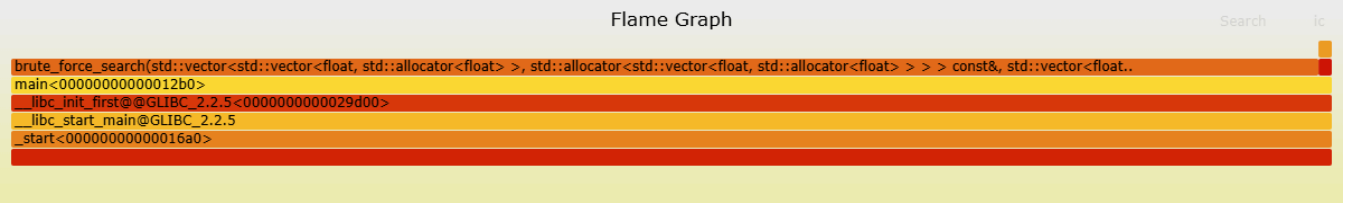
## 4. 看结果

```
# 4.1 文本热点
google-pprof --text ./pgvector_demo cpu.prof | head -20

# 4.2 火焰图（一步到位）
google-pprof --collapsed ./pgvector_demo cpu.prof | \
    ~/FlameGraph/flamegraph.pl > pgvector_demo.svg
```

用浏览器打开 `pgvector_demo.svg`，就能看到 **1536 维 L2 距离计算** 占用了绝大多数 CPU，与真实 pgvector 暴力扫描时的瓶颈完全一致。

```
Total: 99 samples
      98  99.0%  99.0%       98  99.0% brute_force_search
       1   1.0% 100.0%        1   1.0% munmap
       0   0.0% 100.0%       99 100.0% __libc_init_first@@GLIBC_2.2.5
       0   0.0% 100.0%       99 100.0% __libc_start_main@GLIBC_2.2.5
       0   0.0% 100.0%       99 100.0% _start
       0   0.0% 100.0%        1   1.0% free@@GLIBC_2.2.5
       0   0.0% 100.0%       99 100.0% main
```




# Valgrind

`Valgrind`是一套开源工具集，核心工具包括：

- **Memcheck**：检测内存管理问题（最常用）
- **Callgrind**：函数调用分析和性能 profiling
- **Cachegrind**：缓存使用分析
- **Helgrind**：检测多线程竞争问题
- **Massif**：堆内存使用分析

## 1. 安装 Valgrind 可视化工具

```
sudo apt update
sudo apt install -y valgrind kcachegrind
```

## 2. 重新编译（保留调试符号）

```
g++ pgvector_demo.cpp -o pgvector_demo \
    -g -O2 -fno-omit-frame-pointer
```

> 不需要链接 `-lprofiler`，valgrind 是插桩式，无需改源码。

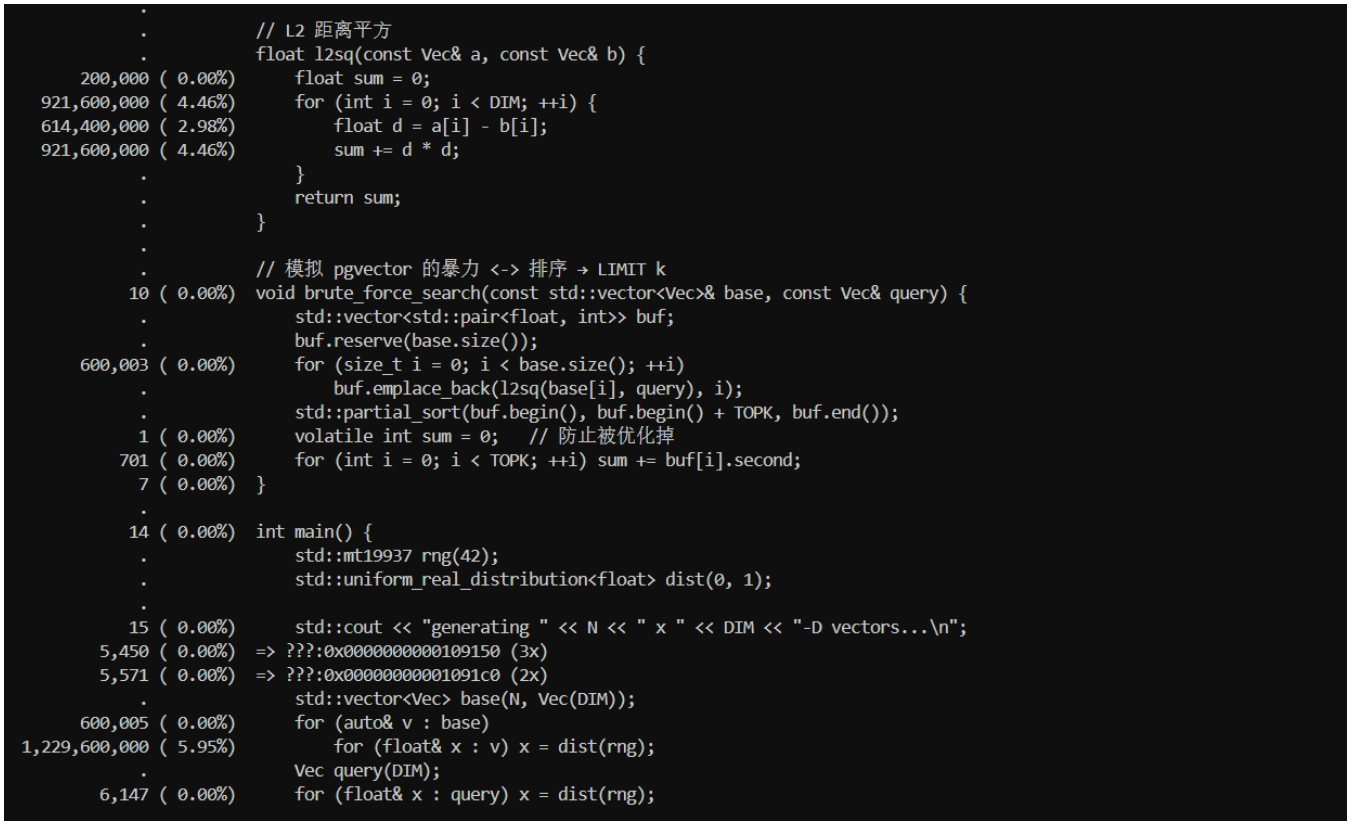
## 3. 运行 callgrind 采样

```
valgrind --tool=callgrind --callgrind-out-file=demo.callgrind
./pgvector_demo
```

跑完会在当前目录生成 `demo.callgrind`。

## 4. 文本速览

```
callgrind_annotate demo.callgrind > annotate.txt
```



```
            .
            .          // L2 距离平方
            .          float l2sq(const Vec& a, const Vec& b) {
      200,000 ( 0.00%)     float sum = 0;
  921,600,000 ( 4.46%)     for (int i = 0; i < DIM; ++i) {
  614,400,000 ( 2.98%)         float d = a[i] - b[i];
  921,600,000 ( 4.46%)         sum += d * d;
            .              }
            .              return sum;
            .          }
            .
            .          // 模拟 pgvector 的暴力 <-> 排序 → LIMIT k
           10 ( 0.00%) void brute_force_search(const std::vector<Vec>& base, const Vec& query) {
            .              std::vector<std::pair<float, int>> buf;
            .              buf.reserve(base.size());
      600,003 ( 0.00%)     for (size_t i = 0; i < base.size(); ++i)
            .                  buf.emplace_back(l2sq(base[i], query), i);
            .              std::partial_sort(buf.begin(), buf.begin() + TOPK, buf.end());
            1 ( 0.00%)     volatile int sum = 0;   // 防止被优化掉
          701 ( 0.00%)     for (int i = 0; i < TOPK; ++i) sum += buf[i].second;
            7 ( 0.00%) }
            .
           14 ( 0.00%) int main() {
            .              std::mt19937 rng(42);
            .              std::uniform_real_distribution<float> dist(0, 1);
            .
           15 ( 0.00%)     std::cout << "generating " << N << " x " << DIM << "-D vectors...\n";
        5,450 ( 0.00%) => ???:0x0000000000109150 (3x)
        5,571 ( 0.00%) => ???:0x00000000001091c0 (2x)
            .              std::vector<Vec> base(N, Vec(DIM));
      600,005 ( 0.00%)     for (auto& v : base)
1,229,600,000 ( 5.95%)         for (float& x : v) x = dist(rng);
            .              Vec query(DIM);
        6,147 ( 0.00%)     for (float& x : query) x = dist(rng);
```

| 工具 | 原理 | slowdown | 精度 | 适合场景 |
| --- | --- | --- | --- | --- |
| **gperftools** | 100 Hz 采样 | ≈ 0 % | 函数级 | 线上/生产压测 |
| **callgrind** | 指令级插桩 | 10–50× | 行级 | 线下深度优化 |