

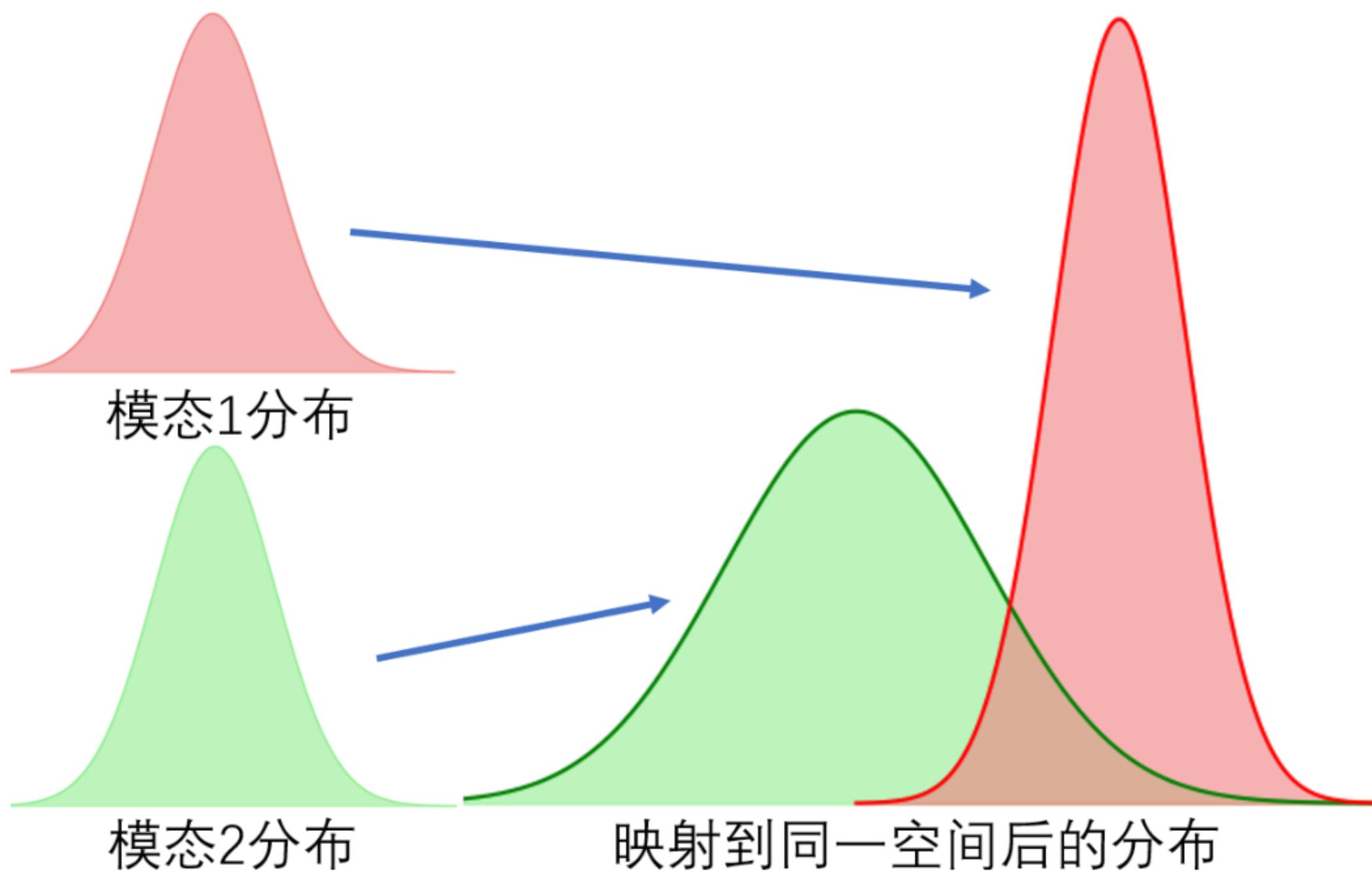
# 1. 导论

## 1.1. 研究背景

### 1 跨模态检索:

- 含义: 使用某个模态的数据作为query, 返回另一个模态中语义相似的内容
- 示例: 输入 "Apple" 后, 返回苹果的照片

### 2 模态差距(gap): 不同模态数据即使映射到同一语义空间(比如用CLIP), 其分布特征仍差距显著



### 3 两种ANN

- 单模态ANN: 查询向量分布 $\xleftrightarrow{\text{ID}}$ 基础数据分布, 即查询来源于与数据库数据相同的分布
- 跨模态ANN: 查询向量分布 $\xleftrightarrow{\text{OOD}}$ 基础数据分布, 即查询来源于与数据库数据不同的分布

## 1.2. 本文的研究

### 1 研究动机: 当前SOTA的ANN都是单模态的, 在OOD负载上表现差

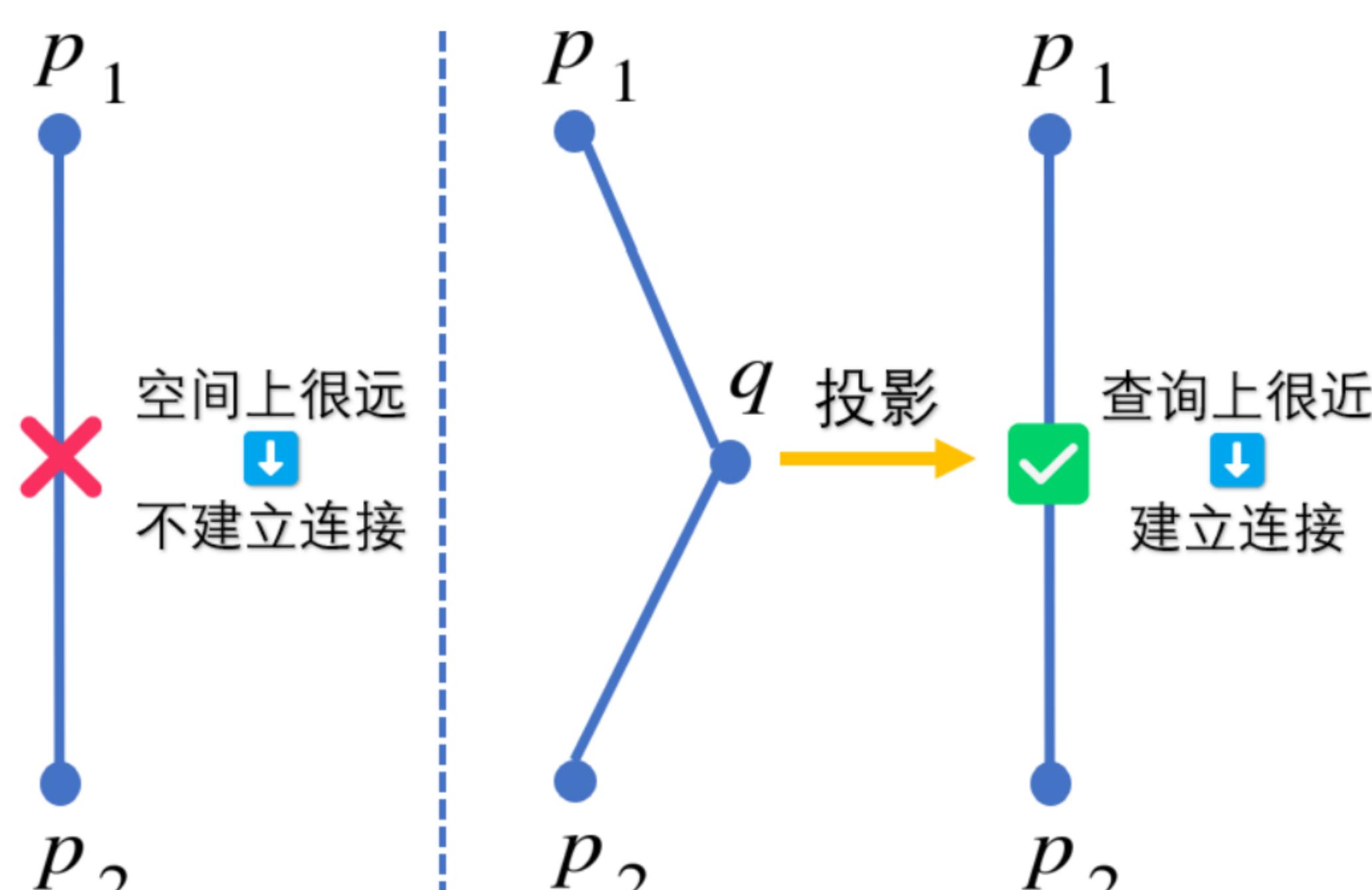
### 2 研究内容

- OOD工作负载分析: 跨模态后性能下降, 源于查询过远+标签分散 $\rightarrow$ 收敛变慢/跳数增加

类型	查询 $\xleftrightarrow{\text{距离}}$ 基础数据	查询最邻近 $i$ $\xleftrightarrow{\text{距离}}$ 查询最邻近	查询 $\xleftrightarrow{\text{分布}}$ 基础数据
单模态ANN	近(基本假设)	近(基本假设)	ID
跨模态ANN	远(实验得到)	远(实验得到)	OOD

- RoarGraph的提出:

- 原理: 让查询参与图构建 $\rightarrow$ 将[查询点 $\leftrightarrow$ 基础点]邻接关系投影到基础点 $\rightarrow$ 形成仅有基础点的图
- 意义: 让空间上很远但是查询上很近的点相连, 从而能高效处理OOD-ANNS



- 效果：在跨模态数据集上实现了QPS和Recall指标的提升

## 1.3. 有关工作

方法	核心思想	优缺点
束搜索终止	利用查询训练分类模型判断何时终止搜索	提升效率，但训练成本较高
图卷积(GCN)	引入GCN学习最优搜索路径	路径优化明显，但训练成本较高
GCN+RL	强化学习与GCN结合引导搜索路由	提升效果显著，但训练成本较高
GraSP	概率模型与子图采样学习边重要性	性能优化明显，但索引构建成本高
ScaNN	结合向量量化和PQ进行分区与压缩	压缩与搜索性能高效，但依赖调参

# 2. 对OOD负载的分析与验证

## 2.1. 初步的背景及其验证

### 2.1.1. 对模态差距的验证

#### ① OOD的量化

距离类型	衡量什么	如何理解
Wasserstein距离	两个分布间的差异	把一个分布搬到另一个的最小代价
Mahalanobis距离	一个向量到一个分布的距离	一个点相对于一个分布的异常程度

#### ① 实验1：用Wasserstein距离衡量OOD特性

- 数据集：基础数据集中抽取的无交叉集 $B_1/B_2$ , OOD的查询集 $Q$
- 结果：Wasserstein( $B_1, Q$ )和Wasserstein( $B_2, Q$ ), 大致是Wasserstein( $B_1, B_2$ )两倍

#### ② 实验2：用Mahalanobis距离衡量OOD特性

- 数据集：满足分布 $P$ 的基础数据，来自ID查询集的 $q_{id}$ ，来自OOD查询集的 $q_{ood}$
- 结果：Mahalanobis( $q_{id}, P$ )<Mahalanobis( $q_{ood}, P$ )

### 2.1.2. SOTA-ANN在OOD任务上的表现

#### ① 对传统的SOTA-ANN

索引方法	在OOD上的表现(相比在ID上)
HNSW	性能显著下降，在BeamSearch过程显著访问更多的结点(要经历更多跳)
IVF-PQ	性能显著下降，需要更多的聚类数才能达到相同的Recall

#### ② 对改进的ANN：针对OOD-ANNS的首个图索引RobustVamana(OOD-DiskANN)

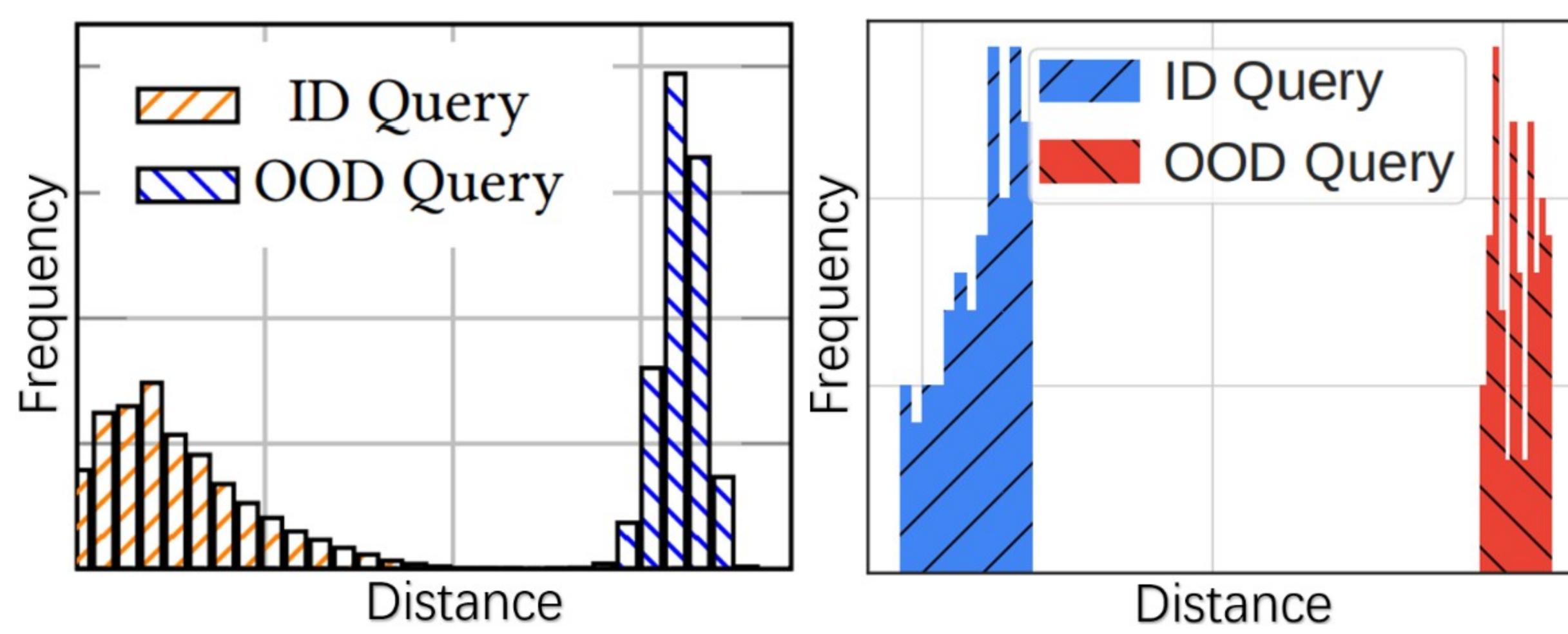
- 原理：先用Vamana建图，然后再用RobustStitch根据查询向量，连接新的边
- 性能：比DiskANN在OOD任务上提升了40%性能，但是查询速度慢了×4-10

## 2.2. 对OOD上ANN工作负载的分析

## 2.2.1. OOD-ANNS和ID-ANNS的两个差异

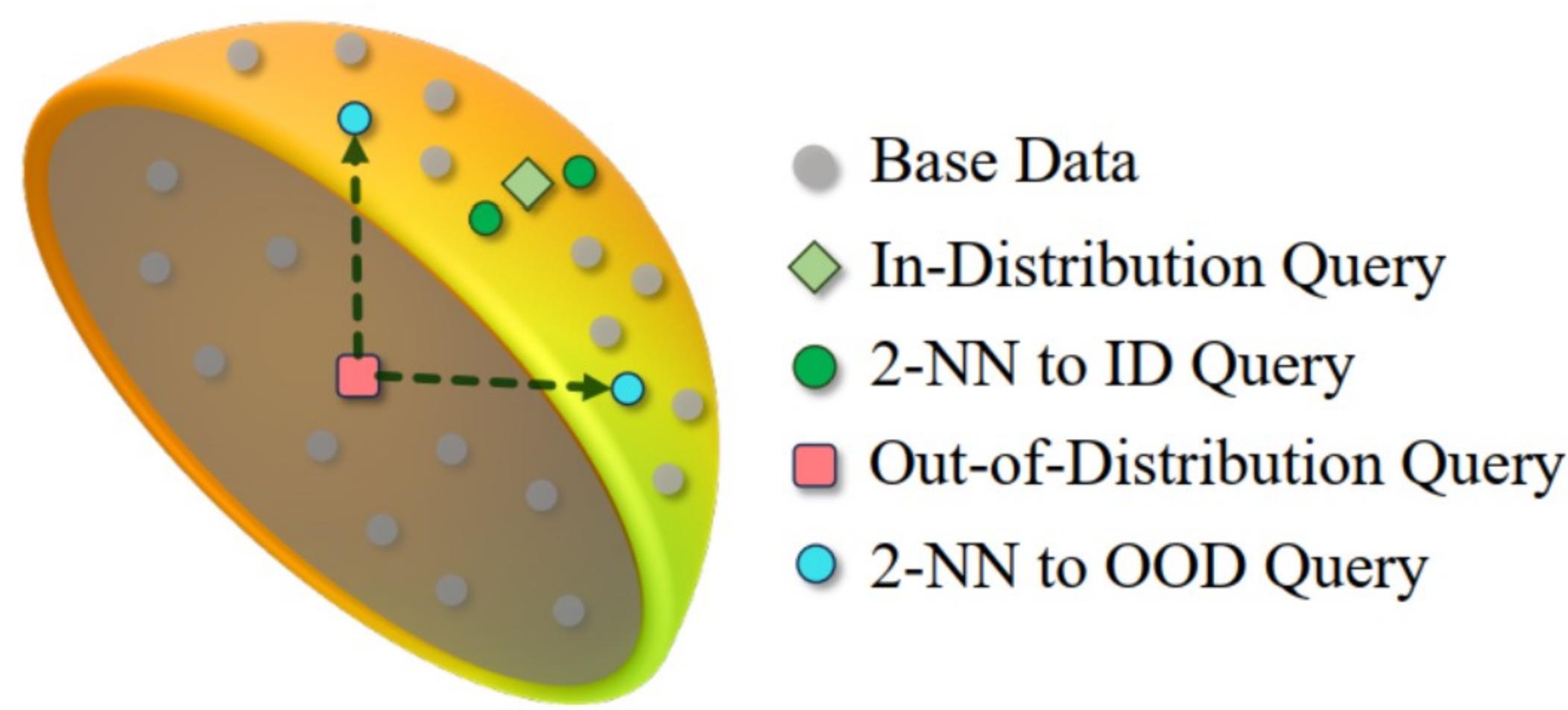
### 1 两种差异及实验结果

1. OOD查询离其最邻近很远：即 $\delta(q_{\text{ood}}, i^{\text{th}}\text{-NN}_{\text{ood}}) \gg \delta(q_{\text{id}}, i^{\text{th}}\text{-NN}_{\text{id}})$ , 左为*i*=1时的分布结果
2. OOD查询的最邻近彼此原理：100<sup>th</sup>-NN互相之间的平均距离，实验结果如右

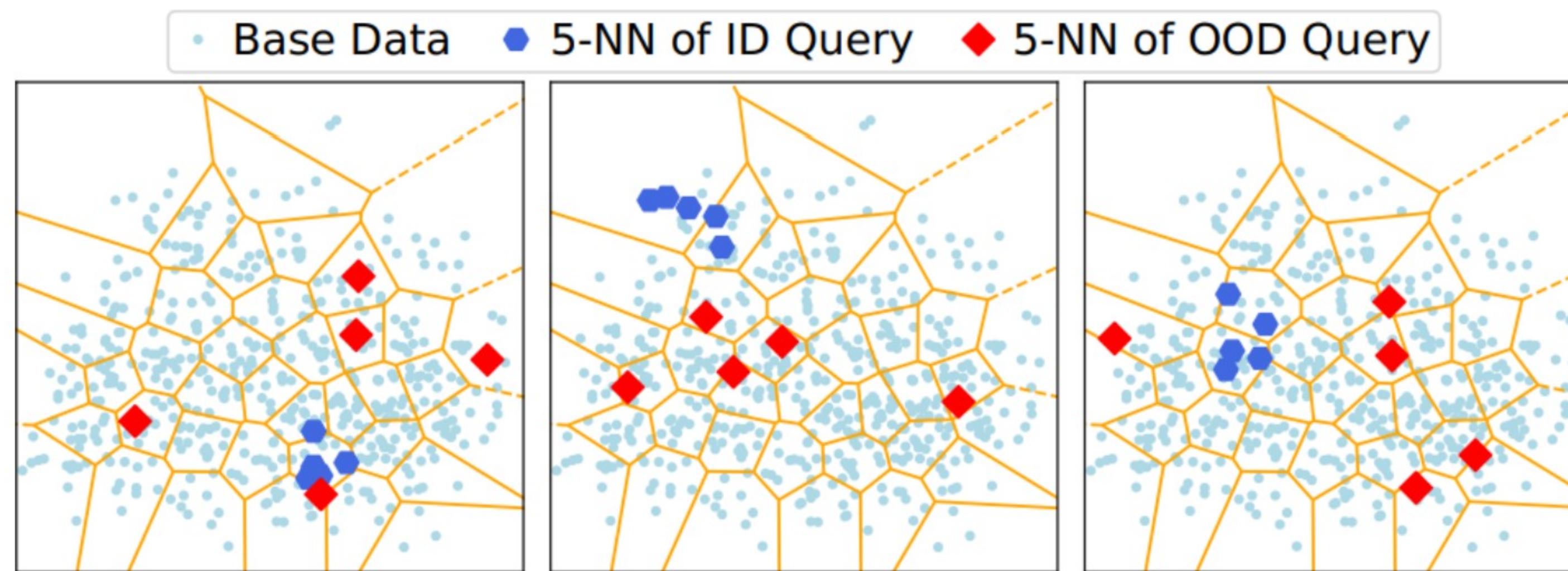


### 2 对差异的直观理解

1. 简单(概念)示例：



- ID查询：查询与其最邻近在球面上，相互靠近
  - OOD查询：查询在球心，其最邻近在球面上(由此距离较远且查询不多+分散分布)
2. 真实示例：真实数据PCA降到二维的视图，ID查询更为集中



## 2.2.2. 为何传统SOTA-ANN在ODD表现不佳

## 0 传统ANN的设计

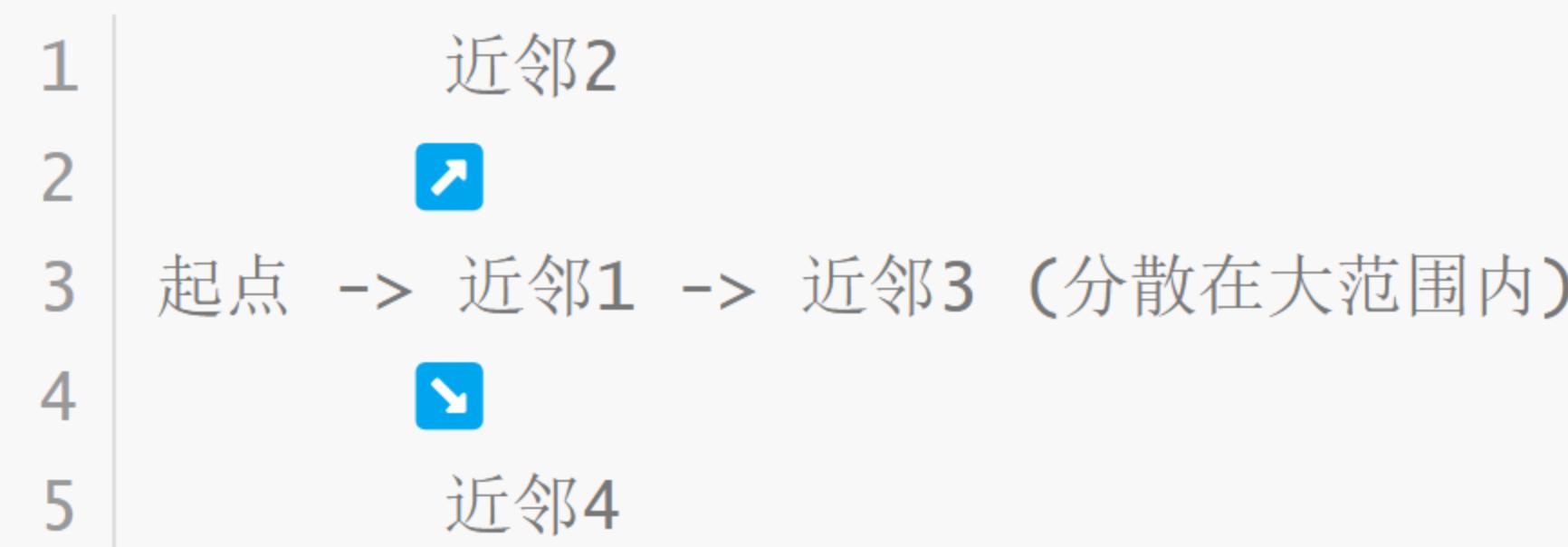
1. 基于两假设：查询/数据同分布 +  $k$ 个最近邻彼此靠近(邻居的邻居是邻居)，刚好全反的
2. 设计的思路：
  - 建图：用BeamSearch来构建KNN图→空间中相近的点转化为图中紧密连接的结点
  - 搜索：从中心点开始GreedySearch

### 1 在基于图ANN上：OOD会使得搜索空间增大

1. 可识别搜索空间：包围当前访问结点 $x$ 的 $B^s(x) + B^k(1^{\text{st}}\text{-NN}, R)$ 
  - 球 $B^k(1^{\text{st}}\text{-NN}, R)$ ：以 $1^{\text{st}}$ -NN为球心， $k$ 邻近间互相距离 $\delta(i^{\text{th}}\text{-NN}, j^{\text{th}}\text{-NN})$ 最大值为半径
  - 球 $B^s(x)$ ：以当前结点 $x$ 为圆心，以 $\delta(x, i^{\text{th}}\text{-NN})$ 的最大值(到最远最邻近的距离)为半径
2. OOD的影响：搜索空间大幅增大
  - 对 $B^k$ ：由于OOD的性质 $R_{\text{ood}} \gg R_{\text{id}}$ ，这一差异在体积层面放大到 $\left(\frac{R_{\text{ood}}}{R_{\text{id}}}\right)^D$ 级别
  - 对 $B^s$ ：由于OOD的性质 $\delta(x, i^{\text{th}}\text{-NN}_{\text{ood}}) \gg \delta(x, i^{\text{th}}\text{-NN}_{\text{id}})$ ，使得体积也大幅膨胀
3. 对搜索过程的影响：
  - 对于ID查询：由于最近邻彼此靠近，GreedySearch可以使 $B^s(x)$ 轻松收敛
  - 对于OOD查询：最近邻方向分散难以收敛，需要更大的Beam宽度/搜索路径等

1 | 起点 -> 近邻1 -> 近邻2 -> 近邻3 (一个小范围内)

◦ 对于OOD查询：最近邻方向分散难以收敛，需要更大的Beam宽度/搜索路径等



### 2 在基于划分IVF上

1. 原理上：IVF先将原数据分簇
  - ID查询：最邻近集中在少数几个相邻簇中
  - OOD查询：最邻近分散在多个不相邻簇中
2. 实验上：OOD查询需要扫描更多的簇，性能下降2.5倍

## 3. RoarGraph

## 3.1. RoarGraph的设计思路

### 1 面向解决三种挑战

1. 边的建立：如何连接查询/基础两类结点，同时避免基础结点度数太高
2. 搜索效率：查询结点要保持极高出度以覆盖基础节点，但同时也会大幅增加跳数/内存开销
3. 连通性：避免出现孤立结点，独立子图

### 1 大致的设计流程

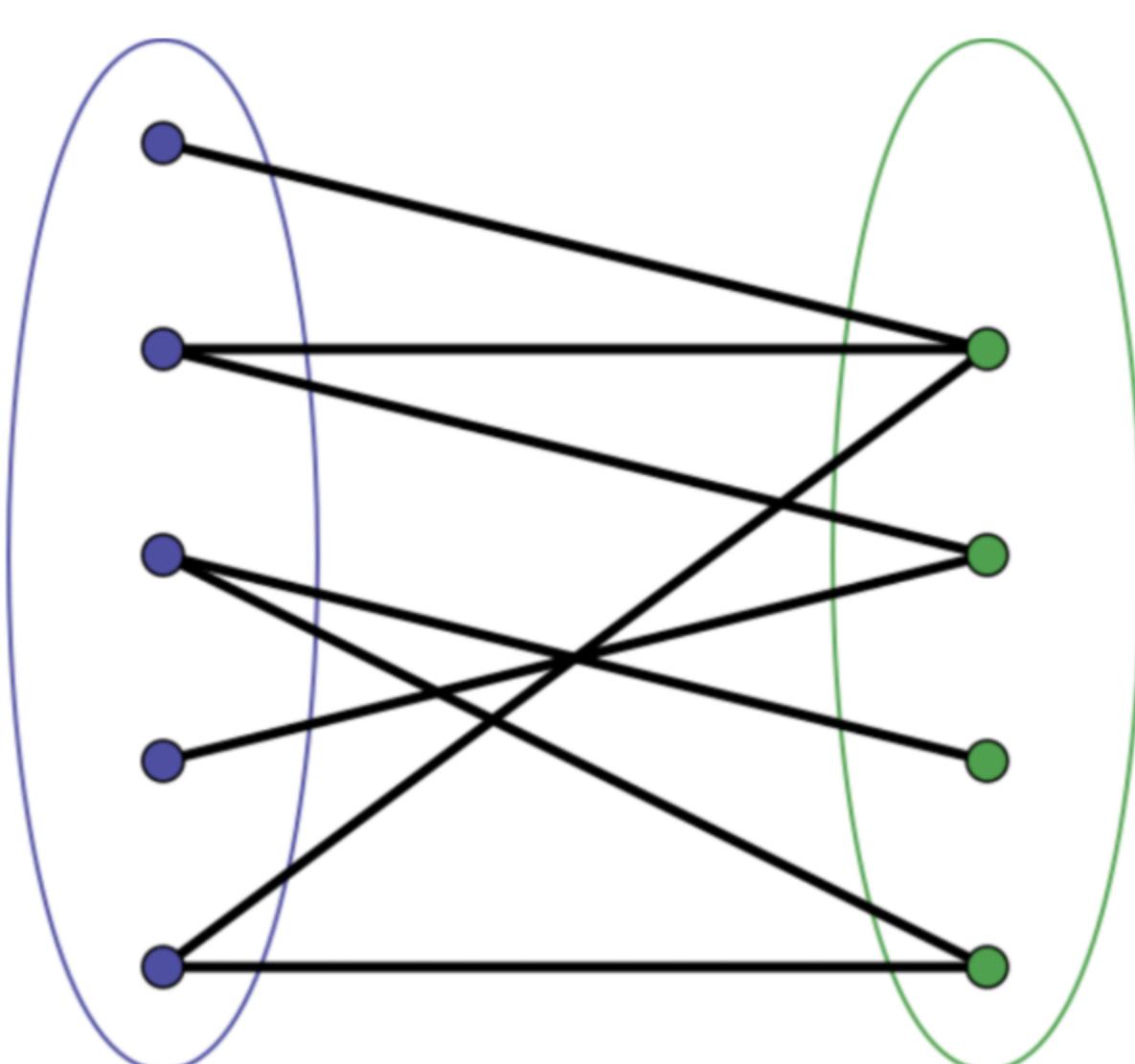
1. 构建：建立查询 $\leftrightarrow$ 基础二分图 $\rightarrow$ 将邻接信息投影到基础点中 $\rightarrow$ 增强连接
2. 查询：同样是用BeamSearch

## 3.2. RoarGraph的构建: 三个阶段

### 3.2.1. 阶段1: 查询 $\leftrightarrow$ 基础二分图构建

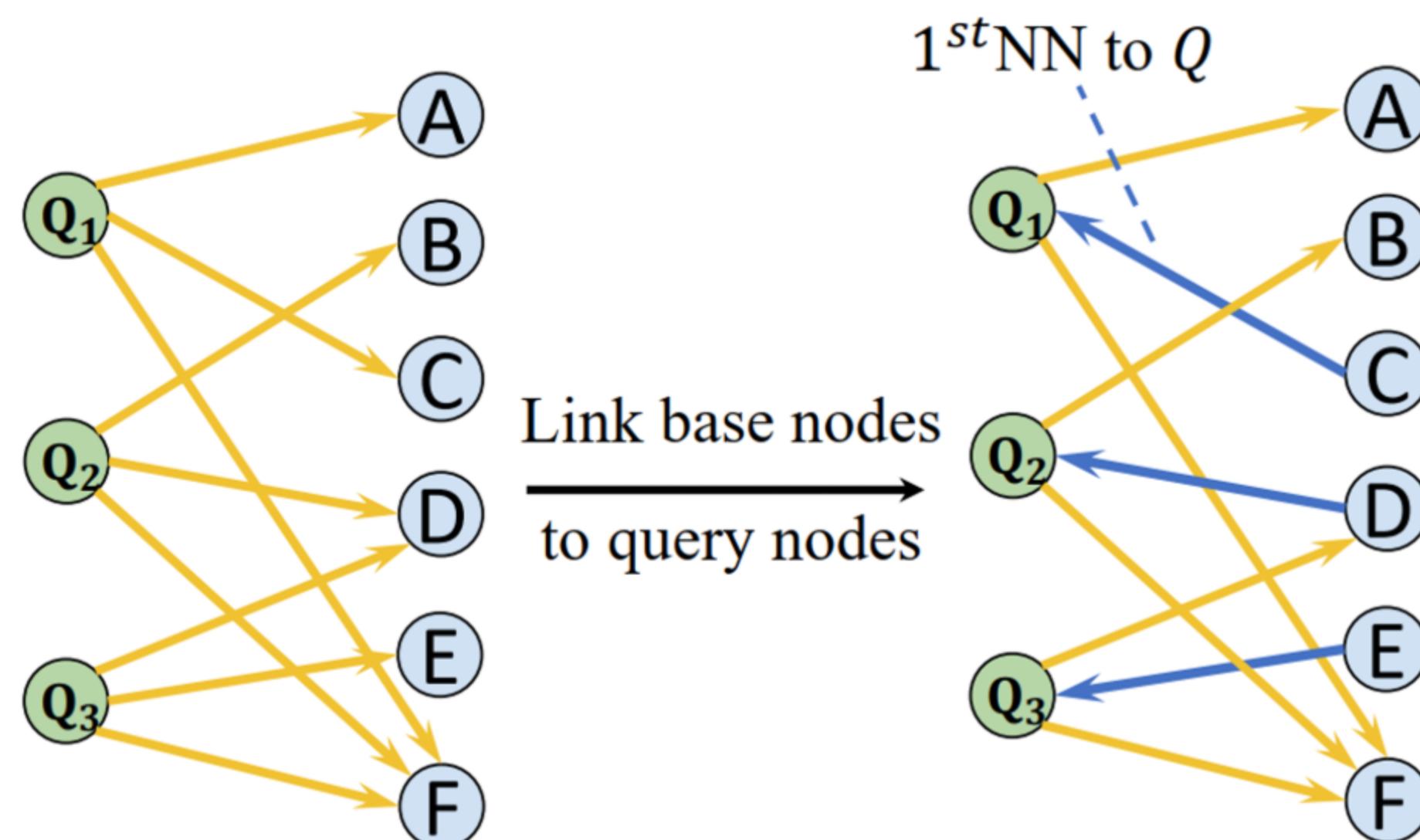
#### 1 二分图概述：

1. 基本概念：将所有的点分为两个集合，所有边必须连接不同子集的点，不能内部连接



2. 在此处：两子集查询结点+基础节点，两种边[查询结点 $\rightarrow$ 基础结点]+[查询结点 $\leftarrow$ 基础结点]

#### 2 构建过程概述



1. 预处理：计算每个查询向量的真实 $N_q$ -NN标签

2. 边构建：

方向	操作
查询点 $\rightarrow$ 基础点	查询点 $\xrightarrow{\text{连接}}$ 查询点的 $N_q$ -NN基础点
基础点 $\rightarrow$ 查询点	查询点 $\xleftarrow{\text{连接}}$ 查询点的1-NN基础点，查询点 $\xrightarrow{\text{断连}}$ 查询点的1-NN基础点

3. 示例：

- 1 预处理：T1  $\rightarrow$  X1, X2, X3 ( $Nq=3$ )
- 2 边构建：T1  $\rightarrow$  X2, X3
- 3 T1  $\leftarrow$  X1

## 2 构建过程分析

### 1. 结点度数的考量:

- 高查询结点出度: 提高 $N_q$ 值, 增加[基础点——>查询点], 使多基础点可由同一查询点联系  
重叠性  
覆盖性
- 低基础节点出度: 为了解决上述挑战1, 目的在于提高二分图上的搜索效率

### 2. 边方向的考虑: 不进行双向连接, 避免二分图搜索时要去检查邻居的邻居( $N_q^2$ )

```
1 | 预处理: T1 -> X1, X2, X3 (Nq=3)
2 | 边构建: T1 -> X1, X2, X3
3 |         T1 <- X1
4 |         T1 <- X2
5 |         T1 <- X3
```

## 3.2.2. 阶段2: 领域感知投影

### 1 一些分析

1. 优化动机: 二分图内存消耗高(额外存储了查询节点), 搜索路径长(需要额外经过查询结点)
2. 关于投影:
  - 目的: 移除二分图中的查询结点, 并保留从查询分布获得的邻近关系
  - 方式: 最简单的可将查询点所连的全部基础点全连接(度数太高), 优化方法如领域感知投影

### 2 投影过程:

#### 1. 预处理:

- 遍历查询点: 获得与查询点相连的最邻近基础点

```
1 | 查询Q -> {B1, B2, B3, B4, B5} (Q连接了5个基础节点)
```

- 选择中心点: 即查询点的1-NN点, 作为Pivot

```
1 | 查询Q -> {B1, B2, B3, B4, B5} (Q连接了5个基础节点)
2 |          ↗
3 |          pivot
```

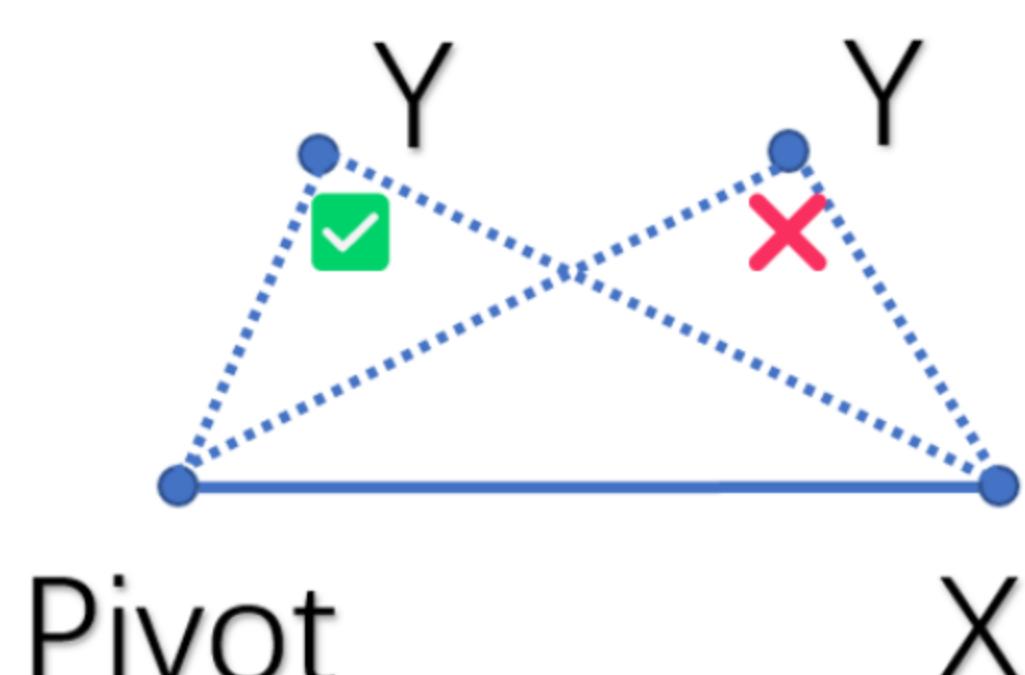
- 排序基础结点: 将余下 $N_q$ -NN点, 按与Pivot的距离排序

#### 2. 感知投影:

- 连接: 让中心点与余下点建立连接

```
1 | B1 -> B2 (最近)
2 | B1 -> B3 (次近)
3 | B1 -> B4 (较远)
4 | B1 -> B5 (最远)
```

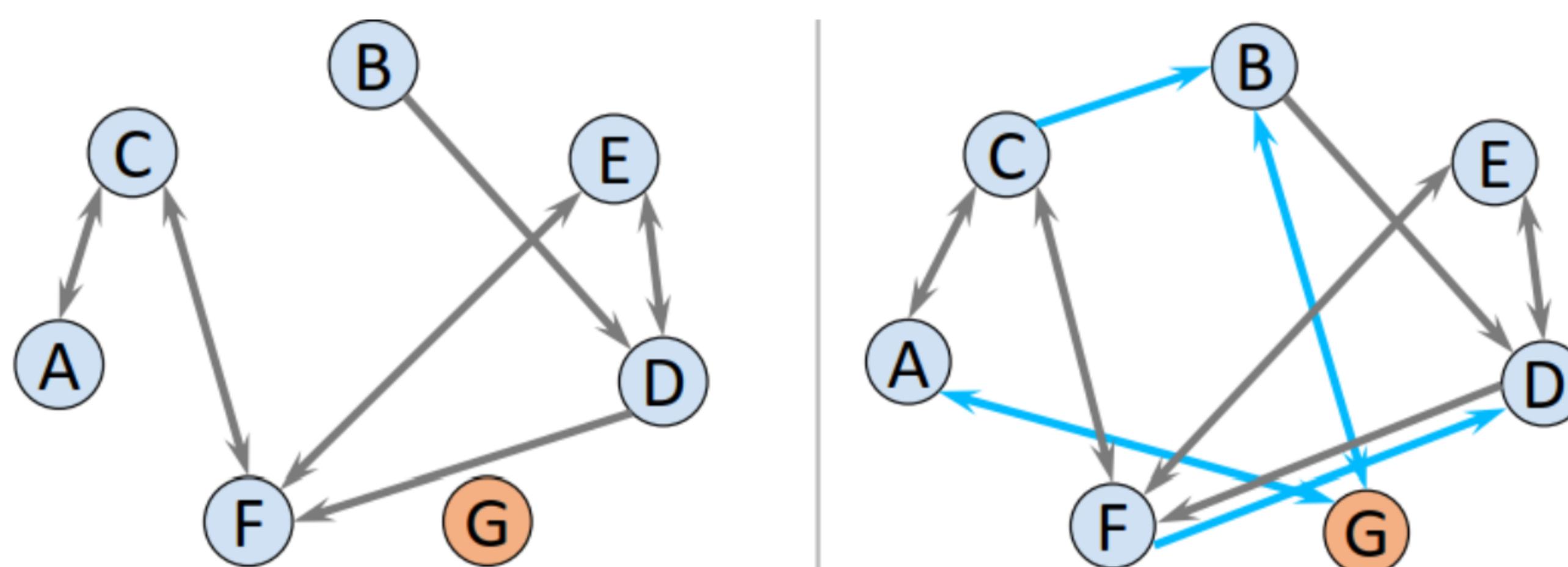
- 过滤: 保证与Pivot连接方向的多样性



条件	含义	操作
$\text{Dist}(X, Y) < \text{Dist}(\text{Pivot}, Y)$	该方向已有连接	则筛掉Y(不与Pivot建立连接)
$\text{Dist}(X, Y) > \text{Dist}(\text{Pivot}, Y)$	代表新的搜索方向	则保留Y(可与Pivot建立连接)

- 填充：当Pivot的出度小于度数限制，则又重新连接之前过滤掉的结点

### 3.2.3. 连通性增强



① 为什么要增强：仅依赖于二分图的覆盖范围，投影图的连通性还太低，对GreedySearch不友好

② 增强的方法：

- 检索：从基础集的Medoid开始，对每个基础点执行BeamSearch得到最邻近(作为候选点)
- 连边：在不超过度数限制的前提下，让该基础点连接一定数量的候选点作

## 3.3. RoarGraph性能的验证

### 3.3.1. 实验设置

① 数据集

数据集	描述	查询集	索引集
Text-to-Image	流行基准数据集，含图像和文本查询向量	官方1w条	余下不重叠数据
LAION	数百万对图像—替代文本对	采样1w条	余下不重叠数据
WebVid	素材网站获取的字幕和视频对	采样1w条	余下不重叠数据

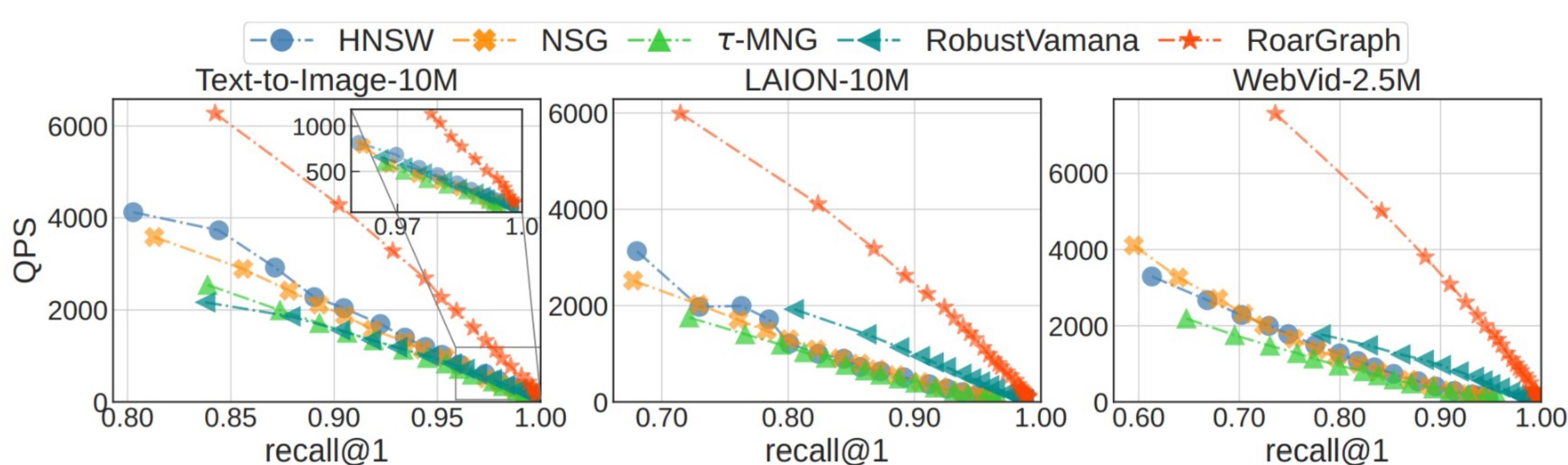
② 超参数设置

模型	超参数列表
HNSW	$M=32$ , efConstruction=500
NSG	$R=64$ , $C=L=500$
$\tau$ -MNG	$R=64$ , $C=L=500$ , $\tau=0.01$
RobustVamana	$R=64$ , $L=500$ , $\alpha=1.0$
RoarGraph	$N_q=100$ (最近邻候选数量), $M=35$ (出度约束), $L=500$ (候选集大小)

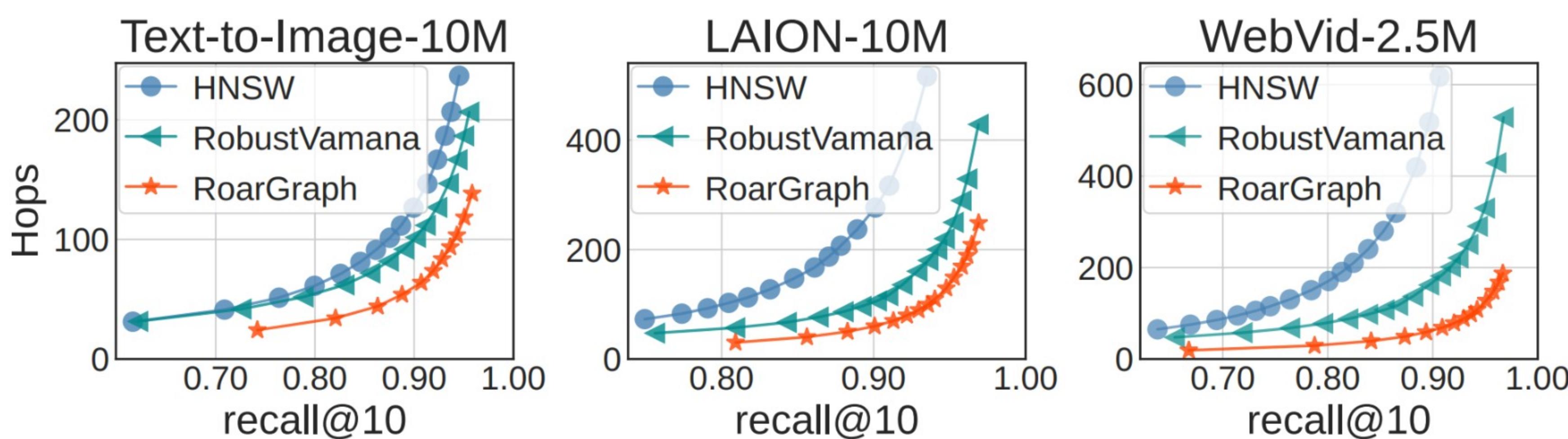
③ 性能指标：Recall@k和QPS(检索速度)

### 3.3.2. 实验结果

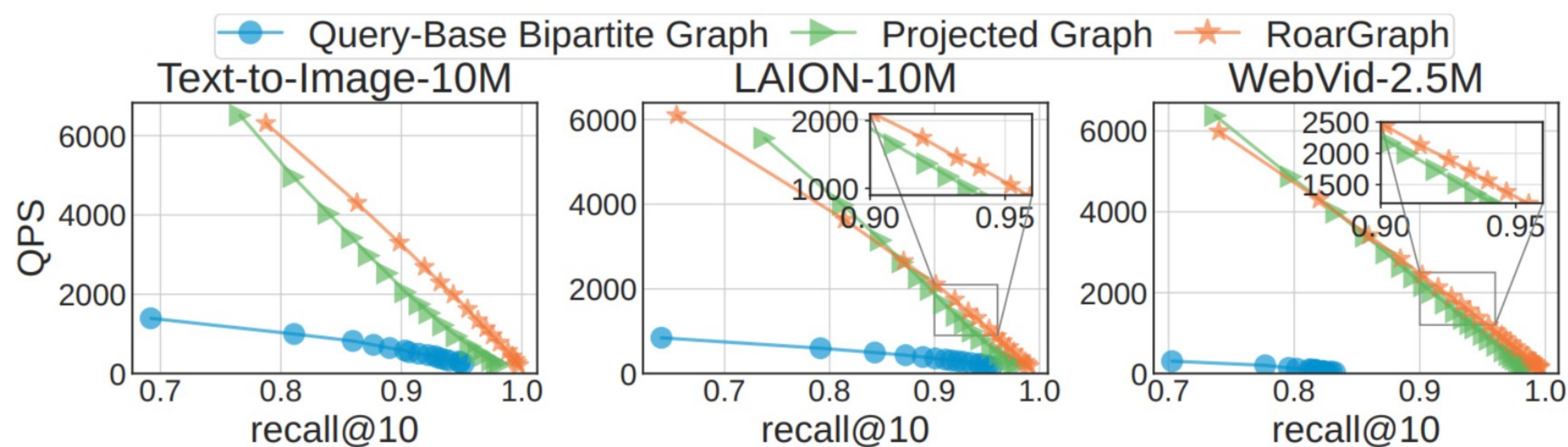
**1 QPS与召回：RoarGraph最优(超过RobustVamana)，HNSW/NSG差不多， $\tau$ -MNG最差**



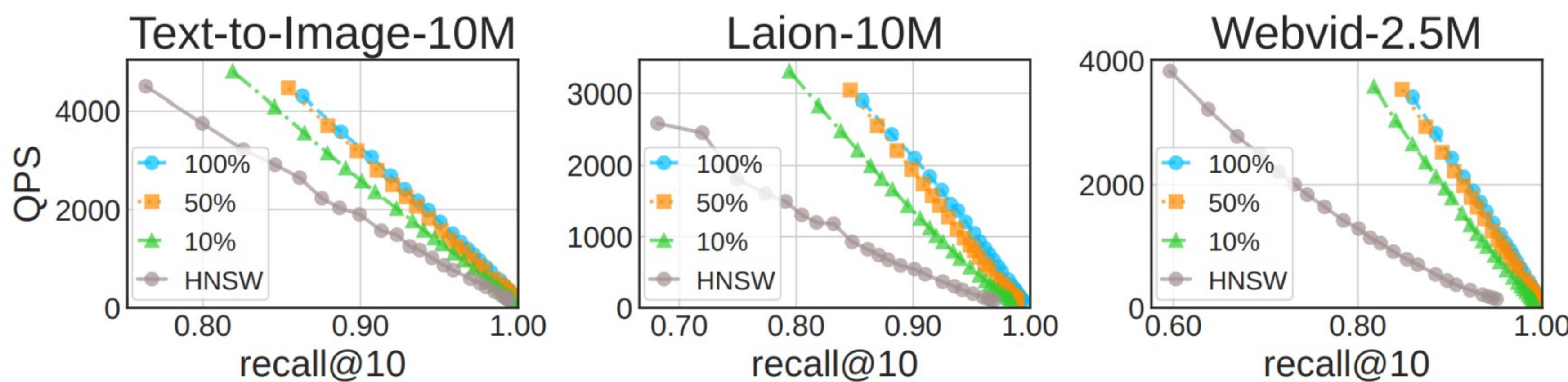
**2 跳数与召回：RoarGraph跳数显著减少，且随Recall@的k增大，减少趋势下降**



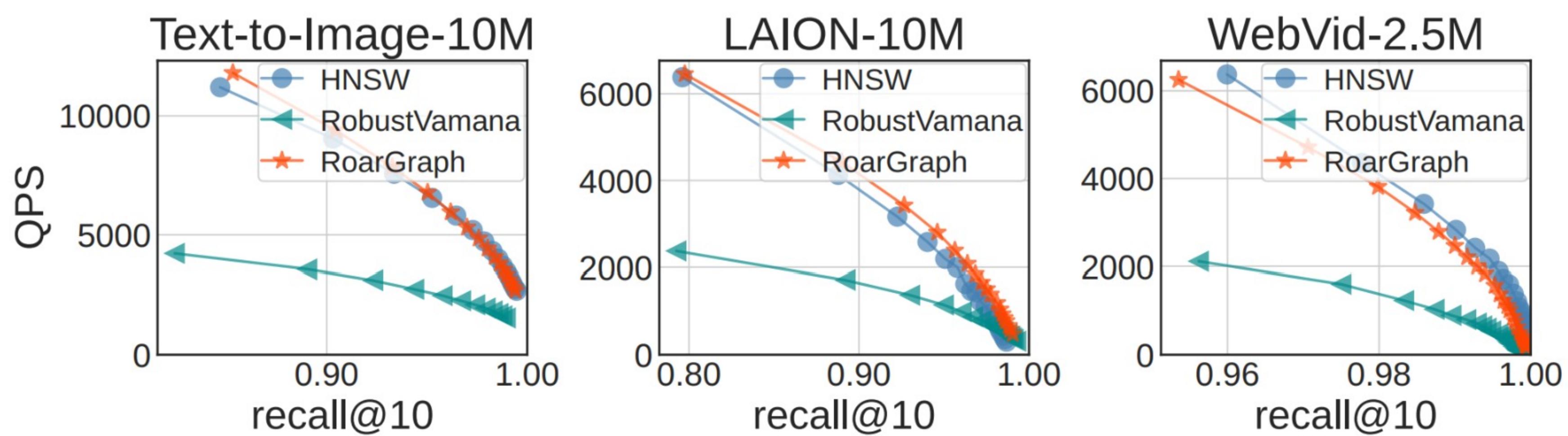
**3 消融实验：对比了二分图/投影图/完整图，可见通过邻域感知投影显著提升性能**



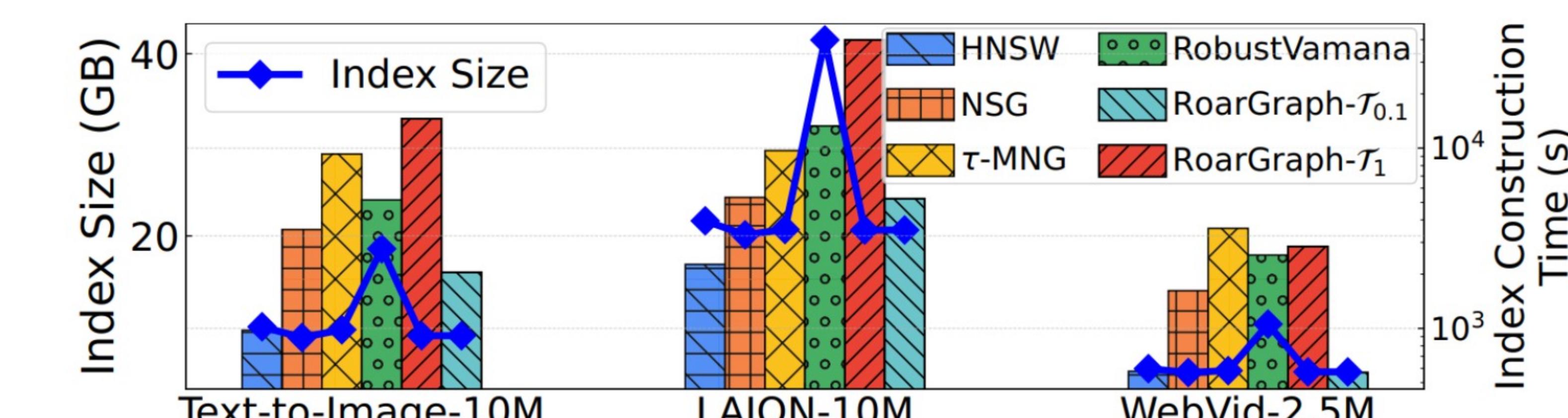
**4 查询集规模：即查询集大小占基础集大小比重对索引性能的影响；可见起始模型对规模并不敏感**



**5 在ID负载上的性能：RoarGraph依旧能打，和HNSW相当**



**6 索引开销成本：使用10%数据可大幅降低构建成本，同时保持搜索性能**



## 3.4. RoarGraph的一些讨论

1 运用场景：结合大量历史查询数据，用多模态深度学习模型生成嵌入，部署在大型检索/推荐系统

2 更新机制：

1. 初始搜索：

- 结点查询：将新插入下新基础节点 $v$ 作为查询，在基础数据集中搜索其最邻近
- 结点筛选：要求最邻近满足，曾在图构建过程中与**至少一个查询点**连接过的基础点
- 反向回溯：对该最邻近点，回溯到与其曾建立过连接的距离最近的查询点 $q$

2. 子图构建：

- 二分子图：将 $q \leftrightarrow N_{\text{out}} \cup v$ 整合为二分子图
- 邻域投影：将 $v$ 作为Pivot按同样的方式，生成投影图

3 删除操作：采用墓碑标记法Tombstones，即被删结点任参与路由，但排除在搜索结果中