# Сравнение методов ANN (LoRANN, IVF-PQ, ScaNN) на датасете GloVe


- Николай Слободчук,
- Александр Гаврись,
- Вадим Смирнов,
- Андрей Юганов.

# Мотивация: Approximate Nearest Neighbors (ANN)

**Задача.** Для запроса $q \in \mathbb{R}^d$ найти ближайшие векторы в базе $X = \{x_i\}_{i=1}^N$.

**Почему точный поиск не подходит.**

- Современные базы: $N \gg 10^6$, размерность $d = 100$–$1000$.
- Точный kNN: $O(N \cdot d)$ операций на запрос.
- Слишком медленно для real-time сценариев.

**Идея ANN (INN).**

- Допускаем небольшую аппроксимацию.
- Получаем ускорение в разы/десятки раз.

**Основные подходы ANN.**

- Графовые методы (HNSW).
- Деревья / partition-based методы.
- Кластеризация и индексы: **IVF**.

**В этой работе.**

- **LoRANN** метод: Сравнение с методом **IVF-PQ** и **ScaNN** .

# IVF-PQ: идея метода

**Задача ANN.** Для запроса $q \in \mathbb{R}^d$ найти ближайшие векторы в большой базе $X = \{x_i\}_{i=1}^N$.

**Нормировка.** Векторы базы и запросы приводятся к единичной длине:

$$\tilde{x} = \frac{x}{\|x\|_2}, \qquad \tilde{q} = \frac{q}{\|q\|_2}.$$

В этом случае евклидово расстояние эквивалентно косинусной близости.

**Основная идея IVF-PQ**

- **IVF (Inverted File):** разбиваем базу на кластеры и ищем только в нескольких подходящих.
- **PQ (Product Quantization):** храним векторы в сжатом виде, чтобы быстро считать приближённые расстояния.

**Цель:** сократить число проверяемых векторов и ускорить поиск при допустимой потере точности.

# IVF-PQ: алгоритм поиска

**fit**

- База разбивается на $L = n_{\text{list}}$ кластеров (k-means).
- Каждый вектор сохраняется в своём кластере — **inverted list**.
- Векторы внутри кластеров сжимаются с помощью Product Quantization.

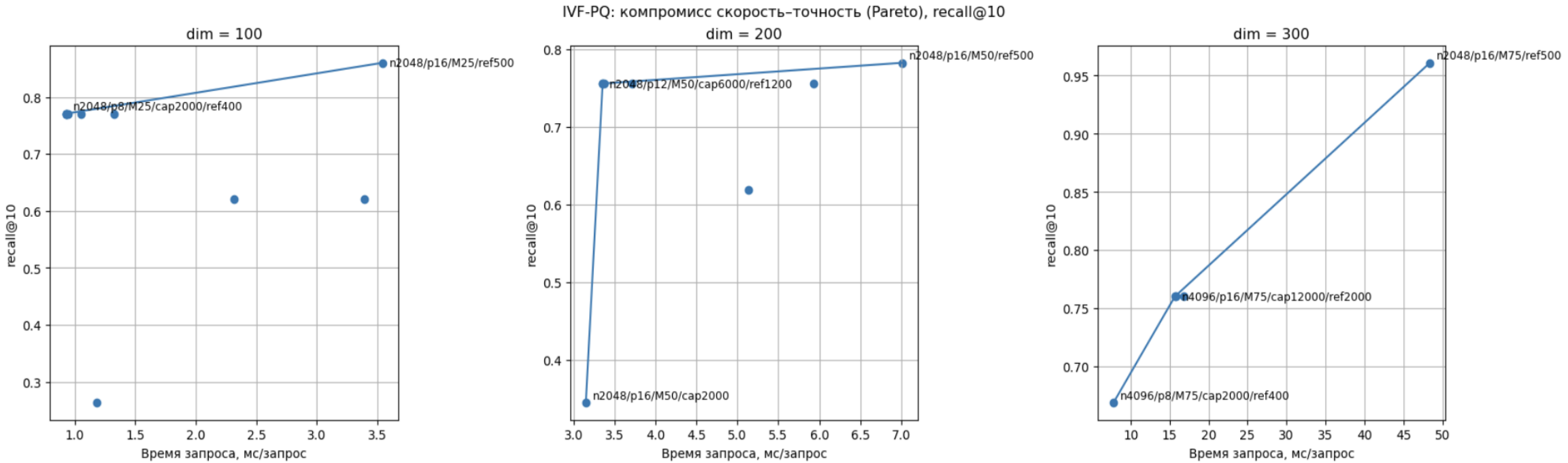
**Поиск (search)**

- Для запроса $q$ выбираются $P = n_{\text{probe}}$ ближайших кластеров:

$$\mathcal{P}(q) = \text{Top-}P\{\|\tilde{q} - c_\ell\|_2\}.$$

- Поиск выполняется только внутри выбранных inverted lists.
- Расстояния считаются приближённо по PQ, затем выбираются top-$k$ ближайших кандидатов.

**Ключевые параметры:** $n_{\text{list}}$ (разбиение базы), $n_{\text{probe}}$ (ширина поиска), $M$ (степень сжатия PQ).




- **Чёткий компромисс скорость–точность (Pareto).**
  Для каждой размерности видно, что увеличение времени запроса приводит к росту `recall@10` — ожидаемое поведение классического ANN-метода.

- **Хорошая работа при малой и средней размерности.**
  При $d = 100$ и $d = 200$ IVF-PQ достигает `recall@10` $\approx 0.75$–$0.8$ при времени запроса порядка 1–5 мс, что приемлемо для real-time приложений.

- **Слабая масштабируемость по размерности.**
  При $d = 300$ время запроса резко возрастает (до десятков миллисекунд), даже при умеренных значениях точности.

# LoRANN: идея метода (ANN по косинусной близости)

**1. Нормировка.** Приводим векторы базы и запросы к единичной длине.

$$\tilde{x} = \frac{x}{\|x\|_2}, \quad \tilde{q} = \frac{q}{\|q\|_2}$$

**2. IVF-разбиение.** Делим базу на $L$ кластеров (n_clusters); центроиды $c_\ell$.

$$L = \texttt{n\_clusters}, \qquad c_\ell \in \mathbb{R}^d$$

**3. Маршрутизация запроса.** Сканируем $P$ кластеров (n_probe) с максимальным $\tilde{q}^\top c_\ell$.

$$\mathcal{P}(q) = \mathrm{Top}{-}P\{\tilde{q}^\top c_\ell\}$$

**4. Узкое место.** Внутри выбранного кластера $\ell$ нужно оценить все $m_\ell$ точек.

$$s_\ell(q) = X_\ell \tilde{q} \in \mathbb{R}^{m_\ell}$$

## Ключевая мысль

**LoRANN ускоряет внутрикластерные оценки**, заменяя точное умножение $m_\ell \times d$ на вычисления в низкоранговом пространстве ранга $r = \texttt{rank}$.

**Ключевые параметры на этапе поиска:** n_clusters (размер кластеров), n_probe (сколько кластеров сканируем).

# LoRANN: математика и источник ускорения

## Обучение (fit)

- Для каждого кластера $\ell$ собираем обучающие запросы $T_\ell$ (параметры: `n_probe_train`, `train_multi_assign`).
- Строим матрицу оценок и находим подпространство ранга $r = \texttt{rank}$:

$$
\begin{aligned}
Y_\ell &= T_\ell X_\ell^\top \in \mathbb{R}^{n_\ell \times m_\ell}, \\
Y_\ell &\approx U_\ell \Sigma_\ell V_\ell^\top, \\
A_\ell &= X_\ell^\top V_\ell, \qquad B_\ell = V_\ell^\top.
\end{aligned}
$$

## Поиск (search)

- Точно внутри кластера и LoRANN-аппроксимация:

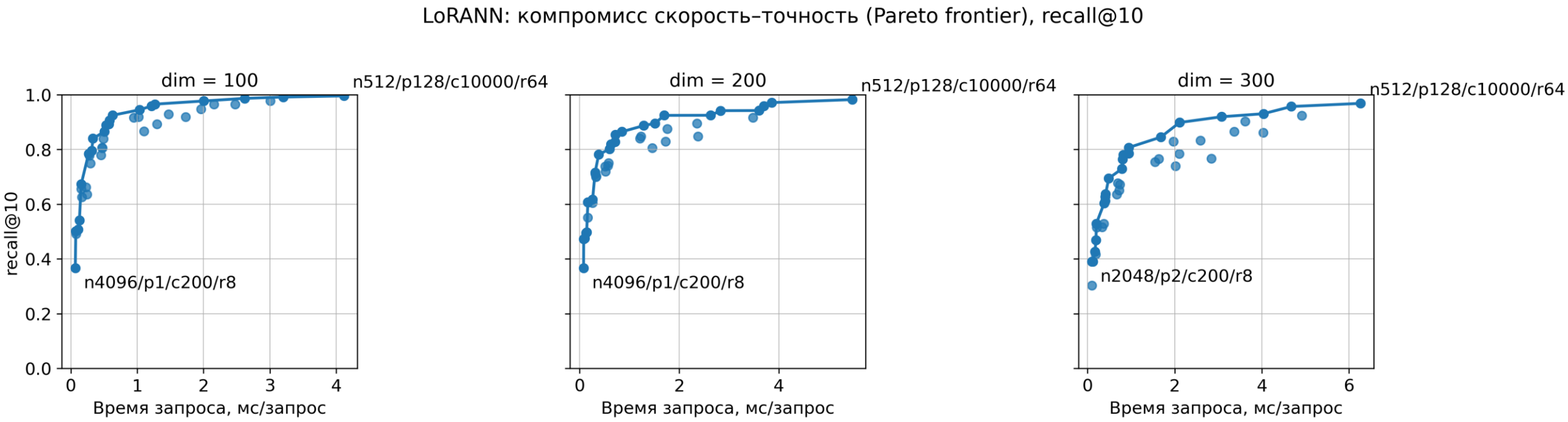
$$
\begin{aligned}
s_\ell(q) &= X_\ell \tilde{q}, \\
\hat{s}_\ell(q) &= V_\ell V_\ell^\top s_\ell(q) = (\tilde{q}^\top A_\ell)\, B_\ell.
\end{aligned}
$$

- Ускорение внутри кластера:

$$
O(d\, m_\ell) \;\to\; O(d\, r + r\, m_\ell), \quad r \ll d.
$$

- Shortlist и rerank: берём top-$C$ кандидатов ($C = \texttt{candidate\_count}$), затем точный rerank $\to$ top-$k$.

# LoRANN – результаты экспериментов на GloVe




- Наилучшие по точности – комбинации с большим числом кластеров и кластеров-для-поиска, то есть n_clusters ≈ 4096–8192, n_probe ≈ 128–256, candidate_count ≥ 5000, rank ≥ 64. Эти параметры давали recall@10 ≈ 0.97–0.99, но время запроса увеличивалось до 5–6 мс/запрос.
- Самые быстрые, но с наихудшей точностью – n_clusters ≈ 1024–2048, n_probe = 1–2, candidate_count ≤ 200, rank = 8. При таких настройках recall@10 падал до 0.35–0.55, зато время запроса было < 0.5 мс/запрос.
- Оптимальные (на Парето-границе) — конфигурации вроде n_clusters ≈ 4096, n_probe ≈ 32–64, candidate_count ≈ 1000–2000, rank = 16–32. Они обеспечивали recall@10 ≈ 0.9–0.95 при 1–2 мс/запрос, давая наилучший баланс точности и скорости.
- Разница между датасетами по размерности эмбеддингов (100 / 200 / 300): при росте размерности растёт время запроса и падает recall.

# Что такое ScaNN и его особенности

ScaNN (Guo et al., 2020) — двухэтапный ANN/MIPS-поиск, заточенный под cosine/inner product в больших коллекциях.

1) Routing (partitioning): быстрый выбор подмножества кандидатов через coarse-векторную квантизацию/дерево.

2) Fast scoring: база хранится в PQ-кодах, запрос остаётся в float; $\langle q, x \rangle$ оценивается через LUT (ADC).

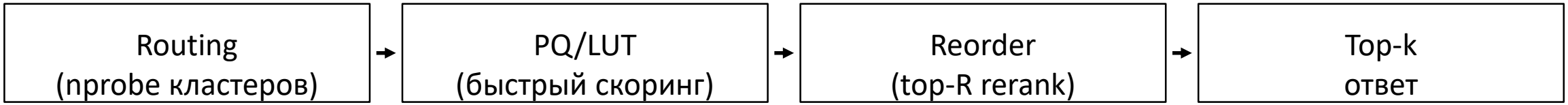
3) Reorder: точный пересчёт dot product на shortlist повышает точность при контролируемом времени.

Ключевая особенность: score-aware (anisotropic) quantization — оптимизация ошибки скоринга, сильнее штрафуется компонент ошибки вдоль направления x.

Память: при $K_s \leq 256$ используется 1 байт на подпространство $\Rightarrow$ M байт/вектор (+ coarse-структура).

# ScaNN: пайплайн поиска

- Цель: top-k по cosine (эквивалентно dot product на L2-нормированных векторах)
- 1) Routing: coarse-кластеризация ограничивает число кандидатов
- 2) Fast scoring: PQ + LUT приближает ⟨q, x⟩
- 3) Reorder: точный rerank на shortlist (пересчёт dot product)




Идея: меньше кандидатов + дешевле скоринг ⇒ выше QPS при приемлемом recall

# Линейная алгебра в ScaNN: anisotropic VQ

- Для MIPS важна ошибка вдоль направления x: она напрямую влияет на ⟨q, x⟩
- ScaNN оптимизирует score-aware loss: сильнее штрафует «параллельную» компоненту ошибки
- Обновление центроида сводится к решению малой СЛАУ (d×d) на каждом

```
Loss (упрощённо):  ||x − c||²  + (γ−1)·((||x||² − ⟨x,c⟩)² / ||x||²)

Центроид (фикс. разбиение S):
(|S|I + (γ−1)·Σ_{x∈S} (xxᵀ / ||x||²)) · c  =  γ·Σ_{x∈S} x

⇒ c = A^{-1} b  (решаем через solve; A — сумма ранга-1 матриц)
```

# Гиперпараметры ScaNN (что влияет на качество/скорость)

**Routing (кандидаты)**

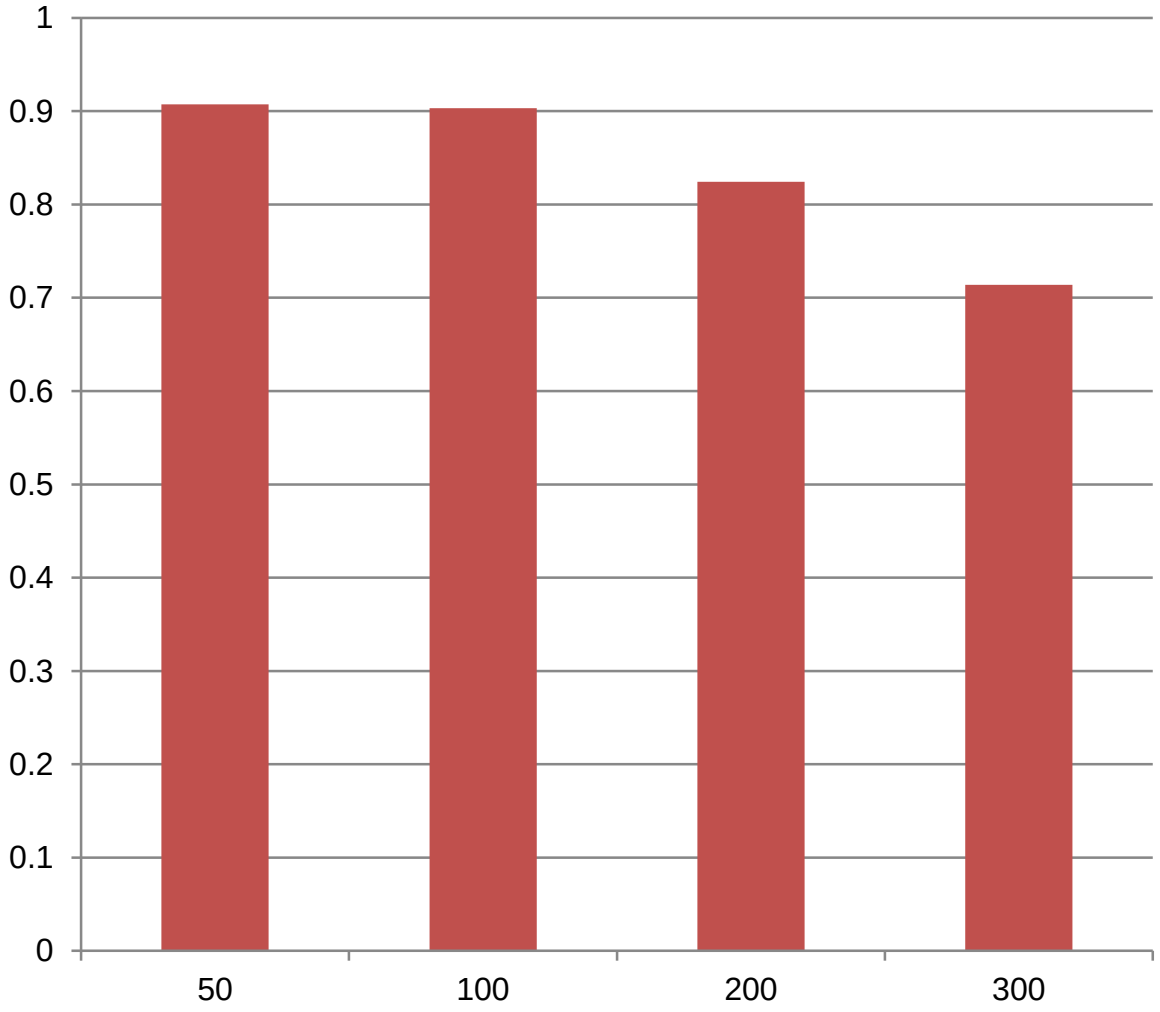
- nlist: число coarse-кластеров
- nprobe: сколько кластеров просматриваем
- candidate_count: кандидаты до PQ/LUT

**Quantization + reorder**

- M (n_subspaces): число подпространств PQ
- Ks (n_codewords): словарь в каждом подпространстве
- γ: степень анизотропии (γ=1 → обычный PQ)
- reorder_k: сколько

Grid search (в ноутбуке): перебираем nlist/nprobe, Ks/M/γ, candidate_count, reorder_k.
Данные: общий pipeline 0_data → одинаковые train/test queries и gt_test для 50/100/200/300.
Метрики: Recall@K + время (fit, search, ms/query).

# ScaNN: лучший Recall@10 по размерности



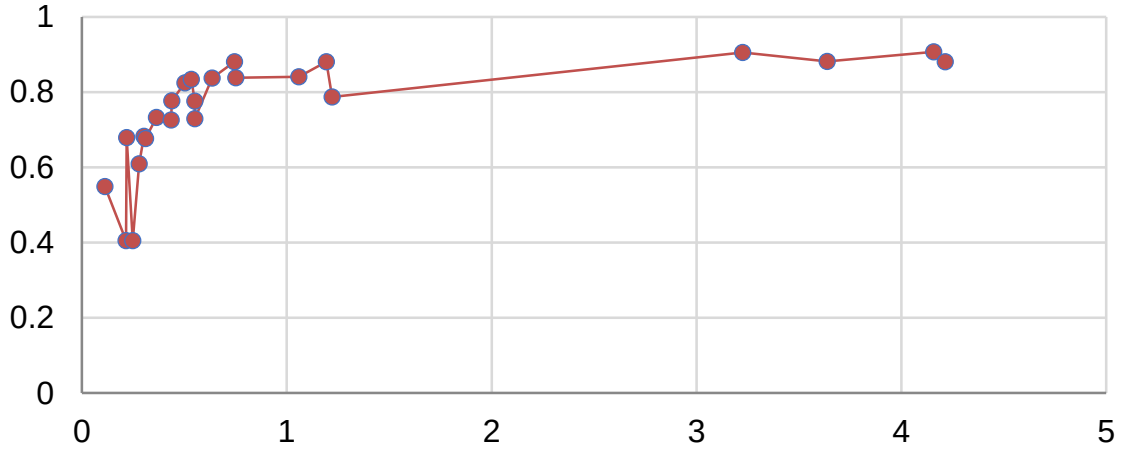

**Лучшая конфигурация (по Recall@10)**

| d | nlist | nprobe | Ks | M | γ | cand | reord |
|---|---|---|---|---|---|---|---|
| 50 | 512 | 32 | 64 | 20 | 1.1 | 20000 | 10000 |
| 100 | 512 | 32 | 64 | 20 | 1.1 | 20000 | 10000 |
| 200 | 1024 | 32 | 64 | 10 | 1.0 | 20000 | 10000 |
| 300 | 1024 | 32 | 64 | 10 | 1.0 | 20000 | 10000 |

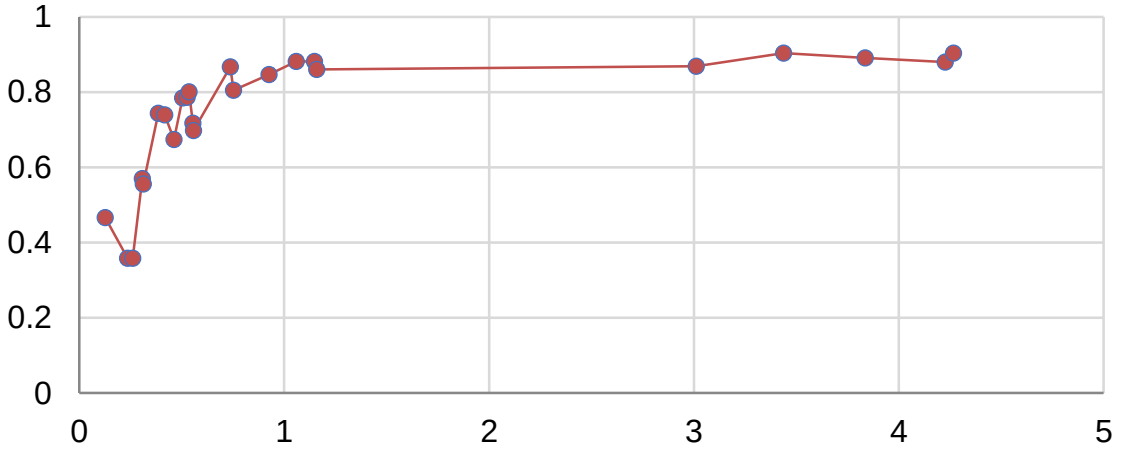
Замеры: Recall@10 на тестовых запросах (общий датасетный pipeline).

# ScaNN: качество vs время поиска (ms/query)

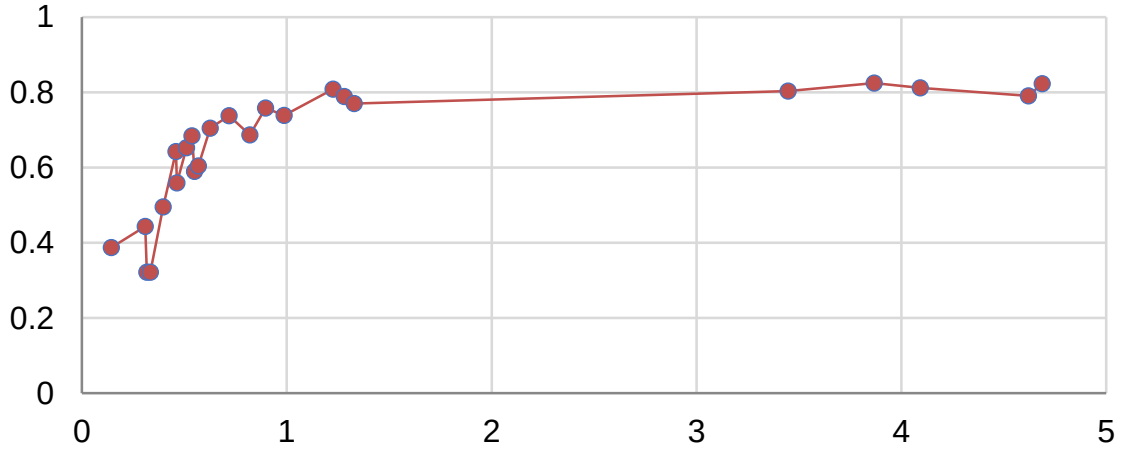
**d=50**



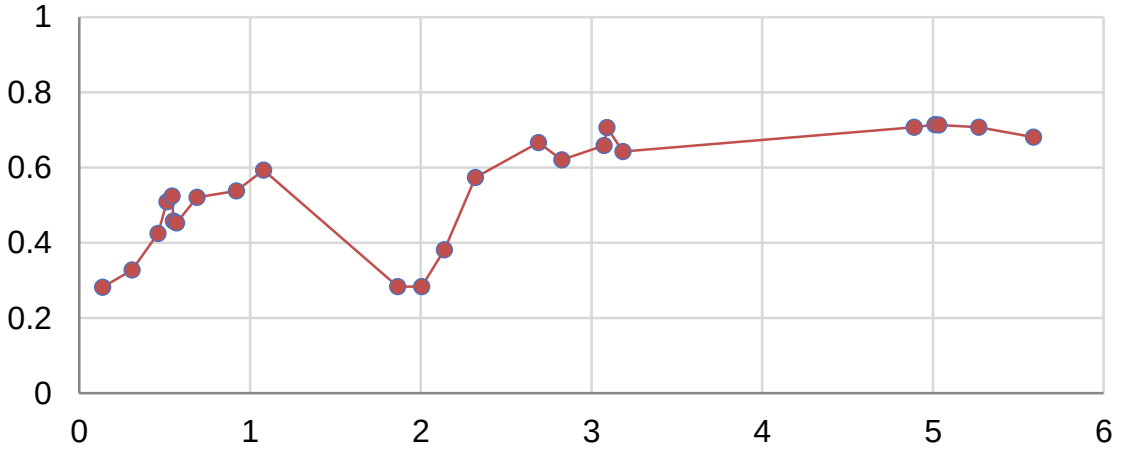

**d=100**




**d=200**




**d=300**



# Заключение

**IVF-PQ.**

- Классический ANN-метод с понятным компромиссом скорость–точность.
- Хорошо работает при малой и средней размерности ($d = 100, 200$).
- Плохо масштабируется по размерности: при $d = 300$ время запроса резко возрастает.

**ScaNN.**

- Современный двухэтапный ANN/MIPS-метод с score-aware квантованием.
- В наших экспериментах показал более низкий recall@10 при сопоставимом или большем времени поиска.
- Чувствителен к настройкам и структуре датасета.

**LoRANN (основной вклад работы).**

- Развивает идею IVF, ускоряя *внутрикластерный поиск*.
- Заменяет полное скалярное произведение на вычисления в низкоранговом подпространстве.
- Демонстрирует лучший Pareto-компромисс скорость–точность на всех размерностях.

# Основные ссылки:

github: https://github.com/nslobodchuk/ann-nla-project/tree/main

Jääsaari A., Oksanen J., Silven O. (2024).
LoRANN: Low-Rank Matrix Factorization for Approximate Nearest Neighbor Search}.
arXiv:2410.18926. https://arxiv.org/abs/2410.18926

Jégou H., Douze M., Schmid C. (2011).
Product Quantization for Nearest Neighbor Search}.
IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).
https://ieeexplore.ieee.org/document/5432202/

Guo R., Sun P., Lindgren E.,
Accelerating Large-Scale Inference with Anisotropic Vector Quantization}.
arXiv:1908.10396. https://arxiv.org/abs/1908.10396