FastRoLSH: Высокопроизводительная система приближенного поиска соседей и семплирования с поддержкой Product Quantization

Аннотация

Данный документ описывает практическую реализацию системы FastRoLSH, объединяющей алгоритмы FastLSH и roLSH с Product Quantization для эффективного приближенного поиска ближайших соседей (ANN) и семплирования. Система имеет поддержку гибридного режима работы, сочетающего преимущества LSH и PQ. В отличие от теоретического описания, акцент здесь сделан на архитектурных решениях, ключевых компонентах кода и их взаимодействии. Система реализована как монолитное серверное приложение на Python с использованием FastAPI, асинхронного доступа к PostgreSQL и GPU-ускорения через PyTorch. Документ отражает фактическую реализацию, включая механизмы индексирования, обработки запросов, семплирования и самооптимизации.

1 Введение

Реализация системы FastRoLSH представляет собой законченный инженерный продукт, готовый к развертыванию и использованию в production-среде. Ядро системы расширено классом `UnifiedQuantizationEngine`, который интегрирует LSH и продуктное квантование в единую архитектуру.

Система поддерживает три режима работы:

- **LSH-only**: классический поиск с использованием LSH
- **PQ-only**: поиск с использованием продуктного квантования
- **Hybrid**: двухэтапный поиск (LSH для кандидатов + PQ для точного расстояния)

Архитектура системы сознательно выбрана как «монолит с состоянием в БД». Это упрощает развертывание и эксплуатацию по сравнению с распределенной системой, сохраняя при этом возможность горизонтального масштабирования на уровне API-серверов (за счет балансировщика нагрузки) и использования мощных серверных машин с GPU для вычислений.

2 Архитектура и компоненты системы

2.1 Общая схема работы

Система состоит из четырех основных уровней:

1. **Уровень хранения данных (PostgreSQL)**: Хранит метаинформацию о датасетах, батчах, сериализованное состояние LSH-таблиц и модели квантования. Обеспечивает персистентность и возможность восстановления состояния сервиса после перезапуска.

2. **Вычислительное ядро (UnifiedQuantizationEngine)**: Интегрирует FastRoLSHsampler и продуктные квантователи. Отвечает за инкрементальное обновление состояния, обработку запросов и семплирование. Для вычислений использует библиотеку PyTorch, что позволяет автоматически задействовать GPU.

3. **Продуктное квантование (ProductQuantizer)**: Реализует методы сжатия и поиска в квантизованном пространстве. Поддерживает базовое квантование и продвинутое с диффузионными картами.

4. **Уровень API (FastAPI)**: Предоставляет внешний интерфейс для управления системой и выполнения операций. Запросы, связанные с добавлением данных, ставятся в фоновую очередь (BackgroundTasks), чтобы не блокировать отклик API.

2.2 Детали реализации

Класс `UnifiedQuantizationEngine` инициализируется с конфигурацией, определяющей режим работы и параметры обоих методов:

```
class UnifiedConfig:
    #                     LSH
    m: int = 100
    k: int = 10
    L: int = 5
    w: float = 1.0
    distance_metric: str = 'euclidean'
    initial_radius: Optional[float] = None
    radius_expansion: float = 2.0
    sampling_ratio: float = 0.1

    #

    pq_num_subspaces: int = 8
    pq_num_clusters: int = 256
    pq_use_diffusion: bool = False
    pq_use_optimized_kmeans: bool = True
    pq_batch_size: int = 1000

    #
    hybrid_mode: str = 'two_stage'  # 'two_stage', 'pq_only', '
    lsh_only'
    hybrid_candidate_multiplier: int = 10
    hybrid_use_compressed: bool = True
```

Класс `FastRoLSHsampler` инициализируется с параметрами, определяющими конфигурацию LSH (`d, m, k, L, w`), и опциональными параметрами для оптимизации roLSH (`initial_radius, radius_expansion`).

- **Инициализация хеш-функций:** Метод `_init_hash_functions` инициализирует проекционные векторы и смещения в зависимости от выбранной метрики расстояния. Для евклидовой метрики дополнительно создается тензор `self.indices` размерности `(L, k, m_sampled)` со случайными индексами признаков для реализации семплирования из FastLSH.
- **Обновление состояния (`update`):** Ключевой метод для инкрементального обучения. Для каждого вектора в батче вычисляется его хеш во всех L таблицах. Глобальный индекс точки рассчитывается как `self.total_points + i`. Индексы точек добавляются в соответствующие бакеты (словари `self.tables[l]`). Параллельно обновляется резервуарная выборка (`_update_reservoir_sample`) и статистика по радиусу (`_update_radius_stats`).
- **Поиск запросов (`batched_query`):** Реализует адаптивный поиск roLSH. Для каждого запроса поиск начинается с начального радиуса. На каждой итерации просматриваются бакеты, соответствующие хешу запроса. Если кандидатов недостаточно, радиус расширяется на коэффициент `radius_expansion`. Для евклидовой метрики это приводит к пересчету хеша с новым эффективным `w`. Для косинусной — к генерации соседних хешей путем переворачивания бит.

3 Методологические аспекты реализации

3.1 Реализация FastLSH

Принцип случайного семплирования признаков реализован полностью. Для каждой хеш-функции в каждой таблице заранее генерируется свой уникальный набор из `m_sampled = int(d * sampling_ratio)` случайных индексов. При вычислении хеша вектора `X` операция `X[:, inds]` эффективно выбирает только нужные признаки перед скалярным произведением на проекционный вектор `self.A[l, k`$_i dx$`].`$O(d)$ `до` $O(m)$ `на одну хеш-функцию.`

3.2 Реализация roLSH

`Адаптивный поиск реализован через итеративное расширение радиуса.`

1. `Инициализация радиуса: Если радиус не задан пользователем, он вычисляется автоматически. На основе первых батчей данных методом _update_radius_stats вычисляется среднее расстояние между точками, и начальный радиус устанавливается равным половине этого значения.`

2. Механизм расширения: В методе batched_query для каждого запроса цикл выполняется до 5 раз. На каждом шаге радиус умножается на radius_expansion. Для евклидовой метрики новый хеш вычисляется в методе _get_expanded_hashes путем деления ширины бакета w на текущий радиус. Для косинусной метрики в методе _get_cosine_expanded_hashes генерируются хеши, отличающиеся на 1, 2, ... бита в зависимости от шага расширения.

3.3 Реализация продуктного квантования

Продуктное квантование реализовано через классы ProductQuantizer и AdvancedProductQuantizer. Основные особенности:

- Разделение на подпространства: Исходное пространство разбивается на num_subspaces подпространств
- Кластеризация: В каждом подпространстве выполняется кластеризация методом K-means на num_clusters кластеров
- Кодирование: Каждый вектор представляется как набор индексов ближайших центроидов в каждом подпространстве
- Диффузионные карты: В расширенной версии используется диффузионное отображение для улучшения качества квантования

3.4 Интеграция LSH и продуктного квантования

Гибридный режим работы реализован через двухэтапный поиск:
1. Этап 1: Быстрый отбор кандидатов с использованием LSH 2. Этап 2: Точное ранжирование кандидатов с помощью продуктного квантования
Вероятность коллизии для комбинированного метода вычисляется как:

$$P_{\text{hybrid}}(r) = P_{\text{LSH}}(r) \cdot P_{\text{PQ}}(r)$$

где $P_{\text{LSH}}(r)$ - вероятность попадания в бакет LSH, $P_{\text{PQ}}(r)$ - точность восстановления расстояния после квантования.

3.5 Семплирование и оптимизация

Система поддерживает стратегии стратифицированного семплирования (sample) на основе LSH-бакетов.

- Пропорциональная стратегия: Из каждого бакета отбирается количество точек, пропорциональное его размеру.
- Сбалансированная стратегия: Из каждого бакета отбирается примерно одинаковое количество точек.

Для обеих стратетий корректно рассчитываются веса для последующего взвешенного обучения, компенсирующие смещение выборки.

Метод optimize_parameters реализует самонастройку системы. На основе резервуарной выборки данных строится эмпирическое распределение pairwise-расстояний. Определяется расстояние R (10-й перцентиль) и cR. Путем численного интегрирования функции _p_elastic находится значение параметра w, которое минимизирует параметр $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, что теоретически гарантирует оптимальное соотношение точности и полноты поиска.

3.6 Оптимизация параметров квантования

Система автоматически подбирает параметры продуктного квантования на основе данных:

- Количество подпространств оптимизируется под размерность данных
- Число кластеров адаптируется под распределение данных
- Используется адаптивное обучение на резервуарной выборке

4 Взаимодействие с базой данных

Состояние системы постоянно синхронизируется с PostgreSQL.

- Метаданные: Таблицы datasets и batches хранят параметры и информацию о processed батчах.
- Состояние LSH: Таблица lsh_tables хранит сериализованные в JSON списки индексов точек для каждого бакета. Метод _save_lsh_tables реализует инкрементальное обновление: изменяет только те бакеты, которые были обновлены в памяти.
- Модели квантования: Таблица quantization_models хранит параметры и обученные модели продуктного квантования.
- Восстановление состояния: Метод load_state_from_db позволяет экземпляру UnifiedQuantizationEngine полностью восстановить свое состояние из БД при перезапуске сервиса.

5 Производительность и оценка качества

Введены новые метрики оценки качества:

- Коэффициент сжатия: Отношение размера исходных данных к размеру квантизованного представления
- Точность восстановления: Среднеквадратичная ошибка восстановления исходных векторов
- Recall@K: Полнота поиска для топ-K результатов

6 Заключение

Представленная реализация системы FastRoLSH с поддержкой продуктного квантования является полной, корректной и готовой к применению. Она успешно объединяет теоретические принципы FastLSH и roLSH с современными методами продуктного квантования, адаптируя их для работы в режиме онлайн-обучения на потоковых данных.

Ключевыми инженерными преимуществами системы являются:

- Поддержка multiple режимов работы (LSH, PQ, Hybrid)
- Полная персистентность состояния в реляционной БД PostgreSQL
- Инкрементальное обновление индекса и метаданных
- Поддержка GPU-ускорения через PyTorch
- Фоновая обработка данных и асинхронный API
- Механизмы самооптимизации параметров на основе поступающих данных
- Автоматическая оптимизация параметров квантования
- Сохранение всех возможностей предыдущей версии
- Улучшенная производительность при работе с высокоразмерными данными

Система обеспечивает значительное улучшение производительности при работе с большими объемами высокоразмерных данных, сохраняя при этом высокую точность поиска и обеспечивая гибкость выбора режима работы в зависимости от конкретных требований приложения.