Стратегии Чанкинга

Автор: Ганеев Рустам Марсович

Что такое Чанкинг?

Чанкинг (Chunking) — это процесс разделения длинного текста на логические, управляемые блоки (чанки) для последующей векторизации.

В разных источниках это называют:

- Слайсинг (Slicing)
- Сегментация
- Фрагментация

Но суть одна: превратить любую книгу в набор абзацев, понятных машине.

Главный компромисс

Маленький чанк

Плюсы: Высокая точность, конкретика, дешевле эмбеддинг.

Минусы: Потеря контекста. "Он пошел туда" — кто он? куда туда?

Большой чанк

Плюсы: Сохраняет контекст, связность повествования.

Минусы: Много "шума", LLM теряет детали ("Lost in the middle"), дорого.

Влияние Чанкинга

- **Retrieval Quality:** Если чанк разрывает мысль, семантический поиск не найдет ответ.
- **Latency:** Больше чанков = дольше поиск. Крупнее чанки = дольше генерация LLM.
- **Cost:** Стоимость векторной БД растет линейно от количества чанков.
- **Hallucinations:** Избыточный контекст (слишком большие чанки) провоцирует модель выдумывать связи.

1. Фиксированная длина

Самый примитивный метод.

Мы просто режем текст каждые N символов или токенов.

```
text = "..."
chunks = [text[i:i+500] for i in range(0, len(text), 500)]
```

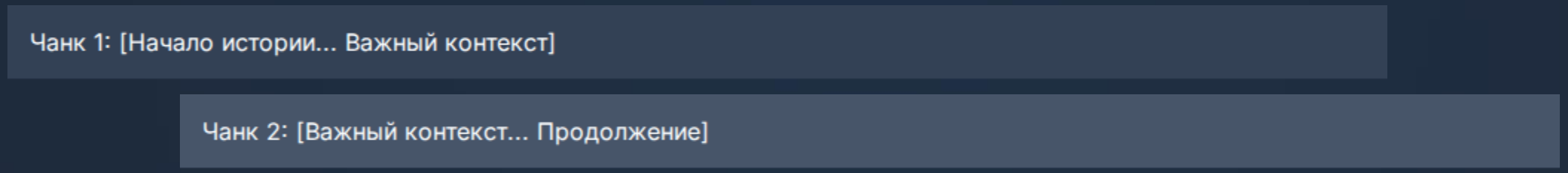
Проблема

Режет слова и предложения посередине. Смысл теряется на границах.

Скользящее окно (Overlap)

Решение проблемы границ — **Overlap** (перекрытие).

Мы захватываем 10-20% предыдущего чанка в следующий.



Это обеспечивает "бесшовность" смысла.

Символы vs Токены

Character Splitting

len(text). Быстро, но неточно для LLM.

500 символов != 500 токенов.

Token Splitting

Используем токенизатор (tiktoken).

Гарантирует, что чанк влезет в контекстное окно модели.

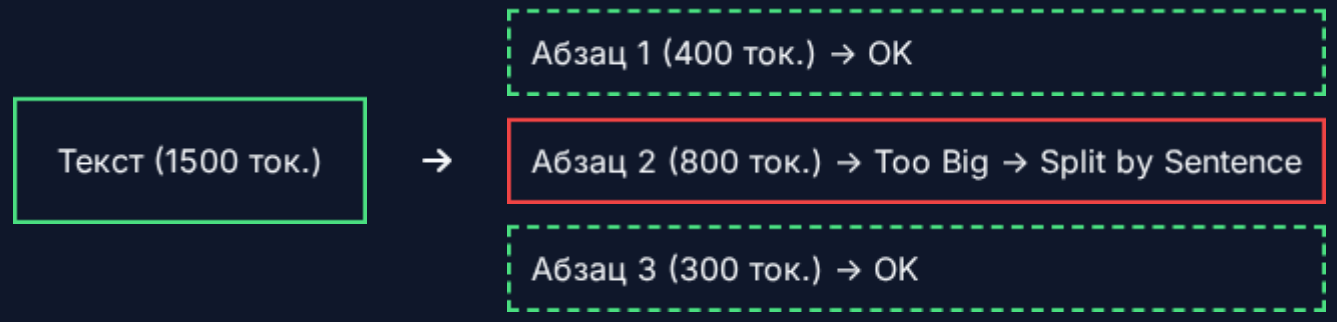
2. Рекурсивный Чанкинг

Стандарт в LangChain (RecursiveCharacterTextSplitter).

Пытается разбить текст по иерархии разделителей:

1. `"\n\n" (Абзацы) — приоритет №1`
2. `"\n" (Строки)`
3. `". " (Предложения)`
4. `" " (Слова)`
5. `"" (Символы) — крайняя мера`

Как работает Recursive?



Чанкинг по структуре (Code/Markdown)

Для кода и Markdown обычный текст-сплиттер не подходит.

Markdown

Разделение по заголовкам (H1, H2, H3). Чанк = целый раздел.

Code (Python/JS)

Разделение по классам и функциям. Нельзя разрывать тело функции посередине.

Разбиение по предложениям

Простой split('.') не работает (напр. "г. Москва", "т.д.").

Нужны NLP-библиотеки:

- **Spacy:** nlp(text).sents
- **NLTK:** sent_tokenize

Это база для более сложных методов (семантического чанкинга).

3. Семантический Чанкинг

"Режем текст не по линейке, а по смыслу."

Идея: границы чанка должны совпадать с границами смены темы.

Аналогия: Фильм

Фиксированный

Режем фильм на куски ровно по 10 минут. Сцена может оборваться на полуслове.

Семантический

Режем фильм по смене сцен. Один кусок = одна законченная сцена (диалог, действие).

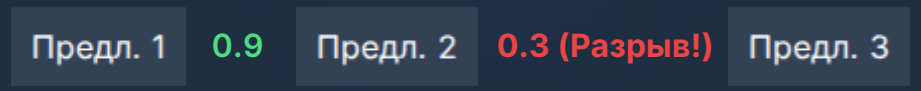
Алгоритм Семантического Чанкинга

1. **Split:** Разбиваем текст на предложения.
2. **Embed:** Для каждого предложения создаем вектор (эмбеддинг).

```
sentences = ["Привет.", "Как дела?", "Погода супер."]
vecs = [model.encode(s) for s in sentences]
```

Алгоритм: Cosine Similarity

Сравниваем косинусное сходство между соседними предложениями.



Высокое сходство -> одна тема. Низкое -> новая тема.

Как найти точку разрыва?

Мы не можем просто сказать "меньше 0.5". Для разных текстов порог разный.

Используем **Перцентили (Percentile Method)**.

Мы говорим: "Сделай разрез там, где сходство ниже, чем у 90% пар предложений".

Это динамический порог, адаптирующийся под документ.

Cluster Semantic Chunking

Подход от исследователей ChromaDB.

Вместо последовательного сравнения, мы можем кластеризовать предложения.

Мы ищем группы предложений, которые образуют плотное облако в векторном пространстве, и объединяем их в чанк.

Плюсы и Минусы Семантики

Плюсы

- Высокая смысловая целостность.
- Меньше шума в RAG.
- Лучшее качество Retrieval.

Минусы

- Медленно (нужно считать эмбеддинги для каждого предложения).
- Дорого (API calls).
- Сложность реализации.

Agentic / LLM Chunking

Самый дорогой, но качественный метод.

Мы отдаем текст самой LLM и просим: "Разбей этот текст на логические секции".

Модель использует свое понимание текста (как человек), чтобы найти идеальные границы.

Propositions (Атомарные факты)

Концепция из статьи "Dense X Retrieval".

Вместо абзацев мы разбиваем текст на **Пропозиции** — минимальные неделимые утверждения.

"Илон Маск, CEO Tesla, купил Twitter."
→
1. Илон Маск является CEO Tesla.
2. Илон Маск купил Twitter.

Graph-Based (Knowledge Graph)

Преобразование текста не в чанки, а в Граф Знаний.

Узлы (Сущности) и Ребра (Отношения).

Позволяет находить сложные связи, которые теряются при простой нарезке текста.

Проблема: Поиск vs Генерация

Для Поиска (Embedding)

Лучше маленькие чанки. Вектор точнее описывает конкретный факт.

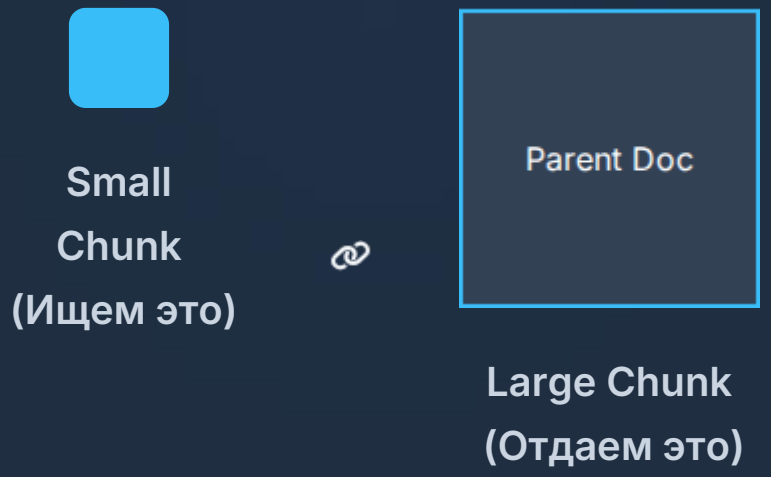
Для LLM (Generation)

Лучше большие чанки. Модели нужен окружающий контекст, чтобы дать развернутый ответ.

Как совместить?

Parent Document Retriever

Решение: Разделить то, что мы ищем, и то, что отдаем в LLM.



Иерархический Чанкинг

Создаем несколько слоев индексов.

- **Summary Index:** Саммари документа для высокоуровневого поиска.
- **Chunk Index:** Детальные чанки для точных вопросов.

Система сначала ищет в саммари, а потом "проваливается" в нужный документ.

Late Chunking (ColBERT)

Продвинутый метод (Jina AI / ColBERT).

Мы не усредняем векторы слов в один вектор чанка.

Мы сохраняем векторы **каждого токена** и сравниваем их (MaxSim).

Это сохраняет контекст даже внутри маленького фрагмента, но требует специальной инфраструктуры.

Multi-Vector Indexing

Один документ может иметь несколько векторов.

- Вектор самого текста.
- Вектор **Questions** (сгенерированных вопросов, на которые отвечает текст).
- Вектор **Summary**.

Поиск идет по всем векторам, а возвращается исходный чанк.

Чанкинг Таблиц

Таблицы нельзя резать посередине.

Стратегия:

1. Вырезаем таблицу целиком.
2. Делаем её текстовое описание (через LLM).
3. Векторизуем описание.
4. При поиске возвращаем сырую таблицу (HTML/Markdown).

Оценка Качества

Как понять, что выбранная стратегия (размер 500 vs 1000) лучше?

Нам нужны метрики, а не интуиция.

Используем фреймворки типа RAGAS или TruLens.

Метрики Поиска

- **Recall (Полнота):** Какой процент релевантной информации мы нашли?
- **Precision (Точность):** Какой процент найденного — полезен? (Меньше шума).
- **IoU (Intersection over Union):** Пересечение найденного с идеальным ответом.

Chunk Attribution

Метрика от Galileo.

Показывает, повлиял ли конкретный чанк на ответ модели.

- **Attributed:** Чанк использован.
- **Not Attributed:** Чанк найден, но бесполезен (шум).

Цель: Минимизировать Not Attributed чанки, чтобы не тратить токены.

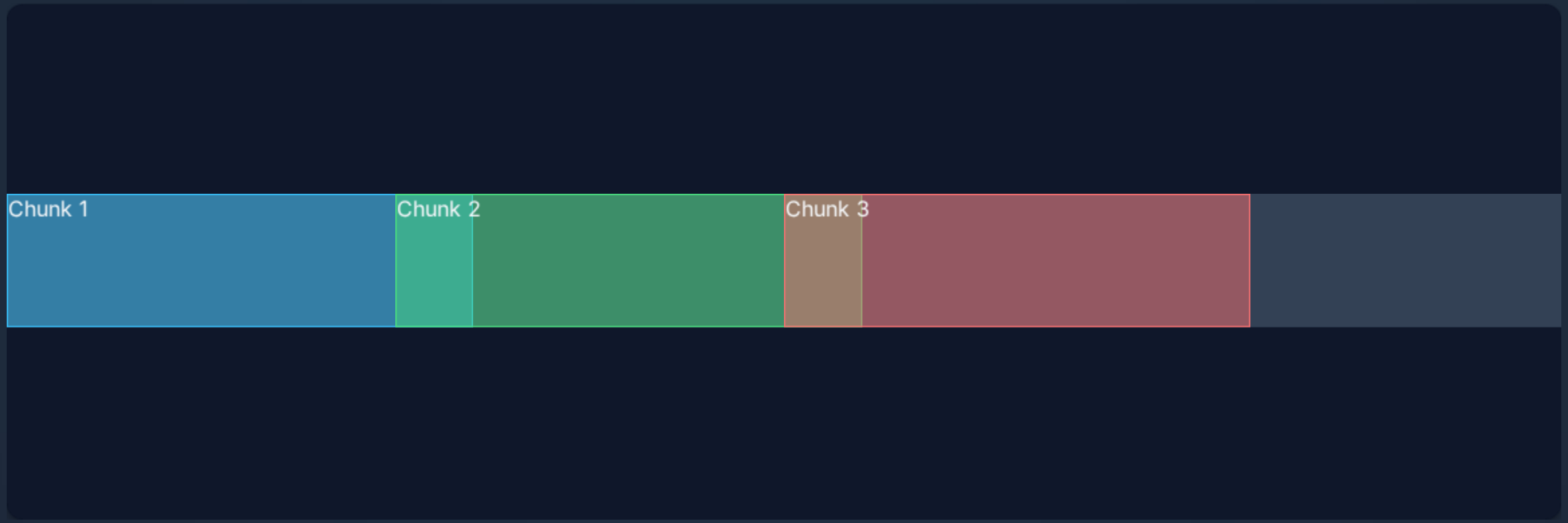
Chunk Utilization

Насколько полно используется чанк?

Если в чанке 1000 слов, а для ответа нужно 1 предложение — утилизация низкая.

Низкая утилизация -> Уменьшайте размер чанка!

Визуализация (Chunkviz)



Инструменты визуализации помогают увидеть перекрытия и "дыры" в разбивке.

Needle in a Haystack Test

Тест "Иголка в стоге сена".

Прячем уникальный факт в случайное место корпуса.

Проверяем, сможет ли RAG найти его при разных стратегиях чанкинга.

Семантический чанкинг обычно выигрывает этот тест.

Подбор параметров

Нет идеального размера чанка.

Grid Search: Запускаем тесты с размерами [128, 256, 512, 1024].

Исследование ChromaDB показало, что часто простые стратегии (Recursive) с размером 200-400 токенов работают на удивление хорошо.

Специфика Домена

- **Юриспруденция:** Чанки по статьям закона. Точность важнее контекста.
- **Художественная лит-ра:** Большие чанки с большим перекрытием.
- **Техподдержка:** Чанки по парам "Вопрос-Ответ".

Золотые Правила

1. Начните с **Recursive Character Splitter** (512 токенов, 10% overlap). Это сильный бейзлайн.
2. Если ответы неточные — попробуйте **Parent Document Retriever**.
3. Если теряется смысл — пробуйте **Semantic Chunking**.
4. Всегда измеряйте (Retrieval Recall) перед изменениями.

Вопросы?

Спасибо за внимание!