브라우징 코파일럿 RAG 도입기

전처리, 임베딩, 하이브리드 서치, 렛츠고!

유병국, 백엔드 엔지니어@LINER, 2023.09.22

목차

전처리 + 임베딩 + 하이브리드 서치

Browsing Copilot 소개

Why RAG?

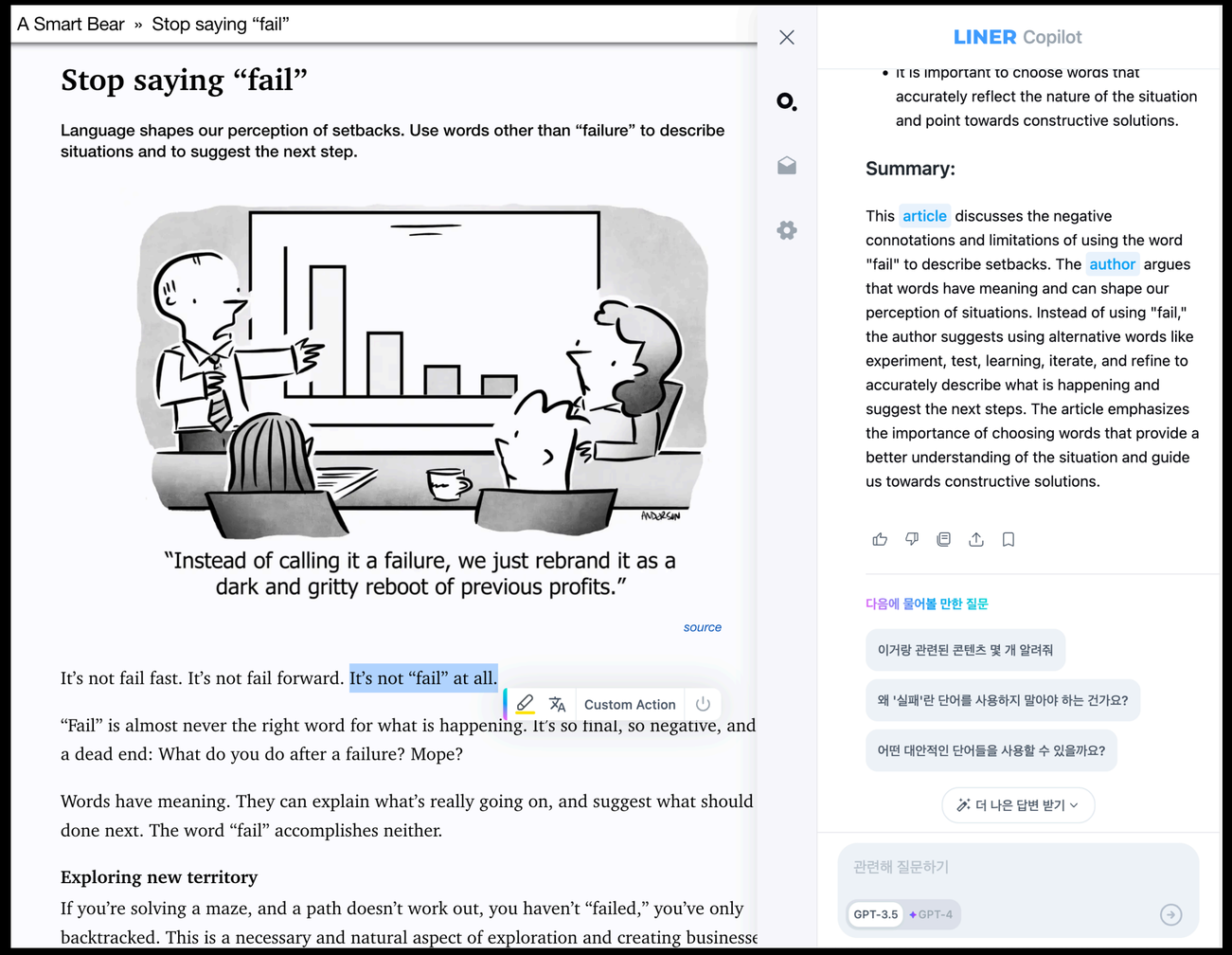
Copilot Sourcing System 👀

Engineering Challenges 🧐

Future Works 😏

Browsing Copilot

브라우저에 설치되는 애플리케이션(Browser Extension)으로 제공됨



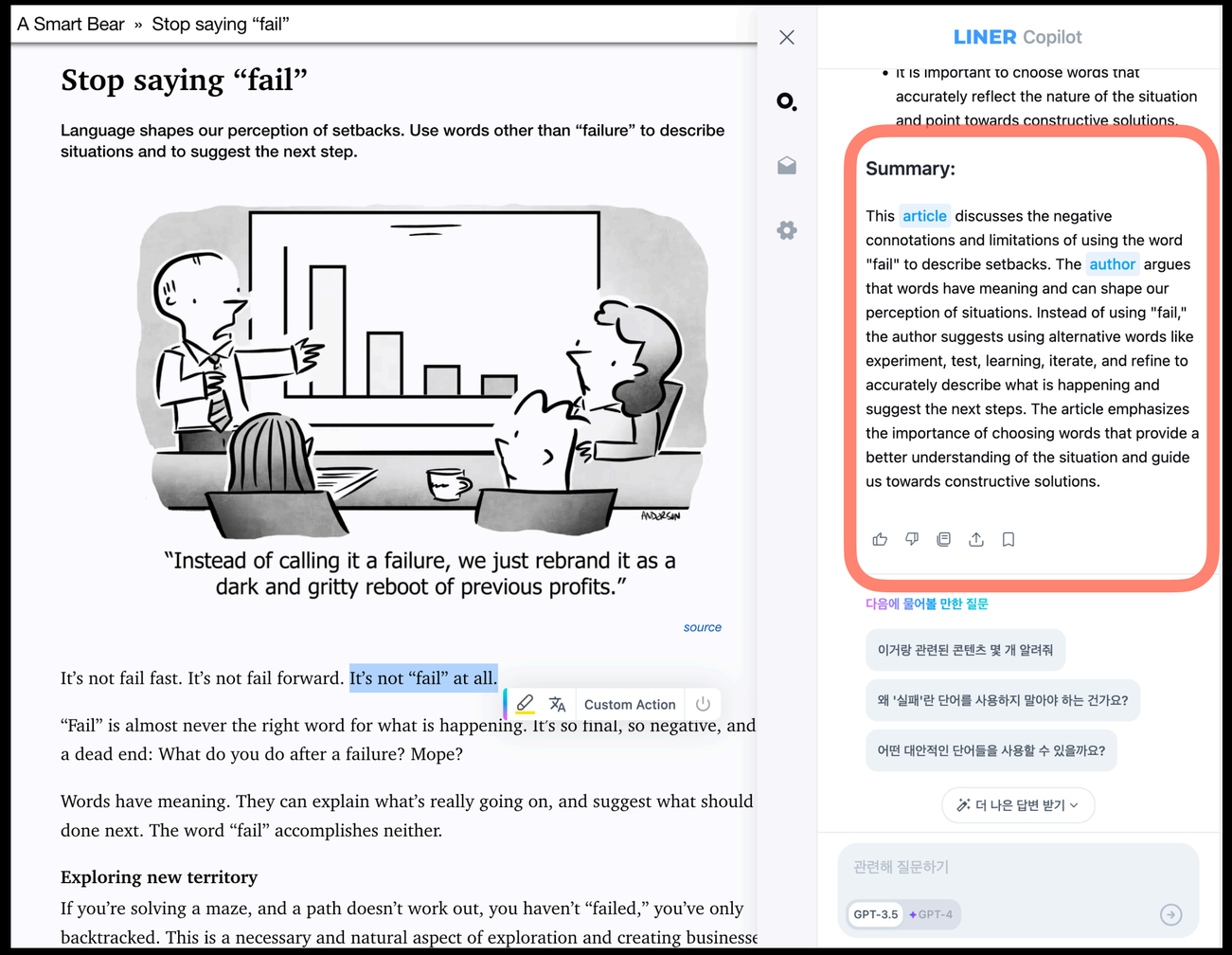
유저의 브라우징 동반자,

AI Assistant for

Web Browsing

Browsing Copilot

유저가 보고 있는 페이지에 대해 다양한 AI 기능을 제공



보고있는 **페이지 요약**

\+

내용에 관한 **대화**

Browsing Copilot

유저가 보고 있는 페이지에 대해 다양한 AI 기능을 제공

콘텐츠 일부에 대하여
하이라이트, 번역,
Custom Action 가능

How to inform LLM about the content?

RAG?
Finetuning?

non-parametric vs parametric

웹 브라우징 패턴은?

usage_bucket	url_count
1 usage	552314
2-4 usage	212563
5-10 usage	41338
11-50 usage	12372
51-100 usage	430
100+ usage	220

= 대부분 처음 보는 문서 (93%)

새로운 콘텐츠에 **즉각 대응**,

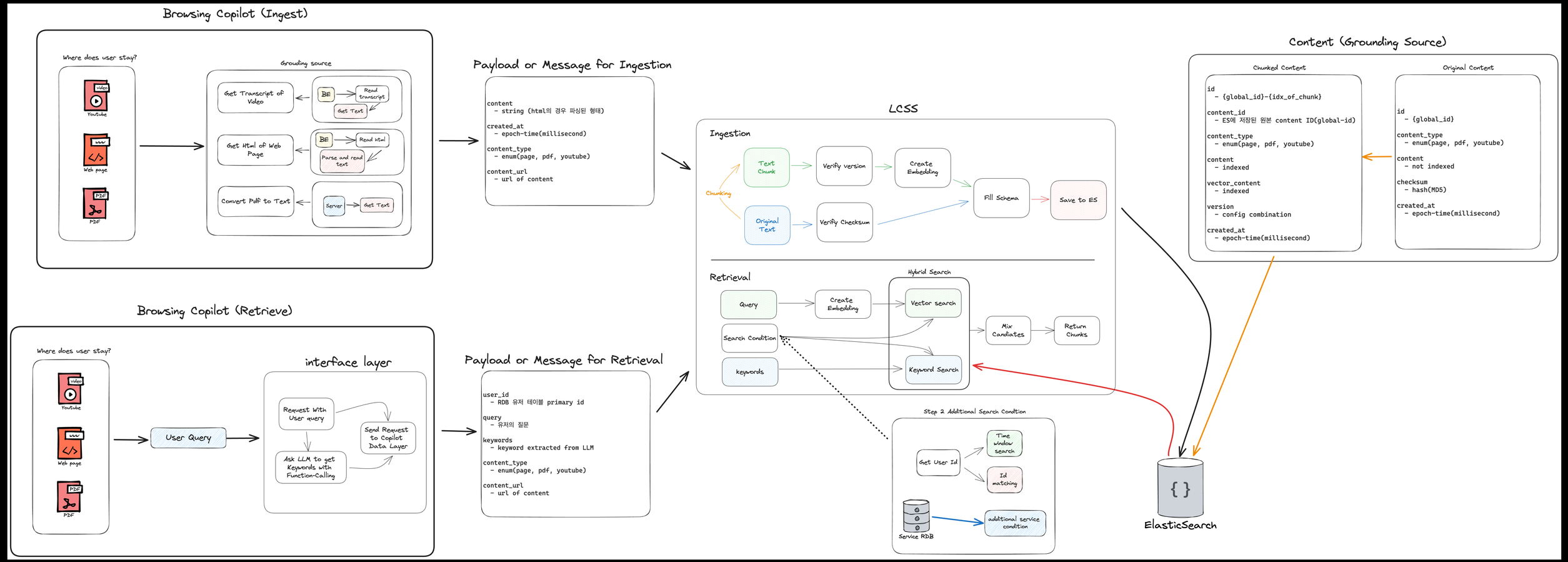
엔지니어링 노하우 레버리지

➡️ **RAG**를 선택

Copilot Sourcing System ⚒️

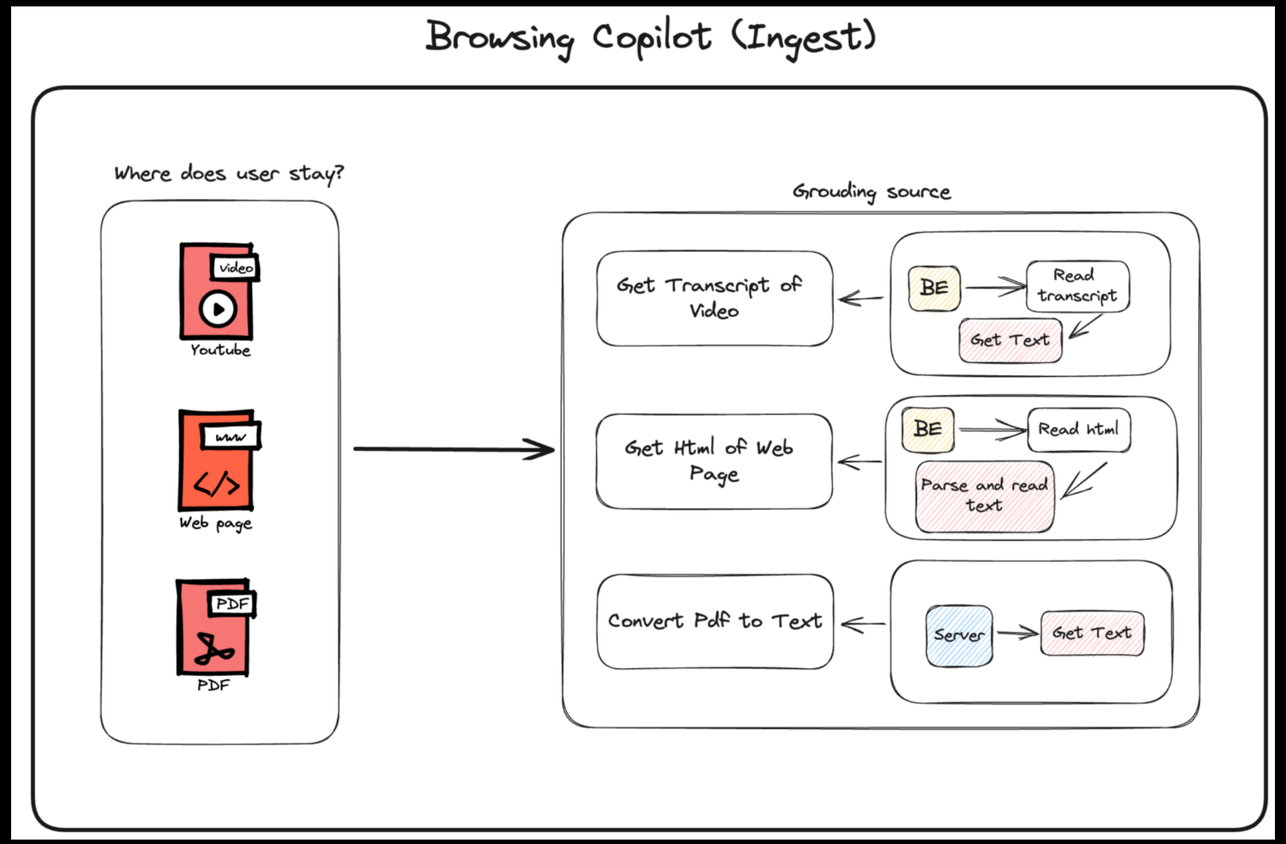
Copilot Sourcing System Architecture

전처리부 + 인입부 + 조회부



콘텐츠 인입 요청 만들기

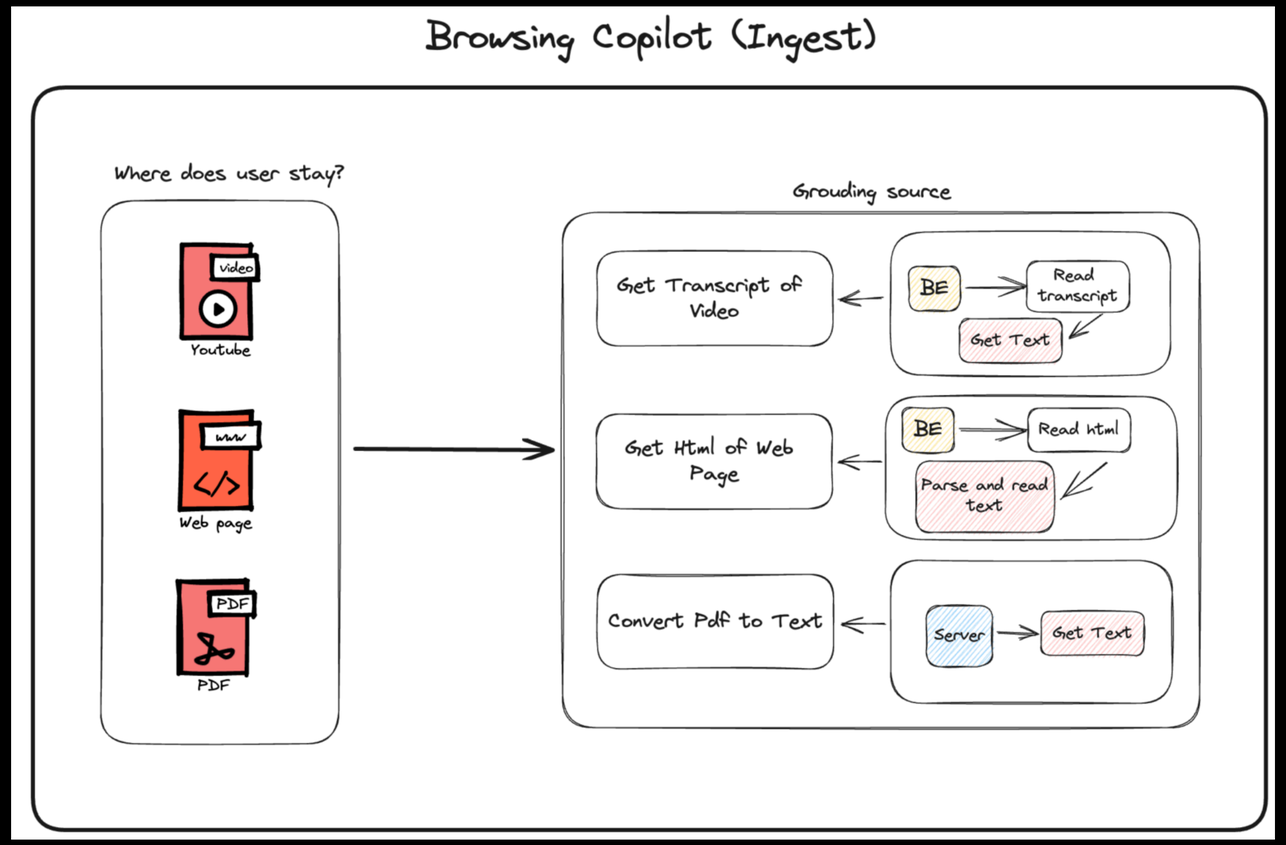
콘텐츠 타입에 따라 다르게 처리



텍스트 형태의 **중간 가공물**을
만드는 것이 목적

콘텐츠 인입 요청 만들기

콘텐츠 타입에 따라 다르게 처리



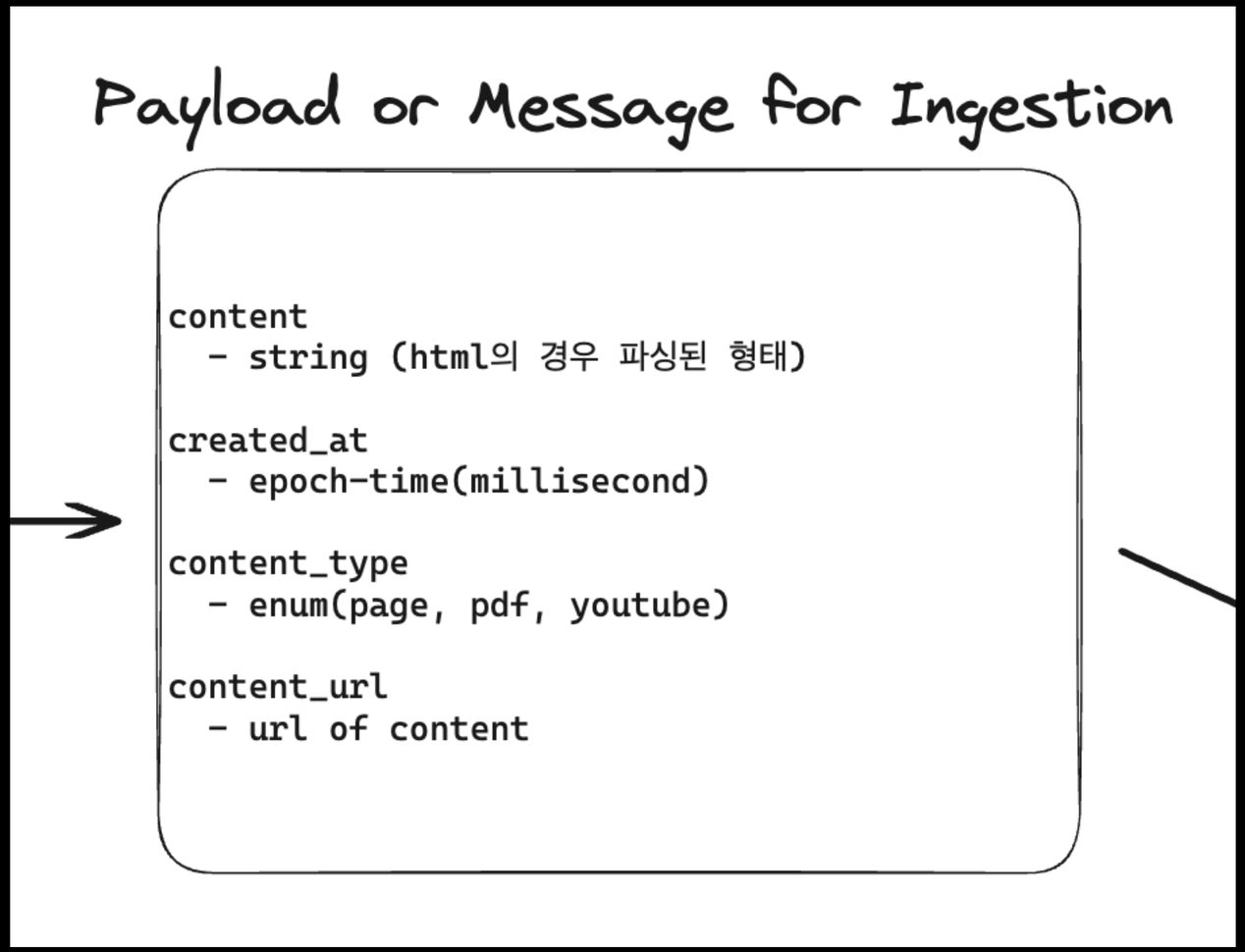
Youtube: 자막을 리스트 전달

HTML: 특정 태그 제거 후 전달

PDF: 전체 텍스트 전달

콘텐츠 인입 요청 만들기

콘텐츠 타입에 따라 다르게 처리



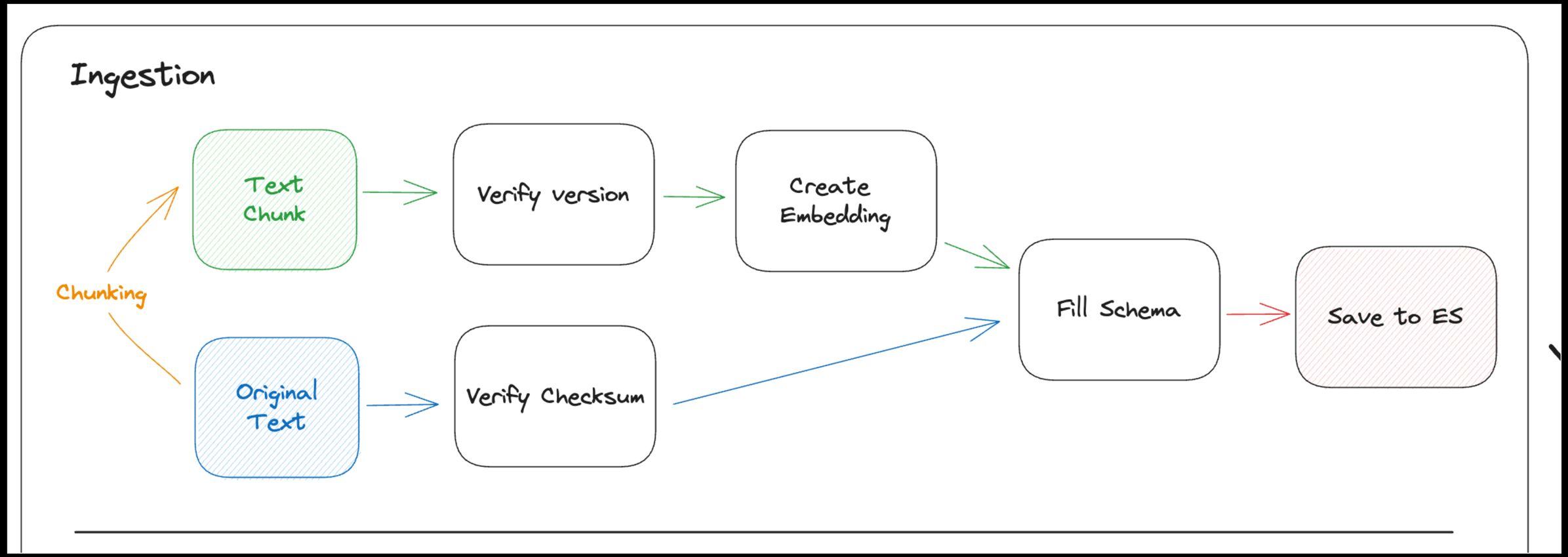
Epoch-time👍

글로벌 서비스라 **타임존이 상이**

Long으로 저장하는게 **편-안**

인입부

Chunk, Original을 각기 다른 프로세스를 통해 저장하는 과정



전처리

순수하게 본문 내용만을 남기기

HTML 메타 태그, 스타일 태그 등 **제거**, 필요 정보 **추출**

Youtube “[mm:ss] 자막” 형태로 **가공**

PDF “<page1>~</page1>” 형태로 **가공**

HTML 전처리, CPU-Heavy

전처리 서비스 별도 구현, 정책 집중화

HTML 메타 태그, 스타일 태그 등 **제거**, 필요 정보 **추출**

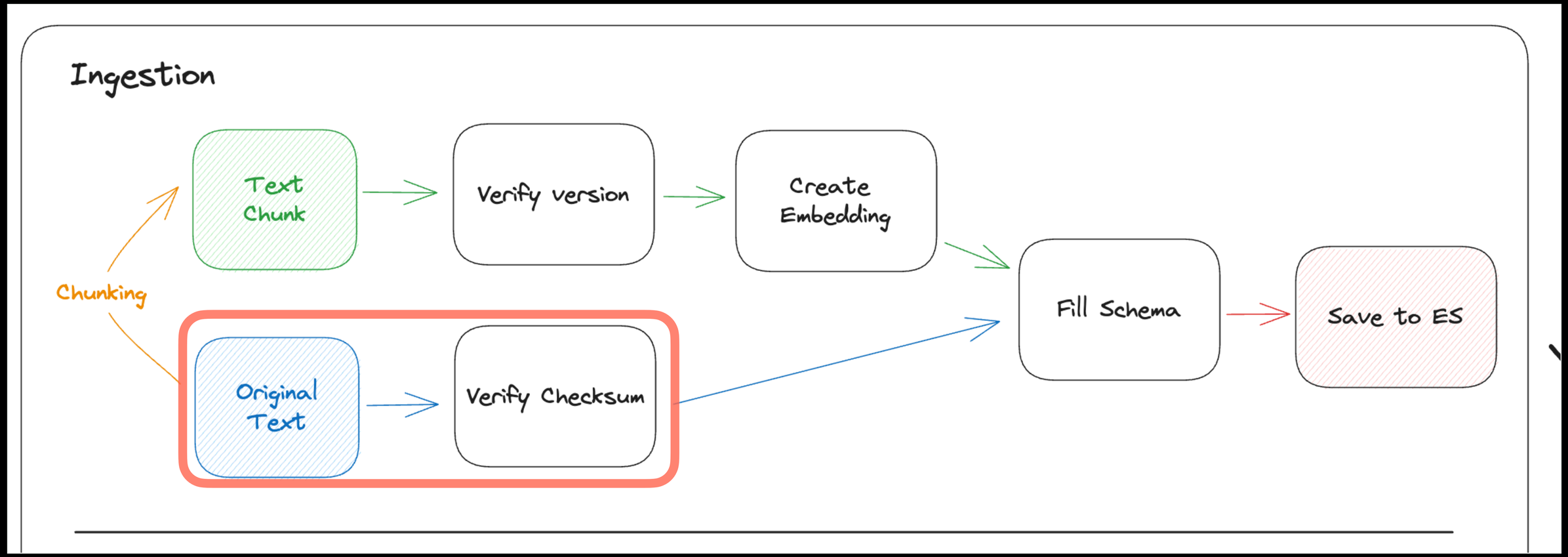
별도의 **Util 서비스(Python🐍)** 구현

전처리 책임/관심사 **분리**

성능 고도화에 **집중**

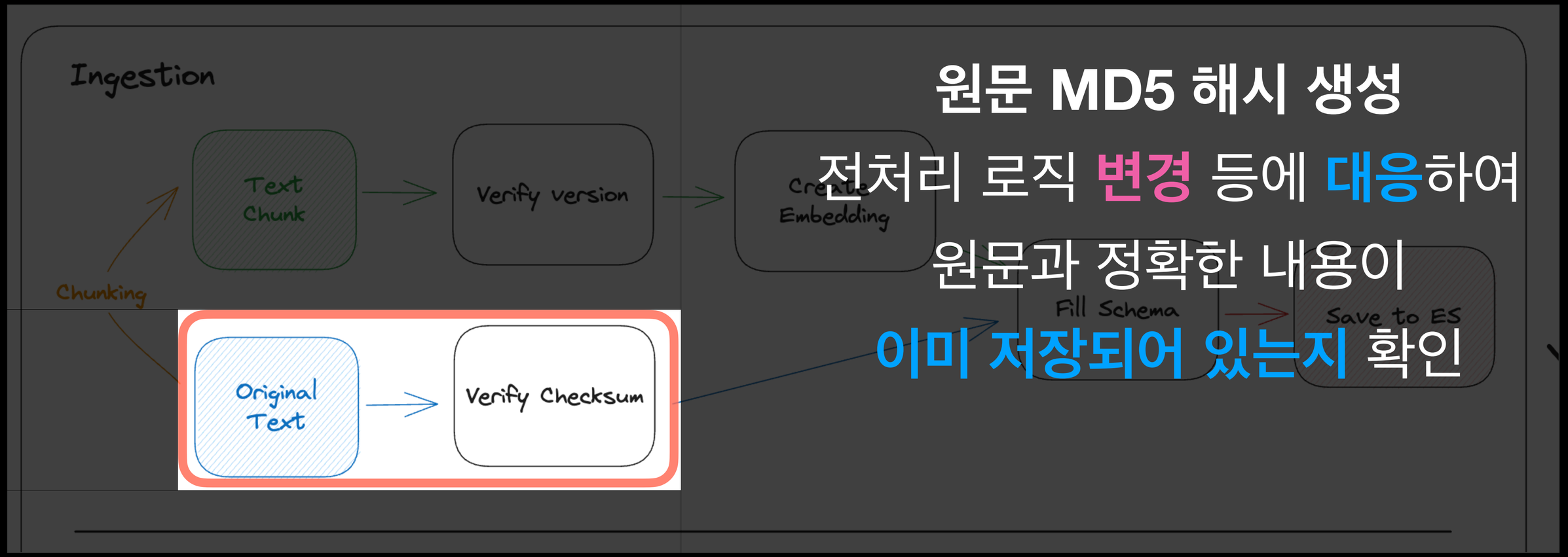
Text-Original

원문 프로세스



Text-Original

체크섬 확인



원문 MD5 해시 생성

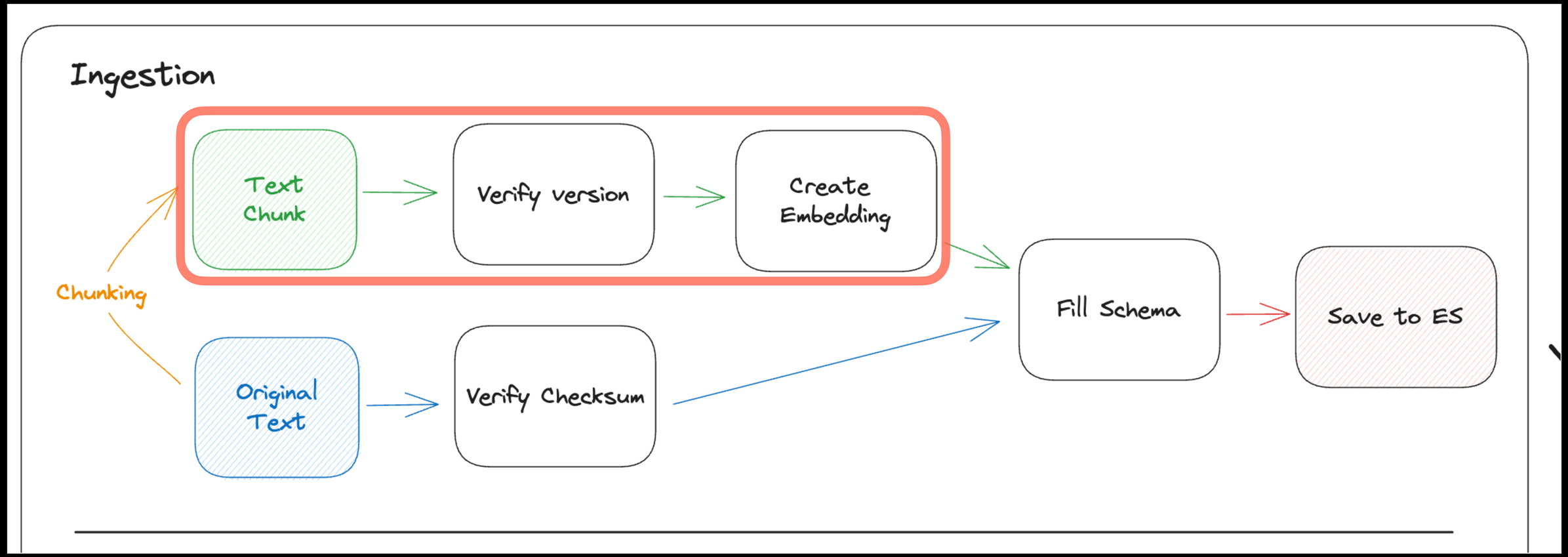
전처리 로직 **변경** 등에 **대응**하여

원문과 정확한 내용이

이미 저장되어 있는지 확인

Text-Chunk

Chunk 프로세스



청크를 나누는 이유

표현형의 효율성

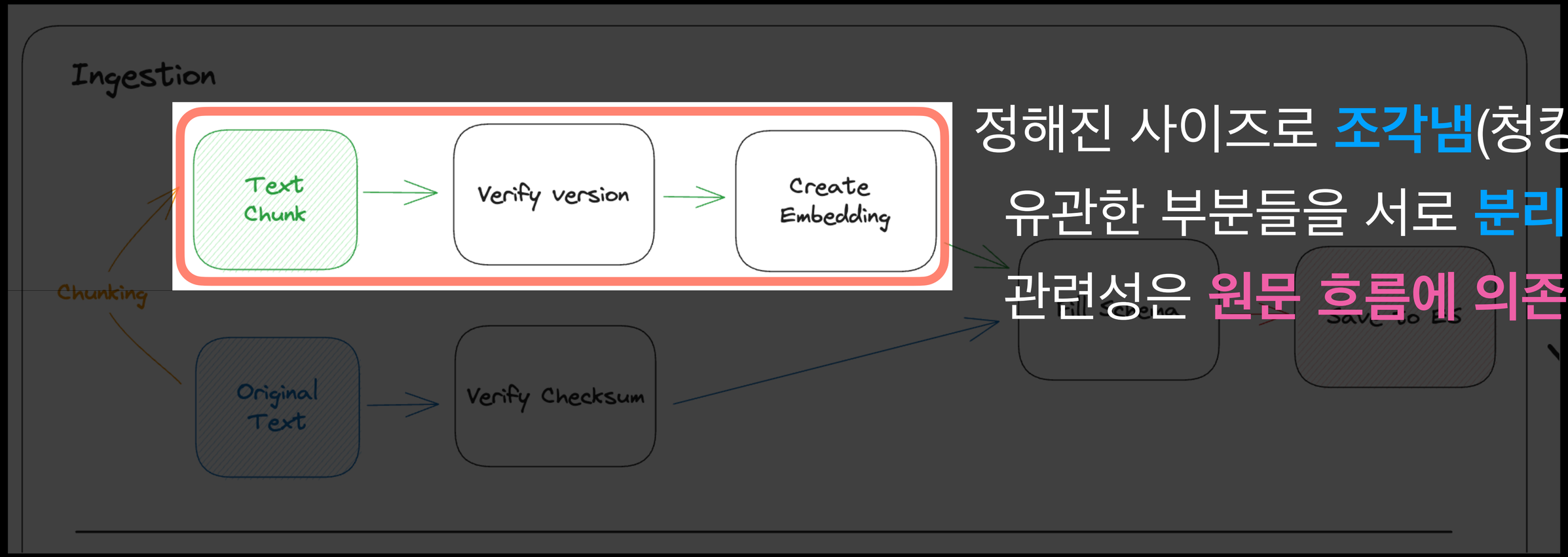
LLM Context 크기 제한

전체 내용이 필요한 것 ❌

임베딩 표현 대상 범위 축소

조각 내기

Chunk 프로세스



정해진 사이즈로 **조각냄**(청킹)

유관한 부분들을 서로 **분리**

관련성은 **원문 흐름에 의존**

조각 내기

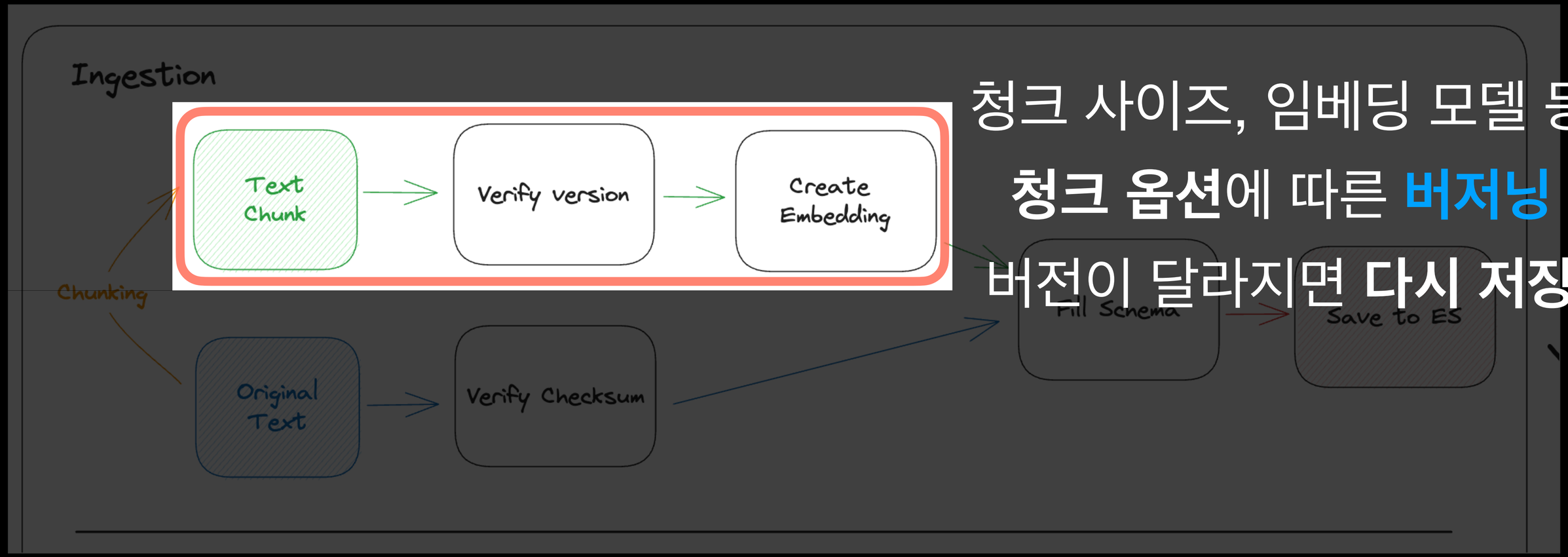
Chunk 프로세스

```
greg-liner
fun <T> List<T>.chunkWithOverlap(chunkSize: Int, overlapSize: Int): List<List<T>> {
    return this.windowed(size = chunkSize, step = chunkSize - overlapSize, partialWindows = true)
}

greg-liner *
fun chunkWithConfig(content: String, model: ModelType): List<String> {
    val tke = registry.getEncodingForModel(model)
    return tke.encodeOrdinary(content) (Mutable)List<Int!>!
        .chunkWithOverlap(chunkTokenSize, chunkOverlapSize) List<List<Int!>>
        .map { it: List<Int!>
            tke.decode(it)
        }
}
```

버전 확인

Chunk 프로세스



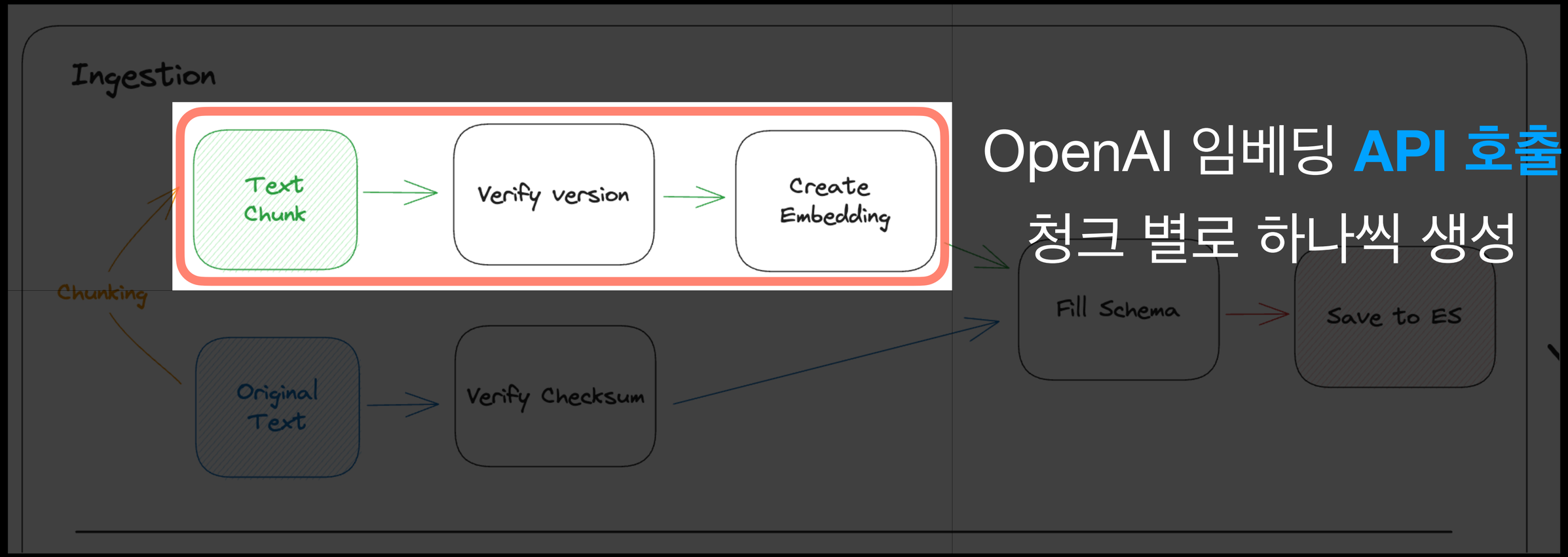
청크 사이즈, 임베딩 모델 등

청크 옵션에 따른 **버저닝**

버전이 달라지면 **다시 저장**

임베딩 생성

Chunk 프로세스



OpenAI 임베딩 **API 호출**

청크 별로 하나씩 생성

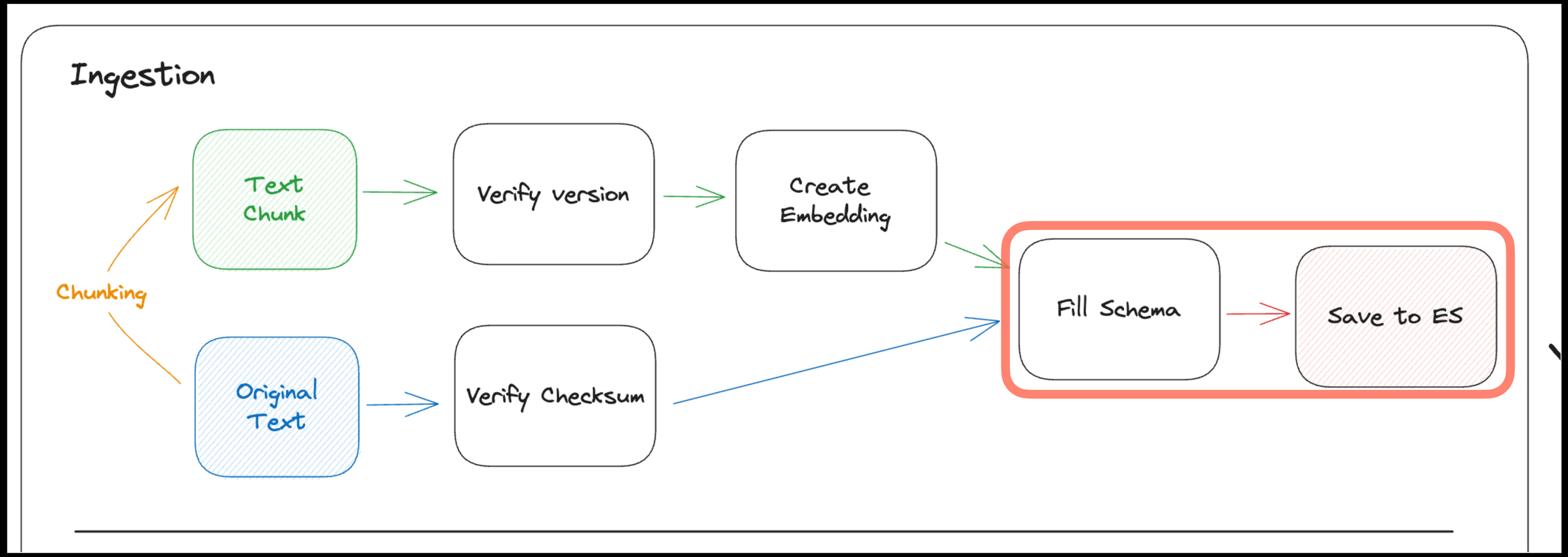
Text-Chunk

Chunk 프로세스

```
if (originalSourceChanges || sourceChangeObserver.chunkedSourceChanges(source)) {
    elasticSearchRepository.deleteChunkBySourceId(source.id) Result<Unit>
        .flatMap { sourceChunkMaker.chunkWithoutVector(source) } Result<List<SourceChunk>>
        .flatMap { sourceChunks ->
            sourceChunks.chunked(chunkBatchSize)
                .flatMap { chunks ->
                    chunks.parallelMap { sourceChunk ->
                        embeddingMaker.makeEmbeddingInternallyMono(sourceChunk.contentChunk) Result<List<Double>>
                            .getOrNull() List<Double>?
                            .let { vectorOfContent ->
                                sourceChunk.copy(
                                    vectorOfContentChunk = vectorOfContent ?: emptyList()
                                )
                            }
```

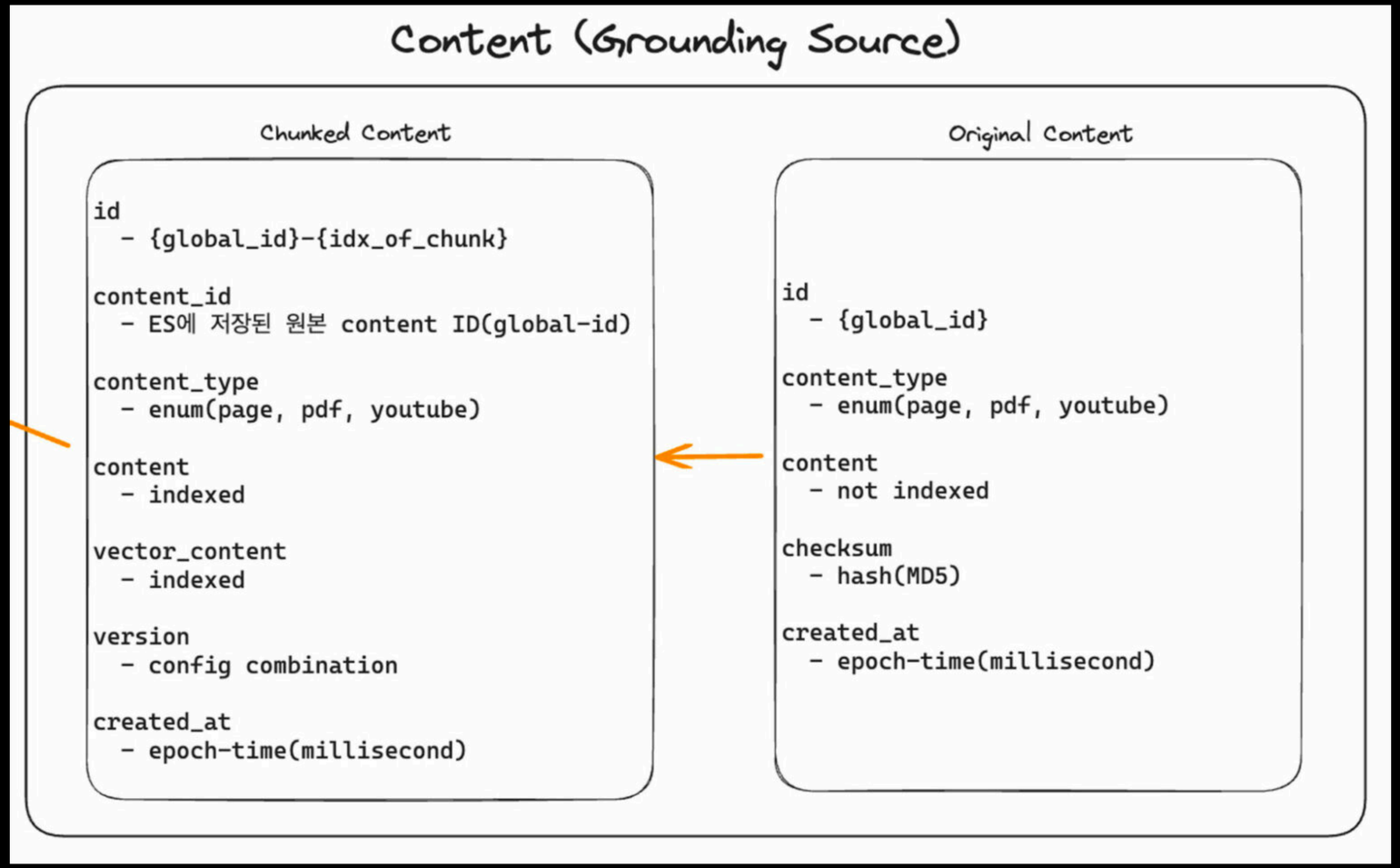
Save to Elasticsearch

청크, 원문 저장 프로세스



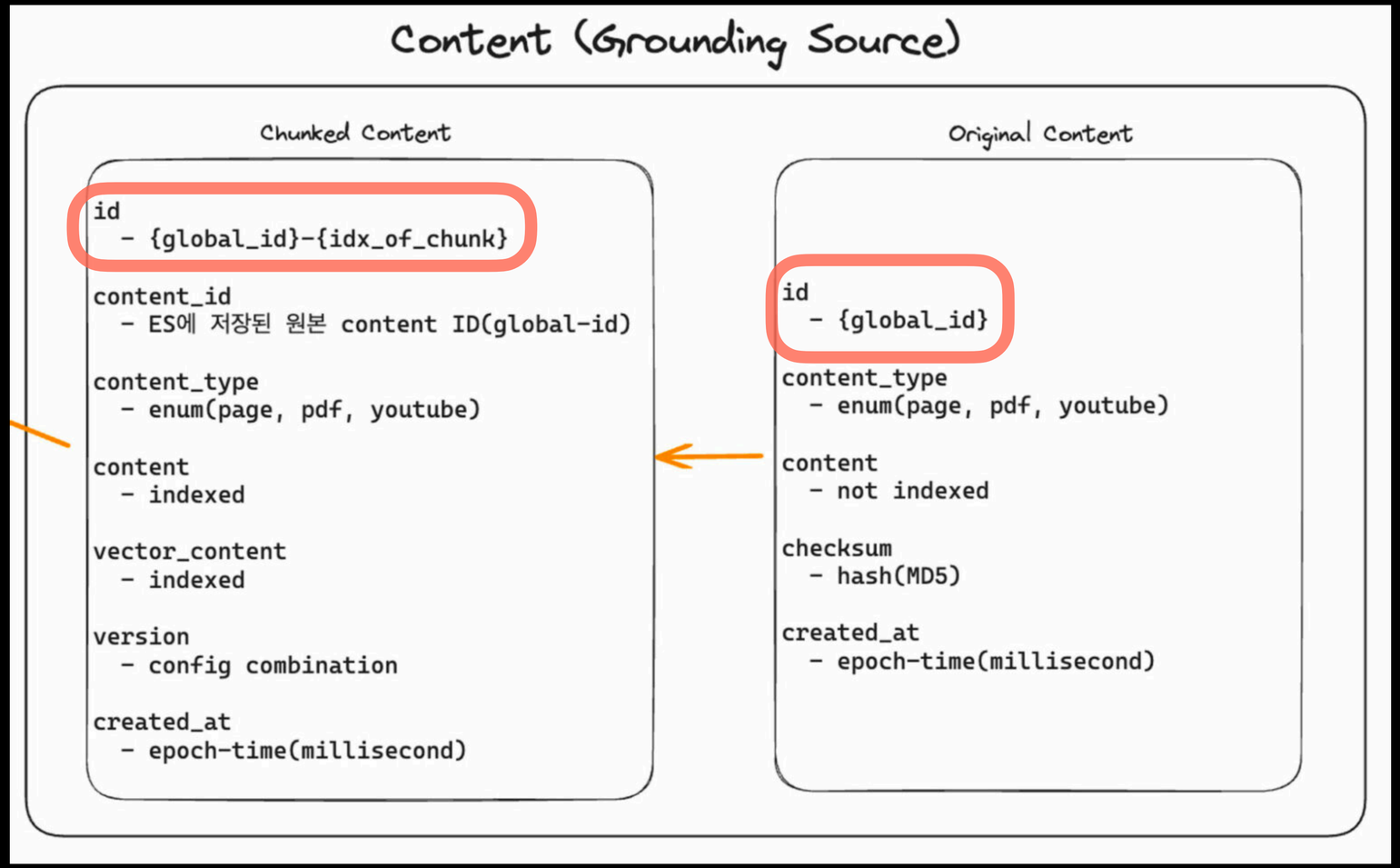
Save to Elasticsearch

청크, 원문 저장 스키마



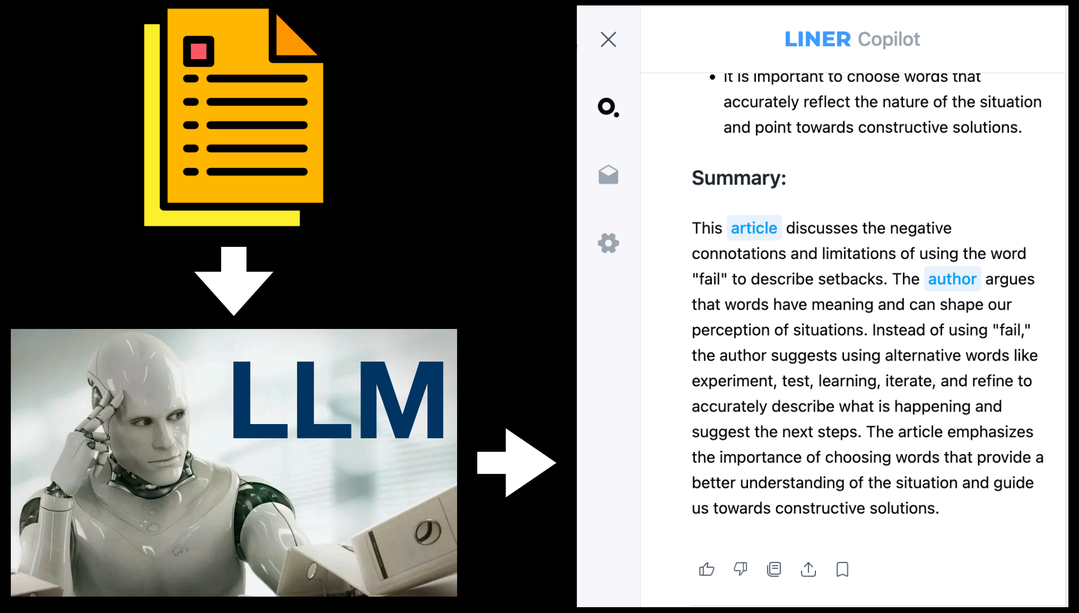
Save to Elasticsearch

청크, 원문 저장 스키마



Elasticsearch _id를 지정된 값으로

즉각 조회 목적



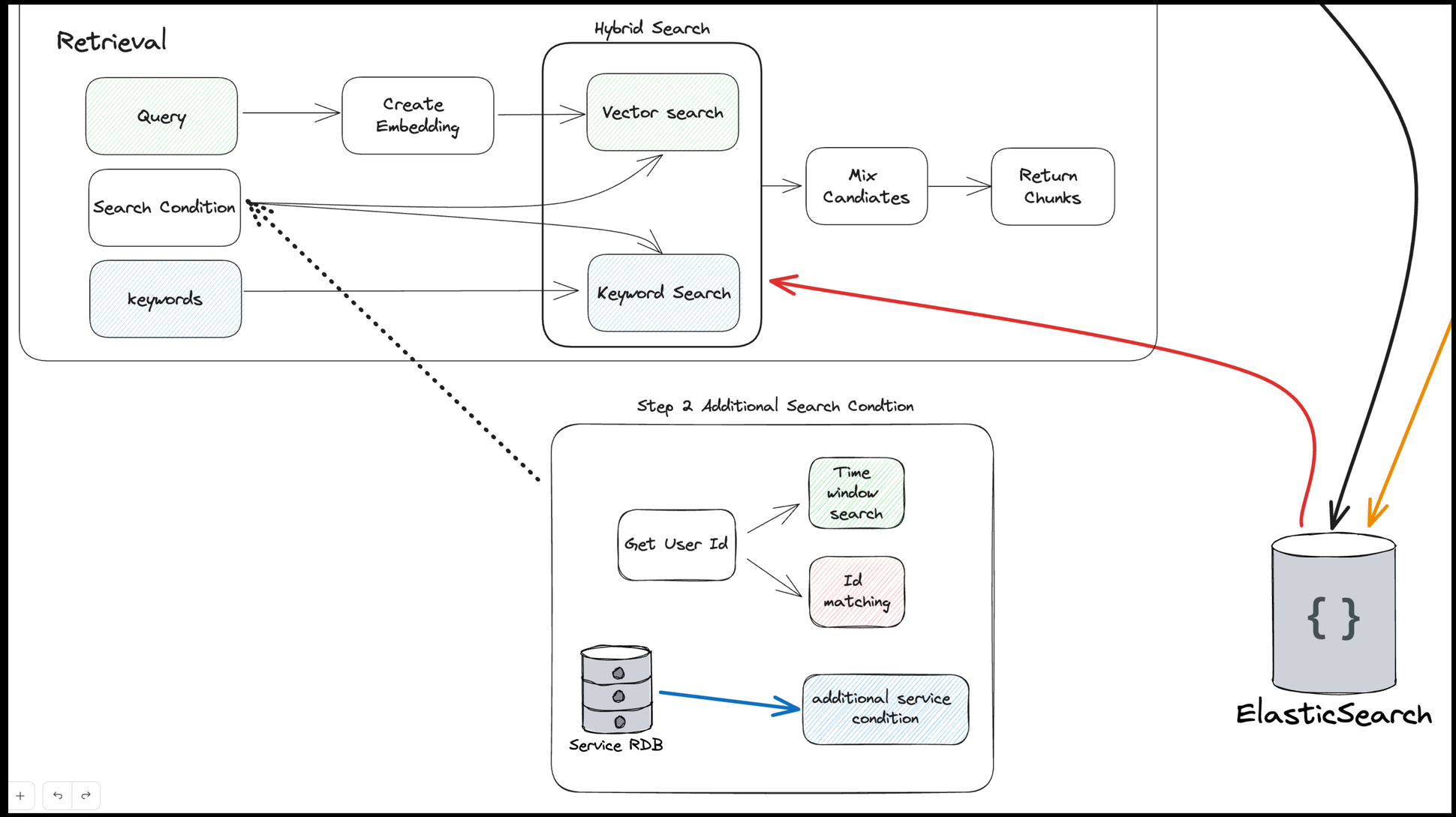
indexing이 느려지면 요약에
사용되는 **원문 검색 지연**

_id를 아는 경우에는
바로 조회가 가능

첫 대화 요약이 나가는 동안
청크 **인덱싱 시간** 벌 수 있음

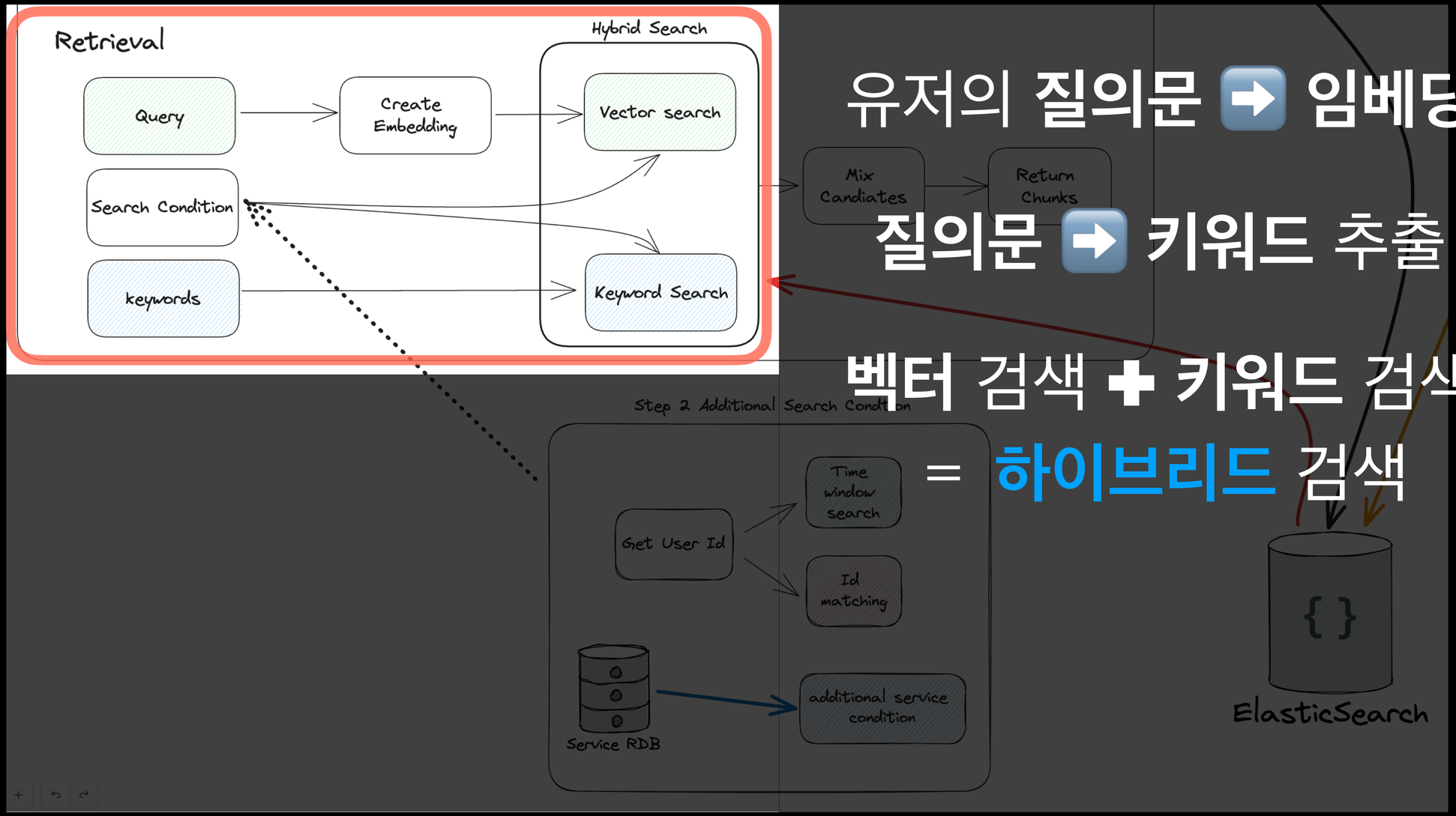
조회부

하이브리드 서치


Retrieval
Hybrid Search
Query
Create Embedding
Vector search
Search Condition
Mix Candiates
Return Chunks
keywords
Keyword Search
Step 2 Additional Search Condtion
Time window search
Get User Id
Id matching
Service RDB
additional service condition
ElasticSearch

하이브리드 검색

청크 검색



하이브리드 검색

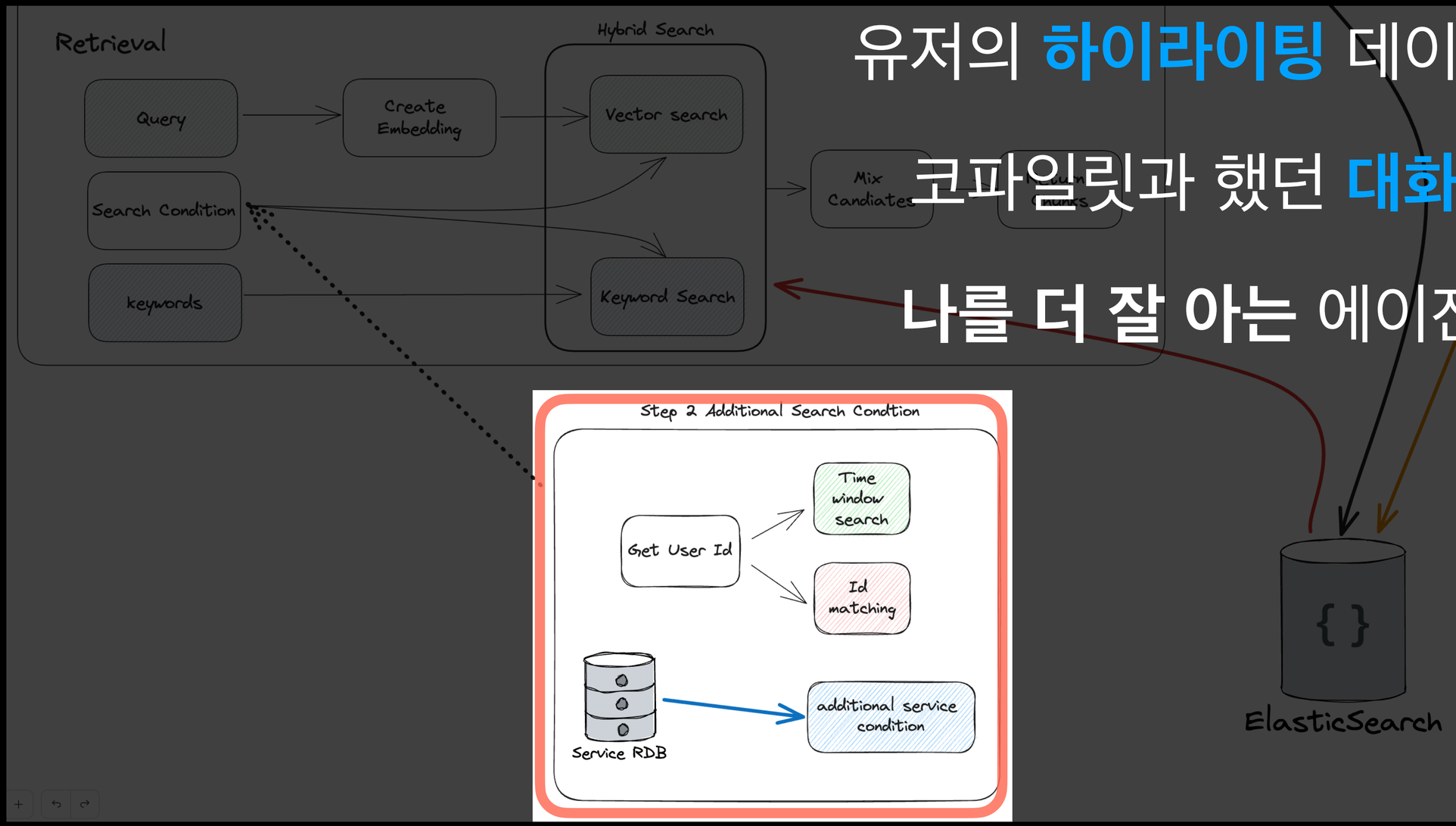
```
.query(
    ScriptScoreQueryBuilder(
        BoolQueryBuilder()
            .filter(
                TermQueryBuilder( fieldName: "content_type", sourceType.name),
            )
            .filter(
                TermQueryBuilder( fieldName: "content_id", sourceId),
            )
            .should(
                MatchQueryBuilder( fieldName: "content", queryTerm)
            ),
        Script(
            Script.DEFAULT_SCRIPT_TYPE,
            Script.DEFAULT_SCRIPT_LANG,
            idOrCode: "cosineSimilarity(params.query, 'content_vector') + _score",
            mapOf("query" to vector),
        ),
```

Bool 쿼리 **should**에 의해
Chunk 내용 **매칭 점수** 계산

Chunk 벡터에 대해
코사인 유사도 계산

스코어 휴리스틱 **개선 여지**

서비스 데이터 맥락 제공 (추가 예정)



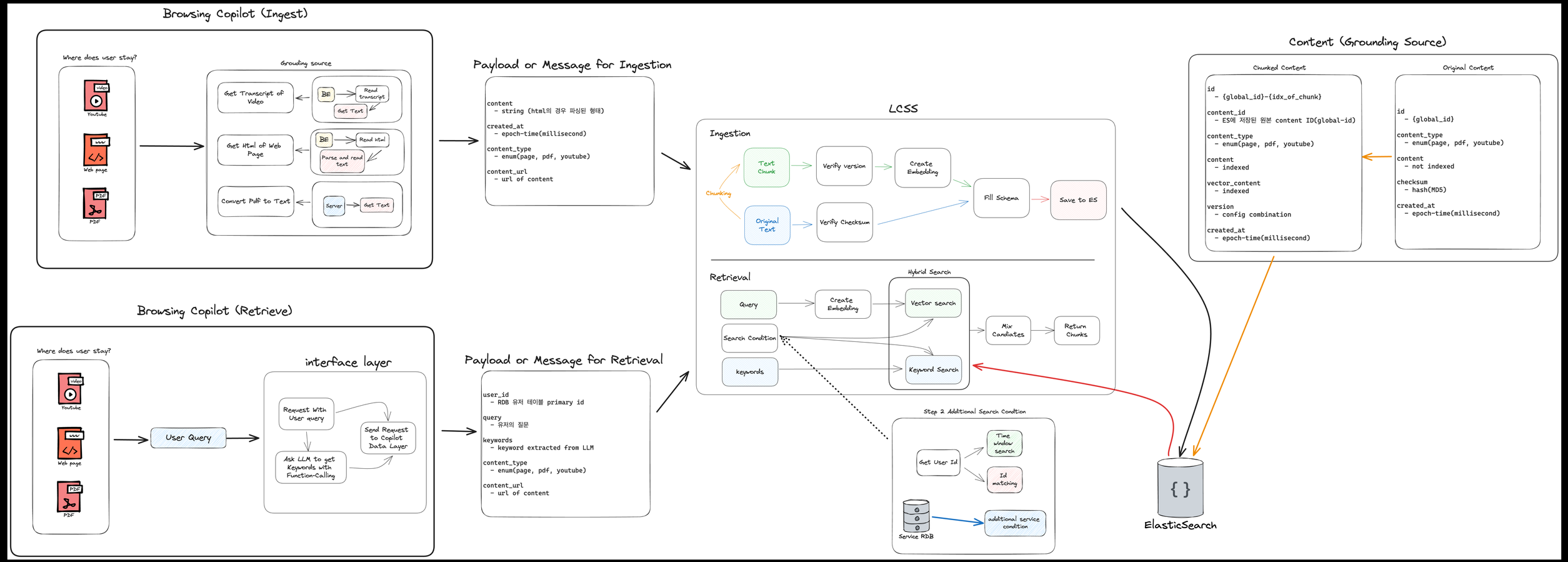
유저의 **하이라이팅** 데이터 제공

코파일릿과 했던 **대화** 제공

나를 더 잘 아는 에이전트로

Copilot Sourcing System Architecture

전처리부 + 인입부 + 조회부

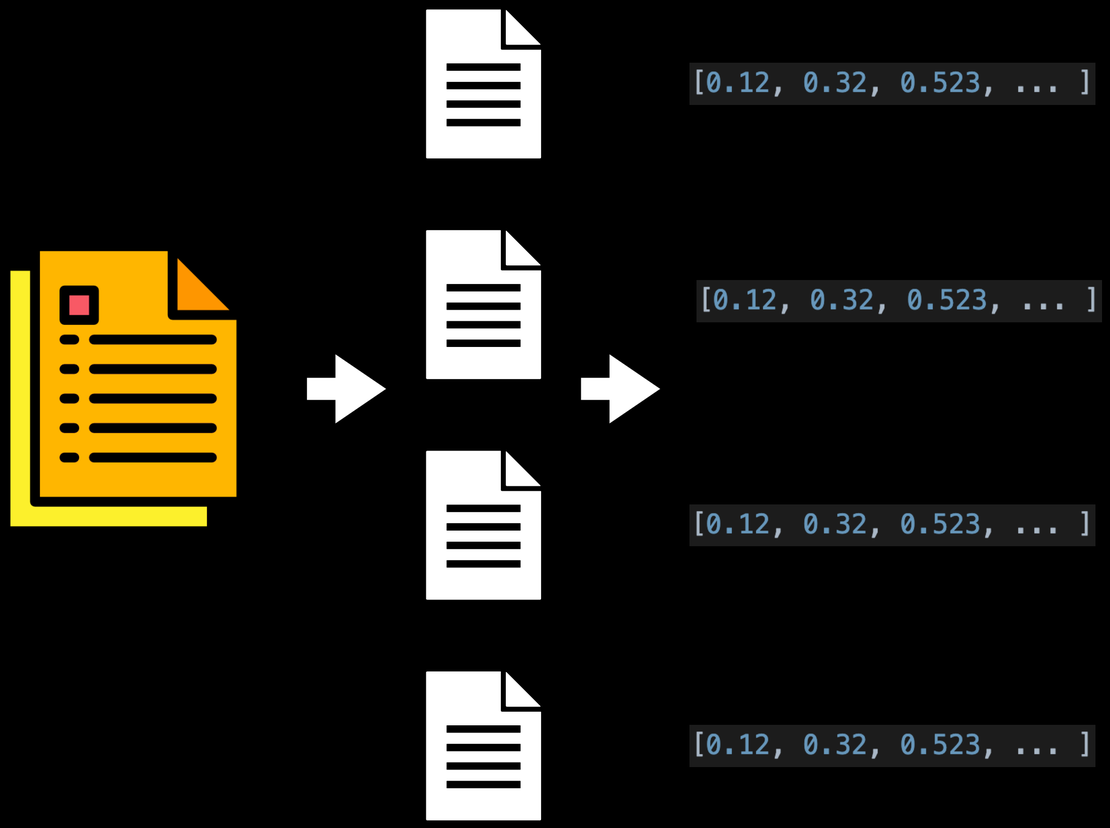


Engineering Challenges 🧐

청킹 사이즈? ?

만약 청크를 잘게 쪼갠다면?

e.g. 문장 단위



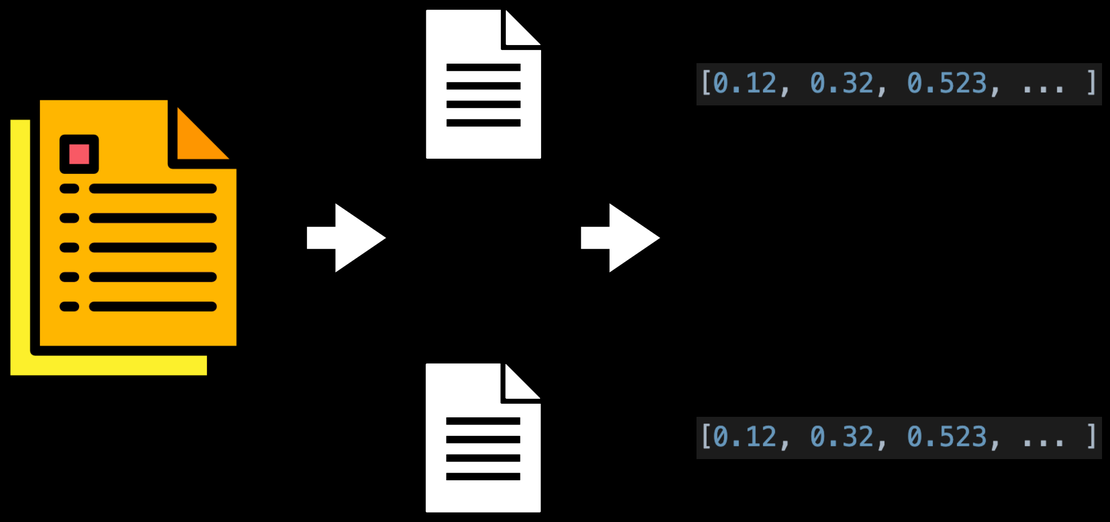
필요한 내용만 조회

임베딩 노이즈 감소

벡터 검색 정확도 상승

만약 청크를 크게 쪼갠다면?

e.g. 페이지 단위



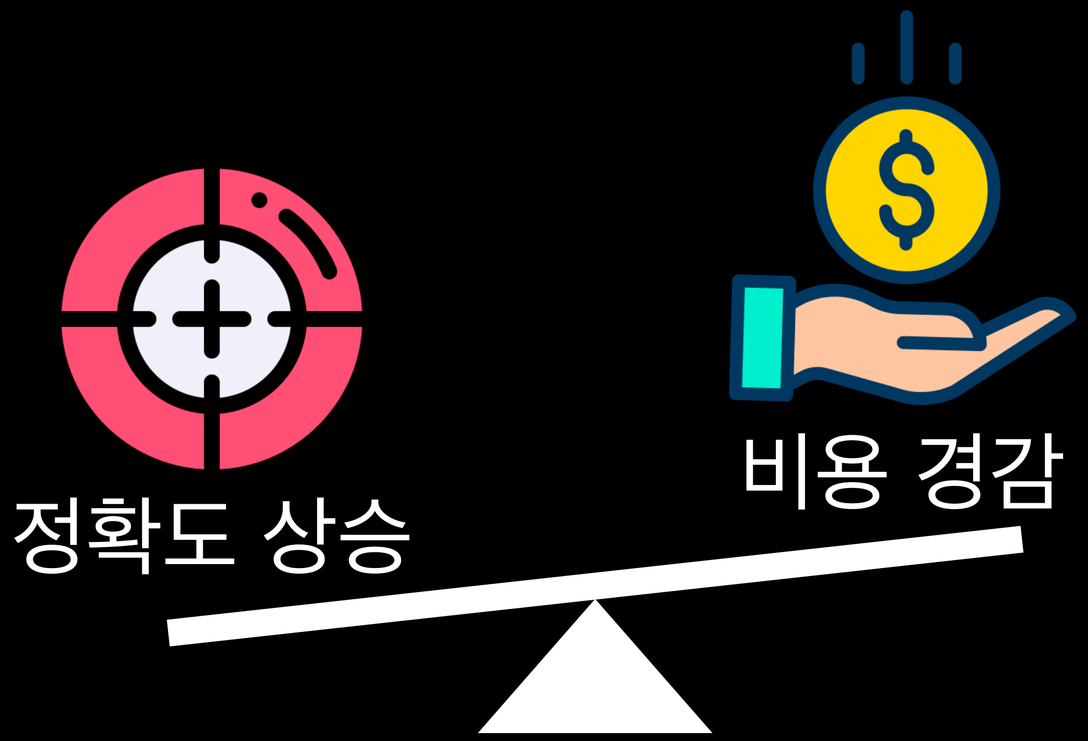
임베딩 생성 비용 경감

청크 저장 비용 경감

문서 검색 비용 경감

청크 사이즈 의사결정

검색 정확도 vs 검색,저장 비용



평가 시스템 이전에는
비용 경감 부분이 **부각**

검색 효용 측정을 위해
평가 시스템 고도화가 필요

IO 작업 문제 없이 처리하기 ⍰

인입부 동시성

Original, Chunk 동시에 처리

```
private suspend fun createSource(
    url: String,
    sourceType: SourceType,
    content: String
): Result<Source> =
    sourceFactory.make(url, sourceType, content)
        .also { source ->
            val originalSourceChanges = sourceChangeObserver.originalSourceChanges(source)
            coroutineScope { this: CoroutineScope
                listOf(
                    async { this: CoroutineScope
                        saveOriginalSource(source, originalSourceChanges)
                    },
                    async { this: CoroutineScope
                        saveChunkedSource(source, originalSourceChanges)
                    }
                ).awaitAll()
```

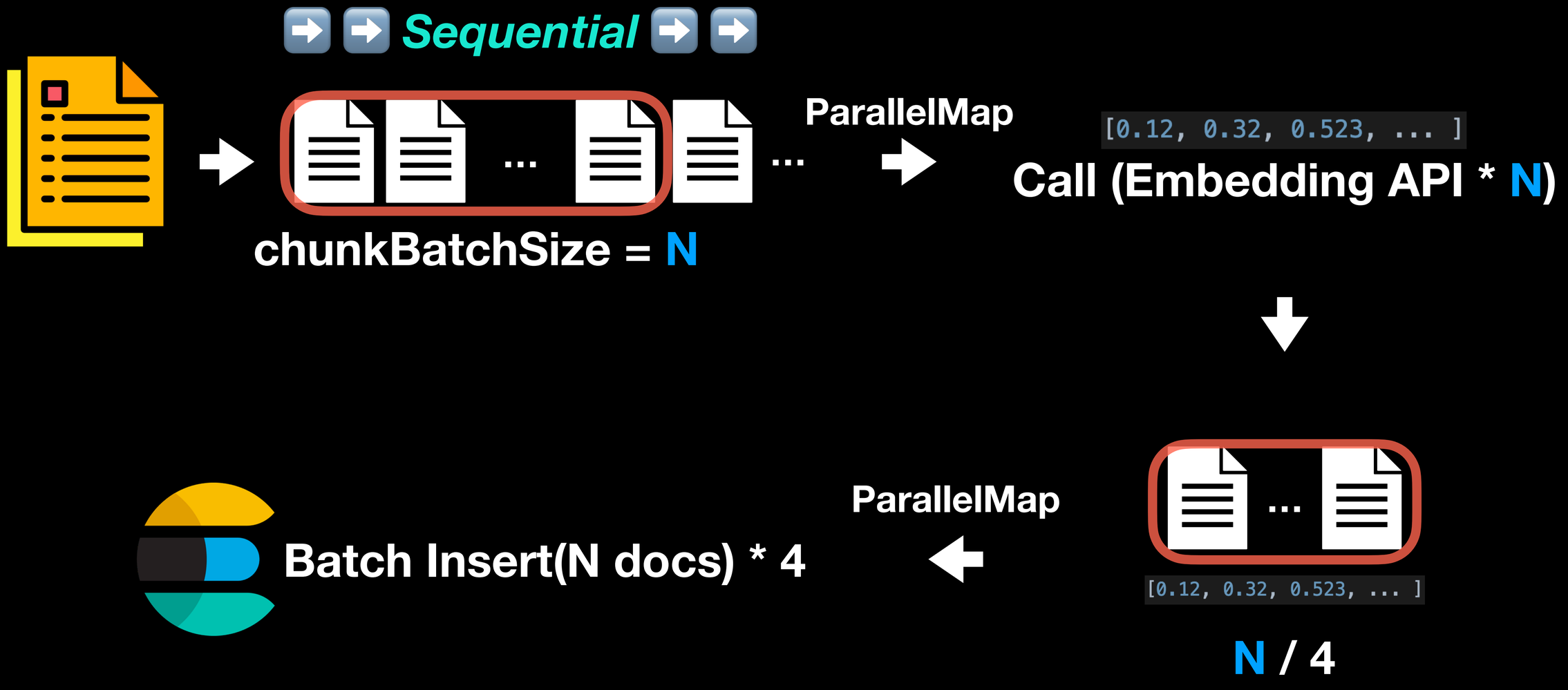
인입부 동시성

부분 동시성 활용

```
.flatMap { sourceChunks ->
    sourceChunks.chunked(chunkBatchSize)
        .flatMap { chunks ->
            chunks.parallelMap { sourceChunk ->
                embeddingMaker.makeEmbeddingInternallyMono(sourceChunk.contentChunk) Result<List<Double>>
                    .getOrNull() List<Double>?
                    .let { vectorOfContent ->
                        sourceChunk.copy(
                            vectorOfContentChunk = vectorOfContent ?: emptyList()
                        )
                    }
            }.chunked( size: chunkBatchSize / 4).parallelMap { it: List<SourceChunk>
                elasticSearchRepository.saveSourceChunks(it)
            }
```

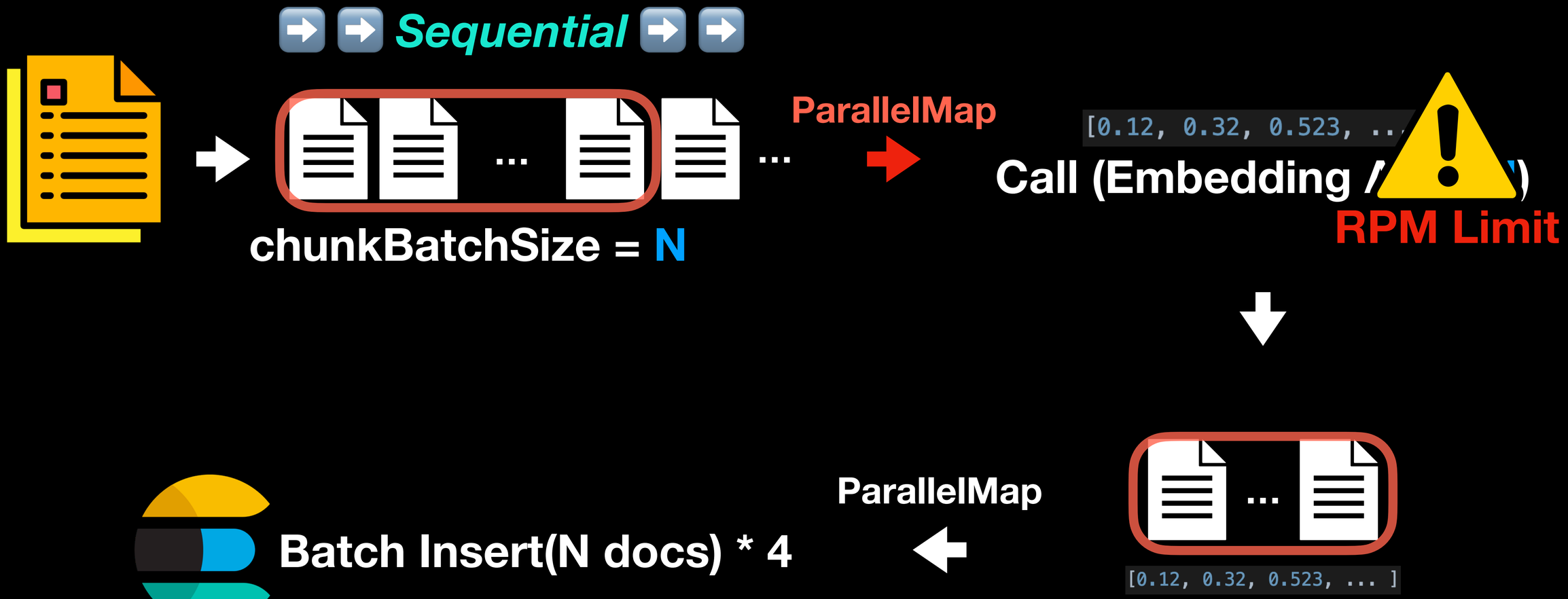
인입부 동시성

부분 동시성 활용



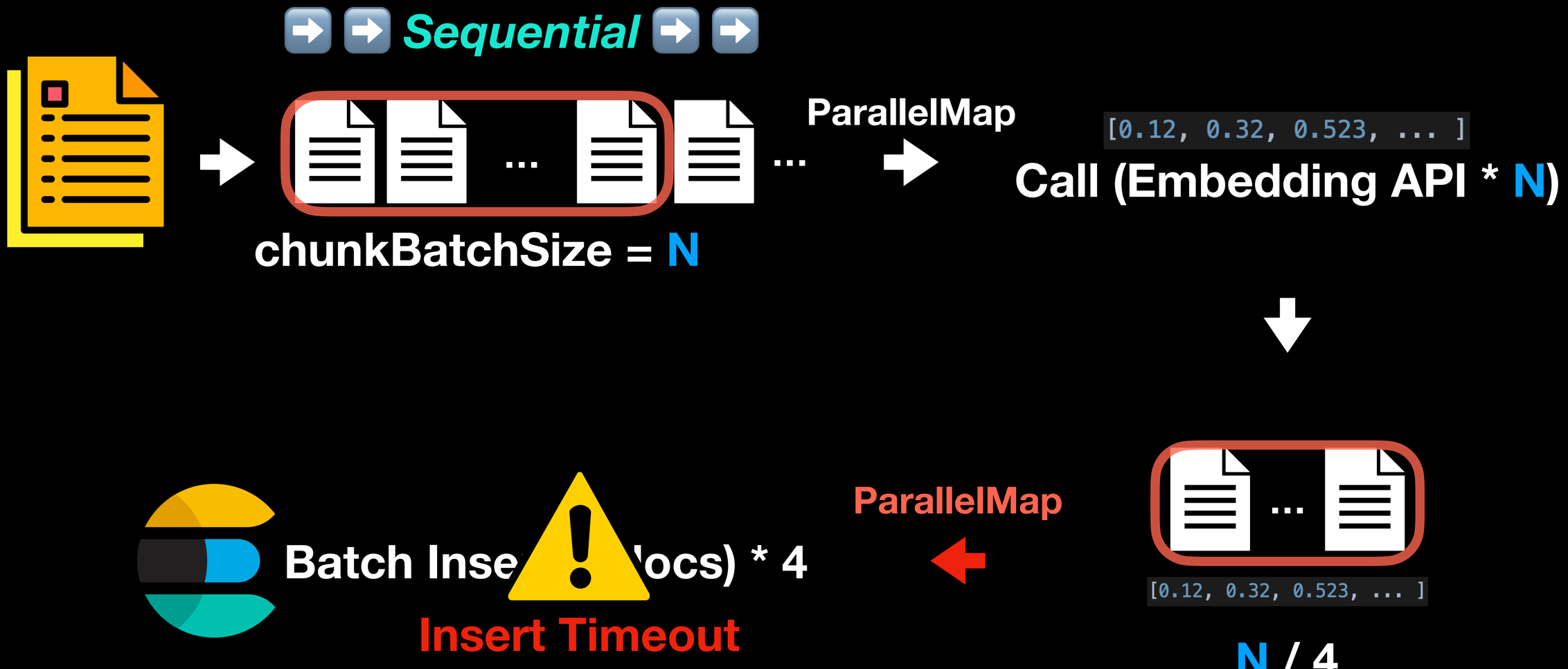
인입부 동시성

임베딩 API RPM 제한



인입부 동시성

ES 인덱싱 큐 사이즈 제한



인입부 동시성

IO 제한에 맞추어 제한된 동시성 구현

RPM Limit

```
.flatMap { sourceChunks ->
    sourceChunks.chunked(chunkBatchSize)
        .flatMap { chunks ->
            chunks.parallelMap { sourceChunk ->
                embeddingMaker.makeEmbeddingInternallyMono(sourceChunk.contentChunk)  Result<List<Double>>
                    .getOrNull()  List<Double>?
                    .let { vectorOfContent ->
                        sourceChunk.copy(
                            vectorOfContentChunk = vectorOfContent ?: emptyList()
                        )
                    }
            }.chunked( size: chunkBatchSize / 4).parallelMap {  it: List<S                k>
                elasticSearchRepository.saveSourceChunks(it)
            }
```

Insert Timeout

검색 키워드 제대로 뽑기 

AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework

Qingyun Wu[1], Gagan Bansal[2], Jieyu Zhang[3], Yiran Wu[1], Shaokun Zhang[1], Erkang Zhu[2], Beibin Li[2], Li Jiang[2], Xiaoyun Zhang[2], and Chi Wang[2]

[1] *Pennsylvania State University*
[2] *Microsoft*
[3] *University of Washington*

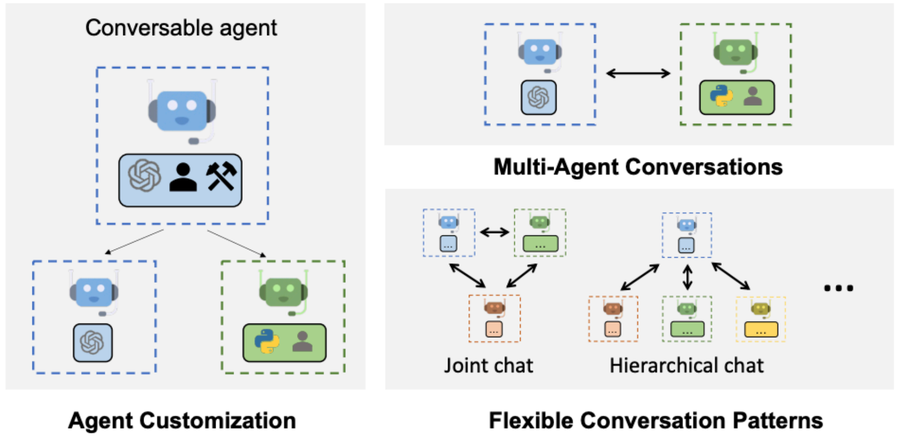


Figure 1: AutoGen enables complex LLM-based workflows using multi-agent conversations. (Left)

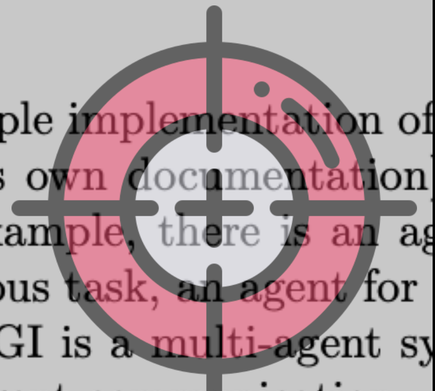
What is the BabyAGI?

본문 일부 제공
키워드 5개+ 추출

babyAGI, AutoGen…

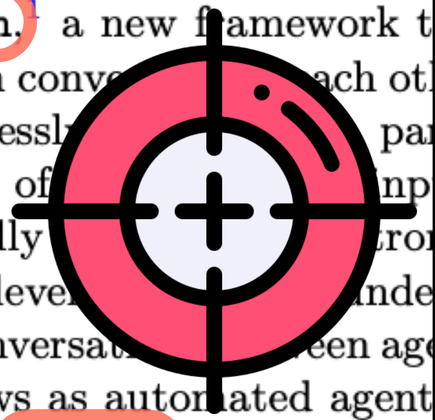
Multi-Agent Systems:

- **BabyAGI**: BabyAGI [6] is an example implementation of in a Python script (according to its own documentation LLM-based agents are used. For example, there is an ag objective and the result of the previous task, an agent for completing tasks/sub-tasks. BabyAGI is a multi-agent sy pattern, i.e., a predefined order of agent communication.
- **CAMEL**: CAMEL [28] is a communicative agent frame



Abstract

This technical report presents `AutoGen`, a new framework t applications using multiple agents that can conv ach ot are customizable, *conversable*, and seamlessl pa various modes that employ combinations of np offers multiple advantages: a) it gracefully ro reasoning abilities of these LLMs; b) it leve nde providing valuable automation through conversat een ag implementation of complex LLM workflows as automated agent examples of how developers can easily use `AutoGen` to effectively



babyAGI, **AutoGen**…

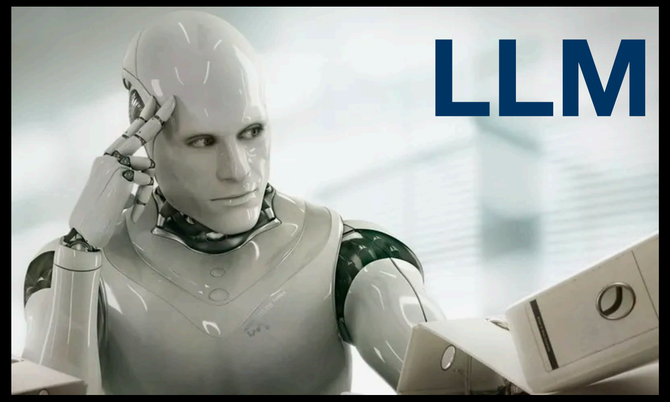
주요 키워드들이 매칭
정답 청크의 스코어가
상대적으로 **낮아짐**

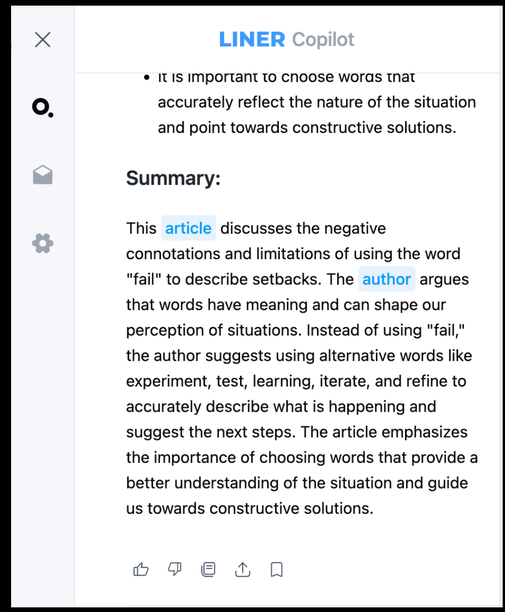
본문을 보지 않고
동의어 수준에서 1~2개만 생성

Future Works 😏

RAG Evaluation 

RAG Flow

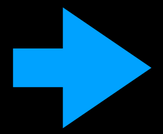


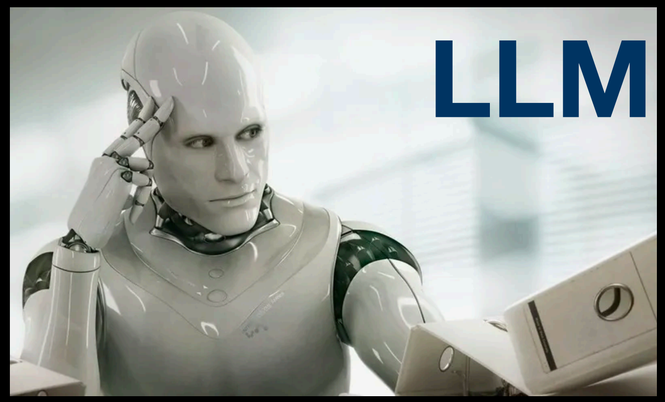


Ground Source | **Prompt + LLM** | **Generation Result**

Retrieval 평가 지표

가져온 것중 **맞게 가져온** 비율 *(signal to noise)*
전체 정답에서 **가져온** 비율 *(recall)*



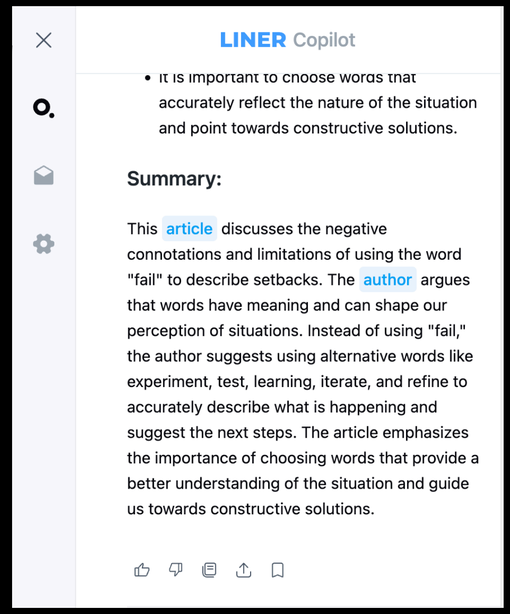


Ground Source

Prompt + LLM

Generation 평가 지표

소스 내용 **그대로 답**했나 ***(anti-hallucination)***
질문과 답변의 **연관성** 체크



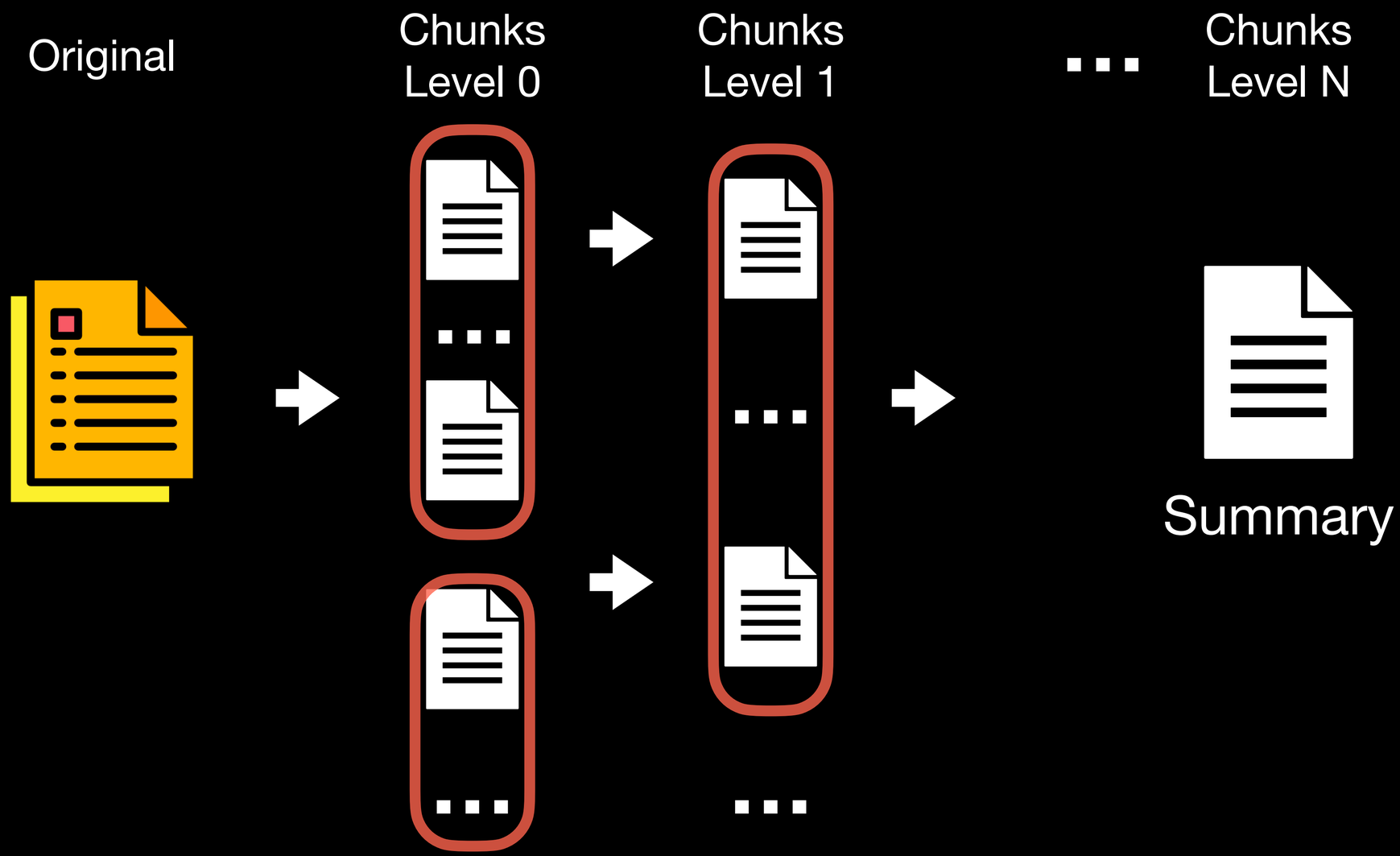
Ground Source **Prompt + LLM** **Generation Result**

콘텐츠에 대한 깊은 이해 🍒

- Hierarchical Search
- 멀티-벡터 서치

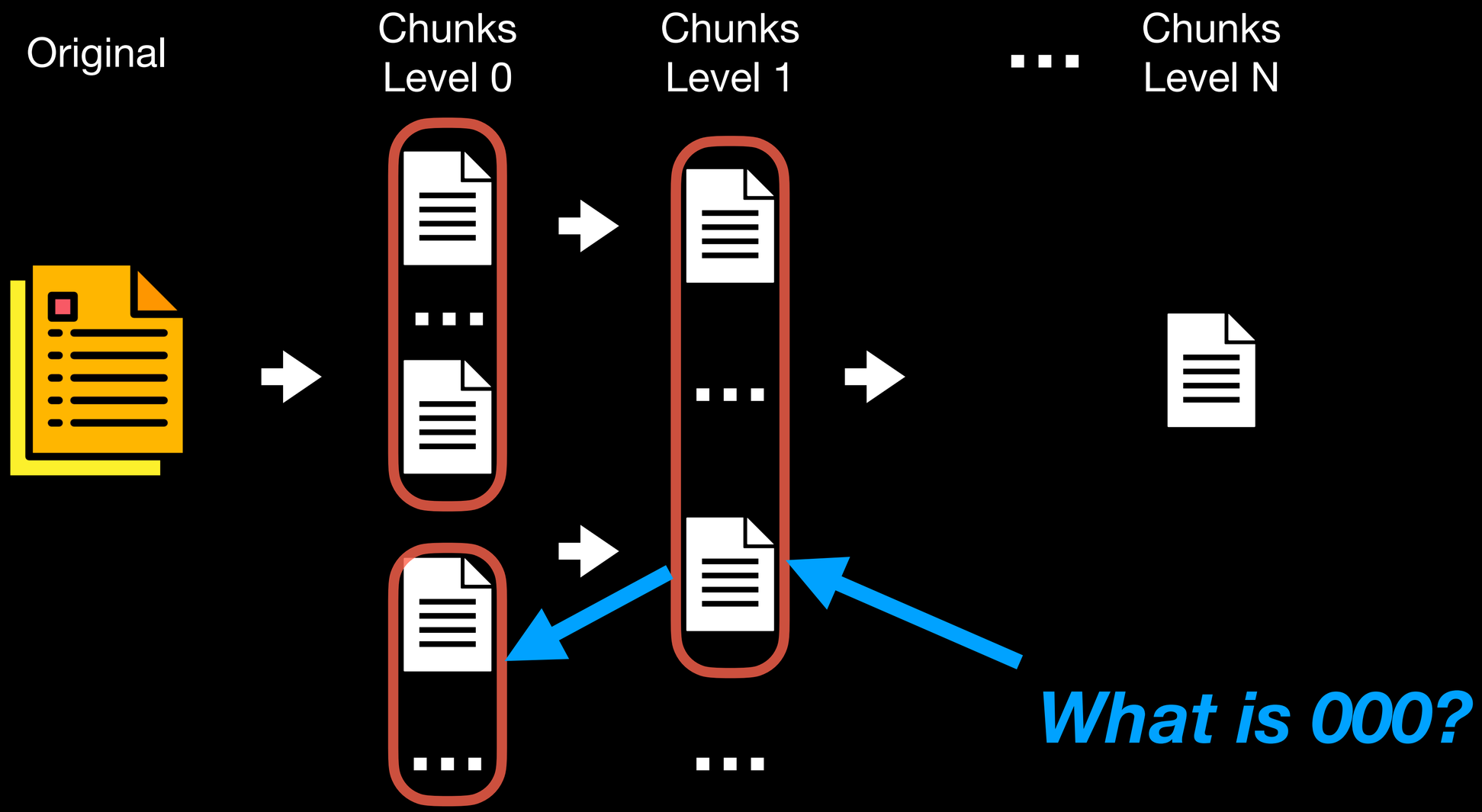
계층 구조 검색

콘텐츠를 트리 형식으로 분석 / 이해



계층 구조 검색

상위 청크에 매칭+검색, 하위로 추가 정보 탐색



멀티 벡터 검색

원문을 다양한 의미의 벡터로 표현

Chunk로 쪼개서 임베딩

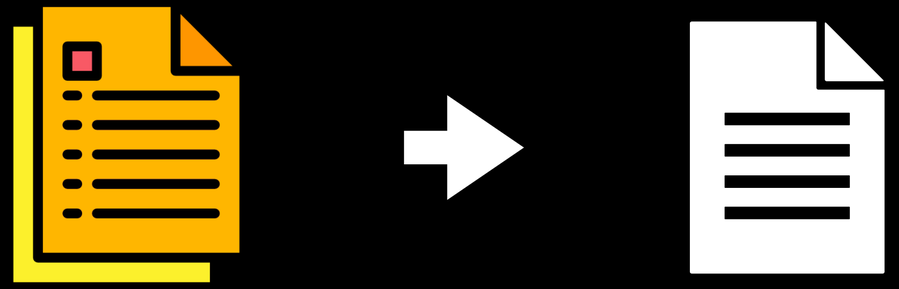
Original

Chunks

멀티 벡터 검색

원문을 다양한 의미의 벡터로 표현

Summary를 만들어 임베딩



Original

Summary

멀티 벡터 검색

원문을 다양한 의미의 벡터로 표현

예상 질문 만들어서 임베딩

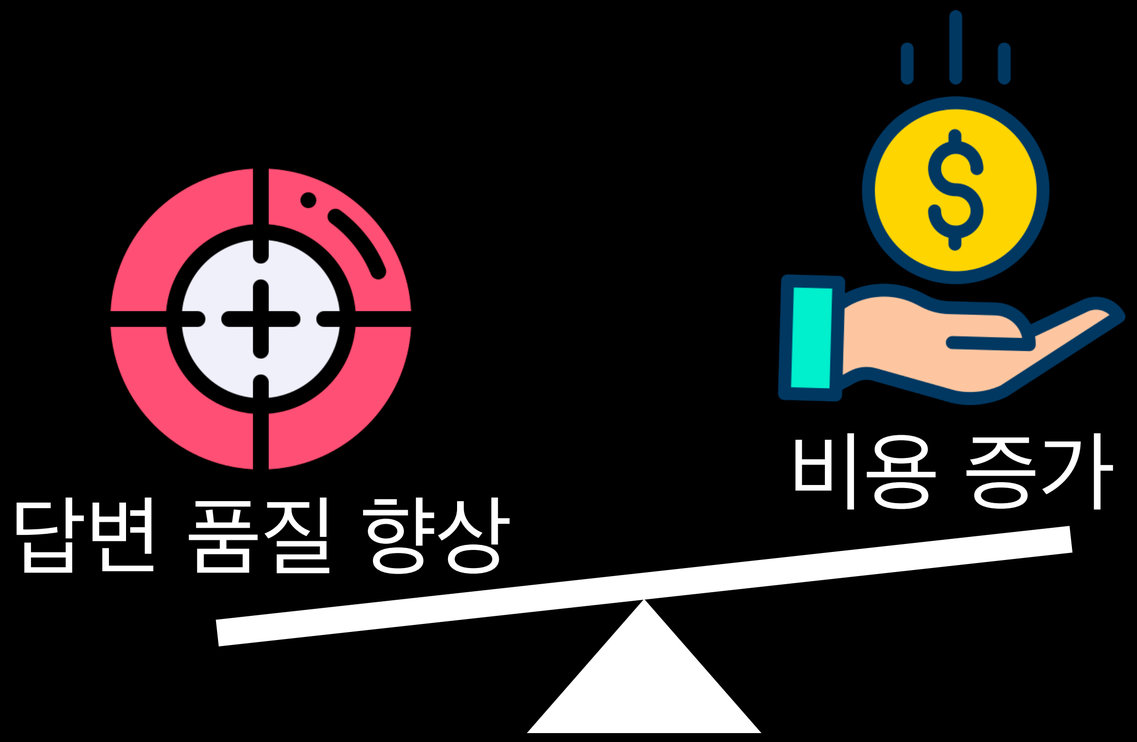
Original

가능한 + 예상 질문

User: What is 000?

어떤 콘텐츠를 심층 분석해야되나?

답변 효용 vs 콘텐츠 분석 비용



자주 쓰는 문서에 대해서는
적용 **가능성** 고려

평가를 통해
답변 향상 **측정 필요**

No Answer
Only Trade-off

Investment vs Return

To Be Continued...