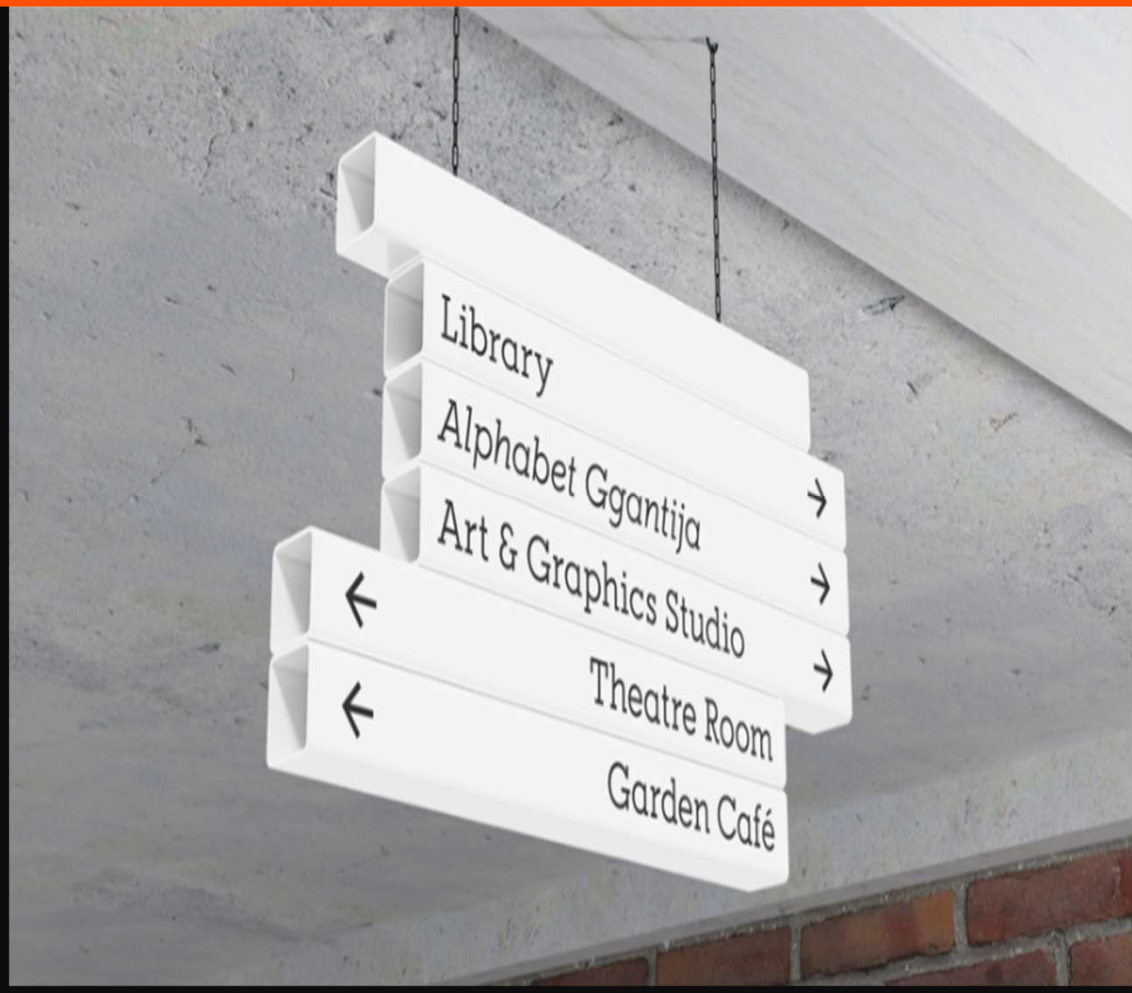

# AI 6기 3팀
# 중급 프로젝트
# 입찰메이트 AI Chatbot

박상진, 김나연, 서유종, 안호성, 장우정

# 목차 a table of contents





# 1 프로젝트 개발 배경 및 목적

RFP 분석의 접근성을 극대화한 **입찰메이트AI** 구현

## 개발 배경 Problem

복잡한 RFP 분석의 어려움: 방대한 기업/정부 제안요청서(RFP)에서 핵심 정보를 신속하게 파악하기 어려움

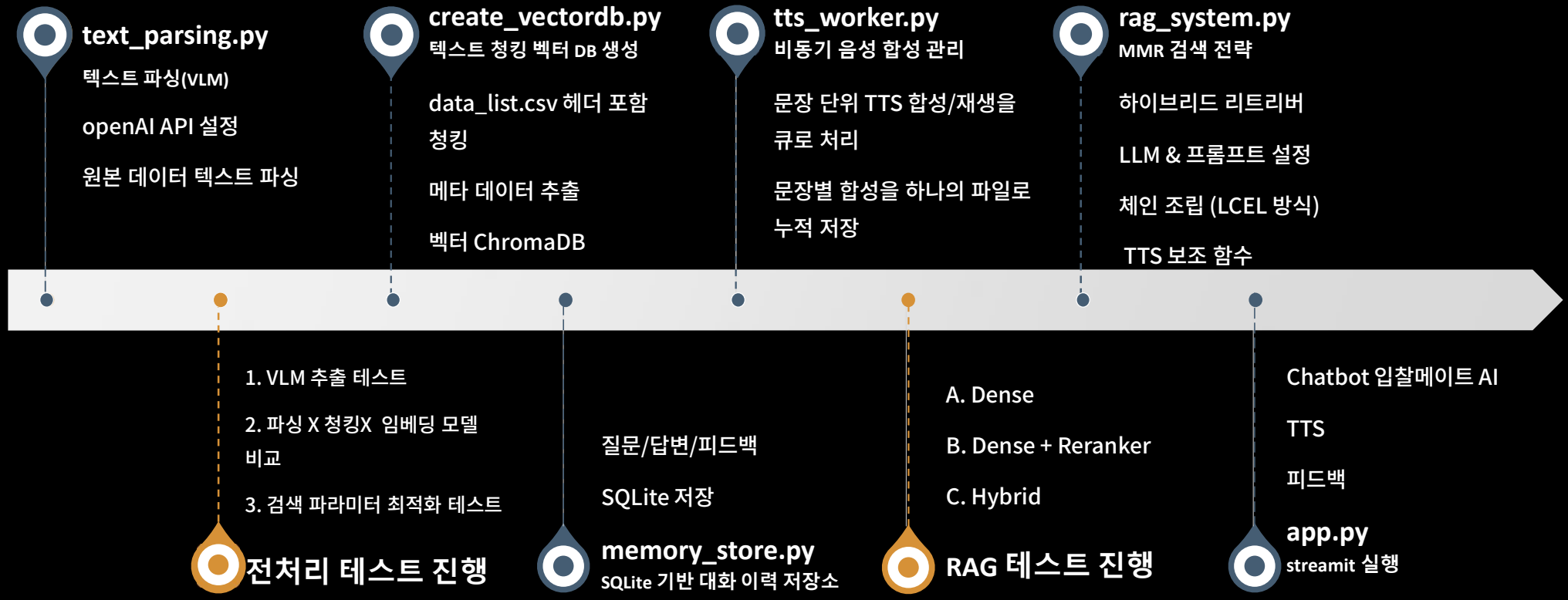
## 핵심 솔루션 Solution

- 직관적 접근성 (Conversational UI): 복잡한 검색 조건 설정 없이 자연어 대화만으로 쉽게 RFP 정보 탐색 가능
- 핵심 정보 파악 (Hybrid Retrieval): 키워드 검색과 벡터 검색을 결합하여 문맥과 수치적 정보 확보
- 사용성 개선 (Streaming TTS): 실시간 음성 합성 기술을 통해 시각적 피로도 해소, 멀티태스킹 지원


## 프로젝트 목표 Goal

복잡한 입찰 공고의 분석 시간 단축 및 업무 효율성 증대

# 2 파이프라인


text_parsing.py
텍스트 파싱(VLM)
openAI API 설정
원본 데이터 텍스트 파싱
create_vectordb.py
텍스트 청킹 벡터 DB 생성
data_list.csv 헤더 포함 청킹
메타 데이터 추출
벡터 ChromaDB
tts_worker.py
비동기 음성 합성 관리
문장 단위 TTS 합성/재생을 큐로 처리
문장별 합성을 하나의 파일로 누적 저장
rag_system.py
MMR 검색 전략
하이브리드 리트리버
LLM & 프롬프트 설정
체인 조립 (LCEL 방식)
TTS 보조 함수
1. VLM 추출 테스트
2. 파싱 X 청킹X 임베딩 모델 비교
3. 검색 파라미터 최적화 테스트
전처리 테스트 진행
질문/답변/피드백
SQLite 저장
memory_store.py
SQLite 기반 대화 이력 저장소
A. Dense
B. Dense + Reranker
C. Hybrid
RAG 테스트 진행
Chatbot 입찰메이트 AI
TTS
피드백
app.py
streamit 실행

# 2 문서 파싱 과정

## 원본 데이터

- HWP 96건
- PDF 4건

>>

## HWP → PDF변환

전체 문서를
PDF 단일 포멧으로 통일

>>

## PDF → MD 변환

표, 그림도 추출하기 위해
.txt 가 아닌 .md 파일로 텍스트 추출

>>

## 표, 그림 추출

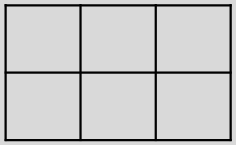
VLM 을 활용하여 표, 그림을 추출
페이지에 형식에 맞게 삽입

```
<!-- tables: start page 24 -->
**[표] 아시아프로젝트마켓(apm.biff.kr) 국/영 문업**
| 사이트 | 1차 메뉴 | 2차 메뉴 | 3차 메뉴 |
|---|---|---|---|
| 아시아프로젝트마켓 | 아시아프로젝트마켓 | 소개<br>참가안내<br>스폰서&어워드<br>파트너<br>연락처 | 소개<br>결산 |
| | 프로젝트 접수 | 2023 모집요강<br>자주하는 질문 | |
| | 2023 프로젝트 | 2023 프로젝트 | 개요 |
| | 역대프로젝트 | 역대 프로젝트 (1998~2022 프로젝트)<br>완성작 | 선정프로젝트 |
| | 뉴스 | 공지사항<br>포토갤러리 | |

**[표] Asian Film Academy (afa.biff.kr) 국/영 문업**
| 사이트 | 1차 메뉴 | 2차 메뉴 |
|---|---|---|
| Asian Film Academy (아카이브) | 아시아영화아카데미 | 소개<br>장학기금<br>주최 및 파트너 |
| | AFA2019 | 프로그램<br>멘토<br>강사진<br>2019 참가자<br>일정표<br>포토앨범<br>행사 장소 |
| | 아카이브 | 졸업생정보(2005~2019)<br>졸업작품<br>포토 갤러리 |
| | 뉴스 | 공지사항 |

<!-- tables: end page 24 -->
```

### 페이지 형식

표의 시작
(<!--tables:start page 1-->

페이지 구분
(<!--page:1-->



표의 끝
(<!--tables:end page 1-->

# 3 테스트 과정

**STEP 1**

**Parsing :**
VLM 기반 이미지-텍스트 멀티모달 추출 비교 테스트

>>

**STEP 2**

**Chunkig , Embedding :**
VLM 모델 2가지 , 팀원별 청킹 , 임베딩 모델 3가지 정량 평가

>>

**STEP 3**

**Parmeter:**
Grid Search 을 이용하여 Chunk Size, K 값 비교

>>

**STEP 4**

**RAG:**
3단계 품질 테스트

A. Dense
Chrmoa MMR(k=5)

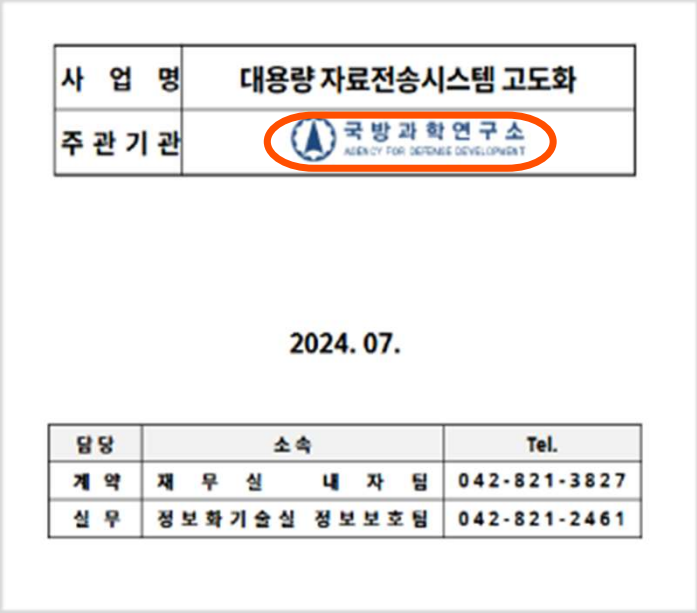
B. Dense + Rerank
Chrmoa -> Reranker 로 재정렬

C. Hybrid
Dense 0.6+ BM25 0.4

# 3 비교 테스트 : Parising

VLM 기반의 이미지-텍스트 멀티모달 추출 구현 테스트

**Q. “자료전송시스템  고도화의 주관기관과 담당 소속 전화번호를 알려줘”**



| 사 업 명 | 대용량 자료전송시스템 고도화 |
|---|---|
| 주 관 기 관 | 국 방 과 학 연 구 소 AGENCY FOR DEFENSE DEVELOPMENT |

2024. 07.

| 담당 | 소속 | Tel. |
|---|---|---|
| 계 약 | 재 무 실 내 자 팀 | 042-821-3827 |
| 실 무 | 정보화기술실 정보보호팀 | 042-821-2461 |

▲원본 pdf 파일


▲ 텍스트만 추출 -> 전화번호는 찾아내지만 정보가 왜곡


▲ 표와 그림까지 추출 -> 정확한 정보 제공

# 3 테스트 과정

**STEP 1**

**Parsing :**
VLM 기반 이미지-텍스트 멀티모달 추출 비교 테스트

>>

**STEP 2**

**Chunkig , Embedding :**
VLM 모델 2가지 , 팀원별 청킹 , 임베딩 모델 3가지 정량 평가

>>

**STEP 3**

**Parmeter:**
Grid Search 을 이용하여 Chunk Size, K 값 비교

>>

**STEP 4**

**RAG:**
3단계 품질 테스트

A. Dense
Chrmoa MMR(k=5)

B. Dense + Rerank
Chrmoa -> Reranker 로 재정렬

C. Hybrid
Dense 0.6+ BM25 0.4

# 3 비교 테스트 : Parsing

팀원 별 청킹 구조 X VLM 모델(2개) X 임베딩 모델 (3개) 비교

## 박상진

```python
def paragraph_chunking(
    text: str,
    doc_id: str,
    min_chars: int = 200,
    max_chars: int = 800,
    overlap: int = 100,
    metadata: dict = None,
) -> List[Chunk]:
    # 1. 빈 줄 기준 문단 분리
    paragraphs = [p.strip() for p in text.split("\n\n") if p.strip()]
    # 2. 짧은 문단 합치기 , 긴 문단 분리
    # 3. overlap 적용
    return go(f, seed, [])
```

특징: 외부 의존성 없는 적응형 청킹

## 김나연

```python
# Context Enrichment
enriched_content = f"""[[사업 개요]]
사업명: {metadata['title']}
발주기관: {metadata['agency']}
공고번호: {metadata['notice_id']}

[[본문]]
{content}"""

# RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200,
    separators=["\n\n", "\n", ". ", " ", ""]
)
```

특징: 메타데이터 주입으로 문맥 보존

## 서유종

```python
from langchain_experimental.text_splitter import SemanticChunker

embeddings = HuggingFaceEmbeddings(model_name="jhgan/ko-sroberta-multitask")

text_splitter = SemanticChunker(
    embeddings,
    breakpoint_threshold_type="percentile"
)
```

특징: 임베딩 기반 의미 분석,
문장 중간 끊김 없이 분할

## 안호성

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=overlap,
    separators=["\n\n", "\n", "□", "。", ".", "!", "?", " ", ""],
)
pieces = [c for c in splitter.split_text(text) if c.strip()]
```

특징: LangChain 라이브러리 사용,
재귀적 분할

## 장우정

```python
class HierarchicalChunkerV2:
    def __init__(self, chunk_size=1000, overlap_ratio=0.2):
        self.hierarchy_patterns = [
            (1, re.compile(r"^[ I II III IV V VI VII VIII IX X ]+[\.\s]")),  # Level 1
            (2, re.compile(r"^(\d+)[\.\)]\s")),                              # Level 2
            (3, re.compile(r"^([가나다라마바사][\.\)]\s")),                  # Level 3
        ]
        self.table_detector = TableDetector()
```

특징: 로마숫자, 가나다 한글 분할,
테이블 감지

# 3 비교 테스트 : Dataset

데이터셋 분포 및 실제 질문

**Dataset1.json**

총 질문 : 40개

특징 : 기술 요건 중심

(예산, 일정, 규격 등 정량데이터)

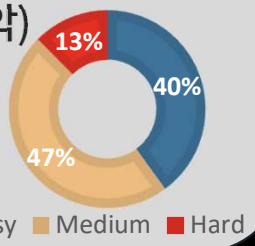
**Dataset2.json**

총 질문 : 40개

카테고리 : 업무 범위 중심

(과업 내용, 제안 요청 사항 등 문맥 파악)

비슷한 난이도




**EASY**

부산국제영화제(BIFF) & ACFM 온라인서비스 재개발 사업의 총 예산은 얼마인가요?

고려대학교 차세대 사업은 다년 사업인가요 단년 사업인가요?

**Medium**

KUSF 사업의 추진 배경이 된 스포츠혁신위원회의 권고문 발표 날짜는 언제인가요?

인천공항운영서비스에서 기존에 사용하던 ERP 패키지의 명칭은 무엇인가요?

**HARD**

예술경영지원센터 컨설팅 사업 추진의 근거가 되는 법률은 무엇인가요?

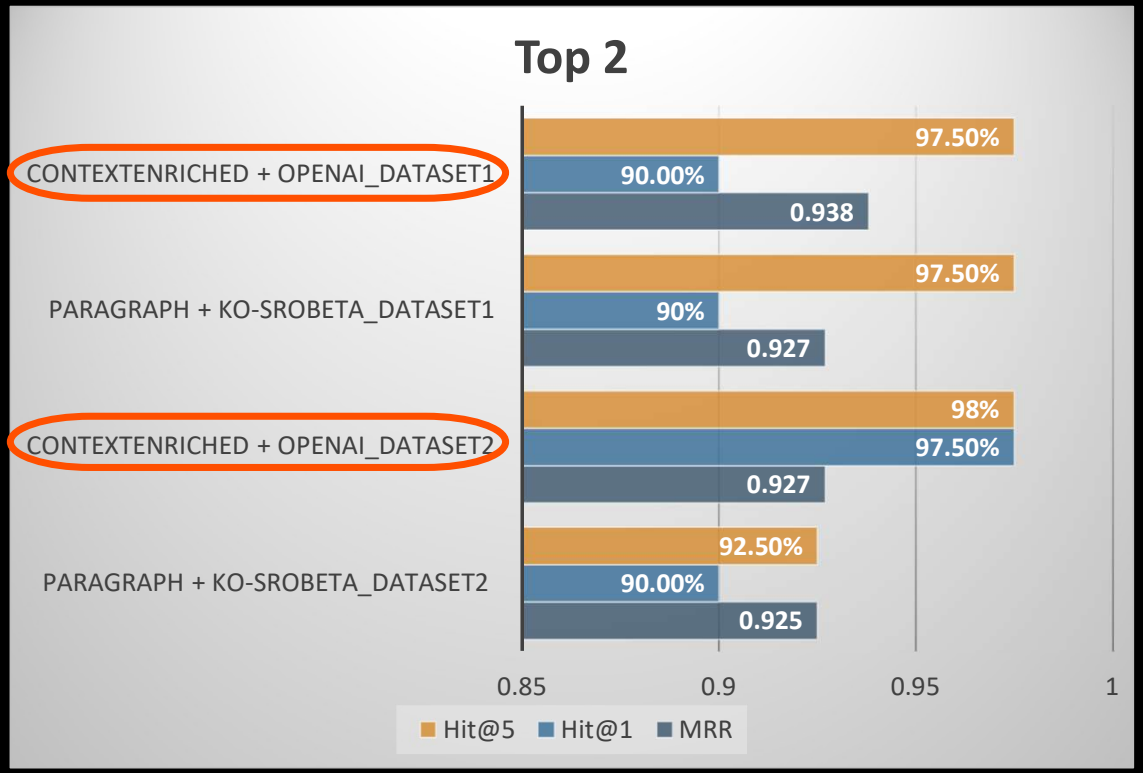
한국한의학연구원 시스템에서 연구계획 관리를 위해 등록해야 하는 기술완성수준 지표 3가지는 무엇인가요?

# 3 비교 테스트 : Chunking, Embeding

VLM 파싱 모델 : QWEN3 vs OpenAI GPT-5
임베딩 모델 : ko-srobeta vs openAI vs MiniLM

| 팀원별 청킹 방법 | QWEN3 청크 수 | OpenAI 청크 수 | 변화율 |
|---|---|---|---|
| 박상진 - Paragraph | 11,764 | 11,332 | -3.7% |
| 김나연 - ContextEnriched | 8,622 | 9,248 | +7.3% |
| 서유종 - Semantic | 8,622 | 8,246 | -4.4% |
| 안호성 - Recursive | 8,622 | 9,248 | +7.3% |
| 장우정 - Hierarchical | 8,622 | 9,437 | +9.5% |

OpenAI 파싱이 7~9% 더 많은 청크 생성
-> 텍스트 추출 품질이 더 세밀하다고 판단




가장 안정적인
ContextEnriched + openAI 으로 결정

# 3 테스트 과정

**STEP 1**

**Parsing :**
VLM 기반 이미지-텍스트 멀티모달 추출 비교 테스트

>>

**STEP 2**

**Chunkig , Embedding :**
VLM 모델 2가지 , 팀원별 청킹 , 임베딩 모델 3가지 정량 평가

>>

**STEP 3**

**Parmeter:**
Grid Search 을 이용하여 Chunk Size, K 값 비교

>>

**STEP 4**

**RAG:**
3단계 품질 테스트

A. Dense
Chrmoa MMR(k=5)

B. Dense + Rerank
Chrmoa -> Reranker 로 재정렬

C. Hybrid
Dense 0.6+ BM25 0.4

# 3 비교 테스트 : Parameter

청킹 사이즈와 K 값을 비교하여 Recall 값 비교
데이터셋: data/dataset1.json, dataset2.json

| Chunk_size / K | 400 | 700 | 900 | 1200 |
|---|---|---|---|---|
| 3 | 52.1% | 58.3% | 54.8% | 49.6% |
| 5 | 56.0% | 67.0% | 60.0% | 60.4% |
| 7 | 60.0% | 71.7% | 68.0% | 66.5% |
| 10 | 67.0% | 72.7% | 70.5% | 71.4% |

Keyword Recall 조합: chunk_size= 700, k=10
Keyword Recall = 72.7%
Hit Rate = 92.5%

QA 포맷 예시

| 필드 | 예시(q001) |
|---|---|
| QID (문항 식별) | q001 |
| Question (검색 질의) | 부산국제영화제(BIFF) & ACFM 온라인서비스 재개발 사업의 총 예산은 얼마인가? |
| Gold(source_docs) | (사)부산국제영화제_2024년.. |
| Gold (chunk_id) | …:chunk1 |
| Gold (answer) | 총 사업예산은 금 243,000,000원(부가세포함) 입니다. |
| Relevant_keywords | [“243,000,000”, “예산”, “부과세“] |
| difficulty | Easy |

# 3 테스트 과정

**STEP 1**

**Parsing :**
VLM 기반 이미지-텍스트 멀티모달 추출 비교 테스트

>>

**STEP 2**

**Chunkig , Embedding :**
VLM 모델 2가지 , 팀원별 청킹 , 임베딩 모델 3가지 정량 평가

>>

**STEP 3**

**Parmeter:**
Grid Search 을 이용하여 Chunk Size, K 값 비교

>>

**STEP 4**

**RAG:**
3단계 품질 테스트

A. Dense
Chroma MMR(k=5)

B. Dense + Rerank
Chroma -> Reranker 로 재정렬

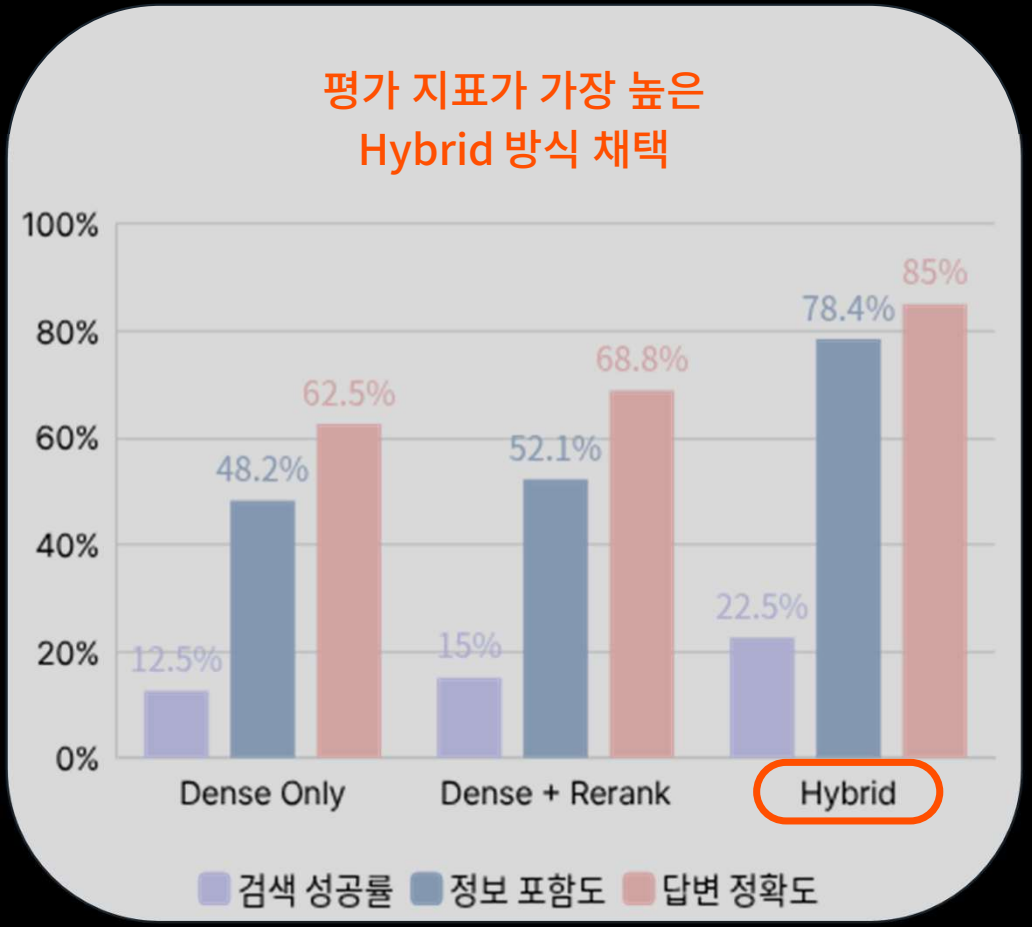
C. Hybrid
Dense 0.6+ BM25 0.4

# 2 비교 테스트 : RAG

검색과 답변 품질 정량 측정

Dense Only vs Dense + Rerank vs Hybrid

| 평가 지표 | 설명 | 평가 대상 |
|---|---|---|
| Retrieval Hit Rate<br>검색 성공률 | 정답 소스 문서가 검색 결과(Top-K)에 포함되었는가? | 검색 정확도 |
| Keyword Recall<br>정보 포함도 | 기대 키워드 중 검색된 텍스트에 포함된 비율 | 정보 충분성 |
| Answer Accuracy<br>답변 정확도 | LLM 담변에 반드시 포함되어야 할 키워드가 있는가? | 생성 품질 |



# 3 RAG 시스템 설계

**STEP 1**

**하이브리드 리트리버**

(1) Chroma Dense
(2) Ensemble

>>

**STEP 2**

**LCEL 체인 조립 · Retrieval → Generation**

>>

**STEP 3**

**LLM 프롬프트 엔지니어링**

>>

**STEP 4**

**실시간 TTS 통합**

# 3 하이브리드 리트리버 - (1) Chroma Dense

```python
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

if not os.path.exists(DB_PATH):
    print(f"오류: DB 경로({DB_PATH})가 존재하지 않습니다.")
    sys.exit()

# [1] Dense Retriever (Chroma)
vectorstore = Chroma(
    persist_directory=DB_PATH,
    embedding_function=embeddings,
    collection_name="bid_rfp_collection"
)

dense_retriever = vectorstore.as_retriever(
    search_type="mmr",
    search_kwargs={"k": 5, "fetch_k": 20} # 하이브리드를 위해 k값 약간 조정
)
```

### Point 01. 임베딩 모델 정의

text-embedding-3-small 모델을 사용하여 자연어 텍스트를 고차원 벡터로 변환합니다. 성능 대비 비용 효율이 우수합니다.

### Point 02. chroma DB 로드

단순 유사도 검색이 아닌 MMR(Maximal Marginal Relevance)을 적용하여, 유사하면서도 내용이 겹치지 않는 다양한 문서를 회수합니다.

### Point 03. MMR 검색 전략

단순 유사도 검색이 아닌 MMR(Maximal Marginal Relevance)을 적용하여, 유사하면서도 내용이 겹치지 않는 다양한 문서를 회수합니다.

# 3 하이브리드 리트리버 - (2) Ensemble

```python
# [2] Sparse Retriever (BM25)
print("BM25 인덱스 생성 중... (데이터 로드)")
try:
    # DB에 저장된 모든 문서를 가져와서 BM25 인덱스를 만듭니다.
    raw_docs = vectorstore.get()
    docs = []
    for i in range(len(raw_docs['ids'])):
        if raw_docs['documents'][i]:
            docs.append(Document(
                page_content=raw_docs['documents'][i],
                metadata=raw_docs['metadatas'][i] if raw_docs['metadatas'] else {}
            ))

    if not docs:
        print("오류: DB에 문서가 비어 있습니다.")
        sys.exit()

    sparse_retriever = BM25Retriever.from_documents(docs)
    sparse_retriever.k = 5  # 키워드 매칭 문서 5개
    print("BM25 인덱스 생성 완료")

except Exception as e:
    print(f"BM25 초기화 실패: {e}")
    sys.exit()

# [3] Ensemble Retriever (Hybrid) - 결합
ensemble_retriever = EnsembleRetriever(
    retrievers=[dense_retriever, sparse_retriever],
    weights=[0.6, 0.4]  # Dense(의미) 60%, Sparse(키워드) 40% 비중
)
```

## Point 01. 키워드 매칭 보완(BM25)

Dense Retriever가 놓치기 쉬운 고유명사, 숫자, 정확한 용어를 BM25 알고리즘(TF-IDF 개선판)으로 보완하여 검색 정확도를 높입니다.

## Point 02. 메모리 내 인덱싱

Chroma DB에 저장된 텍스트 데이터를 로드하여 BM25용 역색인(Inverted Index)을 실시간으로 구축합니다. Document 객체로 변환이 필수적입니다.

## Point 03. 앙상블 가중치 튜닝

Dense(의미)와 Sparse(키워드)의 비중을 0.6 : 0.4로 설정하여, 문맥적 이해를 우선하되 키워드 정확성도 놓치지 않도록 균형을 맞춥니다.

# 3 RAG 시스템 설계

STEP 1

**하이브리드 리트리버**

(1) Chroma Dense
(2) Ensemble

>>

STEP 2

**LCEL 체인 조립 · Retrieval → Generation**

>>

STEP 3

**LLM 프롬프트 엔지니어링**

>>

STEP 4

**실시간 TTS 통합**

# 3 LCEL 체인 조립 · Retrieval → Generation

```python
# (2) 전체 RAG 체인 구성
# 변경점: retriever -> ensemble_retriever로 교체
setup_and_retrieval = RunnableParallel(
    {
        "context": contextualized_question | ensemble_retriever,
        "input": lambda x: x["input"],
        "chat_history": lambda x: x["chat_history"],
    }
)

def format_context_for_prompt(input_dict):
    return {
        "context": format_docs(input_dict["context"]),
        "input": input_dict["input"],
        "chat_history": input_dict["chat_history"]
    }

# 최종 체인: 검색 -> 포맷팅 -> 답변생성 -> 파싱
rag_chain = setup_and_retrieval.assign(
    answer= format_context_for_prompt | qa_prompt | llm | StrOutputParser()
)
```

## Point 01. 병렬 실행 구조

RunnableParallel을 사용하여 검색(Retrieval)과 데이터 패스스루를 동시에 처리합니다. 하나의 `input` 에서 여러 데이터를 파생시켜 컨텍스트 딕셔너리를 구성합니다.

## Point 02. LCEL 파이프라인

Linux 파이프라인(`|`) 구문을 활용하여 로직의 흐름을 선언적으로 정의합니다. 데이터 포맷팅, 프롬프트 주입, LLM 호출, 파싱이 물 흐르듯 연결됩니다.

## Point 03. ASSIGN을 통한 확장

assign 메서드는 이전 단계의 모든 결과(`context`, `history` 등)를 보존하면서, 새로 생성된 `answer` 키를 결과값에 추가합니다.

# 3 RAG 시스템 설계

STEP 1

**하이브리드 리트리버**

(1) Chroma Dense
(2) Ensemble

>>

STEP 2

**LCEL 체인 조립 · Retrieval → Generation**

>>

STEP 3

**LLM 프롬프트 엔지니어링**

>>

STEP 4

**실시간 TTS 통합**

# 3 LLM 프롬프트 엔지니어링 (1)

```python
# ==========================================
# 2. LLM & 프롬프트 설정
# ==========================================
try:
    llm = ChatOpenAI(model=SELECTED_MODEL, temperature=0)
except Exception as e:
    print(f"모델 초기화 오류: {e}")
    sys.exit()

# 2-1. 질문 재구성 (Contextualize Query)
contextualize_q_system_prompt = """
채팅 기록과 최신 사용자 질문이 주어지면,
이 질문이 채팅 기록의 맥락을 참조하고 있을 경우
채팅 기록 없이도 이해할 수 있는 '독립적인 질문'으로 재구성하세요.
질문에 답하지 말고, 질문을 재구성하거나 그대로 반환하기만 하세요.
"""

contextualize_q_prompt = ChatPromptTemplate.from_messages([
    ("system", contextualize_q_system_prompt),
    MessagesPlaceholder("chat_history"),
    ("human", "{input}"),
])

# 질문 재구성 체인
history_aware_retriever = contextualize_q_prompt | llm | StrOutputParser()

# 2-2. 답변 생성 (QA)
qa_system_prompt = """
당신은 공공 입찰(RFP) 분석 전문가 '입찰메이트'입니다.
아래의 [검색된 문서]를 사용하여 질문에 답변하세요.
```

## Point 01. 맥락 인식 질문 재구성

"그거 예산은?"과 같은 모호한 후속 질문을 이전 대화 이력을 참고하여 "A 프로젝트의 예산은 얼마인가요?"라는 완전한 형태의 질문으로 변환합니다.

## Point 02. 페르소나 및 제약조건

시스템 프롬프트에 '입찰 분석 전문가' 역할을 부여하고, "문서에 없는 내용은 답변 금지" 등의 규칙을 명시하여 환각(Hallucination)을 방지합니다.

## Point 03. 동적 이력 주입

MessagesPlaceholder를 사용하여 길이가 가변적인 대화 기록(chat_history)을 프롬프트 템플릿 내 적절한 위치에 유동적으로 삽입합니다.

# 3 LLM 프롬프트 엔지니어링 (2)

```
규칙:
1. 문서를 기반으로 사실만 답변하고, 모르면 "문서에 해당 내용이 없습니다"라고 하세요.
2. 예산, 기간, 날짜 등 숫자를 기재하세요. (숫자 표기 규칙 참고)
3. 답변은 자연스러운 문장으로만 작성하세요. 목록/불릿/표는 쓰지 마세요.
4. 답변은 존댓말로 작성하세요.
5. 문장은 길지 않게 끊어 읽기 쉬운 길이로 유지하세요.
6. 문단은 2~3문장마다 빈 줄(개행 2개)로 구분하세요.
7. 괄호는 쓰지 말고, 목록/헤더/컨텍스트 인용은 문장으로 풀어 작성하세요.
8. 특수문자(% 등)는 한국어로 풀어서 쓰세요.
9. 출력은 10줄을 넘기지 않게 하세요.

영어 표기 규칙:
- 영어 단어는 한국어 음역으로만 표기하세요.
- 예: dashboard -> 대시보드, dataset -> 데이터셋, isp -> 아이에스피, system -> 시스템.

숫자 표기 규칙:
- 금액은 반드시 한글 화폐식으로 작성하세요.
- 예: 35,750,000원 -> 3천 5백 7십 5만원
- 날짜는 'YYYY년 MM월 DD일' 형식으로 작성하세요.
- 예: 2024-06-24 11:00:00 -> 2024년 6월 24일
- 기간은 'N개월', 'N주', 'N일' 형식으로 작성하세요.

[검색된 문서]:
{context}
"""

qa_prompt = ChatPromptTemplate.from_messages([
    ("system", qa_system_prompt),
    MessagesPlaceholder("chat_history"),
    ("human", "{input}"),
])
```

## Point 01. 맥락 인식 질문 재구성

"그거 예산은?"과 같은 모호한 후속 질문을 이전 대화 이력을 참고하여 "A 프로젝트의 예산은 얼마인가요?"라는 완전한 형태의 질문으로 변환합니다.

## Point 02. 페르소나 및 제약조건

시스템 프롬프트에 '입찰 분석 전문가' 역할을 부여하고, "문서에 없는 내용은 답변 금지" 등의 규칙을 명시하여 환각(Hallucination)을 방지합니다.

## Point 03. 동적 이력 주입

MessagesPlaceholder를 사용하여 길이가 가변적인 대화 기록(chat_history)을 프롬프트 템플릿 내 적절한 위치에 유동적으로 삽입합니다.

# 3 RAG 시스템 설계

STEP 1

**하이브리드 리트리버**

(1) Chroma Dense
(2) Ensemble

>>

STEP 2

**LLM 프롬프트 엔지니어링**

>>

STEP 3

**LCEL 체인 조립 · Retrieval → Generation**

>>

STEP 4

**실시간 TTS 통합**

# 3 실시간 TTS 통합 (1)

```python
def split_sentences_buffered(buffer: str) -> tuple[list[str], str]:
    # 소수점 보호
    protected = re.sub(r"(?<=\d)\.(?=\d)", "<DOT>", buffer)

    sentences: list[str] = []
    buf: list[str] = []
    i = 0
    while i < len(protected):
        ch = protected[i]
        buf.append(ch)

        # 문장 구두점 기준 분리
        if ch in ".!?。 ! ?":
            sentence = "".join(buf).replace("<DOT>", ".").strip()
            if sentence:
                sentences.append(sentence)
            buf = []
            i += 1
            continue

        # 줄바꿈/문단 경계 기준 분리
        if ch == "\n":
            # 연속 개행을 하나의 경계로 처리
            while i + 1 < len(protected) and protected[i + 1] == "\n":
                i += 1
                buf.append("\n")
            sentence = "".join(buf).replace("<DOT>", ".").strip()
            if sentence:
                sentences.append(sentence)
            buf = []
        i += 1

    remainder = "".join(buf).replace("<DOT>", ".").strip()
    return [s for s in sentences if s], remainder
```

### Point 01. 문장 단위 버퍼링

LLM의 토큰 단위 출력을 모아서 문장이 완성될 때까지 대기합니다. 마침표나 물음표 등 종결 문자가 나오면 즉시 분리하여 TTS 엔진으로 넘깁니다.

### Point 02. 비동기 큐 (Queue) 처리

TTSWorker는 별도 스레드에서 동작하며 큐에 들어온 문장을 순차적으로 음성 변환합니다. 메인 루프의 텍스트 생성 속도를 저하시키지 않습니다.

### Point 03. 레이턴시 최소화

전체 답변이 완성되길 기다리지 않고, 첫 문장이 완성되는 즉시 오디오 재생을 시작하여 사용자가 느끼는 대기 시간(TTFb)을 획기적으로 줄입니다.

# 3 실시간 TTS 통합 (2)

```
336         tts_worker = TTSWorker(
337             model_path=TTS_MODEL_PATH,
338             bert_path=TTS_BERT_PATH,
339             config_path=TTS_CONFIG_PATH,
340             out_dir=out_dir,
341             device="cpu",
342             player_cmd=player_cmd,
343             sanitize_fn=_sanitize_answer,
344             split_fn=_split_sentences_for_tts,
345         )
346         tts_worker.start()
347         _TTS_WORKER = tts_worker
348
349         for chunk in rag_chain.stream({"input": query, "chat_history": chat_history}):
350             if "answer" in chunk:
351                 text = chunk["answer"]
352                 # 스트리밍 텍스트는 즉시 출력한다.
353                 print(text, end="", flush=True)
354                 full_response += text
355                 tts_buffer += text
356
357                 sentences, tts_buffer = split_sentences_buffered(tts_buffer)
358                 for sent in sentences:
359                     tts_worker.enqueue(sent)
360
361             if "context" in chunk:
362                 source_documents = chunk["context"]
363
364         if tts_buffer.strip():
365             sentences, remainder = split_sentences_buffered(tts_buffer.strip())
366             for sent in sentences:
367                 tts_worker.enqueue(sent)
368             if remainder:
369                 tts_worker.enqueue(remainder)
370
371         tts_worker.close()
```

## Point 01. 문장 단위 버퍼링

LLM의 토큰 단위 출력을 모아서 문장이 완성될 때까지 대기합니다. 마침표나 물음표 등 종결 문자가 나오면 즉시 분리하여 TTS 엔진으로 넘깁니다.

## Point 02. 비동기 큐 (Queue) 처리

TTSWorker는 별도 스레드에서 동작하며 큐에 들어온 문장을 순차적으로 음성 변환합니다. 메인 루프의 텍스트 생성 속도를 저하시키지 않습니다.

## Point 03. 레이턴시 최소화

전체 답변이 완성되길 기다리지 않고, 첫 문장이 완성되는 즉시 오디오 재생을 시작하여 사용자가 느끼는 대기 시간(TTFb)을 획기적으로 줄입니다.

# 4 TTS 시스템 설계

STEP 1

**시스템 아키텍처 개요**

>>

STEP 2

**TTS 워커 : 비동기 처리의 핵심**

>>

STEP 3

**ONNX 추론 파이프라인 (7단계)**

>>

STEP 4

**G2P 와 BERT**

# 4 TTS 시스템 아키텍처 개요

비동기 워커와 ONNX 추론 엔진의 계층적 통합

서비스 계층
(TTS Worker)

비동기 큐 관리
스레드 기반 병렬 처리
사용자 중단 제어

음성 생성 요청 >

< 음성 반환

추론 계층
(ONNX Engine)

BERT 특징 추출
ONNX 모델 추론
G2P 발음 변환

# 4 TTS 시스템 설계

STEP 1

시스템 아키텍처 개요

>>

STEP 2

TTS 워커 : 비동기 처리의 핵심

>>

STEP 3

ONNX 추론 파이프라인 (7단계)

>>

STEP 4

G2P 와 BERT

# 4 TTS 워커 : 비동기 처리 핵심

Queue 와 Thread 를 통한 순서 보장 및 반응성 유지

## 직렬화

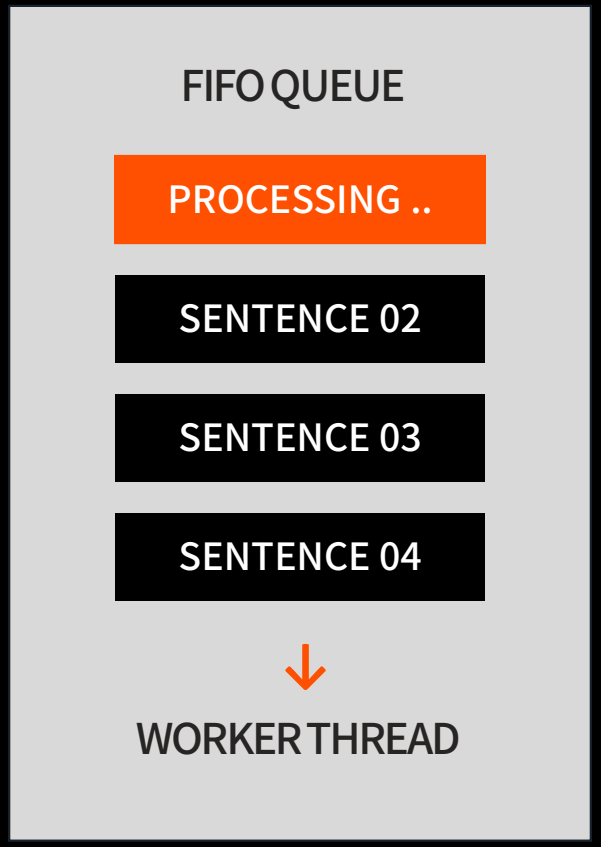
FIFO 큐를 통해 문장 처리 순서를 엄격히 유지하여
오디오 겹침을 방지하고 대화의 흐름을 보장합니다.

## 비차단 실행

별도의 워커 스레드에서 합성을 수행하여
메인 UI나 RAG 프로세스의 지연 없이 실시간 응답을 가능하게 합니다.

## 즉시 중단

사용자의 새로운 입력 시 현재 재생 중인 프로세스를
즉시 종료하고 큐를 비워 최상의 반응성을 제공합니다.



# 4 TTS 시스템 설계

STEP 1

시스템 아키텍처 개요

>>

STEP 2

TTS 워커 : 비동기 처리의 핵심

>>

STEP 3

ONNX 추론 파이프라인 (7단계)

>>

STEP 4

G2P 와 BERT

# 4 ONNX 추론 파이프라인

텍스트 정규화부터 음성 합성까지 7단계 프로세스

| | |
|---|---|
| 01. 텍스트 정규화 | 입력 텍스트의 기호, 숫자, 약어 등을 표준 발음 형식으로 정규화(Normalize)합니다. |
| 02. G2P 변환 | 입력 텍스트를 발음 단위인 Phoneme 시퀀스로 변환합니다. |
| 03. BERT 특징 추출 | BERT ONNX 모델을 통해 문맥적 의미가 담긴 Phone-level feature를 생성합니다. |
| 04. 입력 텐서 구성 | Phoneme ID, Language, BERT 특징 등을 결합하여 TTS 모델 입력을 생성합니다. |
| 05. TTS 모델 추론 | 최적화된 ONNX 세션을 통해 텍스트 특징으로부터 최종 음성 파형을 합성합니다. |
| 06. 문장 결합 | 긴 문장은 구두점 기준으로 분할 합성 후 np.concatenate로 자연스럽게 이어붙입니다. |
| 07. 오디오 출력 | 최종 합성된 파형을 반환하거나 WAV 파일로 저장하여 재생 준비를 마칩니다. |

# 4 TTS 시스템 설계

STEP 1

시스템 아키텍처 개요

>>

STEP 2

TTS 워커 : 비동기 처리의 핵심

>>

STEP 3

ONNX 추론 파이프라인 (7단계)

>>

STEP 4

G2P 와 BERT

# 4 G2P 와 BERT

고품질 음성 합성을 위한 언어학적 특징 추출 (예시음성)

## Phoneme PRONUNCIATION DATA

텍스트를 발음 단위(Phoneme)로 변환합니다.
이는 모델이 문자가 아닌 실제 발음을 이해하는 기초가 됩니다.

## Word2Ph Mapping ALIGNMENT STRATEGY

단어 레벨의 BERT 토큰 특징을 Phoneme 길이로 확장하여 정렬합니다.
문맥 정보가 각 발음 단위에 정확히 매핑되도록 보장하는 핵심 과정입니다.

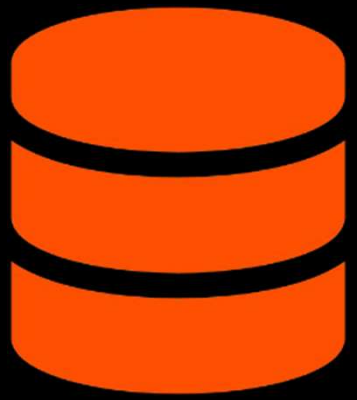
## BERT Features CONTEXTUAL EMBEDDING

한국어 문맥 특징은 BERT 모델을 통해
전달됩니다. 이를 통해 단순 합성을 넘어
풍부한 감정과 표현력을 확보합니다.

# 5 SQLite 시스템 설계

Key Function

RAG 시스템을 위한
SQLite 기반
대화 기록과
피드백 관리



# 5 SQLite 대화 기록과 피드백 관리

## 데이터베이스 스키마 설계

SQL SCHEMA

```
CREATE TABLE IF NOT EXISTS chat_log (
 id INTEGERPRIMARY KEY AUTOINCREMENT,
 question TEXTNOT NULL,
 answer TEXTNOT NULL,
 rating INTEGER,
 created_at TEXTNOT NULL
);
```

### Point 01. ID (Primary Key)

테이블 내 고유 행 식별자로 데이터 무결성을 보장합니다.

### Point 02. Question & Answer

사용자 질문과 모델 답변을 쌍으로 저장하여 대화 맥락을 보존합니다.

### Point 03. Rating (Feedback)

좋아요(1), 싫어요(-1) 피드백을 수집하여 성능 지표로 활용합니다.

### Point 04. Created At

YYYY-MM-DD HH:MM:SS 형식의 타임스탬프를 기록합니다.

# 6 결과 및 기대효과

Goal

**데이터 활용 가치 및 기대효과**

# 6 데이터 활용 가치 및 기대효과

지속적인 RAG 성능 향상을 위한 데이터 기반 구조

TERMINAL

```
-- 최근 로그 20건 조회
SELECT question, answer, rating
FROM chat_log
ORDER BY id DESC
LIMIT 20;

-- 피드백(좋아요/싫어요)만 보기
SELECT rating, created_at
FROM chat_log
WHERE rating IS NOT NULL
ORDER BY id DESC;
```

## Point 01. 성능 평가 (Evaluation)

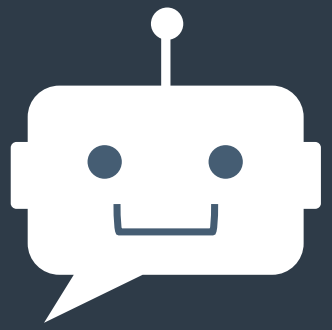
사용자 피드백을 정량화하여 RAG 모델의 답변 정확도와 신뢰도를 객관적으로 측정합니다.

## Point 02. 데이터셋 구축 (Dataset)

'좋아요'를 받은 양질의 답변을 선별하여 향후 GRPO 등 강화학습을 위한 고품질 데이터셋으로 활용합니다.

## Point 03. 문제 진단 (Diagnosis)

'싫어요' 로그를 분석하여 검색(Retrieval) 또는 생성(Generation) 단계의 병목 지점을 파악합니다.



# 감사합니다.