Portfolio

다양한 프로젝트 경험을 통해 문제 해결 능력과 적응력을 갖춘 개발자 **엄요한**입니다.

데이터 기반의 서비스 개선을 통해 다수의 프로젝트의 수행 경력을 담았습니다.

sdjadygks@naver.com

About Me

저를 소개합니다

엄요한 Um yohan

개발자

+010-9427-1829

sdjadygks@naver.com

1992.04.12

융통성과 협업의 힘으로 성장하는 개발자

전자공학 전공을 기반으로 Java·Spring·Oracle 등 백엔드 기술과 HuggingFace 기반 멀티모달 AI 실습 경험을 쌓아왔습니다. GS리테일 KIOSK, 고객사 관리 시스템, 지마켓 AML 시스템 등 다양한 프로젝트를 수행하며 변화에 유연하게 대응하고 협업 속에서 성과를 만들어냈습니다. 새로운 환경에도 빠르게 적응하며 문제를 함께 해결하는 과정을 통해 성장해왔습니다.

Education & Experience

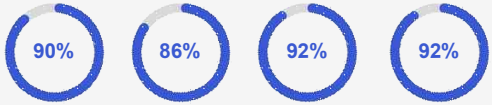
기획자의 기반이 된 배움의 여정

2011.03 ~ 2021.02	**국민대학교** 전자공학부 졸업
2021.06 ~ 2022.01	**KH아카데미** 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정
2025.03 ~ 2025.09	**KG아이티뱅크** 생성형 AI 기반 서비스 개발자 양성과정
2020.03 ~ 2021.02	**스마트임베디드시스템 연구실** 학부연구생 국가, 기업 과제 보조 및 연구
2022.02 ~ 2024.09	**리테일테크** SI 및 솔루션 개발자

Core Skills

강점을 만드는 기술과 역량

프로그래밍 언어 사용능력



다양한 프로그래밍 언어 사용에 능숙하며,
새로운 기술 숙지를 위해 노력하고 있습니다.

프레임워크 활용 능력

현업에서 많이 사용하는 Spring 프레임워크 뿐만 아니라 .NET
프레임워크 등 다양한 프레임워크 활용 능력이 있습니다.

AI 능력

AI 트렌드와 논문을 꾸준히 분석하며 기술 흐름을 학습하는 습관
을 가지고 있습니다.

Etc.

- ✓ HTML / Jinja2 / Thymeleaf / CSS / JavaScript / jQuery
- ✓ AI model 활용 능력 - yolo / Whisper / Llama 등
- ✓ RAG

Project

진행 프로젝트 소개

Alkademia

AI + Akademia

프로젝트명 AI-agent 와 RAG를 활용한 학습 관리 플랫폼

주요목표 AI 기술을 활용한 교육 영상 편집 및 관리 기능
과 학습 컨텐츠 제공

개요

- ✓ 기간: 2주
- ✓ 역할: 메인 기획자 (설계, 핵심 기능 정의, 운영정책 수립 등)

핵심

- ✓ 영상 편집 기능 및 관리 강화 : LLM모델을 활용 전체 영상에서 소주제와 요약 그리고 구간을 추출
- ✓ 학습 컨텐츠 제공 : AI-agent와 RAG를 활용해 추가적으로 학습 정보 제공 및 영상내의 출처 제공

Github

- ✓ https://github.com/yohan412/ai-proj

Timeline

2025.10.01 ~ 2025.10.02

기획

주제 선정 및 계획

2025.10.03 ~ 2025.10.05

관련 문서 작성 및 설계

화면설계서, 요구사항 분석서 및 명세서 등 작성 및 개발 설계

2025.10.05 ~ 2025.10.10.

개발

프로그램 구현

2025.10.11 ~ 2025.10.12

마무리

테스트 및 UML, ERD 등 문서 작성

주요 서비스 기능 및 소스코드(자막 생성 기능)

```python
def transcribe_file(
    file_path: str,              # 영상 파일 경로
    language: Optional[str],     # 언어 코드 (None이면 자동 감지)
    whisper_model: str,          # 모델 크기 (tiny, base, small, medium, large)
    use_fp16: bool               # FP16 사용 여부 (GPU 전용)
) -> Tuple[float, List[Dict[str, Any]], str]:

    # 1. 모델 로드 (캐시됨)
    model = _get_model(whisper_model, use_fp16)

    # 2. 음성 인식 실행
    segments_iter, info = model.transcribe(
        file_path,
        language=language,          # None이면 자동감지
        beam_size=5,                # 빔 서치 크기 (정확도 향상)
        vad_filter=True,            # VAD 활성화
        vad_parameters=dict(
            min_silence_duration_ms=500  # 500ms 이상 무음은 제거
        ),
    )

    # 3. 세그먼트 수집
    out: List[Dict[str, Any]] = []
    for seg in segments_iter:
        out.append({
            "start": _round(seg.start),
            "end":   _round(seg.end),
            "text":  (seg.text or "").strip()
        })

    # 4. 메타데이터 추출
    duration = _round(getattr(info, "duration", 0.0))
    lang_code = getattr(info, "language", None) or (language or "")

    return duration, out, lang_code
```

whisper_model: 모델 크기 선택
- large: 최고 정확도 (1550M 파라미터) <- 해당 모델사용

use_fp16: GPU에서 FP16 정밀도 사용 여부 (메모리 절약)

VAD 필터: 무음 구간 자동 제거로 자막 품질 향상

Beam Search: beam_size=5 로 정확도 향상

주요 서비스 기능 및 소스코드(챕터 분석 기능)

```
# 파이프라인 로드 (캐시됨)
pipe = _get_pipe(
    model_id, load_in_4bit, 0.3, max_new_tokens, hf_token,
    max_gpu_mem=max_gpu_mem, max_cpu_mem=max_cpu_mem,
    offload_dir=offload_dir, low_cpu_mem=low_cpu_mem,
    torch_dtype_name=torch_dtype_name
)

# 1단계: 시간 구간 추출
boundaries = _extract_time_boundaries(segments, duration, pipe)

# 구간 검증 및 병합 (너무 짧거나 많은 구간 처리)
boundaries = _validate_and_merge_boundaries(boundaries, duration, min_duration=60.0)

# 2단계: 각 구간의 제목/요약 생성
chapters = []
for i, boundary in enumerate(boundaries):
    start = float(boundary.get('start', 0))
    end = float(boundary.get('end', duration))

    # 시간 범위 검증
    start = max(0.0, min(start, duration))
    end = max(start + 10.0, min(end, duration))

    # 해당 구간의 제목/요약 생성
    metadata = _generate_chapter_metadata(segments, start, end, lang, pipe)

    chapters.append({
        "start": start,
        "end": end,
        "title": metadata.get("title", "Untitled"),
        "summary": metadata.get("summary", "")
    })

return chapters
```

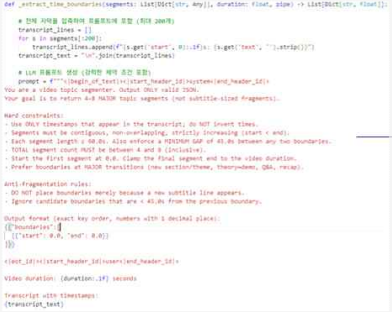
챕터분석에 사용하는 LLM모델은 meta-llama/Llama-3.2-3B-Instruct 로 성능의 한계가 있기때문에 챕터분석시 구간분류 및 주제 요약 작성을 한번에 처리하지 못하기 때문에 로직을 2단계로 나눠 처리

→ 1단계 LLM모델을 활용해 구간만 추출하는 함수

→ 구간 검증 및 병합 함수 (60초 미만 구간 병합)

→ 2단계 LLM모델을 활용해 추출된 구간내의 자막으로 주제 및 요약을 도출하는 함수

주요 서비스 기능 및 소스코드(챕터 분석 기능)



1단계 구간 분류를 위한 프롬프트

```
# LLM 호출 (온도 낮게 설정하여 일관된 결과 생성)
outputs = pipe(prompt, max_new_tokens=800, temperature=0.2)
text = _extract_text(outputs)

# JSON 파싱 (강화된 파싱 로직)
try:
    # 불필요한 공백/개행 제거
    json_text = text.strip()

    # JSON 정규화 (배열 형식, 마지막 쉼표 제거 등)
    json_text = re.sub(r',\s*]', ']', json_text)
    json_text = re.sub(r',\s*}', '}', json_text)

    # boundaries 키 확인 및 추가
    if not json_text.startswith('{"boundaries"'):
        if json_text.startswith('['):
            json_text = '{"boundaries": ' + json_text + '}'

    # Incomplete JSON 완성
    if not json_text.endswith(']}'):
        last_complete = json_text.rfind('},')
        if last_complete != -1:
            json_text = json_text[:last_complete+1] + ']}'

    # 파싱
    obj = json.loads(json_text)
    boundaries = obj.get("boundaries", [])

    # 마지막 구간 누락 체크 (30초 이상 남았으면 추가)
    if boundaries:
        last_end = boundaries[-1].get('end', 0)
        if last_end < duration - 30:
            boundaries.append({"start": last_end, "end": duration})

    return boundaries
```

LLM 출력값이 JSON형태가 아닐때 JSON형태로 만들어주는 파싱 로직

```
except Exception as e:
    # 실패 시 균등 분할 (폴백)
    num_chapters = 6
    chunk_duration = duration / num_chapters
    return [
        {"start": i * chunk_duration, "end": (i + 1) * chunk_duration}
        for i in range(num_chapters)
    ]
```

오류등의 이유로 출력에 이상이 생겼을시 균등하게 6개의 구간을 생성하는 로직

주요 서비스 기능 및 소스코드(챕터 분석 기능)

```
    if lang_name == "Korean":
        prompt = f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>

당신은 교육 콘텐츠 요약 전문가입니다. 주어진 자막을 읽고 반드시 다음 JSON을 생성하세요:

출력 형식(JSON):
{{"title": "제목", "summary": "요약 내용"}}

제약 조건:
1. 제목
   - 3~5개의 단어
   - 7~30자 이내
   - 핵심 키워드 위주, 문장 형태 금지
   - 반드시 순수 한글만 사용
2. 요약
   - 반드시 1문장 또는 2문장만 작성
   - 1문장일 경우 전체 길이 80~120자
   - 2문장일 경우 각 문장은 40~60자, 두 문장의 합은 80~120자
   - 마침표(.)는 최대 두 번까지만 허용
   - 한 번 언급한 내용은 반복하지 말 것
   - 반드시 순수 한글만 사용
3. 공통 규칙
   - 출력에는 영어, 숫자, 알파벳, 외래어, 기호 절대 포함 금지
   - JSON 이외의 다른 글자나 설명은 절대 출력하지 말 것

<|eot_id|><|start_header_id|>user<|end_header_id|>

자막 내용:
{transcript[:1200]}

<|eot_id|><|start_header_id|>assistant<|end_header_id|>

"""
```

2단계 구간 분류를 위한 프롬프트, 한글 영상의 경우 한글로 된 주제,요약문을 출력 받기 위해 입력값의 형태도 주로 한글로 작성된 프롬프트를 작성해야 해서 한글/영어 프롬프트 개별 작성

```
# LLM 호출 (최적 파라미터)
outputs = pipe(
    prompt,
    max_new_tokens=200,
    temperature=0.25,      # 반복 방지 + 영어 혼재 최소화
    repetition_penalty=1.1,
    top_p=0.9,
)
text = _extract_text(outputs)

# JSON 파싱 (강화된 파싱 로직)
try:
    # 다양한 형식 처리 (따옴표 문자열, 불완전 JSON 등)
    json_text = text.strip()

    # summary 키 누락 처리
    if '"title"' in json_text and '"summary"' not in json_text:
        title_match = re.search(r'"title"\s*:\s*"([^"]+)"', json_text)
        if title_match:
            title = title_match.group(1)
            content_match = re.search(r',\s*"([^"]+)', json_text[title_match.end():])
            if content_match:
                summary = content_match.group(1)
                return {"title": title, "summary": summary}

    # 정규 JSON 파싱
    json_text = re.sub(r',\s*}', '}', json_text)
    if json_text.startswith('{') and not json_text.endswith('}'):
        if '"summary"' in json_text and json_text.count('"') % 2 == 1:
            json_text = json_text + '"}'

    obj = json.loads(json_text)
    title = obj.get("title", "").strip()
    summary = obj.get("summary", "").strip()

    if title and summary:
        return {"title": title, "summary": summary}
    else:
        raise ValueError("제목 또는 요약 비어있음")
```

주제와 달리 문장의 형태를 가지는 요약문에서 외국어가 혼재되는 경우가 있어 비교적 낮은 temperature를 설정하였고 마지막 문장이 반복되는 경우도 있어 repetition_penalty를 설정

LLM 출력값이 JSON형태가 아닐때 JSON형태로 만들어주는 파싱 로직

주요 서비스 기능 및 소스코드(상세 설명 기능)

```python
@app.post("/explain")
def explain_chapter():
    """챕터 상세 설명 API"""
    try:
        data = request.get_json()

        # 요청 데이터 추출
        stored_name = data.get("stored_name")
        start_time = float(data.get("start_time", 0))
        end_time = float(data.get("end_time", 0))
        lang = data.get("lang", "ko")

        # 영상 자막 데이터 로드
        segments = load_segments_from_cache(stored_name)

        # LLM 파이프라인 로드
        pipe = _get_pipe(...)

        # 설명 생성
        from services.explainer import generate_explanation
        explanation = generate_explanation(
            segments=segments,
            start_time=start_time,
            end_time=end_time,
            lang=lang,
            pipe=pipe,
            _extract_text=_extract_text
        )

        # 미완성 문장 제거
        cleaned_explanation = trim_incomplete_last_sentence(explanation)

        return jsonify({
            "explanation": cleaned_explanation,
            "segment_count": len([
                s for s in segments
                if s.get('start', 0) >= start_time and s.get('end', 0) <= end_time
            ])
        })
```

상세 설명 작성 기능의 경우 자막을 새로 추출해서 작성하지 않고 DB에 저장된 자막을 활용하는 형태로 구현

상세 설명 작성을 위한 LLM활용 함수

토큰등의 제한의 이유로 미완성된 마지막 문장 삭제

JSON형태로 만든후 출력

주요 서비스 기능 및 소스코드(상세 설명 기능)

```
    # 3. 언어 라벨 변환
    lang_name = _lang_label_from_code(lang)  # "ko" → "Korean"

    # 4. 언어별 프롬프트 생성 (한글 예시)
    if lang_name == "Korean":
        explanation_prompt = f"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>

당신은 교육 컨텐츠 분석 전문가입니다. 주어진 내용을 분석하고 명확한 설명을 한글로 제공하세요.

중요 규칙:
- 핵심 개념을 요약하고 설명하세요
- 자연스러운 한글을 사용하세요
- 교육적 인사이트를 포함한 3-5문장으로 작성하세요
- 영어 단어를 절대 사용하지 마세요
- 전문 용어도 한글로 풀어 쓰세요

<|eot_id|><|start_header_id|>user<|end_header_id|>

다음 내용을 분석하고 주요 아이디어를 한글로 설명하세요:

{transcript_text[:1500]}

<|eot_id|><|start_header_id|>assistant<|end_header_id|>

"""
    else:
        # 영어 프롬프트 (생략 - 동일한 구조)
        explanation_prompt = f"""..."""
```

상세 설명 작성을 위한 프롬프트, 한글 영상의 경우 한글로 된 상세 설명을 출력 받기 위해 입력값의 형태도 주로 한글로 작성된 프롬프트를 작성해야 해서 한글/영어 프롬프트 개별 작성

자막의 주된 언어가 영어일경우 영어프롬프트 입력

```
def trim_incomplete_last_sentence(text: str) -> str:

    if not text:
        return text

    # 1. 유효한 문장 종결 부호: . 또는 !만 허용 (? 제외)
    last_period = text.rfind('.')
    last_exclamation = text.rfind('!')

    # 2. 가장 마지막 유효한 문장 종결 부호의 위치
    last_sentence_end = max(last_period, last_exclamation)

    # 3. 유효한 문장 종결 부호가 없으면 빈 문자열 반환
    if last_sentence_end == -1:
        return ""

    # 4. 마지막 유효한 문장 종결 부호 뒤의 텍스트 확인
    after_last = text[last_sentence_end + 1:].strip()

    # 5. 뒤에 텍스트가 있으면 (미완성 문장) 잘라냄
    if after_last:
        return text[:last_sentence_end + 1].strip()

    # 6. 이미 완전한 문장으로 끝나면 그대로 반환
    return text
```

토큰등의 제한의 이유로 미완성된 마지막 문장 삭제

주요 서비스 기능 및 소스코드(AI 챗봇 기능)

```python
if lang == "ko":
    # KoNLPy 시도 (한글 전용)
    try:
        from konlpy.tag import Okt
        okt = Okt()

        # 명사만 추출
        nouns = okt.nouns(question)
        print(f"[Simple QA] KoNLPy 원본 명사: {nouns}")

        # 메타 명사 필터링 (질문/요청 관련 단어 제거)
        meta_nouns = {
            '설명', '대해', '질문', '답변', '내용', '정보', '이야기', '얘기',
            '것', '거', '뭐', '무엇', '어떻게', '왜', '언제', '어디', '누구',
            '방법', '이유', '시간', '장소', '사람', '알려', '해줘', '주세요'
        }

        # 1-5자 + 메타 명사 제외
        keywords = [
            n for n in nouns
            if 1 <= len(n) <= 5 and n not in meta_nouns
        ]

        print(f"[Simple QA] 메타 명사 제거 후: {keywords}")

    except Exception as e:
        print(f"[Simple QA] KoNLPy 실패 ({e}), 패턴 매칭 사용")
        # 폴백: 패턴 매칭
        keywords = extract_keywords_pattern(question, lang)
```

1단계는 키워드 추출 단계로 KoNLPy 라이브러리를 활용해 질문에서 명사를 추출 리스트에 넣음

```python
final_sources = []

if keywords and len(segments) > 0:
    print(f"[Simple QA] RAG 검색 시작...")

    # VideoRAG 인스턴스 생성 (FAISS 인덱스 자동 빌드)
    rag = VideoRAG(stored_name, segments)
    all_sources = []

    # 각 키워드별로 검색
    for kw in keywords:
        results = rag.search(kw, top_k=1)  # 키워드당 1개씩
        all_sources.extend(results)

    # 중복 제거 (start 기준)
    seen = set()
    unique_sources = []
    for src in all_sources:
        if src['start'] not in seen:
            seen.add(src['start'])
            unique_sources.append(src)

    # 점수 순 정렬 후 상위 3개
    unique_sources.sort(key=lambda x: x['score'])
    final_sources = unique_sources[:3]

    print(f"[Simple QA] 자막 출처: {len(final_sources)}개")
    for i, src in enumerate(final_sources):
        print(f"  {i+1}. [{src['start']:.1f}s] {src['text'][:50]}...")
else:
    print(f"[Simple QA] 키워드 없음 또는 자막 없음, 출처 스킵")
```

2단계는 추출된 키워드를 가지고 해당 영상의 자막데이터에서 출처검색

```python
wiki_results = []
if keywords:
    for kw in keywords:
        print(f"[Simple QA]   - '{kw}' 검색 중...")
        wiki_result = search_wikipedia(kw, lang=lang, sentences=2)

        # [Wikipedia] 접두사 제거
        cleaned = wiki_result.replace("[Wikipedia] ", "")

        # "검색 실패" 메시지가 아니면 추가
        if "검색 실패" not in cleaned:
            separator = "=" * 30
            wiki_results.append(
                f"{separator}\n📌 {kw}\n{separator}\n{cleaned}"
            )

# 키워드가 없거나 모든 검색 실패 시 질문 전체로 검색
if not wiki_results:
    print(f"[Simple QA]   - 질문 전체로 검색...")
    wiki_answer = search_wikipedia(question, lang=lang, sentences=3)
    answer = wiki_answer.replace("[Wikipedia] ", "")
else:
    # 각 키워드 결과 결합
    answer = "\n\n".join(wiki_results)

print(f"[Simple QA] Wikipedia 답변: {answer[:100]}...")
```

3단계는 추출된 키워드를 외부에서 독립적으로 개별 검색후 정보 반환

주요 서비스 기능 및 소스코드(AI 챗봇 기능)

```python
class VideoRAG:
    """영상 자막 기반 RAG 시스템"""

    def __init__(self, stored_name: str, segments: List[Dict]):
        """
        Args:
            stored_name: 영상 파일명
            segments: 자막 세그먼트 리스트
                      [{"start": 0.0, "end": 3.4, "text": "..."}]
        """
        self.stored_name = stored_name
        self.segments = segments

        # Sentence Transformer 임베더 로드
        self.embedder = SentenceTransformer(
            'sentence-transformers/all-MiniLM-L6-v2'
        )

        self.index = None
        self.texts = []

        # FAISS 인덱스 자동 빌드
        self._build_index()
```

영상 자막 기반 RAG 클래스 임베더 모델로는 sentence-transformers/all-MiniLM-L6-v2 활용

```python
def _build_index(self):
    """FAISS 인덱스 생성"""
    print(f"[RAG] 인덱스 생성 시작 - {len(self.segments)}개 세그먼트")

    # 1. 자막 텍스트 추출 (시간 정보 포함)
    self.texts = [
        f"[{s['start']:.1f}s-{s['end']:.1f}s] {s['text']}"
        for s in self.segments
    ]

    # 2. 임베딩 생성 (384차원 벡터)
    embeddings = self.embedder.encode(
        self.texts,
        show_progress_bar=False
    )

    # 3. FAISS 인덱스 생성 (L2 거리 기반)
    dimension = embeddings.shape[1]  # 384
    self.index = faiss.IndexFlatL2(dimension)
    self.index.add(embeddings.astype('float32'))

    print(f"[RAG] 인덱스 생성 완료 - {self.index.ntotal}개 벡터")
```

영상 자막을 벡터형태의 데이터로 변환

```python
def search(self, query: str, top_k: int = 3) -> List[Dict]:
    """
    쿼리와 가장 유사한 자막 세그먼트 검색

    Args:
        query: 검색 쿼리 (키워드)
        top_k: 반환할 결과 개수

    Returns:
        검색 결과 리스트
        [{"start": 0.0, "end": 3.4, "text": "...", "score": 0.5}]
    """
    # 1. 쿼리 임베딩
    query_embedding = self.embedder.encode(
        [query],
        show_progress_bar=False
    )

    # 2. FAISS 검색 (L2 거리)
    distances, indices = self.index.search(
        query_embedding.astype('float32'),
        top_k
    )

    # 3. 결과 반환
    results = []
    for i, idx in enumerate(indices[0]):
        if idx < len(self.segments):
            seg = self.segments[idx]
            results.append({
                "start": seg['start'],
                "end": seg['end'],
                "text": seg['text'],
                "score": float(distances[0][i])  # L2 거리 (낮을수록 유사)
            })

    return results
```

키워드들 또한 벡터형태의 데이터로 변환후 자막벡터와 L2거리를 계산하여 거리가 짧은 자막에 대해 출력

소감

이번 프로젝트는 제가 처음으로 LLM(Large Language Model)을 직접 다뤄본 도전적인 경험이었습니다. 처음에는 서버 컴퓨터의 성능 제약으로 인해 Llama 3.2 3B라는 작은 모델을 선택할 수밖에 없었고, 그로 인해 기대했던 수준의 성능이 나오지 않아 많은 어려움을 겪었습니다.

하지만 이 한계는 오히려 저에게 모델 성능의 본질적인 개선이 무엇인지를 깊이 고민하게 만들었습니다. 단순히 큰 모델을 사용하는 것이 아니라, 파라미터 튜닝과 프롬프트 튜닝을 통해 작은 모델에서도 의미 있는 성능을 이끌어낼 수 있다는 것을 직접 경험했습니다. 이를 통해 “좋은 결과는 모델의 크기보다 개발자의 이해와 시도에서 비롯된다”는 사실을 몸소 깨달을 수 있었습니다.

또한 이번 프로젝트를 통해 기초적인 AI 에이전트(AI Agent)와 RAG(Retrieval-Augmented Generation) 기술을 함께 다뤄볼 수 있었던 점도 매우 흥미로웠습니다. 이 기술들은 현재에도 빠르게 발전하며 주목받고 있는 분야로, 단순히 실험적인 수준을 넘어 실제 서비스에 적용할 수 있는 다양한 활용 방안을 구상하게 되었습니다. 앞으로는 이러한 기술들을 더 깊이 이해하고, 다양한 문제 해결에 창의적으로 접목시켜보고 싶습니다.

이 프로젝트는 단순한 개발 경험을 넘어, 제게 끊임없이 배우고 성장하는 개발자의 자세를 일깨워 준 소중한 여정이었습니다.

향후 개발 계획

LLM모델 교체

성능 향상을 위한 LLM모델 교체 및 프롬프트 수정

MCP Server 구축 및 MCP Agent 도입

교육 컨텐츠 확장 및 강화를 위한 MCP Server 구축 및 MCP Agent 도입

클라우드 서버 도입

LLM모델 교체 및 사용자들의 안정적인 이용을 위해 클라우드 서버 도입

Thank you

함께 더 나은 **디지털 경험**을 만들기 위해.

제 포트폴리오를 봐주셔서 감사합니다.
지속적인 자기계발을 통해 성장하며, 동료와 협력해 더 큰 성과를 만드는 개발자로 성장하겠습니다.

sdjadygks@naver.com