Local_LLM_구현

개요

- 로컬 환경(Mac M3, 16GB RAM)에서 LLM 모델(Ollama + Qwen)을 구동하고, Next.js 프론트엔드와 연동하여 AI 기능을 실시간으로 제공하는 프로젝트입니다.
- 주요 기능은 문서 요약, 번역, 문법/문체 개선 등이며, 외부 유료 API를 사용하지 않고 로컬에서 처리하도록 구성되어 있습니다.

목적

- 외부 유료 AI API 없이 로컬에서 AI 기능 구현
- 사용자가 입력한 문서를 즉시 요약/번역/교정
- 프론트엔드에서 간단히 요청(body로 `prompt`와 `text`만 전달) 가능
- 실시간 스트리밍으로 처리 결과를 즉시 렌더링

주 기능 (MVP)

- 문서 요약: 3~5문장으로 핵심 내용 요약
- 문법/맞춤법 검사: 문서 검토 및 수정 제안
- 번역: 한글 → 영어 (외부 입력 가능)
- 문체 개선: 글의 문체 개선 및 전문적/명확하게 다듬기
- 스트리밍 결과 제공: 프론트에서 결과를 실시간으로 확인(UX 효과 증대)

처리 흐름(요약)

1. Next.js (Client)

- 에디터에서 문서 텍스트 입력/붙여넣기
- `callAI(prompt, text, language)` 호출
- fetch API를 통해 Next.js API(route)로 POST 요청

2. Next.js API (Server)

- 요청 본문에서 `prompt`와 `text` 추출
- 서버 기본 프롬프트와 외부에서 전달받은 prompt를 조합
- 언어 지정 가능 (예: 한글 요약, 영어 번역)
- Ollama 로컬 LLM 서버에 POST 요청(`stream: true`)
- ReadableStream을 통해 결과를 클라이언트로 전송

3. Ollama LLM

- `qwen2.5:3b` 모델 사용
- 요청 받은 prompt 기반 텍스트 생성
- chunk 단위 스트리밍 출력

4. 클라이언트 스트리밍 처리

- `res.body.getReader()`로 스트림 읽기
- TextDecoder로 chunk 디코딩
- state를 업데이트하여 실시간 렌더링

설치 및 연동 과정

1. 환경 준비

- **Mac 모델**: M3 MacBook Air
- 메모리: 16GB
- **IDE**: VSCode
- **Node.js / NPM**: Next.js 15 버전 사용
- **Python**: 필요 시 설치 (간단한 테스트용)

2. Ollama 설치

1) Homebrew로 Ollama 설치:

```
brew install ollama
```

2) Ollama 서버 시작:

```
brew services start ollama
# 실행한 상태에서만 ollama pull 가능

# 서버가 뜨지 않는다면 프로세스 확인
ps aux | grep ollama

# 서버가 떠 있지 않다면 강제 실행
ollama serve
# 정상인 경우 Listening on http://127.0.0.1:11434 등 메세지 출력
# starting runner
# cmd="... ollama runner --ollama-engine --port 59031"
```

- 59031은 내부 엔진 포트
  - LLM 추론용 워커 프로세스
  - Ollama 서버가 자동으로 띄움
  - 절대 직접 접근 안 함
  - 매번 실행할 때마다 포트 바뀔 수 있음. 따라서 신경 X

```
# 구조
[ FE / curl / axios ]
          ↓
   http://127.0.0.1:11434   ← 실 사용 ollama 포트
          ↓
     Ollama Server
          ↓
     LLM Runner (59031)     ← 내부 전용
          ↓
      Apple Silicon (M3)
```

3) ollama 모델 다운로드

```
ollama pull qwen2.5:3b
# ❌ 7b → 돌아가긴 하지만 요약/번역에 살짝 답답할 수 있음
# ✅ 3b → 체감 속도 훨씬 좋음
```

4) ollama 설치 확인

```
ollama list

# 삭제를 원할 경우
ollama rm qwen2.5:3b
```

3. 테스트

- 터미널

```
# 테스트 실행
ollama run qwen2.5:3b
# 테스트 문장 예시
# $ 다음 글을 한글로 요약해줘:
# $ Artificial intelligence is transforming the software industry...
# or

# curl 테스트
curl http://127.0.0.1:11434/api/generate \
  -d '{
    "model": "qwen2.5:3b",
    "prompt": "다음 문장을 한국어로 요약해줘: Artificial intelligence is changing software."
  }'
```

- python

```
# python 설치
brew install python
python3 --version # 설치 확인

# 프로젝트 폴더 만들기
mkdir local-ai
cd local-ai

# VSCode(IDE) 켜기
code .

# Python 가상환경 만들기
python3 -m venv venv
source venv/bin/activate # 활성화
# 성공 시 터미널 앞에서 (venv) 붙음

# 필요 라이브러리 설치
pip install fastapi uvicorn
```

- 예시용 코드 작성

```
# main.py

from fastapi import FastAPI
from pydantic import BaseModel
import requests

app = FastAPI()

class SummarizeRequest(BaseModel):
    text: str

@app.post("/summarize")
def summarize(req: SummarizeRequest):
    response = requests.post(
        "http://localhost:11434/api/generate",
        json={
            "model": "qwen2.5:7b-instruct-q4",
            "prompt": f"다음 글을 한 줄로 요약해라:\n\n{req.text}",
            "stream": False
        }
    )

    result = response.json()
    return {
        "summary": result["response"]
    }
```

```
# requests 설치
pip install requests

# 서버 실행
uvicorn main:app --reload
# 성공 시
# Uvicorn running on http://127.0.0.1:8000
```

- swagger 테스트

```
# python 서버 실행 상태에서 swagger 호출
http://127.0.0.1:8000/docs
# or
http://localhost:8000/docs
```

4. Next.js 연동

1) route.ts

```
import { NextRequest, NextResponse } from 'next/server';

type AIRequest = {
    prompt?: string;
    text: string;
    lang?: string; // 'ko', 'en', 'zh' 등
};

export async function POST(req: NextRequest) {
```

```
    const { prompt, text, lang = 'en' } = (await req.json()) as AIRequest;

    if (!text || !text.trim()) {
        return NextResponse.json({ error: '텍스트가 필요합니다.' }, { status: 400 });
    }

    const languageInstruction = lang
        ? `결과는 반드시 ${lang === 'ko' ? '한국어' : lang === 'en' ? '영어' : '중국어'}로 작성하세요.`
        : '결과는 반드시 한국어로 작성하세요.';

    const basePrompt = prompt?.trim() || '다음 문서를 3~5문장으로 요약하라. 핵심만 간결하게 작성하라.';

    const fullPrompt = [basePrompt, languageInstruction, '[문서 시작]', text, '[문서 끝]'].join('\n');

    const ollamaRes = await fetch('http://127.0.0.1:11434/api/generate', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            model: 'qwen2.5:3b',
            prompt: fullPrompt,
            stream: true,
            options: {
                temperature: 0.2, // (0 ~ 1)낮을수록 안정적 요약/정형화된 문장 출력
                top_p: 0.9, // (0 ~ 1) 누적 확률 필터링, 낮을수록 집중적이고 안전한 출력
                num_ctx: 2048 // 컨텍스트 길이, 한 번에 모델이 참고할 수 있는 토큰 수 제한, 한 번에 볼 수 있는 문서 길이 제한
            }
        })
    });

    if (!ollamaRes.body) return new Response('No stream', { status: 500 });

    // 데이터 스트리밍
    const stream = new ReadableStream({
        async start(controller) {
            // 스트림을 읽는 리더 생성
            const reader = ollamaRes.body.getReader();
            const decoder = new TextDecoder(); // 바이트(`Uint8Array`) → 문자열 변환
            const encoder = new TextEncoder(); // 문자열 → 바이트로 변환 (컨트롤러 전송 시 필요)
            let buffer = '';

            while (true) {
                // 스트림에서 데이터 읽음
                // value: 읽은 바이트 배열
                const { done, value } = await reader.read();
                if (done) break; // true == 루프 종료

                // 디코딩 후 buffer에 누적
                buffer += decoder.decode(value, { stream: true });
                const lines = buffer.split('\n');
                buffer = lines.pop()!;

                for (const line of lines) {
                    if (!line.trim()) continue;
                    const json = JSON.parse(line) as any;

                    // 스트림 컨트롤러를 통해 클라이언트로 전송
                    if (json.response) controller.enqueue(encoder.encode(json.response)); // 문자열 → 바이트로 변환
                }
            }

            controller.close(); // 데이터 다 읽으면 스트림 종료
        }
    });

    return new Response(stream, {
        headers: {
            'Content-Type': 'text/plain; charset=utf-8',
            'Cache-Control': 'no-cache'
        }
    });
}
```

- Qwen 시리즈는 중국 AI 기업 **Alibaba DAMO Academy**에서 개발한 LLM
- 다국어 지원 가능하나 기본 학습 데이터에 중국어 비중이 높아서, 한글이나 영어 번역 시 중국어 출력이 나올 수 있어 영문으로 시작하는 Text의 경우 중국어 출력 비중이 높음(중국어가 기본 언어)
- 따라서 언어팩 별 출력용 안정화를 위해 `lang` 추가(default: `ko`)
- Client 단 호출 시 `prompt, text, lang` 파람만 입력, Server 단에서 ollama 호출을 위한 prompt 전체 변환 로직 적용
- `stream: true`: 한 번에 생성 결과를 받지 않고 모델이 생성하는 토큰/문장 단위로 순차 전송, 실시간 렌더링 가능

2) component.tsx

```
const callAI = async prompt => { // prompt: 요청사항
    const text = getEditorText(); // 대상 text

    if (!text.trim()) {
        alert('에디터에 텍스트를 먼저 입력해주세요.');
        return;
    }

    setAiLoading(true);
    setAiResult('');

    try {
        const res = await fetch('/api/ai', { method: 'POST', body: JSON.stringify({ prompt, text }) });

        bufferRef.current = '';

        const reader = res.body!.getReader();
        const decoder = new TextDecoder();

        while (true) {
            const { done, value } = await reader.read();
            if (done) break;

            bufferRef.current += decoder.decode(value);
            setAiResult(bufferRef.current); // 실시간 렌더링
        }
    } catch (error) {
        console.error('AI 호출 오류:', error);

        if (error.response) {
            // 서버가 에러 응답을 보낸 경우
            setAiResult(`오류: ${error.response.data.error || '서버 오류가 발생했습니다.'}`);
        } else if (error.request) {
            // 요청은 보냈지만 응답을 받지 못한 경우
            setAiResult('오류: 서버로부터 응답이 없습니다.');
        } else {
            // 요청 설정 중 오류가 발생한 경우
            setAiResult(`오류: ${error.message}`);
        }
    } finally {
        setAiLoading(false);
    }
};
```

결과

- 로컬 환경에서 AI 모델 구동 가능
- 프론트에서 단순히 `prompt` + `text` 전달로 AI 기능 사용 가능
- 실시간 스트리밍으로 사용자가 작성 중에도 결과 확인 가능
- 언어 주입 기능 추가로 번역/요약 등 다양한 활용 가능
- Ollama 서버를 통해 Qwen 모델 직접 호출

향후 계획

- AI 사용 유의사항 추가
- 언어 다양성 추가
- 서버 부하도 측정