#바이브 코딩(Vibe Coding)
#AI 자동화 시스템 자체 개발

바이브 코딩(Vibe Coding)을 통한 AI 자동화 시스템 자체 개발기

글. 김우중 MBC 강원영동 기술국 부장

방송기술인으로서 우리는 매일 주조정실과 송출실에서 시스템의 안정성을 지키기 위해 묵묵히 자리를 지킨다. 1초의 블랙 화면도 허용하지 않는 무결점의 세계, 그것이 우리 엔지니어들의 자부심이자 사명이다.
하지만 시선을 조금만 돌려 콘텐츠가 만들어지는 보도국의 풍경을 보았을 때, 필자는 기술적인 난제와는 또 다른 '어려움'을 목격하게 되었다. 매일 뉴스가 끝나면 보도국 뉴미디어 담당자들은 또 다른 전쟁을 치른다. 방송된 뉴스 전체 영상을 가져와 앵커 멘트와 리포트 구간을 일일이 확인하며 수동으로 자르고, 썸네일을 만들고, 태그를 입력하여 유튜브에 업로드하는 모습이었다. 이는 창의적인 기획이라기보다는, 마우스 클릭과 인내심이 요구되는 고된 반복 노동에 가까웠다.

'이런 정형화된 패턴의 업무 프로세스를 기술적으로 덜어줄 수는 없을까?'

이 작은 의문이 모든 것의 시작이었다. 내 옆의 동료들이 단순 노동에서 벗어나 더 가치 있는 콘텐츠 기획에 집중할 수 있도록 엔지니어로서 해법을 제시하고 싶었다. 하지만 현실적인 장벽은 높았다. 필자는 하드웨어와 방송 장비에는 익숙하지만, C언어나 파이썬(Python)을 유려하게 다루는 전문 소프트웨어 개발자가 아니었기 때문이다. 복잡한 자동화 프로그램을 밑바닥부터 설계한다는 것은 차원이 다른 문제였다. 이때 필자가 찾은 해답이 바로 '바이브 코딩(Vibe Coding)'이었다.

Part 1 패러다임의 전환 : 문법(Syntax)을 넘어 의도(Vibe)로

과거의 코딩이 프로그래밍 언어의 엄격한 문법을 지키며 벽돌을 하나씩 쌓아 올리는 조적공의 작업이었다면, 생성형 AI 시대의 코딩은 설계도와 작업의 분위기(Vibe)를 AI에 전달하는 현장 감독(Director)의 역할로 변화했다. 이것이 바로 바이브 코딩의 핵심이다.

바이브 코딩은 단순히 "코드 짜줘"라고 명령하는 것이 아니다. 자신이 만들고자 하는 프로그램의 작동 방식(Flow)과 목적을 AI에 자연어로 설명하고, AI와 끊임없이 대화하며 결과물을 다듬어가는 공학적 프로세스다. 코딩 지식이 전무 했던 필자는 이 방식을 통해 낯선 프로그래밍 언어의 장벽을 넘어설 수 있었다.

그림 1. 전통적 코딩 vs. 바이브 코딩 프로세스 비교

Part 2 바이브 코딩 실전 : 자연어 기반 시스템 아키텍처 설계

단순히 AI에 코드를 요청하는 수준을 넘어, 실제 상용 레벨의 프로그램을 만들기 위해서는 '바이브 코딩'을 체계적인 엔지니어링 프로세스로 접근해야 한다. 본 프로젝트에서는 다음과 같은 [4단계 기술적 방법론]을 정립하여 개발의 정확도와 안정성을 확보했다.

Step 1. 컨텍스트 주입(Context Injection)과 페르소나(Persona) 정의

LLM(거대언어모델)은 백지상태에서는 범용적인 답변만 내놓는다. 방송 현장에 특화된, 당장 실행 가능한 코드를 얻기 위해서는 AI에 정확한 역할과 맥락을 부여해야 한다. 필자는 이 과정을 무료 버전(채팅창)과 유료 버전(전용 기능)으로 나누어 접근했다.

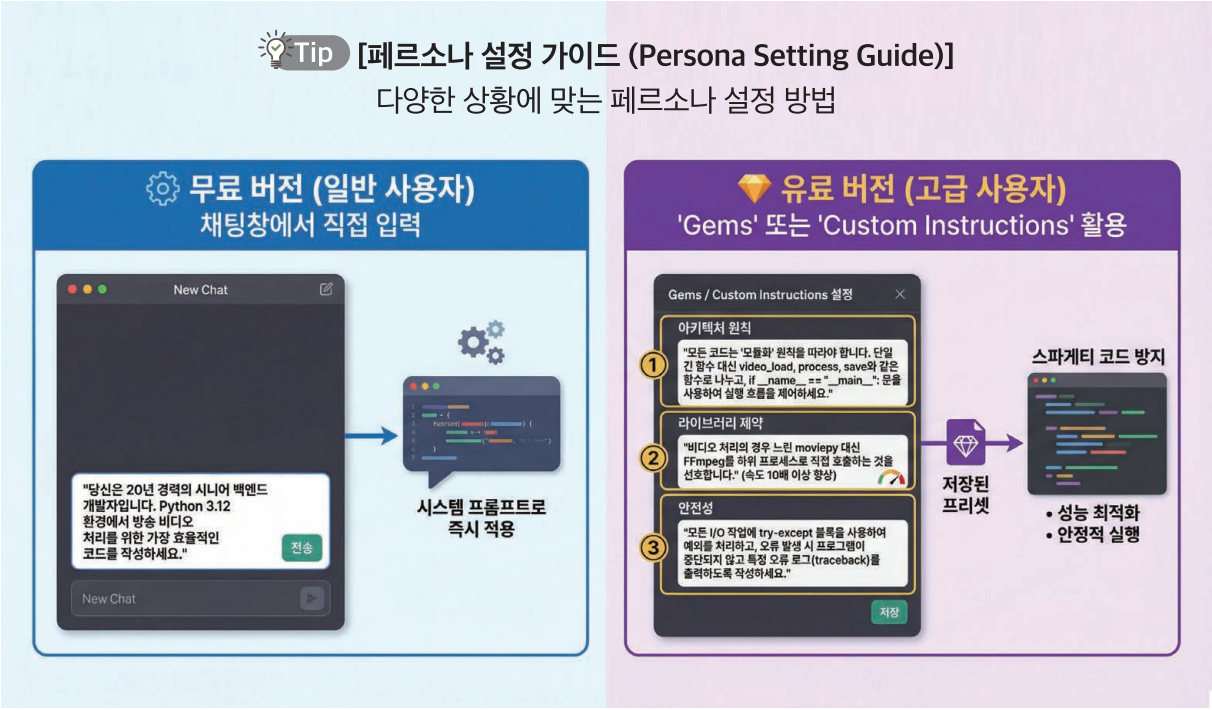

그림 2. 무료 버전 vs. 유료 버전 페르소나 설정 방법 비교

이처럼 페르소나를 정교하게 설정하면, AI는 변수명 하나를 짓더라도 초보자 수준이 아닌, 현업 엔지니어가 작성한 것 같은 고성능, 고효율 코드를 제안한다.

Step 2. 논리 흐름도(Logic Flow) 작성

의사 코드(Pseudo-code)를 통한 로직 검증복잡한 기능을 한 번에 코딩하게 하면 환각(Hallucination) 현상이 발생하거나 논리적 오류가 생기기 쉽다. 이를 방지하기 위해 Chain-of-Thought(CoT) 기법을 활용했다. 바로 코드를 생성하지 않고, 먼저 한글로 된 논리 흐름도(Logic Flow)를 작성하게 하는 것이다.

Step 3. 모듈화(Modularization) 설계

전체 프로그램을 하나의 파일(app.py)에 몰아넣는 스파게티 코드를 방지하기 위해, 기능별로 모듈을 분리하도록 유도했다. video_processor.py(영상 처리), gemini_client.py(AI 통신), gui_main.py(UI) 등으로 파일을 쪼개어 개발하고 이를 통합(Integration)함으로써 유지보수 용이성을 확보했다.

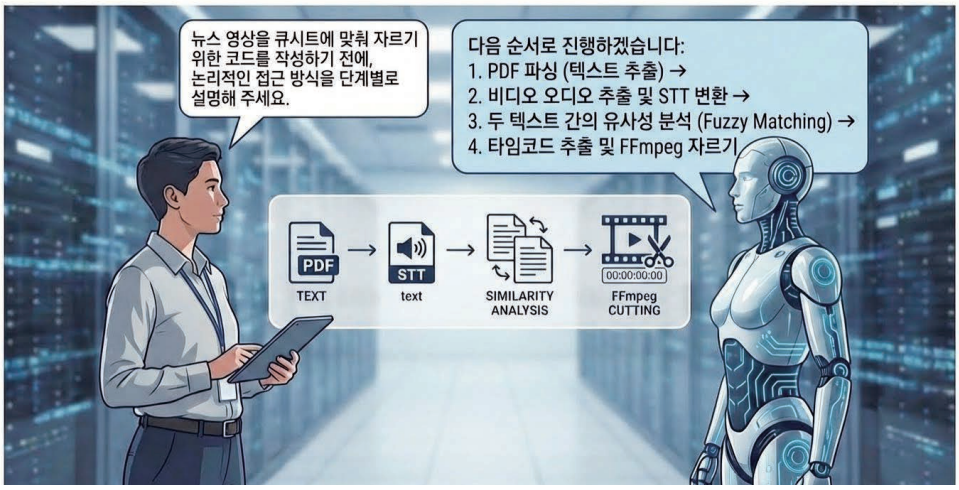

그림 3. AI와의 대화를 통한 논리적 개발 과정 인식

Step 4. 에러 로그 주입 및 AI를 통한 리팩토링

반복적 디버깅(Iterative Debugging)과 협업 코드가 한 번에 완벽하게 실행되는 경우는 드물다. 이때 에러 메시지를 해석하려 애쓰지 않고 터미널의 에러 로그(Log) 전체를 AI에 그대로 복사해 주고 "원인을 분석하고 수정해 줘"라고 요청한다. 이는 마치 숙련된 동료와 프로그래밍을 하는 것과 같은 효과를 낸다.

구분	전통적 코딩(Traditional Coding)	바이브 코딩(Vibe Coding)
핵심 역량	문법(Syntax) 암기, 알고리즘 구현력	프롬프트 설계(Prompt Design), 시스템 기획력
개발 단위	함수(Funtion) 및 라인(Line) 단위 작성	기능(Feature) 및 모듈(Module) 단위 생성
디버깅	StackOverflow 검색 및 코드 분석	에러 로그 주입 및 AI를 통한 리팩토링(Refactoring)
개발 속도	구현 80%, 설계 20%	구현 20%, 설계 및 검증 80%

표 1. 전통적 코딩 vs. 바이브 코딩 비교

Part 3 도구의 선택: ChatGPT vs. Gemini 3.0

성공적인 바이브 코딩을 위해서는 강력한 AI 파트너가 필요하다. 개발 과정에서 양대 LLM인 ChatGPT(O1/Canvas)와 Gemini 3.0(Pro/Flash)을 모두 활용해 보았으며, 각기 다른 강점을 확인했다.

구분	ChatGPT(O1/Canvas)	Gemini 3.0(Pro/Flash)
강점	논리적 추론 및 코드 리팩토링	멀티모달(Video/Audio) 처리 및 긴 문맥 이해
특징	복잡한 알고리즘의 논리적 오류를 찾거나 기존 코드를 더 깔끔하게 다듬는데 탁월하다.	대용량 코드/문서(Long Context1-column capabillity) 분석력이 뛰어나다. 특히 영상/음성 처리를 위한 라이브러리 활용에 강점이 있다.
활용	초기 알고리즘 검증 및 디버용	전체 프로젝트 구조 설계 및 미디어 파이프라인 구축 메인 엔진

표 2. ChatGPT와 Gemini 3.0 비교

본 프로젝트는 영상 처리가 핵심이었기에, 멀티모달 기능이 강화된 Gemini 계열을 메인 엔진으로 채택하여 개발을 진행했다.

Part 4 바이브 코딩의 핵심: '코딩'이 아니라 '시스템 이해도'다

바이브 코딩을 진행하며 뼈저리게 느낀 점은, AI가 만능 해결사가 아니라는 사실이다. AI는 방대한 코드를 학습했지만, 우리 방송사의 구체적인 워크플로우나 하드웨어 특성은 알지 못한다. 이런 이유로 종종 엉뚱한 코드를 내놓거나, 논리적인 막다른 골목에 다다르곤 한다.

그림 4. 개발에서의 AI: 위험 요소와 인간의 역할

바로 이 지점에서 방송기술인(System Expert)의 진가가 발휘된다.

1

AI의 막힘(Blocking)을 뚫는 것은 도메인 지식(Domain Knowledge) 개발 도중 영상 처리 속도가 너무 느려 AI가 해결책을 찾지 못하고 맴도는 상황이 있었다. 일반적인 프로그래머라면 코드 최적화를 고민했겠지만, 필자는 하드웨어적인 관점에서 "파이썬 라이브러리 대신, FFmpeg를 서브프로세스로 직접 호출해서 GPU 가속을 태워라"는 구체적인 해결책을 제시했다. AI는 이 지시를 받자마자 즉시 고성능 코드를 구현해냈다. AI가 '엔진'이라면, 그 엔진이 엉뚱한 곳으로 가지 않도록 길을 알고 방향을 잡아주는 내비게이션은 결국 시스템을 꿰뚫고 있는 엔지니어의 몫이다.

2

엉뚱한 결과(Hallucination)를 걸러내는 AI는 때로 존재하지 않는 함수를 쓰거나, 방송 규격에 맞지 않는 포맷을 제안하기도 한다. 이때 코드가 맞는지 틀리는지 검증할 수 있는 능력은 코딩 실력이 아니라 "이 신호가 우리 주조정실 시스템에서 송출 가능한 규격인가?"를 판단할 수 있는 방송기술에서 나온다.
결국, 바이브 코딩 시대에 가장 강력한 경쟁력은 파이썬 문법 암기가 아니라, 자신이 일하는 시스템의 구조를 완벽하게 장악하고 있는 현장 장악력이다.

Part 5 결과물: MBC 강원영동 자체 개발 'AI Director'

이러한 과정을 통해 탄생한 것이 바로 'AI Director'와 'AI Shorts Maker'다

AI Director (AI Youtube Auto Upload System)

가장 먼저 해결하고 싶었던 과제는 데일리 뉴스 업로드였다.

- STT 매칭 기술 : 기존의 영상 변화 감지(Scene Detection) 방식은 앵커가 계속 화면에 나올 경우 컷을 나누지 못하는 단점이 있었다. 이를 해결하기 위해 Gemini를 활용, 오디오(Audio)를 텍스트로 변환하고 큐시트 내용과 대조하여 정확한 편집점을 찾아내는 알고리즘을 구현했다.
- 성과 : 기존 4시간 가까이 걸리던 작업을 30분 이내로 단축하여 업무 효율을 높였다.

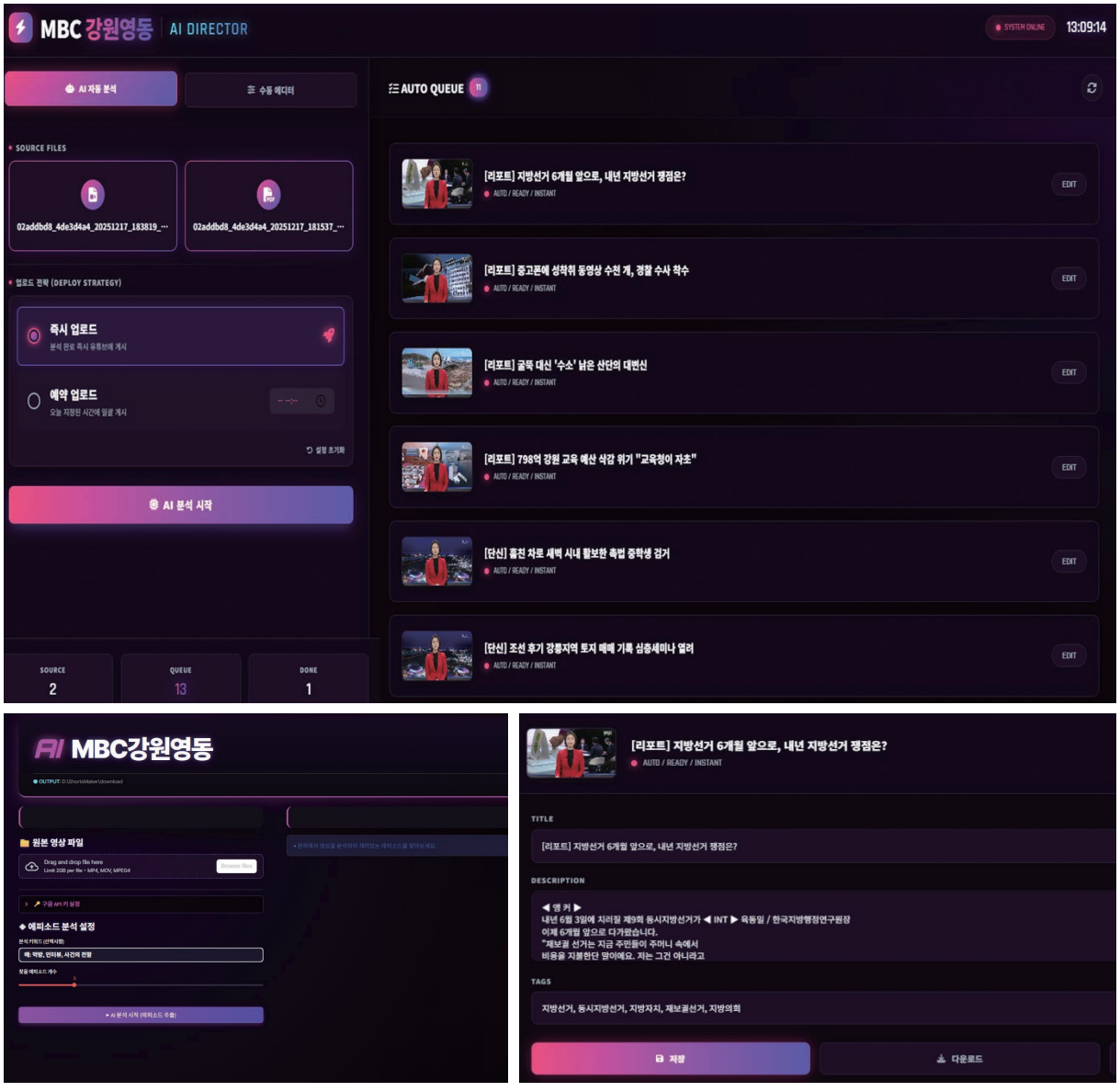

그림 5. 자체 개발한 'AI Director' 메인 대시보드 UI

AI Shorts Maker

Youtube 숏폼(Shorts) 대응을 위한 모듈도 개발했다.

- 기능 : 원본 영상에 대해 원하는 개수만큼의 작은 에피소드로 제작되고, 제작된 에피소드는 다시 1분 이내의 숏폼(Shorts)으로 제작 가능하다.

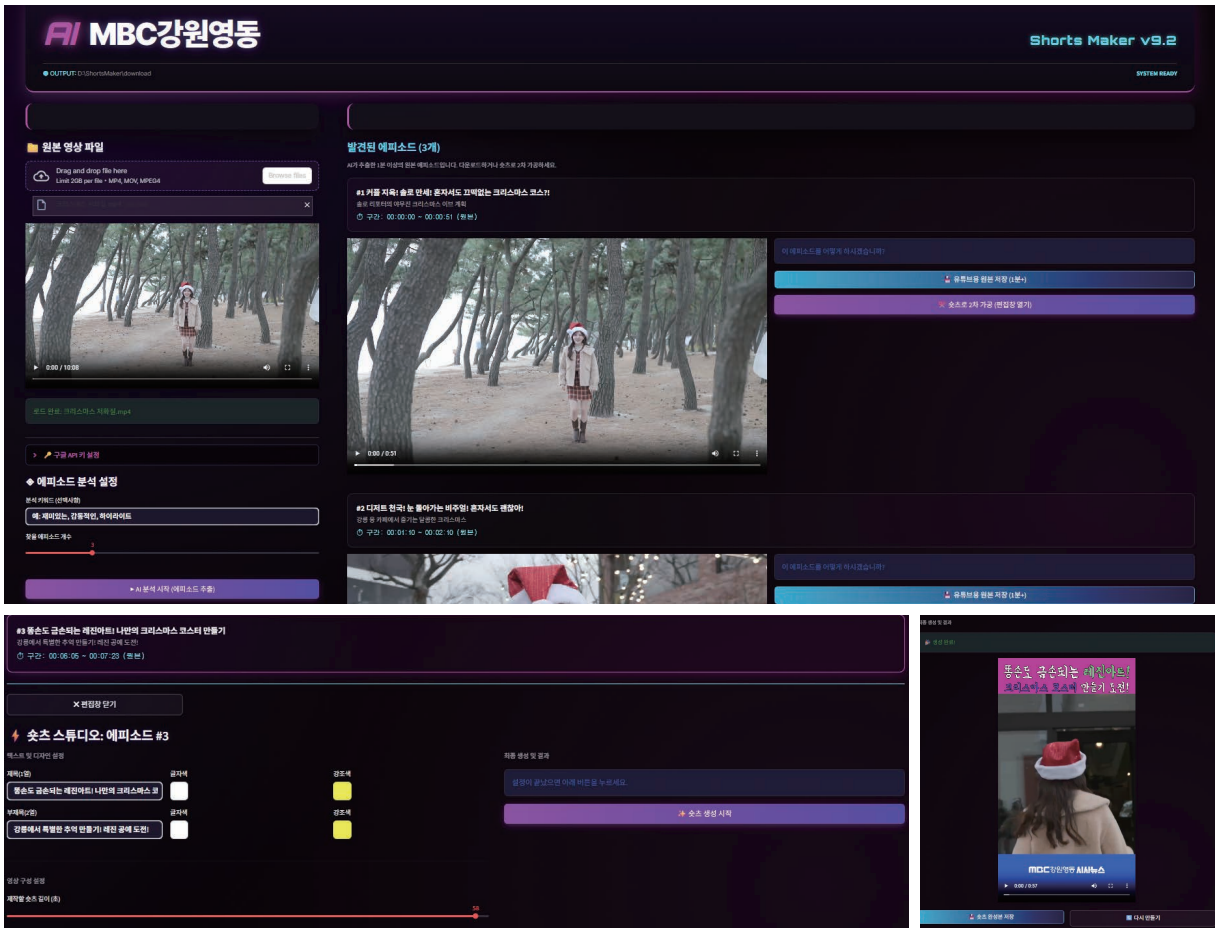

그림 6. AI Shorts Maker 구동 화면 및 결과물

Part 6 에필로그: 아이디어가 곧 코드가 되는 세상

이번 프로젝트를 통해 얻은 가장 큰 수확은 프로그램 그 자체가 아니다. 시스템을 정확하게 아는 사람이 AI를 도구로 쥐었을 때, 상상 이상의 결과물을 만들 수 있다는 확신이다.

코딩을 전공한 개발자는 방송 현장의 가려운 곳을 알지 못한다. 반면, 우리 방송기술인들은 신호의 흐름부터 파일의 구조, 현업 동료들의 고충까지 시스템의 모든 맥락을 이해하고 있다. 바이브 코딩은 이 현장의 지혜를 실행 가능한 도구로 바꿔주는 마법 지팡이일 뿐이다. 마법을 부리는 주체는 여전히 우리다. 방송기술인들이 AI라는 새로운 파트너와 함께 엔지니어링의 새로운 영토를 개척해 나가기를 기대해 본다.

그림 7. AI Shorts Maker 시연 모습