

Contents

바이브 코딩으로 시작하는 생명정보학 실습	7
이 책에 대하여	7
구성	7
서론	7
바이브 코딩이란?	7
왜 생명정보학에 바이브 코딩인가?	7
이 책의 구성	8
이 책의 대상 독자	8
1장. 개발 환경 구성	8
1.1 WSL (Windows Subsystem for Linux)	8
WSL이란?	8
WSL 설치	8
1.2 Visual Studio Code 설치	9
설치 방법	9
설치 확인	9
유용한 확장 프로그램	11
VS Code에서 WSL 연동	11
터미널이란?	11
새 프로젝트 시작하기	13
1.3 VS Code에서 Claude Code 사용하기	14
필수 조건	14
설치 방법	14
시작하기	14
주요 기능	15
주요 단축키	17
1.4 uv와 micromamba	17
uv	17
micromamba	17
uv vs micromamba	18
1.5 정리	18
2장. Git & GitHub	18
2.1 버전 관리 시스템 (Version Control System)	18
버전 관리가 없다면?	18
2.2 Git이란?	18
Git 기본 용어	19

GitHub 기본 용어	19
2.3 Git 설치 및 설정	21
Git 설치	21
사용자 정보 설정	21
2.4 GitHub CLI (gh) 설치 및 설정	21
2.5 GitHub에 저장소 만들기	22
마크다운(Markdown)이란?	22
2.6 Claude Code로 Git 사용하기	23
혼자 작업하기	23
함께 작업하기	25
Stash 활용하기	28
2.7 GitHub에서 협업하기	31
다른 개발자의 프로젝트에 기여하기	31
Fork	31
Pull Request	31
2.8 바이브 코딩에서의 Git	33
2.9 정리	33
3장. Docker	35
3.1 Docker란?	35
왜 Docker가 필요한가?	35
가상 머신과의 차이	36
3.2 Docker 설치	36
설치 확인	37
3.3 Docker 기본 개념	37
이미지 (Image)	37
컨테이너 (Container)	37
Dockerfile	38
Docker Compose	38
3.4 자주 사용하는 Docker 명령어	39
3.5 Docker와 바이브 코딩	40
3.6 정리	40
4장. Python 데이터 분석 기초	40
4.1 왜 Python인가?	40
4.2 프로젝트 환경 구성	40
4.3 pandas — 테이블 데이터 처리	41
핵심 개념	41
코드 예시	41
4.4 NumPy — 수치 연산	42
핵심 개념	42
코드 예시	43
4.5 Matplotlib — 기본 시각화	43
핵심 개념	43
코드 예시	43
4.6 Seaborn — 통계 시각화	44
Matplotlib과의 차이	44
코드 예시	45
4.7 SciPy — 과학 계산과 통계 검정	45
코드 예시	47
4.8 AI와 함께하는 데이터 분석	48
워크플로우	48
핵심 포인트	48
4.9 정리	48

5장. 생명정보 데이터 분석 실전	48
5.1 전장 유전체 시퀀싱(WGS) 분석	48
WGS란?	48
주요 파일 형식	49
분석 도구	49
Claude Code로 WGS 분석하기	49
Python으로 VCF 분석하기	50
5.2 단일세포 RNA-seq 분석	50
단일세포 분석이란?	50
환경 구성	51
AnnData — 단일세포 데이터 구조	51
MuData — 멀티오믹스 데이터	51
Scanpy 분석 워크플로우	52
Claude Code로 단일세포 분석하기	54
5.3 핵심 개념 정리	55
WGS 분석	55
단일세포 분석	55
5.4 정리	56
6장. Snakemake를 이용한 워크플로우 관리	56
6.1 워크플로우란?	56
Snakemake의 장점	56
다른 워크플로우 도구와의 비교	56
6.2 환경 구성	57
6.3 Snakemake 핵심 개념	57
Rule (규칙)	57
Wildcard (와일드카드)	57
DAG (방향성 비순환 그래프)	58
expand() 함수	58
Conda 환경 통합	58
Config 파일	58
로그 파일	58
6.4 Claude Code로 Snakefile 작성하기	58
RNA-seq 파이프라인 생성	58
Conda 환경 분리 요청	60
Config 파일 분리	60
파이프라인 확장	60
실행과 디버깅	60
6.5 단일세포 분석 파이프라인	60
6.6 정리	61
7장. 생명정보 웹 툴 제작 기초	61
7.1 웹 도구, 아직 만들어야 하는가?	61
7.2 이 책에서 사용하는 기술 스택	62
SvelteKit	62
Tailwind CSS	63
PostgreSQL	63
7.3 프로젝트 생성	64
Node.js 설치	64
SvelteKit 프로젝트 초기화	66
Tailwind CSS	66
7.4 Docker 환경 구성	66
compose.yml과 Dockerfile 생성	66
7.5 환경 변수 (.env)	67

7.6 프로젝트 디렉토리 구조	67
CLAUDE.md 작성	68
7.7 개발 서버 실행	69
Docker를 사용하는 경우	69
Docker 없이 로컬에서 실행하는 경우	70
7.8 정리	70
8장. 랜딩 페이지 디자인	71
8.1 랜딩 페이지란?	71
8.2 랜딩 페이지의 구조	71
Header, Body, Footer	71
Navigation Bar (Navbar)	72
Hero Section	72
Carousel	72
Features Section	74
8.3 UI 컴포넌트	74
8.4 AI를 활용한 디자인 목업 생성	75
이미지 생성 AI를 활용한 디자인	75
Claude Code의 design 스킬	75
8.5 SvelteKit에서 랜딩 페이지 구현	75
레이아웃 구성	75
컴포넌트 분리	78
랜딩 페이지 조립	78
8.6 정리	79
9장. 일반 페이지 디자인	79
9.1 일반 페이지란?	79
9.2 일반 페이지의 구조	80
공동 레이아웃	80
Page Header (Breadcrumb 포함)	80
Sidebar	80
9.3 도구 페이지에서 자주 사용되는 컴포넌트	80
입력 폼 컴포넌트	80
출력/결과 컴포넌트	81
시각화 컴포넌트	81
9.4 일반 페이지 디자인 패턴	83
단일 도구 페이지	83
다중 도구 페이지	84
탭 기반 결과 표시	85
9.5 SvelteKit에서 일반 페이지 구현	85
파일 기반 라우팅	85
Reverse Complement 예시	86
9.6 AI를 활용한 일반 페이지 디자인	88
9.7 정리	88
10장. BLAST 검색 도구 만들기	89
10.1 BLAST란?	89
이 장에서 사용하는 도구와 개념	89
완성된 도구의 모습	90
10.2 BLAST+ 설치와 데이터베이스 준비	90
Docker에 BLAST+ 추가	91
예시 데이터베이스 준비	91
compose.yml 업데이트	91
10.3 백엔드 API 구현	92

BLAST 실행 함수	92
API 엔드포인트	92
10.4 검색 결과를 데이터베이스에 저장	92
데이터베이스 스키마와 저장 로직	92
10.5 프론트엔드 구현	92
검색 폼과 결과 화면	92
10.6 AI를 활용한 확장	93
프롬프트 예시	93
10.7 정리	94
11장. MCP와 Claude for Life Sciences	94
11.1 MCP란?	94
MCP가 필요한 이유	94
MCP 이전 vs 이후	94
MCP의 구조	95
11.2 Claude Code에서 MCP 서버 설정하기	95
claude mcp add 명령	95
설정 확인	96
여러 MCP 서버 동시 사용	96
11.3 MCP 마켓플레이스	97
mcpmarket.com	97
11.4 생명정보학 MCP 서버	97
bioRxiv / medRxiv	97
UniProt	98
PubMed	98
fetch (범용 웹 접근)	98
MCPmed	98
11.5 Claude for Life Sciences	98
Scientific Connectors	98
Agent Skills	99
주요 활용 분야	99
11.6 MCP와 바이브 코딩	99
11.7 정리	99
12장. Co-scientist 에이전트 설계	100
12.1 AI Co-scientist란?	100
Generative Agents	100
The AI Scientist	100
12.2 Co-scientist의 구성 요소	101
12.3 자동으로 업데이트되는 기억	101
CLAUDE.md	102
자동 메모리	103
12.4 나만의 MCP 서버 만들기	103
Python FastMCP	103
Claude Code로 MCP 서버 만들기	103
Claude Code에 연결	104
12.5 커스텀 에이전트 정의	104
Claude Code로 에이전트 만들기	104
연구 역할별 에이전트 구성	105
12.6 Hooks로 자동 규칙 만들기	105
Hook의 구조	105
Hook 이벤트	106
Claude Code로 Hook 만들기	106
12.7 Skills로 전문 능력 부여하기	106

Claude Code로 Skill 만들기	106
12.8 멀티에이전트 워크플로우	106
새로운 항암 표적 탐색	107
단일세포 분석 워크플로우	107
12.9 Co-scientist의 한계와 원칙	107
설계 원칙	107
데이터 보안	107
12.10 정리	107
13장. AI로 논문 작성하기	108
13.1 AI 논문 작성의 현실	108
올바른 사용과 잘못된 사용	108
13.2 투명하고 재현 가능한 AI 논문 작성	108
5단계 워크플로우	108
13.3 Paper Writing Assistant	110
주요 특징	110
투명성 보고서	110
13.4 Co-scientist로 논문 작성 보조하기	110
문헌 리뷰 → 논문 작성 워크플로우	110
섹션별 작성 요청	111
Figure 생성	111
Discussion 작성	111
13.5 AI 논문 작성 시 주의사항	112
학술지별 AI 사용 정책 확인	112
반드시 사람이 검증	112
알려진 한계	112
13.6 정리	112
14장. AI Co-scientist와 함께 분석하기	113
14.1 AI 시대의 과학적 발견	113
14.2 AI 연구팀 구성하기	113
팀장으로서의 연구자	113
팀원 역할 설계	113
팀원 에이전트 생성	114
14.3 프로젝트 환경 구축	115
프로젝트 디렉토리 생성	115
Docker 환경 설정	115
CLAUDE.md 작성	115
MCP 서버 연결	115
알림 시스템 설정	115
14.4 실전 예제: 폐암 단일세포 전사체 분석	116
Phase 1: 데이터 준비	116
Phase 2: 데이터 이해	117
Phase 3: 가설 도출	118
Phase 4: 가설 검증	118
Phase 5: 결과 종합 및 논문 작성	118
14.5 알림 시스템 활용 팁	119
플랫폼별 Webhook 형식	119
다른 Hook 이벤트 활용	119
14.6 연구자의 역할: 무엇을 판단해야 하는가	119
체크포인트	119
14.7 이 책을 마치며	120
14.8 정리	120

바이브 코딩으로 시작하는 생명정보학 실습

저자: 박정빈

이 책에 대하여

이 책은 AI 코딩 에이전트(Claude Code)를 활용하여 생명정보학 분석과 웹 도구 개발을 실습하는 안내서이다. 코드를 한 줄 한 줄 외우는 대신, 핵심 개념을 이해하고 AI에게 자연어로 지시하는 **바이브 코딩(Vibe Coding)** 방식을 다룬다.

프로그래밍 경험이 없거나 적은 생명과학 연구자를 주요 독자로 상정하며, 개발 환경 구성부터 Git, Docker, Python 데이터 분석, 단일세포 RNA-seq 분석, Snakemake 워크플로우 자동화, SvelteKit 기반 웹 도구 제작, MCP 프로토콜, AI Co-scientist 설계, 그리고 AI 논문 작성까지 실무에 필요한 전 과정을 단계별로 안내한다.

구성

파트	내용	챕터
I. 개발 기초 도구	개발 환경, 버전 관리, 컨테이너	1~3장
II. 파이썬으로 분석하기	데이터 분석, 단일세포 분석, 워크플로우	4~6장
III. 생명정보 웹 툴 만들기	SvelteKit, UI 디자인, BLAST 도구	7~10장
IV. AI Co-scientist 만들기	MCP, 에이전트 설계, 논문 작성, 실전 분석	11~14장

서론

바이브 코딩이란?

2025년 2월, OpenAI의 공동 창립자 안드레이 카파시(Andrej Karpathy)가 트위터에 “바이브 코딩(Vibe Coding)”이라는 용어를 처음 사용했다. 코드의 문법을 외우고 한 줄 한 줄 직접 작성하는 대신, AI에게 자연어로 지시하여 프로그램을 만드는 새로운 개발 방식이다.

바이브 코딩에서 사람의 역할은 코드를 작성하는 것이 아니라, **무엇을 만들지 결정하고, AI가 만든 결과를 검증하는 것이다.** “BLAST 검색 결과를 테이블로 보여주는 웹 페이지를 만들어줘”라고 요청하면, AI가 코드를 생성하고 파일을 수정하고 테스트까지 수행한다. 사람은 결과를 확인하고 방향을 조정한다.

이 방식이 가능해진 것은 AI 코딩 에이전트의 발전 덕분이다. 단순한 코드 자동 완성을 넘어, 파일 시스템을 탐색하고, 터미널 명령을 실행하고, 여러 파일을 동시에 수정할 수 있는 에이전트가 등장했다. 이 책에서 사용하는 Claude Code가 대표적이다.

바이브 코딩의 힘을 보여주는 사례가 있다. 2025년 Anthropic이 개최한 Claude Code 해커톤에서 5명의 수상자 중 소프트웨어 개발자는 2명뿐이었다. 금상은 캘리포니아 건설 허가 법률 전문가가, 동상은 벨기에 심장내과 과장이, 우간다 인프라 엔지니어가 수상했다. 각자 자신의 도메인에서 발견한 문제를 Claude Code로 해결한 것이다. **어떻게 만드는가(코딩 기술)보다 무엇이 필요한가(문제 인식)가 AI 시대의 핵심 경쟁력**이라는 점을 잘 보여준다.

왜 생명정보학에 바이브 코딩인가?

생명정보학은 바이브 코딩과 특히 잘 어울리는 분야이다. 그 이유는 다음과 같다.

도메인 지식이 코딩 실력보다 중요하다. BLAST 검색 결과에서 E-value가 의미하는 바를 이해하는 것, 단일세포 데이터에서 미토콘드리아 유전자 비율이 높은 세포를 제거해야 하는 이유를 아는 것 — 이런 판단은 AI가 대신하기 어렵다. 반면 그 판단을 코드로 구현하는 작업은 AI가 할 수 있다.

도구와 포맷이 다양하다. FASTA, FASTQ, BAM, VCF, AnnData 등 생명정보학에서 다루는 파일 포맷은 수십 가지이고, 각각을 처리하는 도구도 다르다. 이 모든 도구의 사용법을 외우는 것은 비효율적이다. AI에게 “이 BAM 파일에서 매핑 퀄리티 30 이상인 리드만 추출해줘”라고 요청하는 것이 더 효과적이다.

재현 가능한 분석이 중요하다. 생명정보학 분석 결과는 논문에 실린다. 다른 연구자가 동일한 결과를 재현할 수 있어야 한다. Docker로 환경을 고정하고, Snakemake로 파이프라인을 자동화하고, Git으로 버전을 관리하는 것 — 이 과정을 AI와 함께 구축하면 훨씬 빠르고 체계적으로 진행할 수 있다.

이 책의 구성

이 책은 네 개의 파트로 구성되어 있다.

파트 I. 개발 기초 도구에서는 개발 환경을 구성하고(1장), Git으로 버전을 관리하고(2장), Docker로 일관된 환경을 만드는 방법(3장)을 다룬다. 모든 후속 장의 기반이 되는 내용이다.

파트 II. 파이썬으로 분석하기에서는 Python 데이터 분석의 기초(4장)부터 Scanpy를 이용한 단일세포 RNA-seq 분석(5장), Snakemake를 이용한 워크플로우 자동화(6장)까지 다룬다. 생명정보학 연구자가 가장 자주 수행하는 분석 작업이다.

파트 III. 생명정보 웹 툴 만들기에서는 SvelteKit과 Tailwind CSS로 프로젝트를 설정하고(7장), 랜딩 페이지(8장)와 일반 페이지(9장)를 디자인한 뒤, 실제로 동작하는 BLAST 검색 도구(10장)를 만든다. 분석 결과를 웹으로 공유하고, 연구실 내부 도구를 만드는 데 활용할 수 있다.

파트 IV. AI Co-scientist 만들기에서는 MCP 프로토콜로 Claude Code의 기능을 확장하고(11장), AI Co-scientist 에이전트를 설계하고(12장), AI를 활용한 논문 작성 워크플로우(13장)를 소개한다.

이 책의 대상 독자

이 책은 프로그래밍 경험이 없거나 적은 생명과학 연구자를 주요 독자로 상정한다. Python 문법을 몰라도, 웹 개발을 해본 적이 없어도 괜찮다. 각 장에서 필요한 개념을 설명하고, 구현은 Claude Code와 함께 진행한다.

다만 생명정보학의 기본 개념 — 유전자, 시퀀싱, 전자체 등 — 은 어느 정도 알고 있다고 가정한다. 이 책은 생명정보학 자체를 가르치는 교재가 아니라, 생명정보학 도구를 AI와 함께 만드는 실습서이기 때문이다.

1장. 개발 환경 구성

코딩을 시작하려면 먼저 작업 공간을 갖춰야 한다. 이 장에서는 코드 편집기(VS Code), AI 코딩 에이전트(Claude Code), 그리고 패키지 관리 도구(uv, micromamba)를 설치하고 설정한다. Windows 사용자는 리눅스 환경(WSL)을 먼저 구성해야 하므로 그 과정도 함께 다룬다.

1.1 WSL (Windows Subsystem for Linux)

생명정보학 도구 대부분은 리눅스 환경을 전제로 개발되었다. 대용량 시퀀싱 데이터를 처리하는 고성능 서버와 클라우드 인프라(AWS, Google Cloud)가 리눅스로 운영되기 때문이다. samtools, BWA, BLAST+, GATK 같은 핵심 도구들이 모두 리눅스에서 만들어졌고, Windows에서는 직접 실행하기 어렵다. macOS는 유닉스 계열이라 대부분의 도구가 그대로 동작하지만, Windows 사용자는 **WSL(Windows Subsystem for Linux)**을 설치하여 리눅스 환경을 갖춰야 한다. macOS나 Linux 사용자는 이 절을 건너뛰어도 된다.

WSL이란?

WSL은 Windows 위에서 리눅스 배포판을 직접 실행할 수 있게 해주는 기능이다. 별도의 가상 머신 없이도 리눅스 터미널과 명령어를 사용할 수 있다. WSL을 사용하면 Windows의 편의성을 유지하면서 리눅스 생태계의 모든 도구를 활용할 수 있다.

WSL 설치

Microsoft Store를 열고 “Ubuntu”를 검색하여 설치한다. 터미널에서 명령어를 입력할 필요 없이, 평소 앱을 설치하듯 받기 버튼을 클릭하면 된다.

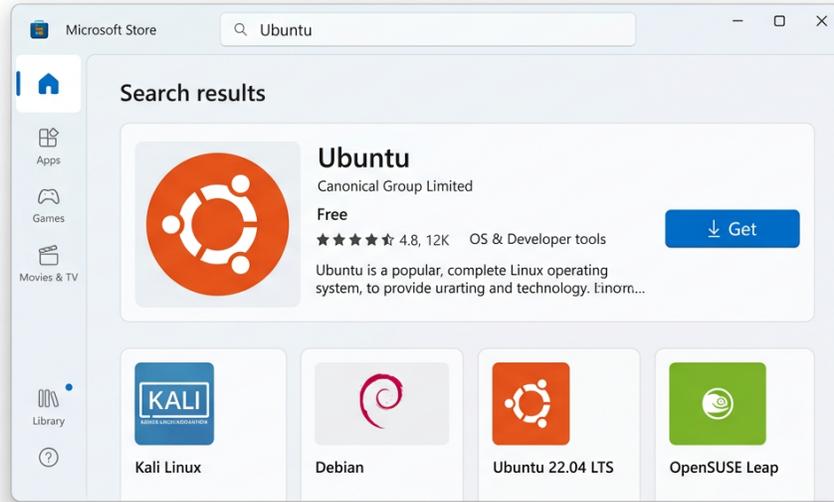


Figure 1: Microsoft Store에서 Ubuntu를 검색하여 설치하는 화면 스크린샷

설치가 완료되면 컴퓨터를 재부팅해야 한다. 재부팅하지 않으면 WSL이 정상적으로 동작하지 않으므로 반드시 재부팅한다.

재부팅 후 Ubuntu를 처음 실행하면, 리눅스 사용자 이름과 비밀번호를 설정하라는 메시지가 나타난다. 이때 설정하는 비밀번호는 `sudo` 명령(관리자 권한 실행)을 사용할 때 필요하므로 잊지 않도록 한다.

1.2 Visual Studio Code 설치

Visual Studio Code(VS Code)는 Microsoft에서 개발한 무료 코드 편집기다. 다양한 프로그래밍 언어를 지원하며 확장 프로그램을 통해 기능을 추가할 수 있다. 이 책에서는 VS Code를 기본 개발 환경으로 사용한다.

VS Code가 개발자들 사이에서 가장 인기 있는 편집기가 된 데는 몇 가지 이유가 있다. 먼저, 무료이면서도 IntelliSense(코드 자동 완성), 디버거, Git 통합 등 유료 IDE 못지않은 기능을 제공한다. 또한 Windows, macOS, Linux를 모두 지원하므로 운영체제에 관계없이 동일한 환경에서 작업할 수 있다. 무엇보다, 이 책에서 사용할 Claude Code가 VS Code 확장 프로그램으로 제공되기 때문에 VS Code를 선택했다.

설치 방법

1. <https://code.visualstudio.com> 에 접속한다
2. 운영체제에 맞는 설치 파일을 다운로드한다 (Windows, macOS, Linux)
3. 다운로드한 파일을 실행하여 설치를 완료한다

설치 확인

설치가 완료되면 VS Code를 실행하여 정상적으로 동작하는지 확인한다. 처음 실행하면 Welcome 탭이 나타나며, 여기서 색상 테마를 선택하거나 기본 설정을 조정할 수 있다.

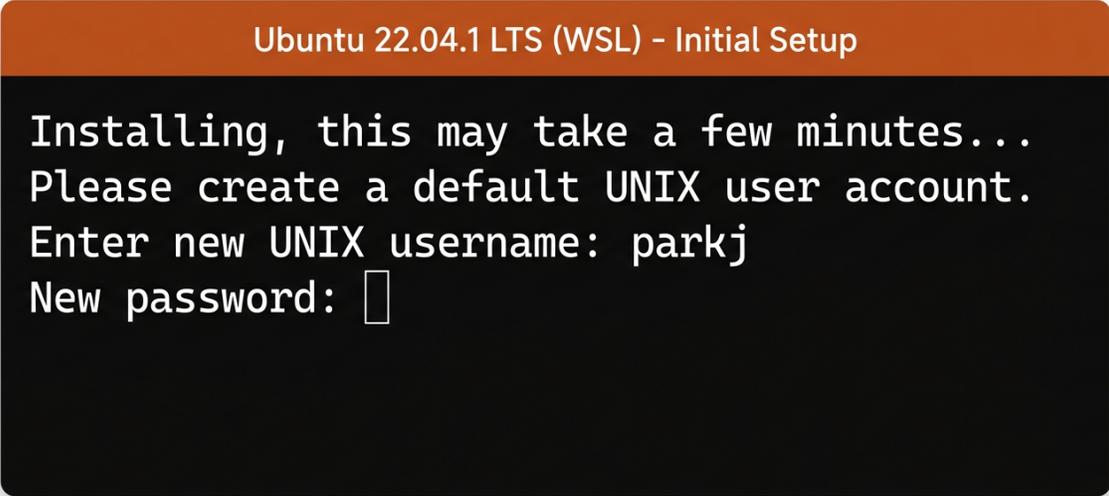


Figure 2: WSL Ubuntu 초기 설정 — 사용자 이름과 비밀번호 입력 화면 스크린샷

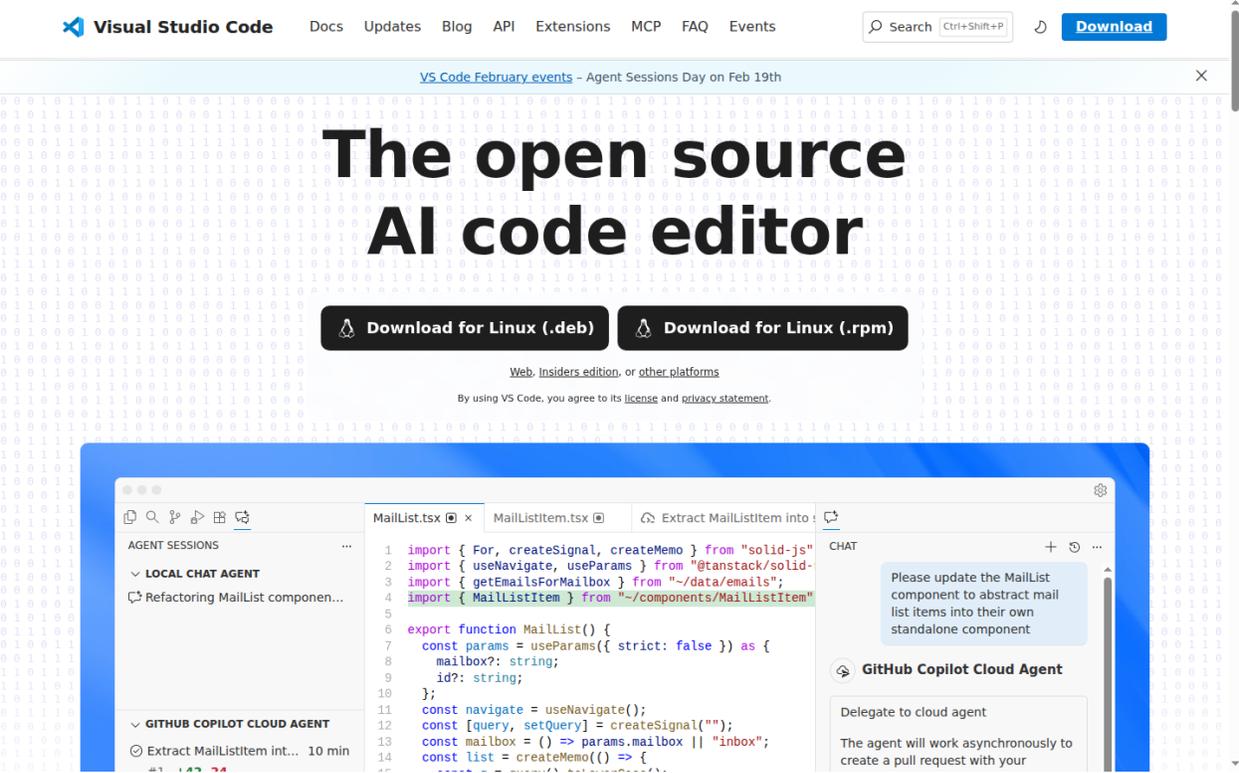


Figure 3: VS Code 공식 웹사이트에서 다운로드 버튼이 보이는 메인 페이지 스크린샷

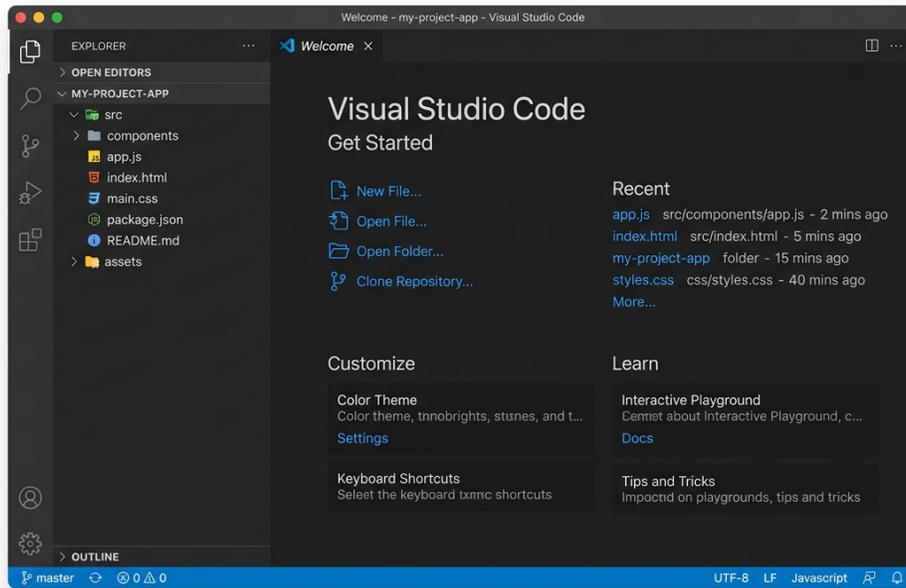


Figure 4: VS Code를 처음 실행했을 때의 Welcome 탭 화면 스크린샷

유용한 확장 프로그램

VS Code의 강력한 장점 중 하나는 확장 프로그램(Extension)이다. 왼쪽 사이드바의 확장 프로그램 아이콘을 클릭하거나 `Cmd+Shift+X` (Mac) / `Ctrl+Shift+X` (Windows/Linux)를 눌러 확장 프로그램 마켓플레이스를 열 수 있다.

확장 프로그램 마켓플레이스에는 수만 개의 확장 프로그램이 등록되어 있다. Python 개발을 위한 Python 확장, 코드 포매팅을 위한 Prettier, Git 히스토리를 시각화하는 GitLens 등이 대표적이다. 앞으로 필요한 확장 프로그램은 이곳에서 검색하여 설치하면 된다. 이 책에서는 WSL 확장과 Claude Code 확장을 주로 사용하게 된다.

VS Code에서 WSL 연동

Windows 사용자는 VS Code에서 WSL 환경을 연결해야 한다. WSL을 설치한 뒤 재부팅했다면, 별도로 확장 프로그램을 검색하여 설치할 필요 없이 바로 연결할 수 있다.

1. VS Code 좌측 하단의 파란색 >< 아이콘을 클릭
2. 메뉴에서 “WSL”을 선택하면 WSL 확장 프로그램이 자동으로 설치된다
3. 설치가 완료되면 >< 아이콘을 다시 클릭
4. “Connect to WSL”을 선택하여 WSL 환경에 연결한다

연결이 완료되면 VS Code 좌측 하단에 “WSL: Ubuntu”라고 표시되며, 터미널을 열면 리눅스 셸이 실행된다. 이후 이 책의 모든 터미널 명령은 WSL 환경에서 실행하면 된다.

이 연결이 되어 있는 상태에서는 VS Code의 파일 탐색기, 터미널, 확장 프로그램이 모두 WSL 환경에서 동작한다. Windows 파일 시스템이 아닌 리눅스 파일 시스템(/home/ /)에서 프로젝트를 생성하고 관리하는 것이 성능 면에서 유리하다.

터미널이란?

터미널(Terminal)은 텍스트 명령으로 컴퓨터와 소통하는 창이다. 마우스로 아이콘을 클릭하는 대신, 명령어를 직접 입력하여 파일을 관리하거나 프로그램을 실행할 수 있다. 이 책에서 “터미널에서 실행한다”라고 하면, 이 텍스트 입력 창에 명령어를 입력한다는 뜻이다.

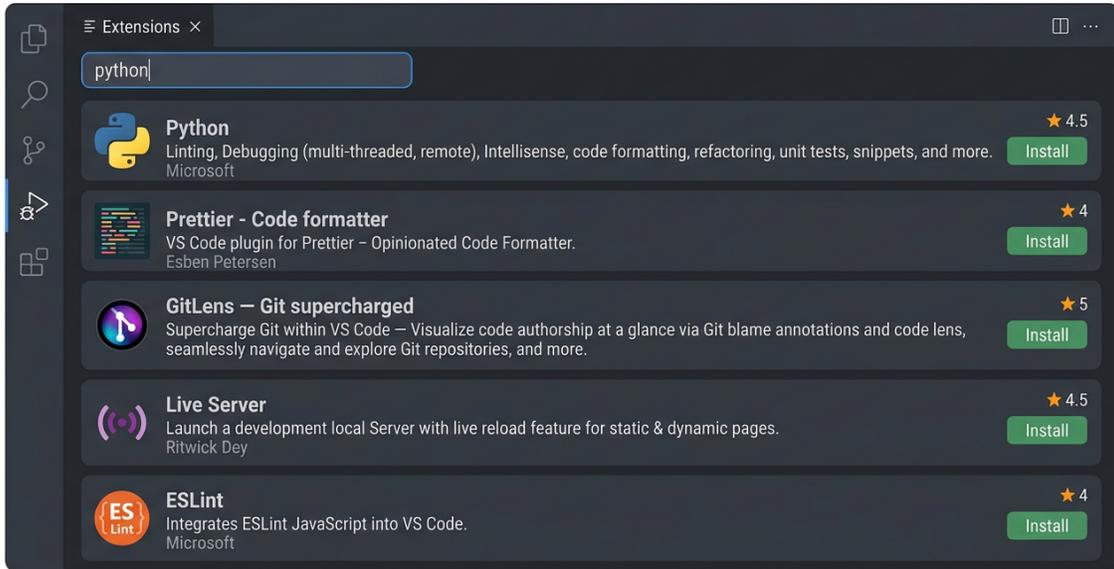


Figure 5: VS Code 확장 프로그램 마켓플레이스 화면 스크린샷

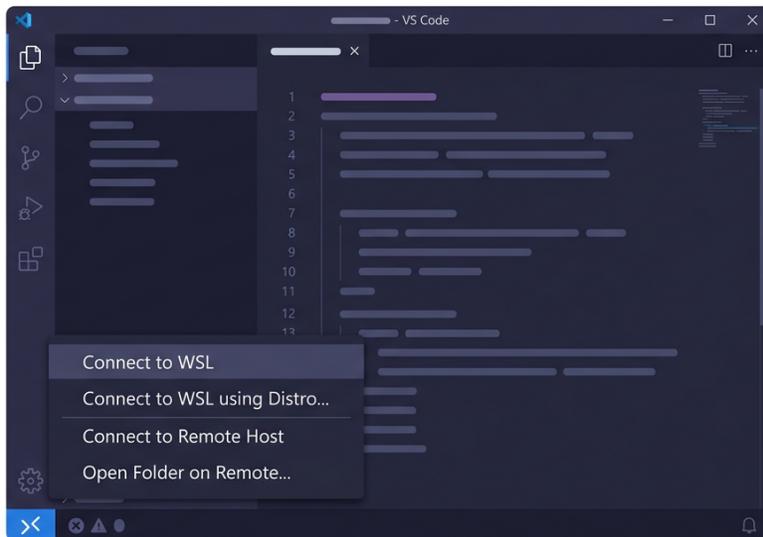


Figure 6: VS Code 좌측 하단 <X> 아이콘을 클릭하여 WSL 메뉴를 선택하는 화면 스크린샷

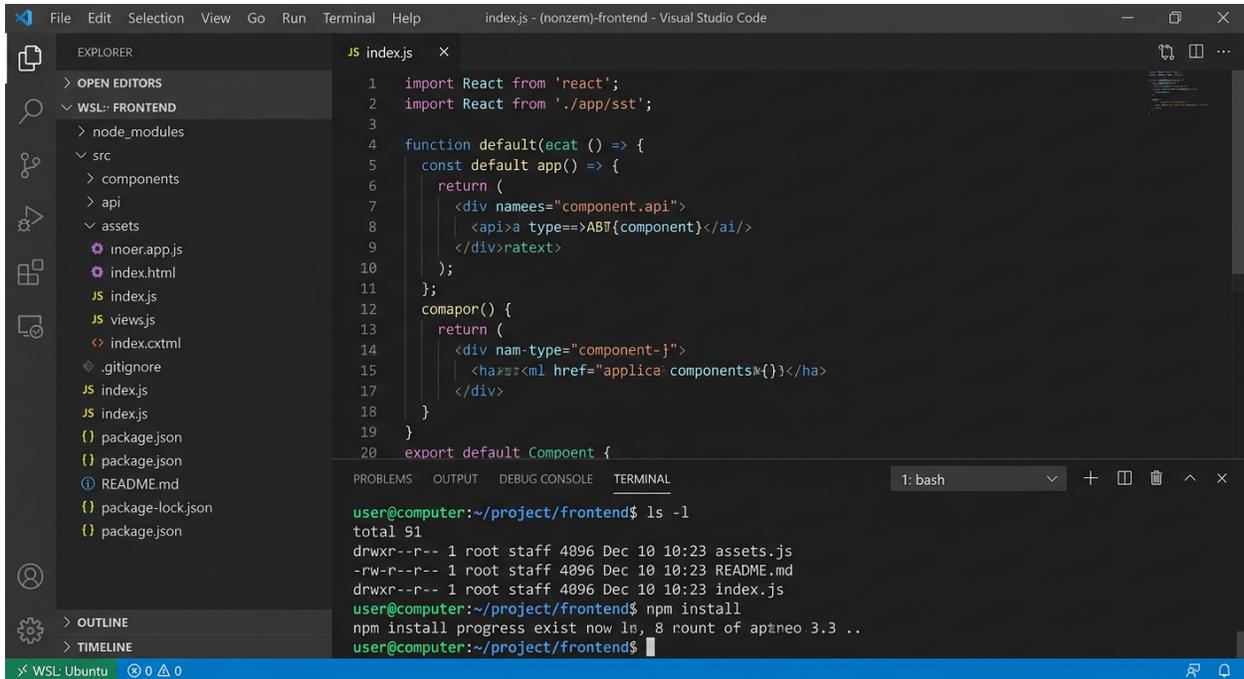


Figure 7: VS Code가 WSL에 연결된 상태 — 좌측 하단에 “WSL: Ubuntu” 표시와 리눅스 터미널이 열린 모습

터미널을 여는 방법은 두 가지가 있다.

방법 1: Ubuntu 앱으로 직접 열기

Windows 시작 메뉴에서 “Ubuntu”를 검색하여 실행하면, 검은 배경의 텍스트 입력 창이 나타난다. 이것이 리눅스 터미널이다. WSL 설치 시 설정한 사용자 이름이 표시되며, 여기에 명령어를 입력할 수 있다.

방법 2: VS Code 내장 터미널 사용 (권장)

VS Code 안에서도 터미널을 열 수 있다. 상단 메뉴에서 터미널 → 새 터미널을 선택하거나, 단축키 `Ctrl+``(백틱)을 누르면 편집기 아래쪽에 터미널 패널이 나타난다. WSL에 연결된 상태라면 자동으로 리눅스 터미널이 열린다.

VS Code 내장 터미널을 권장하는 이유는 코드 편집과 터미널 명령을 한 화면에서 모두 처리할 수 있기 때문이다. 파일을 수정하면서 바로 아래 터미널에서 실행 결과를 확인할 수 있어 효율적이다.

macOS나 Linux 사용자는 시스템에 기본 설치된 터미널 앱을 사용하거나, 마찬가지로 VS Code 내장 터미널을 사용하면 된다.

새 프로젝트 시작하기

이 책에서는 여러 프로젝트를 새로 만들게 된다. 새 프로젝트를 시작할 때는 항상 다음 과정을 거친다:

1. VS Code 터미널에서 프로젝트 디렉토리를 생성한다:

```
mkdir my-project
```

2. VS Code에서 파일 → 폴더 열기(`Ctrl+K Ctrl+O`)를 선택하여 생성한 디렉토리를 연다.

VS Code가 프로젝트 디렉토리를 기준으로 열려야 Claude Code가 프로젝트의 전체 구조를 파악하고 정확하게 동작할 수 있다. 이 패턴은 앞으로 반복적으로 등장하므로 익숙해지는 것이 좋다.

1.3 VS Code에서 Claude Code 사용하기

Claude Code는 Anthropic이 만든 AI 코딩 에이전트다. VS Code 확장 프로그램으로 설치하여 사용할 수 있으며, 코드 작성, 디버깅, 리팩토링 등 다양한 작업을 AI의 도움을 받아 수행할 수 있다.

Claude Code의 핵심은 **에이전트** 방식으로 동작한다는 것이다. 단순한 코드 자동 완성이 아니라, 파일을 읽고, 수정하고, 터미널 명령을 실행하고, Git 작업까지 수행할 수 있다. 마치 옆에 앉아서 함께 코딩하는 동료처럼 동작한다. 이 책 전체에서 Claude Code를 활용하여 생명정보학 도구를 만들게 된다.

필수 조건

- VS Code 1.98.0 이상
- Claude 정기구독 (Max 플랜 권장, <https://claude.ai> 에서 구독)

설치 방법

VS Code에서 `Cmd+Shift+X` (Mac) 또는 `Ctrl+Shift+X` (Windows/Linux)를 눌러 확장 프로그램 보기를 열고, “Claude Code”를 검색한 후 설치를 클릭한다.

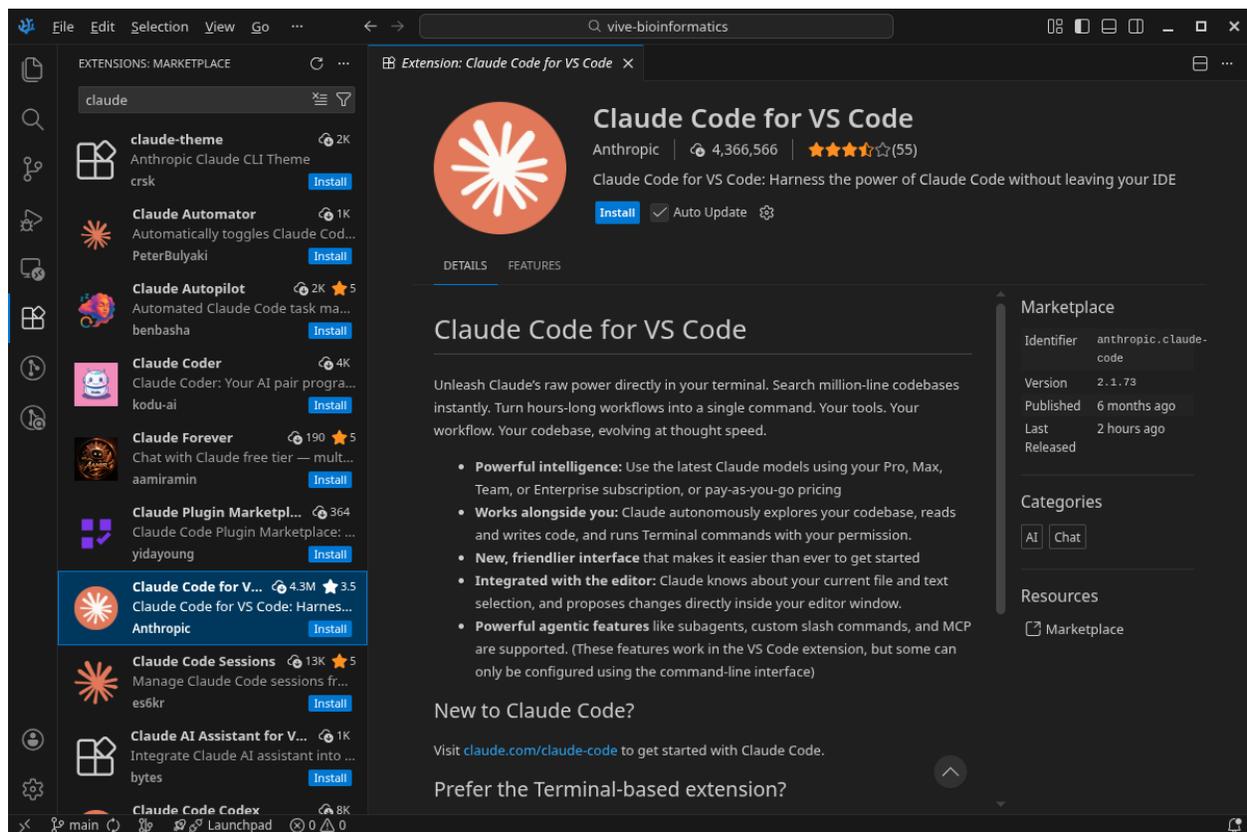


Figure 8: VS Code 확장 프로그램 마켓플레이스에서 Claude Code를 검색하여 설치하는 화면 스크린샷

시작하기

Claude Code 패널 열기 편집기 오른쪽 위 모서리의 **Spark 아이콘**을 클릭하여 Claude Code 패널을 연다. 또는 다음 방법으로 열 수 있다:

- **명령 팔레트**: `Cmd+Shift+P` (Mac) 또는 `Ctrl+Shift+P` (Windows/Linux)를 누르고 “Claude Code”를 입력
- **상태 표시줄**: 창 오른쪽 아래의 **Claude Code**를 클릭

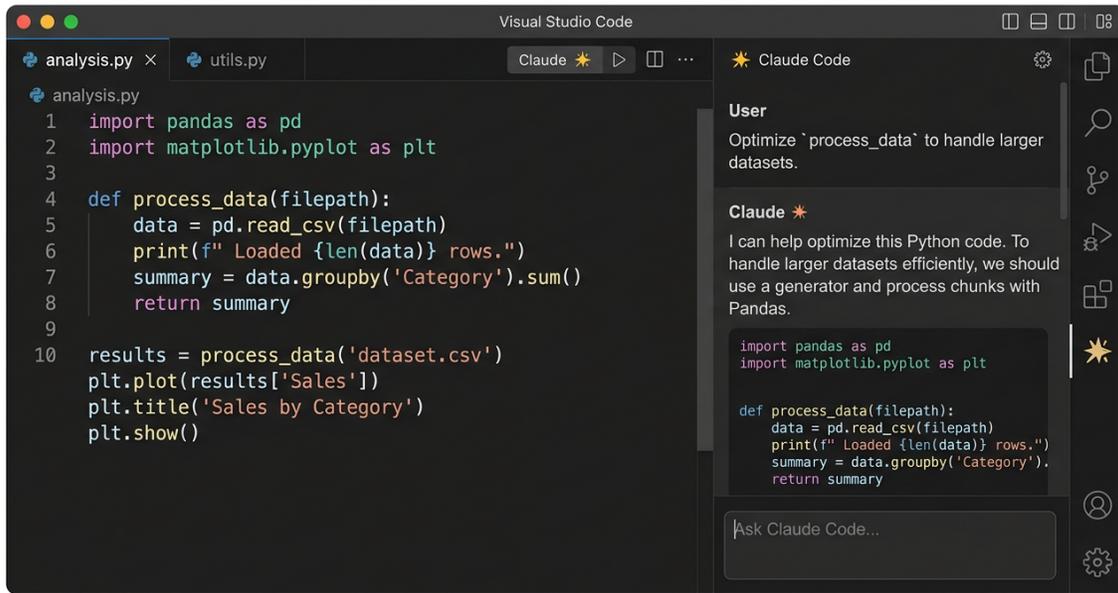


Figure 9: VS Code 편집기에서 Spark 아이콘 위치와 Claude Code 패널이 열린 모습 스크린샷

프롬프트 보내기 Claude에게 코드에 대해 질문하거나, 디버깅을 요청하거나, 변경 사항 작성을 요청한다. 편집기에서 텍스트를 선택하면 Claude가 해당 코드를 자동으로 인식한다.

- @ 뒤에 파일명을 입력하면 특정 파일을 참조할 수 있다 (예: @auth.js)
- Option+K (Mac) / Alt+K (Windows/Linux)로 현재 파일과 선택 영역의 참조를 삽입할 수 있다

@- 기능은 특히 유용하다. 프로젝트에 파일이 수십 개 있을 때, Claude가 어떤 파일을 참조해야 하는지 명확히 알려줄 수 있기 때문이다. 예를 들어 @src/lib/server/db.ts API 라고 요청하면, Claude는 해당 파일의 패턴을 따라 일관된 코드를 작성해 준다.

변경 사항 검토 Claude가 파일을 편집하려고 하면, 원본과 제안된 변경 사항을 나란히 비교하는 diff 화면이 나타난다. 수락하거나 거부하거나, Claude에게 다른 방법으로 수정하도록 요청할 수 있다.

이 diff 화면은 바이브 코딩에서 매우 중요한 역할을 한다. 시가 제안한 코드를 무조건 수락하는 것이 아니라, 변경 사항을 확인하고 이해한 뒤 수락 여부를 결정할 수 있기 때문이다. 특히 데이터베이스 스키마 변경이나 보안 관련 코드는 반드시 diff를 확인하는 습관을 들이는 것이 좋다.

주요 기능

권한 모드 프롬프트 상자 하단의 모드 표시기를 클릭하여 전환한다:

모드	설명
일반 모드	각 작업 전에 권한을 요청
Plan Mode	수행할 작업을 설명하고 승인을 기다림
자동 수락 모드	요청하지 않고 바로 편집

처음에는 **일반 모드**로 시작하여 Claude가 어떤 작업을 수행하는지 하나씩 확인하면서 익숙해지는 것을 권장한다. 작업 흐름에 익숙해지면 자동 수락 모드로 전환하여 더 빠르게 작업할 수 있다. Plan Mode는 복잡한 리팩토링처럼 큰 변경을 수행하기 전에 계획을 먼저 확인하고 싶을 때 유용하다.

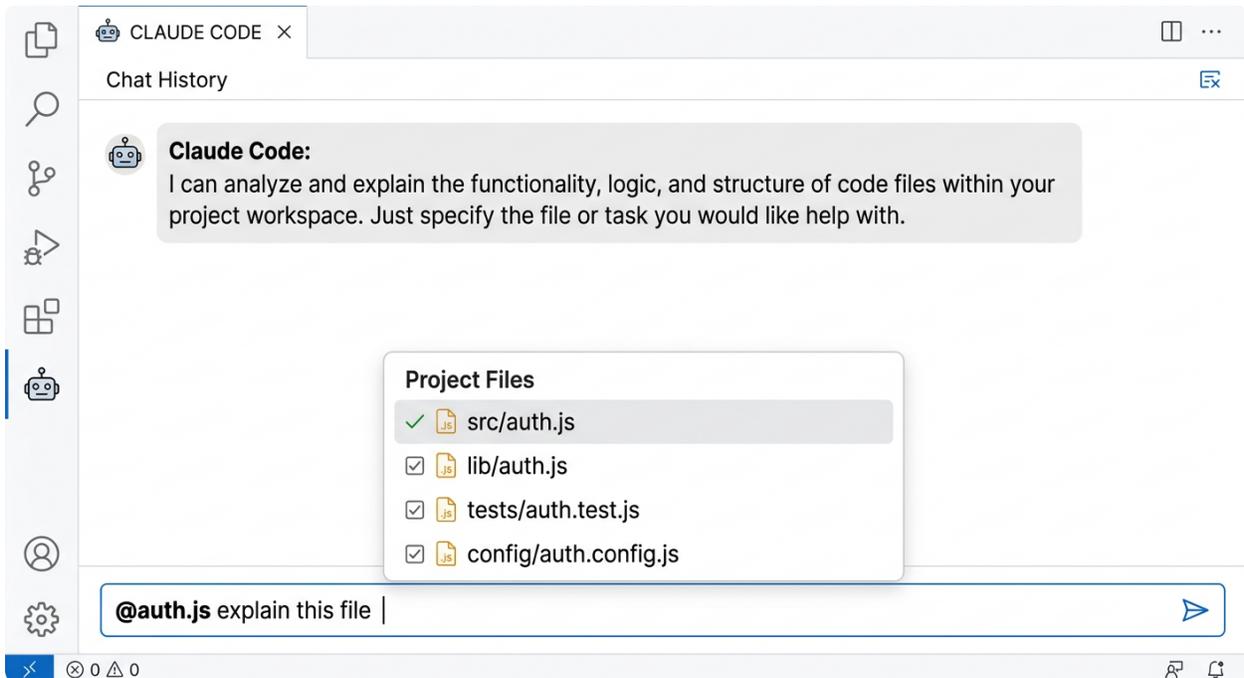


Figure 10: Claude Code 패널에 프롬프트를 입력하고 @-멘션으로 파일을 참조하는 모습 스크린샷

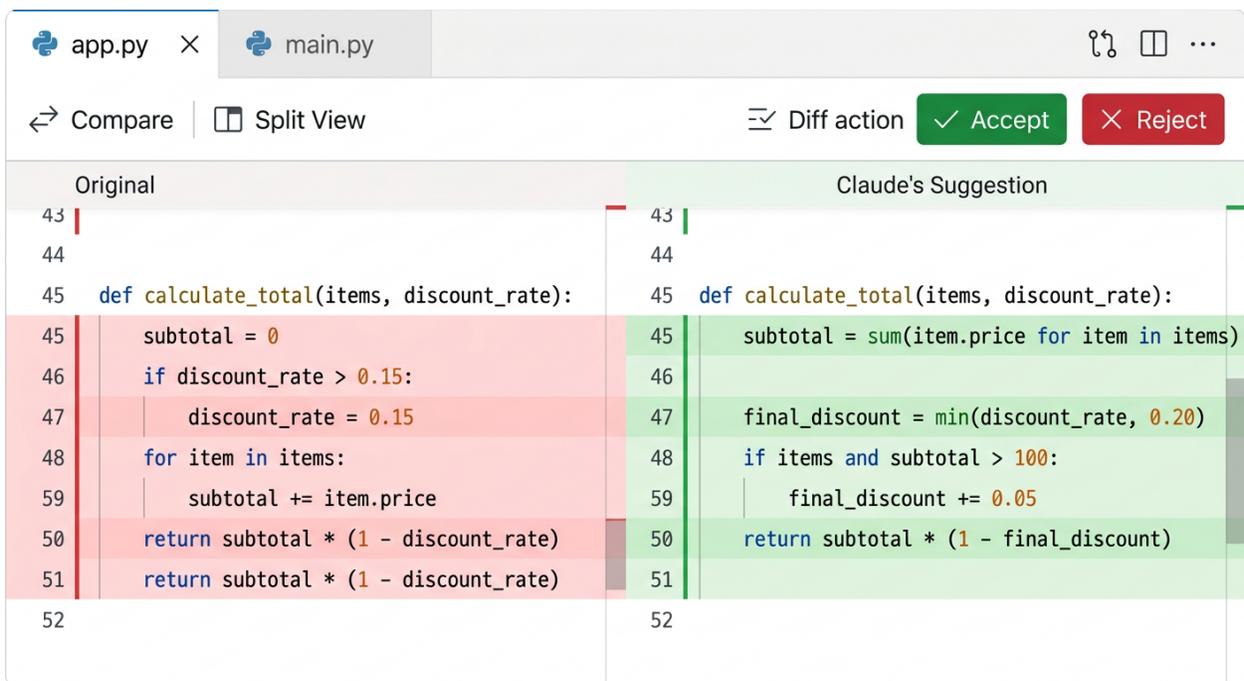


Figure 11: Claude가 제안한 코드 변경 사항의 diff 화면과 수락/거부 버튼이 있는 스크린샷

파일 및 폴더 참조 @- 을 사용하여 특정 파일이나 폴더에 대한 컨텍스트를 Claude에게 제공한다:

- @auth.js 이 파일의 로직을 설명해줘
- @src/components/ 이 폴더의 구조를 분석해줘

여러 대화 실행 명령 팔레트에서 **새 탭에서 열기** 또는 **새 창에서 열기**를 사용하여 여러 대화를 동시에 실행할 수 있다. 각 대화는 독립적인 기록과 컨텍스트를 유지한다. 예를 들어 한 탭에서는 프론트엔드 UI를 수정하고, 다른 탭에서는 백엔드 API를 개발하는 식으로 병렬 작업이 가능하다.

Git 통합 Claude Code는 Git과 통합되어 커밋, 브랜치 관리, PR 생성 등을 자연어로 요청할 수 있다. Git에 대한 자세한 내용은 2장에서 다룬다.

주요 단축키

명령	단축키 (Mac / Windows·Linux)	설명
포커스 전환	Cmd+Esc / Ctrl+Esc	편집기와 Claude 사이 전환
새 탭에서 열기	Cmd+Shift+Esc / Ctrl+Shift+Esc	새 대화를 탭으로 열기
새 대화	Cmd+N / Ctrl+N	새 대화 시작 (Claude 포커스 시)
@-멘션 삽입	Option+K / Alt+K	현재 파일 및 선택 영역 참조 삽입

1.4 uv와 micromamba

생명정보학 작업에는 Python 패키지 관리 도구가 필요하다. 이 책에서는 **uv**(Python 패키지 매니저)와 **micromamba**(Conda 호환 환경 매니저)를 사용한다. 이 도구들의 설치도 Claude Code에게 맡길 수 있다.

uv

uv는 Rust로 작성된 초고속 Python 패키지 매니저다. 기존의 pip보다 10~100배 빠르며, 가상 환경 생성과 패키지 설치를 한 번에 처리할 수 있다. 2024년 Astral사에서 공개한 이후 빠르게 Python 커뮤니티의 표준으로 자리잡고 있다.

Claude Code에게 설치를 요청한다:

uv를 설치해줘

Claude가 `curl -LsSf https://astral.sh/uv/install.sh | sh` 명령을 실행하고, 셸 설정까지 자동으로 처리해 준다.

uv 설치 후에는 **글로벌 CLAUDE.md**에 uv 사용 지침을 추가하는 것이 좋다. 글로벌 CLAUDE.md(`~/.claude/CLAUDE.md`)는 모든 프로젝트에서 Claude Code가 참조하는 설정 파일이다. 여기에 “패키지 설치에 uv를 사용할 것”이라고 명시해 두면, 어떤 프로젝트에서든 Claude Code가 pip 대신 uv를 사용한다.

글로벌 CLAUDE.md에 다음 규칙을 추가해줘: Python 패키지 설치 시 pip 대신 uv를 사용할 것, 가상 환경 생성 시 uv venv를 사용할 것

micromamba

micromamba는 Conda의 경량화 버전으로, Conda와 동일한 패키지 저장소(conda-forge, bioconda)를 사용하지만 훨씬 빠르고 가볍다. 원래 Conda는 Anaconda 배포판에 포함된 패키지 매니저인데, 설치 용량이 크고 속도가 느리다는 단점이 있었다. micromamba는 C++로 작성되어 이런 문제를 해결했다.

생명정보학 도구 중 상당수가 Conda/Bioconda 채널을 통해 배포되므로 micromamba가 필요하다. PyPI에 없는 도구 — 예를 들어 samtools(시퀀스 데이터 처리), STAR(RNA-seq 정렬), bedtools(게놈 구간 연산) 등 — 는 Bioconda 채널에서만 설치할 수 있다.

Claude Code에게 설치를 요청한다:

micromamba를 설치하고, conda 명령으로도 사용할 수 있게 alias 설정해줘

Claude가 micromamba 설치와 `alias conda="micromamba"` 설정을 자동으로 처리해 준다. 이 alias가 있으면 시가 `conda install` 명령을 생성해도 micromamba가 실행된다.

uv vs micromamba

	uv	micromamba (conda)
용도	Python 패키지 설치	Python + 비Python 도구 설치
속도	매우 빠름	빠름 (conda보다 훨씬 빠름)
사용 상황	pandas, matplotlib 등 순수 Python 패키지	samtools, STAR, snakemake 등 바이너리 도구
채널	PyPI	conda-forge, bioconda

팁: 일반적인 Python 패키지는 uv로 설치하고, 생명정보학 전용 도구(samtools, STAR, bedtools 등)는 micromamba로 설치하는 것이 좋다. 두 도구는 서로 충돌하지 않으므로 함께 사용해도 문제없다.

1.5 정리

- Windows 사용자는 WSL을 가장 먼저 설치
 - Microsoft Store에서 Ubuntu를 검색하여 설치 후 재부팅
- VS Code 설치 및 WSL 연동
 - 좌측 하단 >< 아이콘에서 WSL 연결 (확장 프로그램 자동 설치)
 - 이후 모든 작업은 WSL 환경에서 진행
- Claude Code 설치 및 기본 사용법 숙지
 - VS Code 확장 프로그램으로 설치
 - @-멘션, 권한 모드, diff 검토 등 주요 기능 익히기
 - 이 책 전체에서 Claude Code를 활용하여 생명정보학 도구를 개발
- uv와 micromamba는 Claude Code에게 설치를 맡기기
 - uv: 빠른 Python 패키지 설치 (PyPI)
 - micromamba: Conda 호환 환경 매니저 (bioconda 채널 활용)
 - `alias conda="micromamba"`로 편의성 확보

2장. Git & GitHub

2.1 버전 관리 시스템 (Version Control System)

버전 관리가 없다면?

버전 관리 시스템 없이 파일을 관리하면 어떻게 될까? 연구자라면 한 번쯤은 다음과 같은 경험이 있을 것이다.

졸업논문, 졸업논문수정1, 졸업논문수정2, 졸업논문완성본, 졸업논문완성본final, 졸업논문완성본final1, 졸업논문최종완성본-final, 졸업논문최종완성본final1... 파일 이름만으로는 어떤 것이 최신 버전인지, 어떤 변경이 있었는지 알 수 없다. 코드도 마찬가지다. 분석 스크립트를 수정하다 이전 버전으로 되돌리고 싶은데, 어느 시점의 코드가 정상적으로 동작했는지 기억나지 않는 경우가 흔하다.

이러한 문제를 해결하기 위해 버전 관리 시스템(VCS, Version Control System)이 등장했다. VCS는 파일의 모든 변경 이력을 기록하여, 언제든 특정 시점의 상태로 되돌아갈 수 있게 해준다. 누가, 언제, 무엇을 변경했는지 추적할 수 있어 협업에도 필수적이다.

2.2 Git이란?

Git은 분산형 버전 관리 시스템이다. 2005년 리누스 토르발즈(Linux 운영체제의 창시자)가 Linux 커널 개발을 위해 만들었으며, 현재 전 세계 소프트웨어 개발의 사실상 표준이 되었다.

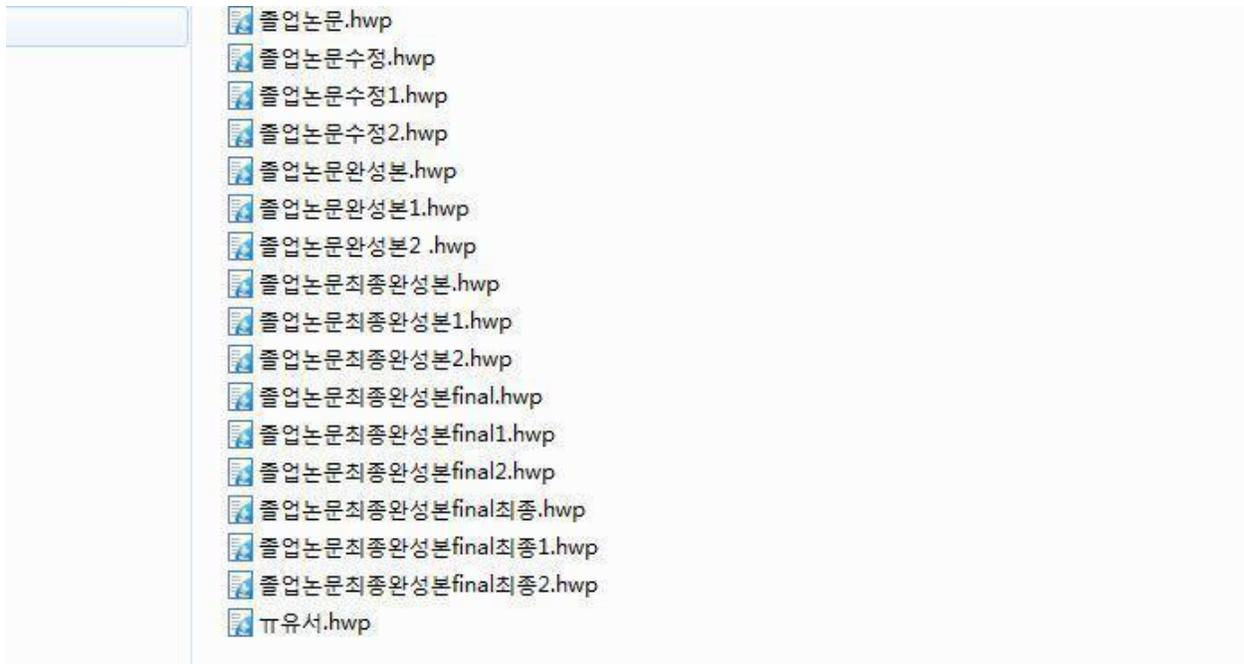


Figure 12: 버전 관리 없이 파일을 관리하는 모습

“분산형”이란 각 사용자가 버전 트리의 전체 사본을 로컬에 가지고 있다는 뜻이다. 따라서 오프라인에서도 커밋, 브랜치 생성, 이력 조회 등 모든 버전 관리 작업이 가능하다. 네트워크가 연결되면 변경 사항을 원격 저장소와 동기화하면 된다. 프로젝트 폴더 안의 `.git` 디렉토리에 버전 트리가 저장된다.

생명정보학에서 Git은 분석 스크립트, Snakemake 파이프라인, 설정 파일 등의 버전을 관리하는 데 널리 사용된다. 이 책에서는 Claude Code를 통해 Git을 사용하므로, 명령어를 외울 필요 없이 개념만 이해하면 된다.

Git 기본 용어

용어	설명
Repository (Repo, 리포지토리)	Git 버전 트리의 저장소. 프로젝트의 모든 파일과 변경 이력이 담겨 있다
Clone (클론)	원격의 버전 트리를 로컬에 처음으로 가져오는 작업
Commit (커밋)	특정 시점의 파일 상태를 기록한 스냅샷. “저장” 버튼과 비슷하지만, 이전 상태로 되돌아갈 수 있다는 점이 다르다
HEAD (헤드)	현재 작업 중인 가장 최신 커밋을 가리키는 포인터
Push (푸시)	로컬의 커밋들을 원격 저장소에 업로드
Pull (풀)	원격 저장소의 최신 변경 사항을 로컬로 다운로드하고 병합
Stash (스태시)	현재 작업 중인 변경사항을 임시로 저장해두고 HEAD 상태로 되돌림
<code>.gitignore</code>	Git이 추적하지 않을 파일을 정의하는 파일. 바이너리 파일, 비밀번호, 대용량 데이터 등을 제외할 때 사용

GitHub 기본 용어

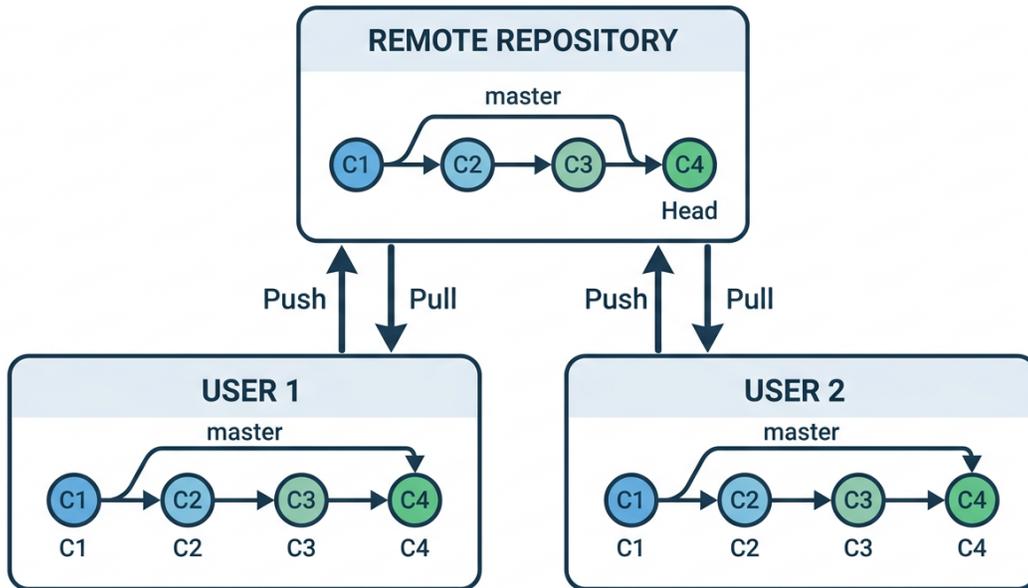


Figure 13: 분산형 VCS 다이어그램 — User1과 User2가 각각 로컬에 버전 트리를 가지고 있고, 원격 저장소와 동기화하는 구조



Figure 14: Git 로고 이미지

용어	설명
GitHub (깃합)	원격 Git 저장소를 호스팅하는 세계에서 가장 유명한 플랫폼
Fork (포크)	다른 사람의 Git 저장소를 내 계정으로 복제. 원본에 대한 쓰기 권한 없이도 자유롭게 수정 가능
Pull Request (PR)	Fork한 저장소의 변경 사항을 원본 저장소에 반영해달라고 요청하는 것
GitHub Actions	GitHub 저장소에서 자동화 작업(테스트, 배포 등)을 실행하는 CI/CD 도구
GitHub Copilot	GitHub에서 개발한 AI 기반 코드 어시스트 도구

2.3 Git 설치 및 설정

Git 설치

Git을 사용하려면 먼저 설치해야 한다. 운영체제별 설치 방법은 다음과 같다.

Linux / WSL (Ubuntu)

```
sudo apt update && sudo apt install git
```

중요: sudo는 시스템 전체에 영향을 미치는 관리자 권한으로 명령을 실행하는 것이므로, 비밀번호를 입력해야 한다. LLM(Claude Code 등)에게 sudo 비밀번호를 전달하는 것은 보안상 위험하므로, sudo가 필요한 설치 작업은 반드시 사용자가 직접 터미널에서 실행한다.

macOS

터미널에서 `git --version`을 입력하면, Git이 설치되어 있지 않을 경우 Xcode Command Line Tools 설치를 자동으로 안내한다. 안내에 따라 설치하면 된다.

설치 확인:

```
git --version
```

사용자 정보 설정

Git을 처음 사용할 때는 이름과 이메일을 설정해야 한다. 이 정보는 커밋에 기록되어, 누가 어떤 변경을 했는지 추적할 수 있게 해준다.

```
git config --global user.name " "
git config --global user.email "hong@example.com"
```

이메일은 GitHub 계정에 등록된 이메일과 동일하게 설정하는 것이 좋다. GitHub에서 커밋 기록이 본인의 프로필과 연결되기 때문이다.

2.4 GitHub CLI (gh) 설치 및 설정

GitHub CLI(gh)는 터미널에서 GitHub의 기능을 사용할 수 있게 해주는 도구이다. Claude Code가 저장소 생성, Pull Request, Issue 관리 등 GitHub 관련 작업을 수행할 때 내부적으로 gh를 사용하므로, 반드시 설치하고 인증을 완료해야 한다.

Linux / WSL (Ubuntu)

```
sudo apt update && sudo apt install gh
```

macOS

```
brew install gh
```

brew는 macOS의 패키지 관리자이다. 설치되어 있지 않다면 <https://brew.sh> 에서 설치할 수 있다.

설치 확인:

```
gh --version
```

GitHub 인증

gh를 설치한 후, GitHub 계정과 연결하는 인증 과정이 필요하다.

```
gh auth login
```

이 명령을 실행하면 대화형으로 인증 방식을 선택하게 된다. 다음과 같이 진행한다:

1. “Where do you use GitHub?” → GitHub.com 선택
2. “What is your preferred protocol for Git operations on this host?” → HTTPS 선택
3. “Authenticate Git with your GitHub credentials?” → Yes 선택
4. “How would you like to authenticate GitHub CLI?” → Login with a web browser 선택
5. 화면에 표시된 one-time code를 복사하고 Enter를 누른다
6. 자동으로 열리는 브라우저에서 코드를 붙여넣고 인증을 완료한다

인증이 완료되면 다음 명령으로 확인할 수 있다:

```
gh auth status
```

2.5 GitHub에 저장소 만들기

Claude Code에게 요청하면 저장소 생성부터 로컬 Clone까지 한 번에 처리할 수 있다:

```
my-analysis라는 이름의 공개 GitHub 저장소를 만들고 클론해줘
```

Claude Code가 내부적으로 `gh repo create` 명령을 실행하여 GitHub에 저장소를 생성하고, 로컬에 Clone까지 완료해 준다.

직접 터미널에서 만들 수도 있다:

```
gh repo create my-analysis --public --clone
```

이 명령은 GitHub에 my-analysis라는 공개 저장소를 만들고, 자동으로 로컬에 Clone까지 해준다. 비공개 저장소를 만들려면 `--public` 대신 `--private`을 사용한다.

GitHub 웹사이트에서도 저장소를 만들 수 있다:

1. <https://github.com> 에 로그인한다
2. 오른쪽 상단의 + 버튼 → “New repository”를 클릭한다
3. 저장소 이름을 입력한다 (예: my-analysis)
4. “Add a README file” 체크박스를 선택한다
5. “Add .gitignore”에서 프로젝트에 맞는 템플릿을 선택한다 (Python 프로젝트라면 Python 선택)
6. “Create repository” 버튼을 클릭한다

마크다운(Markdown)이란?

README 파일의 확장자 `.md`는 **마크다운(Markdown)**을 의미한다. 마크다운은 간단한 기호로 문서의 서식을 지정하는 경량 마크업 언어이다. GitHub에서 README.md 파일을 자동으로 렌더링하여 보기 좋게 표시해 주며, 이 책에서도 CLAUDE.md, 에이전트 정의 파일 등 다양한 곳에서 마크다운을 사용한다.

기본 문법은 다음과 같다:

```
# 1
## 2
### 3
```

```
** **, * *, ` `
```

- 1
- 2

1. 1
2. 2

[](<https://example.com>)

>

마크다운은 일반 텍스트로 작성하면서도 구조적인 문서를 만들 수 있어, 프로그래머들 사이에서 문서 작성의 사실상 표준이 되었다. 자세한 문법은 외울 필요 없이, 필요할 때 Claude Code에게 물어보면 된다.

웹에서 저장소를 만든 경우, 로컬에 Clone하여 작업을 시작한다:

```
gh repo clone /my-analysis
```

또는 Claude Code에게 요청할 수도 있다:

`https://github.com/사용자명/my-analysis` 이 저장소를 클론해줘

참고: 반드시 원격 저장소로 GitHub을 쓸 필요는 없다. Bitbucket, GitLab, Phorge 등 다른 서비스도 활용 가능하다. 다만 이 책에서는 가장 널리 사용되는 GitHub을 기준으로 설명한다.

2.6 Claude Code로 Git 사용하기

이 책에서는 Git 명령어를 직접 입력하는 대신, Claude Code에게 자연어로 요청하여 Git을 사용한다. Git의 개념을 이해하고 있으면, 세부 명령어는 Claude Code가 알아서 처리해 준다.

혼자 작업하기

가장 기본적인 워크플로우이다. 원격 저장소를 Clone한 뒤, 파일을 수정하고 Commit, 그리고 Push하는 과정이다.

1단계: Clone

원격 저장소를 로컬로 복제한다. Claude Code에게 저장소 주소를 알려주면 된다.

`https://github.com/사용자명/저장소명` 이 저장소를 클론해줘

2단계: Commit

파일을 수정한 뒤, Claude Code에게 커밋을 요청한다. Claude가 변경 사항을 분석하여 적절한 커밋 메시지를 자동으로 생성해 준다.

변경 사항을 커밋해줘

커밋 메시지를 직접 지정하고 싶다면 이렇게 요청할 수도 있다:

“분석 스크립트에 QC 단계 추가” 메시지로 커밋해줘

커밋은 Git에서 가장 기본적인 단위이다. Claude Code는 내부적으로 변경된 파일을 스테이징(`git add`)하고 커밋(`git commit`)하는 과정을 한 번에 처리해 준다. “수정” 같은 모호한 메시지보다 “BLAST 결과 파싱 함수에서 빈 결과 처리 추가”처럼 구체적인 메시지가 좋는데, Claude Code에게 맡기면 변경 내용을 분석하여 이런 구체적인 메시지를 자동으로 작성해 준다.

3단계: Push

커밋한 내용을 원격 저장소에 업로드한다.

푸시해줘

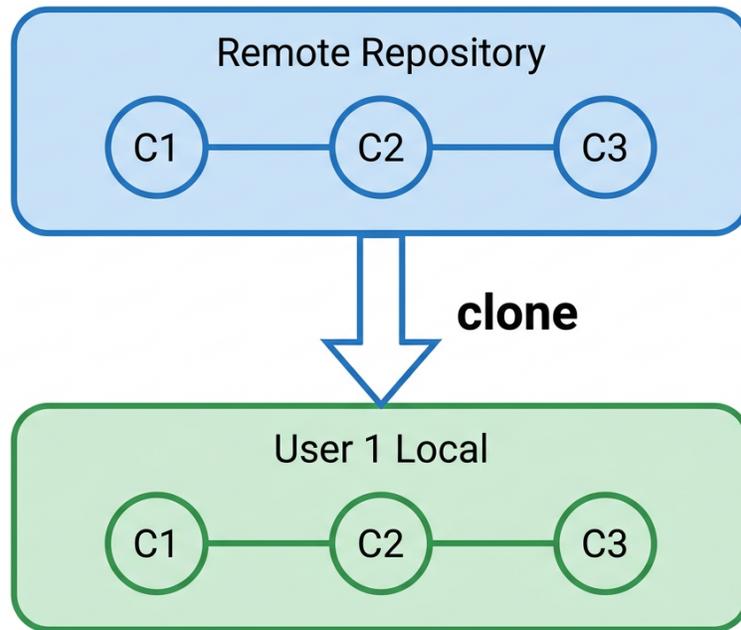


Figure 15: 시나리오 1 다이어그램 — User1이 원격 저장소에서 Clone하는 모습

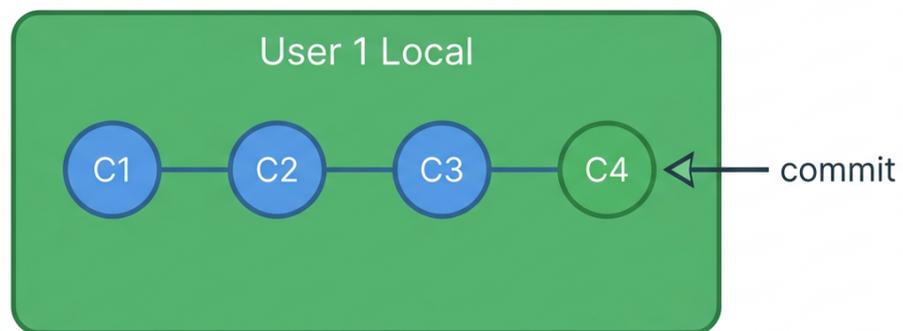


Figure 16: 시나리오 1 다이어그램 — User1이 로컬에서 Commit하는 모습

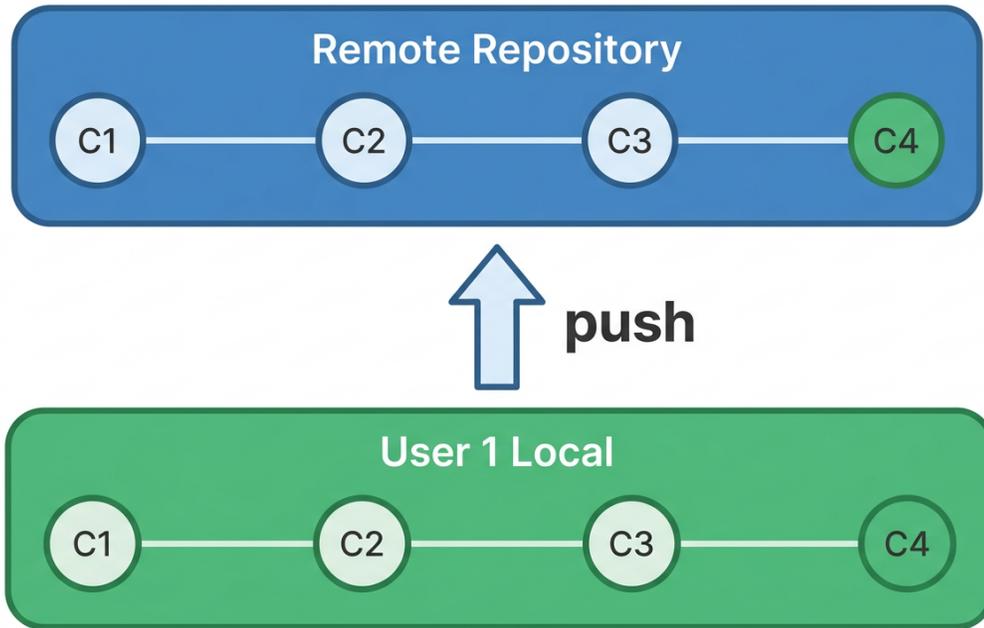


Figure 17: 시나리오 1 다이어그램 — User1이 원격 저장소로 Push하는 모습

함께 작업하기

두 명 이상이 같은 저장소에서 작업할 때 발생할 수 있는 상황이다. 이 시나리오를 이해하면 협업 과정에서 생기는 대부분의 문제를 해결할 수 있다.

1단계: User1과 User2가 각각 Clone

두 사람이 같은 원격 저장소를 각자의 컴퓨터에 Clone한다. 이 시점에서 두 사람의 로컬 저장소는 동일한 상태이다.

2단계: User1이 먼저 Commit & Push

User1이 파일을 수정하고 커밋한 뒤, 원격 저장소에 Push한다. 이제 원격 저장소는 User1의 변경 사항이 반영된 상태이다.

3단계: User2도 Commit 후 Push 시도 — 실패!

User2도 파일을 수정하고 커밋한 뒤 Push를 시도하지만, User1이 이미 Push한 변경 사항이 있으므로 **Push가 거부된다**. Git은 원격 저장소의 이력이 로컬보다 앞서 있으면 Push를 허용하지 않는다. 데이터 손실을 방지하기 위한 안전장치이다.

4단계: User2가 Pull

먼저 원격의 변경 사항을 Pull로 가져온다. Git이 자동으로 User1의 변경 사항과 User2의 변경 사항을 병합(merge)하려고 시도한다.

원격 저장소의 최신 변경 사항을 가져와줘

5단계: 충돌(Conflict) 해결

같은 파일의 같은 부분을 수정했다면 **merge conflict(병합 충돌)**이 발생한다. Git이 자동으로 병합할 수 없는 경우이다. 충돌이 발생하면 해당 파일에 다음과 같은 표시가 나타난다.

```
<<<<<<< HEAD
User2
=====
User1
```

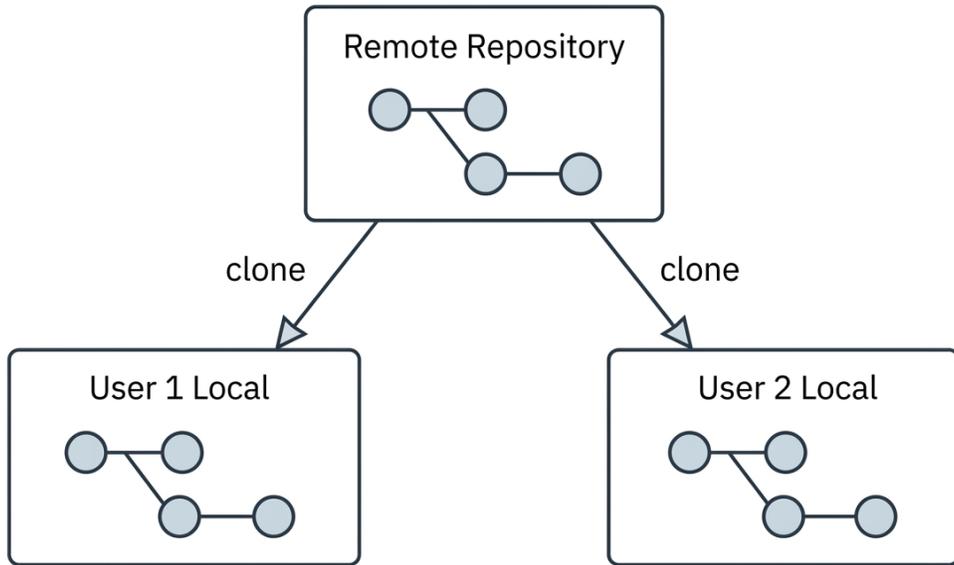


Figure 18: 시나리오 2 다이어그램 — User1과 User2가 동일한 원격 저장소를 Clone하는 모습

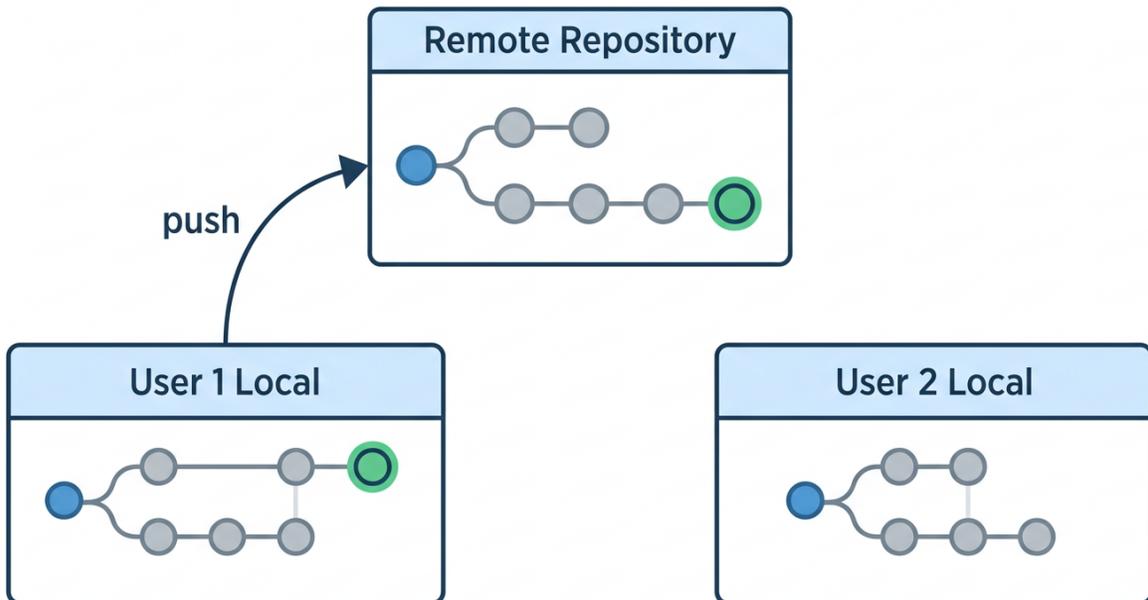


Figure 19: 시나리오 2 다이어그램 — User1이 Commit 후 Push하는 모습

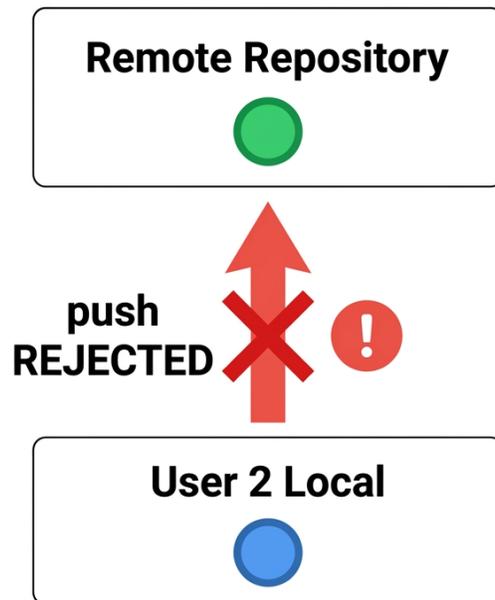


Figure 20: 시나리오 2 다이어그램 — User2의 Push가 거부되는 모습

```
~/projects/git-demo $ git push origin main
[remote: ] Verifying... Done.
[remote: ] Enumerating objects: 7, done.
[remote: ] Counting objects: 100% (7/7), done.
[remote: ] Delta compression using up to 12 threads.
[remote: ] Compressing objects: 100% (3/3), done.
[remote: ] Writing objects: 100% (3/3), 481 bytes | 481.00 KiB/s, done.
[remote: ] Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
[error] error: failed to push some refs to 'github.com:user/git-demo.git'
[hint] hint: Updates were rejected because the remote contains work that you do
[hint] hint: not have locally. This is usually caused by another repository pushing
[hint] hint: to the same ref. You may want to first integrate the remote changes
[hint] hint: (e.g., 'git pull ...') before pushing again.
[hint] hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Figure 21: Git Push 거부 시 터미널 에러 메시지 스크린샷

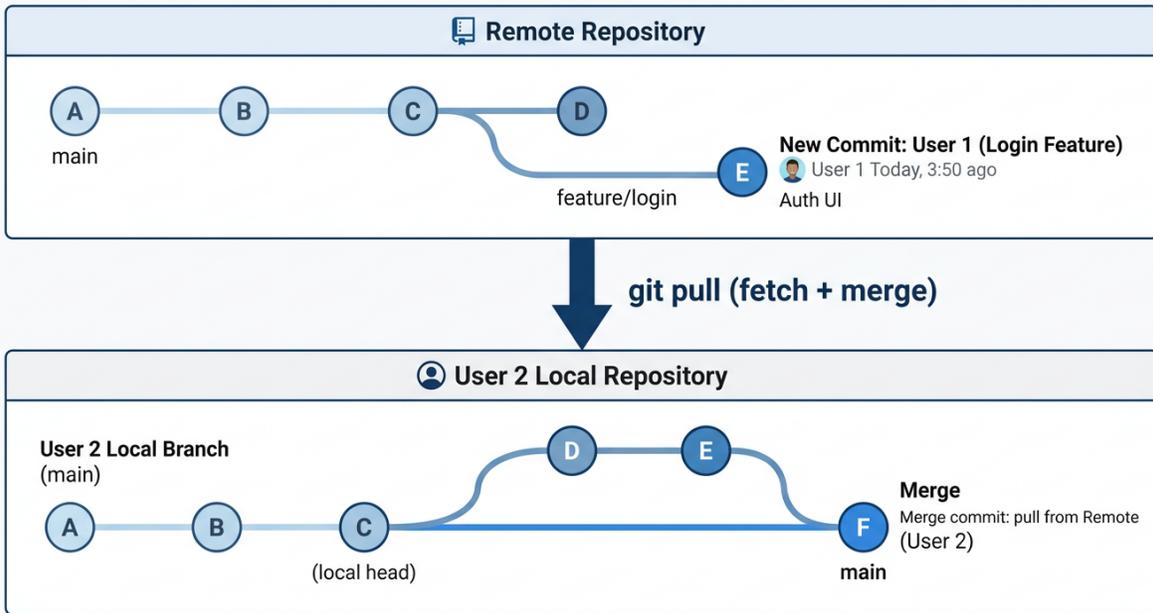


Figure 22: 시나리오 2 다이어그램 — User2가 Pull하는 모습

```
>>>>>> origin/main
```

<<<<<< HEAD와 ===== 사이는 내(User2)의 변경 사항이고, =====과 >>>>>> 사이는 원격(User1)의 변경 사항이다.

충돌이 발생하면 Claude Code에게 해결을 요청할 수 있다:

merge conflict를 해결해줘. 양쪽 변경 사항을 모두 반영해줘

Claude Code가 충돌 마커를 분석하고, 양쪽 변경 사항을 적절히 병합한 뒤, 결과를 커밋해 준다. 물론 VS Code에서 직접 해결할 수도 있다. 충돌이 발생한 파일을 열면 “현재 변경 사항 수락”, “상대방 변경 사항 수락”, “양쪽 모두 수락” 같은 버튼이 표시되어 편리하게 해결할 수 있다.

참고: GitHub 웹 인터페이스에서도 merge conflict를 해결할 수 있다. <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-on-github>

6단계: Merge 후 Push

충돌을 해결하고 merge 커밋을 생성한 뒤, 최종적으로 Push한다.

푸시해줘

Stash 활용하기

파일을 수정했으나 아직 Commit하지 않은 상태에서, Pull을 깜빡 잊었을 때 사용하는 방법이다. 작업을 시작하기 전에 Pull을 실행하는 습관을 들이면 이 상황을 피할 수 있지만, 현실에서는 종종 발생한다.

Git stash: 로컬의 변경사항을 임시로 저장하고 HEAD로 되돌린다. 마치 작업 중인 서류를 서랍에 넣어두는 것과 비슷하다. Claude Code에게 이렇게 요청하면 된다:

현재 변경 사항을 임시 저장하고, 원격에서 최신 코드를 가져온 다음, 임시 저장한 내용을 다시 적용해줘

```
JS app.js

import React, { useState } from 'react';

function Header() {
  const [user, setUser] = useState(null);
  <<<<<< HEAD
  return (
    <header>
      <h1>Welcome to Our App!</h1>
      <button>Log In</button>
    </header>
  );
  =====
  return (
    <header className="main-header">
      <h1>Welcome back, {user ? user.name : 'Guest'}!</h1>
      {user ? <button>Log Out</button> : <button>Log In</button>}
    </header>
  );
  >>>>>> feature/user-profile
}

export default Header;
```

Figure 23: Merge conflict 표시가 된 파일 내용 스크린샷 — «««« HEAD, =====, »»»» 마커가 보이는 모습

```
src/utills/calculator.js

1 import app from 'too';
2 import from 'main';
3
4 export default from calculators;
5
6
7 <<<<<< HEAD
8 const add = (a, b) => a + b;
9
10 =====
11 >>>>>> main
12 const add = (a, b) => {
13   return a + b
14 };
15
16 export default add;
17
18
19
20
```

Figure 24: GitHub 웹에서 merge conflict 해결하는 화면 스크린샷

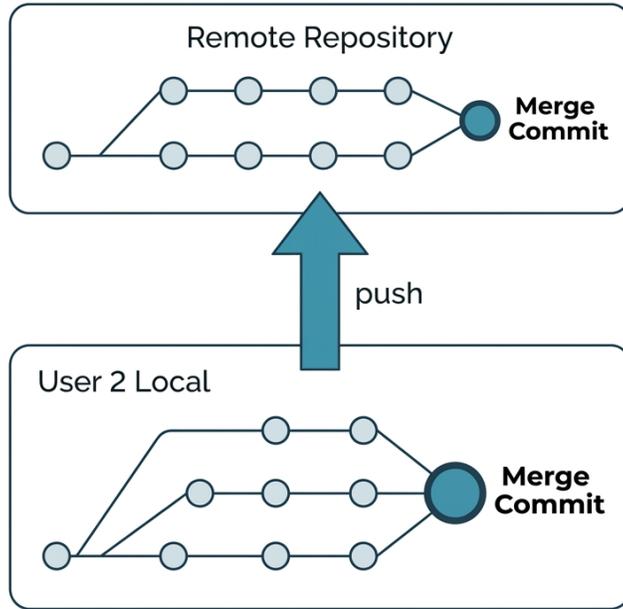


Figure 25: 시나리오 2 다이어그램 — User2가 Merge 후 Push하는 모습, 버전 트리에 merge branch가 표시됨

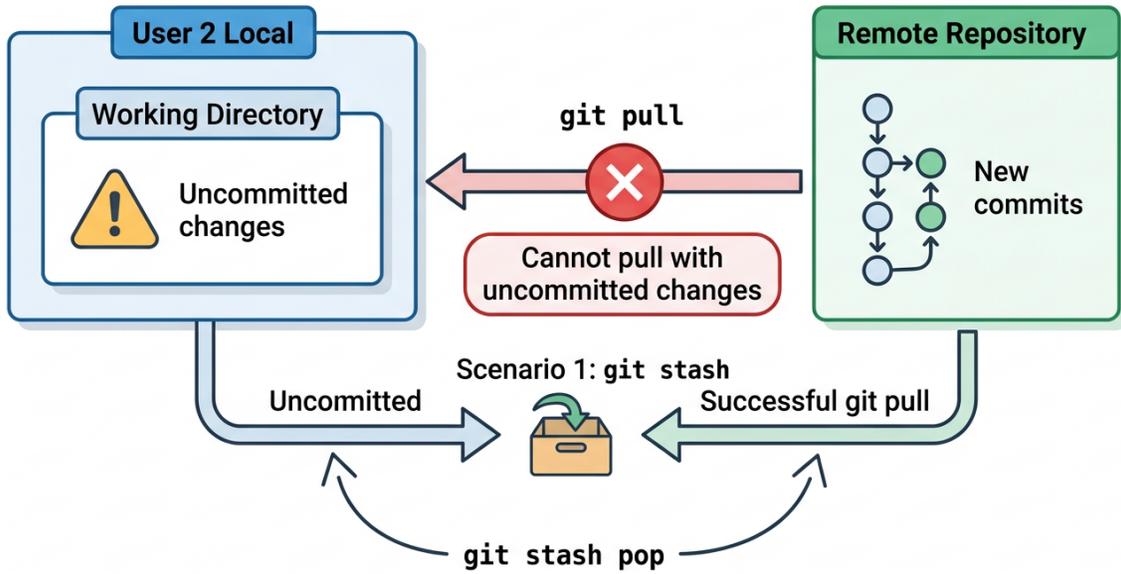


Figure 26: 시나리오 3 다이어그램 — User2가 파일을 수정했으나 Commit 전에 Pull이 필요한 상황

Claude Code가 내부적으로 `git stash` → `git pull` → `git stash pop` 순서로 처리해 준다. 만약 conflict가 발생하면 앞의 시나리오와 동일한 방식으로 해결을 요청할 수 있다.

```
$ git stash
Saved working directory and index state WIP on
main: abc1234 latest commit
$ git pull
Updating abc1234..def5678
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Figure 27: git stash 실행 전후 터미널 스크린샷

2.7 GitHub에서 협업하기

다른 개발자의 프로젝트에 기여하기

오픈소스 생태계에서는 다른 사람의 프로젝트에 기여(contribution)하는 문화가 활발하다. 예를 들어 Scanpy에서 버그를 발견했다면, 직접 수정하여 프로젝트에 반영을 요청할 수 있다. 이 과정은 다음과 같다.

1. 다른 개발자의 저장소를 **Fork**한다
2. Fork한 저장소를 로컬에 **Clone**한다
3. 자유롭게 수정한 후 GitHub에 **Push**한다 (이때 Push되는 저장소는 내가 Fork한 저장소)
4. GitHub 상에서 **Pull Request**를 보낸다
5. 상대방이 변경 사항을 **Merge**해 주면 기여 완료

Fork

Fork는 다른 개발자의 프로젝트(쓰기 권한 없음)를 내 계정의 프로젝트(쓰기 권한 있음)로 복제하는 것이다. GitHub 웹사이트에서 저장소 페이지 오른쪽 상단의 “Fork” 버튼을 클릭하면 된다.

Fork된 저장소는 원본과 독립적이므로, 어떤 실험적인 수정을 하더라도 원본에 영향을 주지 않는다. 이 점이 Clone과의 차이이다 — Clone은 원본 저장소를 그대로 가져오는 것이고, Fork는 내 계정에 별도의 복사본을 만드는 것이다.

Pull Request

Fork한 저장소에서 변경 사항을 Push한 뒤, 원본 저장소에 Pull Request(PR)를 열어 변경 사항의 반영을 요청한다. Claude Code를 사용하면 PR 생성도 간편하다:

이 변경 사항으로 PR을 만들어줘

```
>_
$ git stash pop
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   docs/api_reference.md
        modified:   frontend/src/app.js
        modified:   frontend/src/components/button.jsx
        modified:   frontend/src/styles.css
        modified:   server/index.js
  (use "git add/rm <file>..." as appropriate to mark resolution)
Dropped refs/stash@{0} (7397b919...)
```

Figure 28: git stash pop 실행 후 터미널 스크린샷

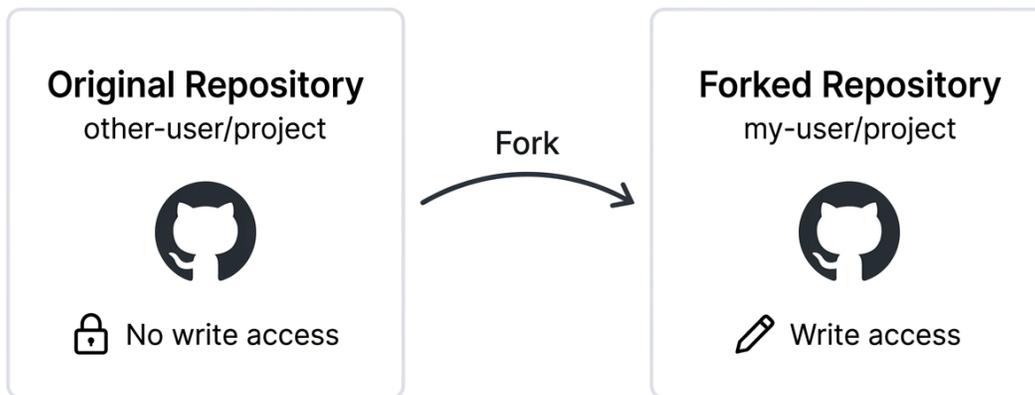


Figure 29: GitHub Fork 개념 다이어그램 — 원본 저장소에서 내 계정으로 Fork되는 구조

Claude Code가 변경 내용을 분석하여 PR 제목과 설명을 자동으로 작성해 준다. 어떤 문제를 해결했는지, 어떤 방식으로 수정했는지를 포함한 설명이 생성된다.

PR이 열리면 원본 저장소의 관리자가 코드를 검토(code review)한다. 수정 요청이 올 수도 있고, 바로 승인될 수도 있다. 이 과정을 통해 코드의 품질을 유지하면서 외부 기여를 받아들일 수 있다.

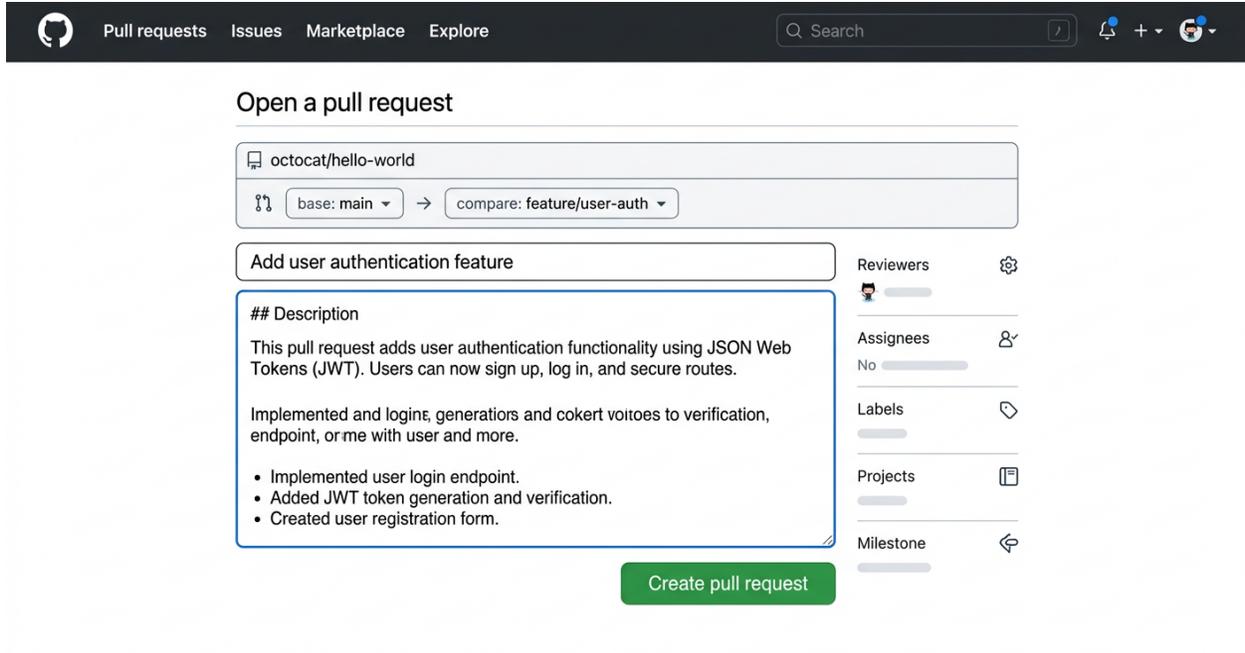


Figure 30: GitHub에서 Pull Request 생성 화면 스크린샷

2.8 바이브 코딩에서의 Git

바이브 코딩에서 Git은 특히 중요한 역할을 한다. 시가 코드를 수정하는 과정에서 예상치 못한 결과가 나올 수 있기 때문이다. Git을 통해 변경 이력을 추적하고, 문제가 생기면 이전 상태로 되돌릴 수 있다.

Claude Code에게 자주 사용하는 Git 관련 요청:

- 현재 변경 사항을 커밋해줘
- 마지막 커밋을 되돌려줘
- feature/blast-search 브랜치를 만들어줘
- main 브랜치로 돌아가줘
- 이 변경 사항으로 PR을 만들어줘
- 최근 커밋 이력을 보여줘

Git의 개념만 이해하고 있으면, 세부 명령어는 Claude Code에게 맡길 수 있다. 중요한 것은 커밋, 브랜치, 머지, PR이 무엇인지 개념적으로 이해하는 것이다. 특히 시가 코드를 수정하기 전에 커밋해두는 습관을 들이면, 언제나 안전하게 이전 상태로 돌아갈 수 있다.

2.9 정리

- 버전 관리 시스템(VCS)은 파일의 변경 이력을 추적하여 협업과 되돌리기를 가능하게 한다
- Git은 분산형 VCS로, 오프라인에서도 모든 버전 관리 작업이 가능하다

Add user authentication feature Open

Conversation Commits (3) Files changed (5)

Implements user sign-up, login, and password reset functionality using JWT.
Resolves issue #42. Added tests for authentication logic.

```
3 app/enaninvectors @@ -2,3, +103 @@
11 11  const tests = require from 'root';
12 12
13 + app.use(authRoutes);
14 + app.use(authRoutes);
14 15
15 - console.log('User logged in');
16 - console.log('User logged in');
17 + console.log('User logged reset');
18 18  });
19 19
20 20  export default, app => {
```

Review requested

- @reviewer1
- @reviewer2
- @reviewer3
- @reviewer4

Figure 31: Pull Request 상세 내용 작성 화면 스크린샷

Pull requests Issues Marketplace Explore

Add user authentication feature (#42) Merged

Conversation Delete branch

3 commits Review approval (Review approved by user1) Pull request successfully merged and closed

Reviewers: user1, user1

Assignees: No assign assignees

Labels: No suggested to chlokes, and labels

Projects: No projects

Milestones: No milestones with rennomed.

Conversations Comments 1

user1 commented 1 ago: Looks good!

user1 added 1 1 ago: Added some tests.

user1 commented 1 ago: Merging now.

Figure 32: Pull Request가 생성된 후의 화면 스크린샷

- 기본 워크플로우: Clone → 수정 → Commit → Push
 - Claude Code에게 자연어로 요청하여 모든 Git 작업을 수행할 수 있다
 - Commit은 자주 하기 (특히 시가 코드를 수정하기 전에 반드시 커밋)
- 협업 시 충돌 해결: Pull → Conflict 해결 → Merge → Push
- GitHub 협업: Fork → Clone → 수정 → Push → Pull Request
- 바이브 코딩에서의 Git: 시가 코드를 수정하므로, 변경 이력 추적과 되돌리기가 더욱 중요하다

3장. Docker

3.1 Docker란?

Docker는 애플리케이션을 컨테이너(Container)라는 격리된 환경에서 실행할 수 있게 해주는 도구다. 컨테이너는 운영체제, 라이브러리, 설정 파일 등을 모두 포함하고 있어, 어떤 컴퓨터에서든 동일한 환경으로 프로그램을 실행할 수 있다.

왜 Docker가 필요한가?

생명정보학 도구를 개발하다 보면 다음과 같은 문제를 자주 겪게 된다:

- “내 컴퓨터에서는 되는데 다른 컴퓨터에서는 안 돼요”
- 파이썬 버전, 라이브러리 버전 충돌
- 운영체제마다 설치 방법이 다름
- 데이터베이스, 웹 서버 등 여러 서비스를 동시에 관리해야 함

이런 문제는 생명정보학에서 특히 심각하다. BLAST+는 특정 버전의 libgomp를 요구하고, Scanpy는 특정 버전의 NumPy와 호환되고, R의 DESeq2는 Bioconductor 버전에 의존한다. 연구실의 서버에서 잘 돌아가던 파이프라인이 동료의 노트북에서 에러를 내는 일은 흔하다.

Docker를 사용하면 이러한 문제를 해결할 수 있다. 개발 환경 자체를 코드(Dockerfile)로 정의하여 누구나 동일한 환경을 재현할 수 있다. “내 컴퓨터에서는 되는데”라는 말이 필요 없어진다.

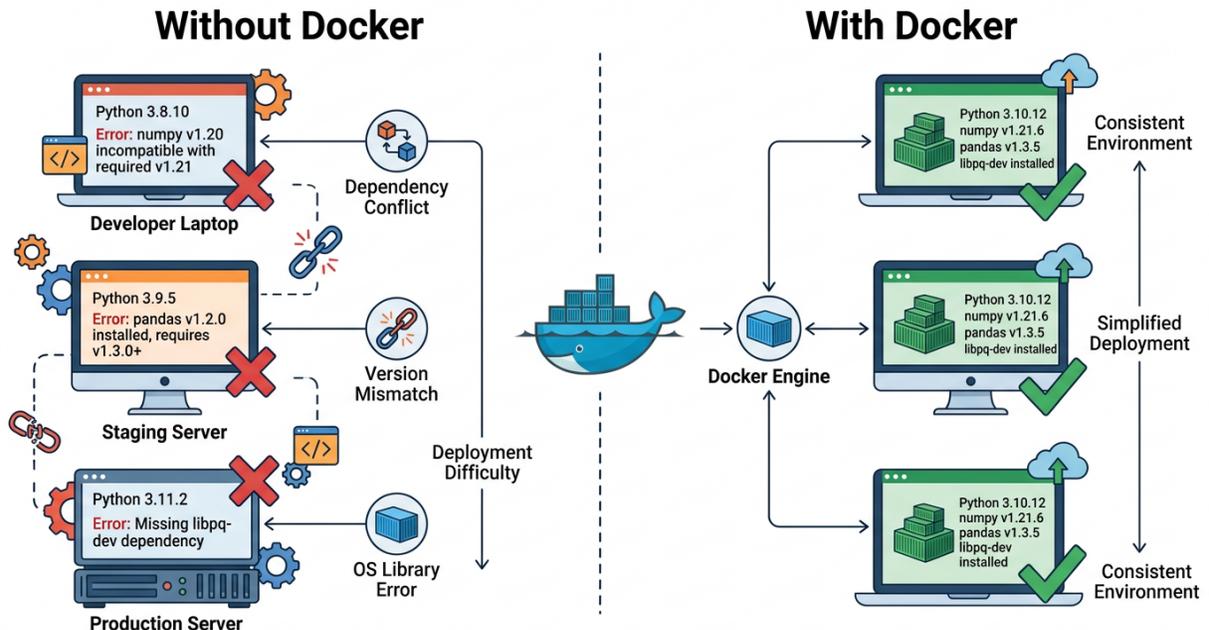


Figure 33: Docker 없이 개발할 때의 환경 차이 문제 vs Docker를 사용할 때의 일관된 환경을 비교하는 다이어그램

가상 머신과의 차이

Docker 컨테이너는 가상 머신(VM)과 비슷해 보이지만, 중요한 차이가 있다. 가상 머신은 운영체제 전체를 포함하므로 무겁고 느린 반면, 컨테이너는 호스트 운영체제의 커널을 공유하므로 가볍고 빠르다.

구체적으로 비교하면:

	가상 머신 (VM)	Docker 컨테이너
크기	수 GB (OS 전체 포함)	수십~수백 MB
시작 시간	수 분	수 초
자원 사용	OS별 메모리/CPU 점유	호스트 커널 공유, 가벼움
격리 수준	완전한 격리 (별도 OS)	프로세스 수준 격리
사용 사례	완전히 다른 OS가 필요할 때	애플리케이션 배포, 개발 환경

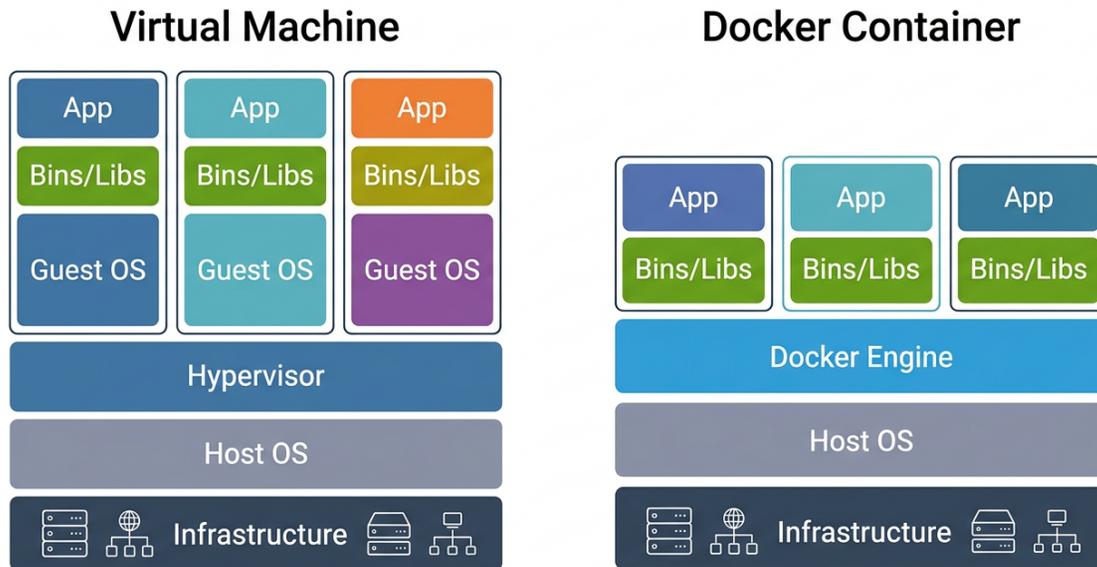


Figure 34: 가상 머신 vs Docker 컨테이너 아키텍처 비교 다이어그램 — VM은 Guest OS 포함, Container는 커널 공유

3.2 Docker 설치

이 책에서는 모든 개발을 WSL(Windows) 또는 네이티브 리눅스/macOS 환경에서 진행한다. Docker도 WSL 내에서 직접 설치한다. Windows 사용자는 1장에서 설치한 WSL Ubuntu 터미널을 열고 진행한다.

참고: Windows에서는 Docker Desktop이라는 GUI 프로그램을 설치하는 방법도 있지만, 이 책에서는 WSL 안에서 Docker Engine을 직접 설치하는 방법을 사용한다. Docker Desktop은 유료 라이선스 정책이 있고, WSL 직접 설치가 더 가볍고 안정적이다.

중요: Docker 설치 **사용자가 직접 터미널에서 수행해야 한다.** 설치 과정에 `sudo`(관리자 권한) 명령이 포함되어 있기 때문이다. `sudo`는 시스템 전체에 영향을 미치는 관리자 권한으로 명령을 실행하는 것이므로, 비밀번호를 입력해야 한다. LLM(Claude Code 등)에게 `sudo` 비밀번호를 전달하는 것은 보안상 위험하므로, `sudo`가 필요한 설치 작업은 반드시 사용자가 직접 수행한다.

터미널에서 다음 명령을 순서대로 실행한다:

```
# Docker GPG
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Docker
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Docker
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# docker (sudo docker )
sudo usermod -aG docker $USER
```

마지막 명령(usermod)은 매번 `sudo docker ...`를 입력하지 않아도 되게 해준다. 이 설정이 적용되려면 WSL 터미널을 완전히 종료했다가 다시 열어야 한다.

설치 확인

터미널에서 다음 명령을 실행하여 Docker가 정상적으로 설치되었는지 확인한다:

```
docker --version
docker run hello-world
```

`docker run hello-world`를 실행하면 Docker가 `hello-world` 이미지를 자동으로 다운로드하고 실행한다. “Hello from Docker!”라는 메시지가 출력되면 설치가 성공한 것이다. 이 과정에서 Docker의 핵심 동작 방식 — 이미지 다운로드 → 컨테이너 생성 → 실행 → 종료 — 을 한 번에 체험할 수 있다.

3.3 Docker 기본 개념

Docker에는 세 가지 핵심 개념이 있다: **이미지**, **컨테이너**, **Dockerfile**. 이 관계를 이해하면 Docker의 대부분을 이해한 것이나 마찬가지다.

이미지 (Image)

Docker 이미지는 컨테이너를 만들기 위한 **설계도**다. 운영체제, 프로그램, 설정 파일 등이 모두 포함되어 있다. Docker Hub(<https://hub.docker.com/>)에서 다양한 공식 이미지를 다운로드할 수 있다.

이미지는 **읽기 전용(read-only)**이다. 한 번 만들어진 이미지는 변경되지 않으며, 이를 기반으로 여러 개의 컨테이너를 만들 수 있다. 이 책에서 사용하는 대표적인 이미지로는 `node:20-alpine`(Node.js 런타임), `postgres:16-alpine`(PostgreSQL 데이터베이스) 등이 있다.

`alpine`이라는 태그가 붙은 이미지는 Alpine Linux라는 경량 리눅스 배포판을 기반으로 한다. 일반 Ubuntu 기반 이미지가 수백 MB인 데 반해, Alpine 기반 이미지는 수십 MB로 훨씬 가볍다.

컨테이너 (Container)

컨테이너는 이미지를 기반으로 **실제로 실행되는 인스턴스**다. 하나의 이미지로 여러 개의 컨테이너를 만들 수 있다. 이미지와 컨테이너의 관계는 프로그램의 설치 파일과 실행 중인 프로세스의 관계와 비슷하다.

```
Terminal: docker
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (and created a container from that image).
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.
```

Figure 35: docker run hello-world 실행 시 “Hello from Docker!” 메시지가 출력되는 터미널 스크린샷

컨테이너는 서로 격리되어 있다. 컨테이너 A에서 어떤 패키지를 설치하거나 파일을 수정해도, 컨테이너 B에는 영향을 주지 않는다. 컨테이너를 삭제하면 내부의 모든 변경 사항도 함께 사라진다. 데이터를 영구적으로 보존하려면 **볼륨(volume)**을 사용해야 한다.

Dockerfile

Dockerfile은 Docker 이미지를 만들기 위한 **레시피 파일**이다. 어떤 기반 이미지를 사용하고, 어떤 파일을 복사하고, 어떤 명령을 실행할지 순서대로 기술한다.

```
FROM node:20-alpine      #      : Node.js 20    Alpine Linux
WORKDIR /app             #
COPY package.json .     # package.json
RUN npm install          #
COPY . .                 #
CMD ["npm", "run", "dev"] #
```

Dockerfile의 각 줄은 **레이어(layer)**를 형성한다. Docker는 이 레이어를 캐시하므로, package.json이 변경되지 않았다면 RUN npm install 단계를 건너뛴다. 그래서 변경 빈도가 낮은 파일(의존성 정의)을 먼저 복사하고, 빈도가 높은 파일(소스 코드)을 나중에 복사하는 것이 효율적이다.

Docker Compose

실제 웹 애플리케이션은 보통 여러 서비스로 구성된다. 예를 들어 이 책에서 만드는 생명정보학 웹 도구는 SvelteKit 앱(프론트엔드 + 백엔드)과 PostgreSQL(데이터베이스)이 함께 동작해야 한다. 이럴 때 **Docker Compose**를 사용한다.

Docker Compose는 여러 개의 컨테이너를 compose.yml 파일 하나로 정의하고 한 번에 실행할 수 있게 해주는 도구다.

```
services:
  web:
    build: .              #      Dockerfile
    ports:
```

```

- "3000:3000"      #   3000      3000
db:
  image: postgres:16      # Docker Hub      PostgreSQL
  environment:
    POSTGRES_PASSWORD: mysecret

```

ports 설정은 호스트(내 컴퓨터)와 컨테이너 사이의 포트를 연결한다. "3000:3000"은 내 컴퓨터의 3000번 포트로 접속하면 컨테이너의 3000번 포트로 전달된다는 의미다. 브라우저에서 http://localhost:3000으로 접속하면 컨테이너 안에서 실행 중인 웹 서버에 연결된다.

실행은 다음 명령 하나로 가능하다:

```
docker compose up
```

이 명령을 실행하면 compose.yml에 정의된 모든 서비스의 이미지를 빌드(또는 다운로드)하고, 컨테이너를 생성하고, 네트워크로 연결한 뒤 실행한다. 하나의 명령으로 전체 개발 환경이 구성되는 것이다.

```

$ docker compose up
[+] Running 2/2
  ✔ Container my-app-web-1   Started           [34m0.3s[0m
  ✔ Container my-app-db-1   Started           [34m0.2s[0m

Attaching to my-app-web-1, my-app-db-1
[33mmy-app-db-1 | [0m PostgreSQL Database directory appears to contain a database; Skipping initialization
[33mmy-app-db-1 | [0m 2024-05-15 10:00:01.123 UTC [1] LOG: starting PostgreSQL 13.4 on x86_64-pc-linux-gn
nu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
[33mmy-app-db-1 | [0m 2024-05-15 10:00:01.124 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
[33mmy-app-db-1 | [0m 2024-05-15 10:00:01.124 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/
.PGSQL.s.PGSQL.5432"
[36mmy-app-web-1 | [0m => Booting Puma
[36mmy-app-web-1 | [0m => Rails 7.0.8 application starting in development
[36mmy-app-web-1 | [0m => Run `bin/rails server --help` for more startup options
[36mmy-app-web-1 | [0m Puma starting in single mode...
[36mmy-app-web-1 | [0m * Puma version: 5.6.7 (ruby 3.2.2-p53) "Birdie's Version"
[36mmy-app-web-1 | [0m * Min threads: 5, max threads: 5
[36mmy-app-web-1 | [0m * Environment: development
[36mmy-app-web-1 | [0m * PID: 1
[36mmy-app-web-1 | [0m * Listening on http://0.0.0.0:3000
[36mmy-app-web-1 | [0m Use Ctrl-C to stop

```

Figure 36: docker compose up 실행 시 웹 서버와 데이터베이스가 동시에 시작되는 터미널 출력 스크린샷

3.4 자주 사용하는 Docker 명령어

명령어	설명
docker compose up	compose.yml에 정의된 모든 서비스 시작
docker compose up -d	백그라운드에서 서비스 시작 (터미널을 다른 용도로 사용 가능)
docker compose down	모든 서비스 종료 및 컨테이너 삭제
docker compose logs	서비스 로그 확인 (에러 디버깅에 유용)
docker compose logs -f	실시간 로그 스트리밍 (Ctrl+C로 중단)
docker ps	실행 중인 컨테이너 목록 확인
docker exec -it < > bash	실행 중인 컨테이너 안으로 접속 (셸 사용)

`docker compose up -d`는 특히 유용하다. `-d` 플래그(detached mode)를 붙이면 서비스가 백그라운드에서 실행되므로, 같은 터미널에서 다른 작업을 계속할 수 있다. Claude Code를 사용할 때는 터미널을 시가 사용하므로, 백그라운드 실행이 편리하다.

`docker exec`는 실행 중인 컨테이너에 직접 접속하는 명령이다. 예를 들어 PostgreSQL 컨테이너에 접속하여 SQL 쿼리를 직접 실행하거나, 앱 컨테이너에서 BLAST+ 명령을 테스트할 때 사용한다.

3.5 Docker와 바이브 코딩

바이브 코딩에서 Docker의 역할은 더욱 중요하다. Claude Code에게 “BLAST+ 검색 도구를 만들어줘”라고 요청하면, Claude는 Dockerfile에 BLAST+ 설치 명령을 추가하고, `compose.yml`에 볼륨 마운트를 설정하고, 환경 변수를 구성해 준다.

이때 사람이 알아야 할 것은 Docker 명령어의 문법이 아니라, 왜 Docker가 필요한지, Dockerfile과 `compose.yml`이 각각 무슨 역할을 하는지이다. 예를 들어:

- “BLAST+를 Docker 컨테이너에 설치해줘” → Claude가 Dockerfile 수정
- “데이터베이스를 영구적으로 보존하게 해줘” → Claude가 볼륨 설정 추가
- “웹 서버 포트를 5173으로 바꿔줘” → Claude가 ports 설정 수정

이런 요청을 할 수 있으려면, 이미지, 컨테이너, 볼륨, 포트의 개념을 이해하고 있어야 한다.

3.6 정리

- Docker는 개발 환경을 코드로 정의하여 일관된 환경을 재현하는 도구
 - “내 컴퓨터에서는 되는데” 문제를 해결
 - 가상 머신보다 가볍고 빠르다
- 세 가지 핵심 개념: 이미지(설계도), 컨테이너(실행 인스턴스), Dockerfile(레시피)
- Docker Compose로 여러 서비스를 한 번에 관리
 - 웹 서버 + 데이터베이스 등을 `compose.yml` 하나로 정의
 - `docker compose up` 한 줄로 전체 환경 실행
- Windows 사용자는 WSL 위에 Docker를 설치하여 사용
- 바이브 코딩에서의 Docker: 코드를 직접 작성할 필요는 없지만, 이미지, 컨테이너, 볼륨, 포트의 개념은 이해해야 시에게 정확한 지시를 내릴 수 있다

4장. Python 데이터 분석 기초

4.1 왜 Python인가?

생명정보학에서 Python은 사실상 표준 프로그래밍 언어다. 데이터 처리, 시각화, 통계 분석, 머신러닝까지 거의 모든 작업을 Python 생태계 안에서 수행할 수 있다. Scanpy(단일세포 분석), Biopython(서열 처리), PyMOL(분자 시각화) 등 생명정보학 전용 라이브러리도 대부분 Python으로 작성되어 있다.

바이브 코딩에서는 이 패키지들의 문법을 암기할 필요가 없다. 대신 각 패키지가 무엇을 할 수 있는지를 알고, 시에게 정확히 요청하는 것이 중요하다. 이 장에서는 데이터 분석에 사용되는 핵심 패키지들의 역할과 주요 개념을 소개한다.

비유하자면, 요리를 주문할 때 레시피를 몰라도 되지만, 메뉴판을 읽을 수 있어야 하는 것과 같다. “볶음밥”과 “리조또”의 차이를 모르면 원하는 음식을 주문할 수 없듯이, “박스 플롯”과 “히트맵”의 차이를 모르면 시에게 원하는 시각화를 요청할 수 없다.

4.2 프로젝트 환경 구성

먼저 터미널에서 프로젝트 디렉토리를 만든다:

```
mkdir python-data-analysis
```

VS Code에서 **파일** → **폴더 열기**로 python-data-analysis 디렉토리를 연다. 새 창이 열리면, 3장에서 배운 Docker를 활용하여 분석 환경을 구성한다. Claude Code에게 다음과 같이 요청한다:

Python 데이터 분석용 Docker 환경을 만들어줘. pandas, numpy, matplotlib, seaborn, scipy를 포함하고, Jupyter Notebook도 사용할 수 있게 해줘. compose.yml과 Dockerfile을 만들어줘.

Claude Code가 Dockerfile과 compose.yml을 생성하여, 필요한 패키지가 모두 설치된 컨테이너 환경을 구성해 준다. Docker를 사용하면 “내 컴퓨터에서는 되는데 다른 컴퓨터에서는 안 된다”는 문제를 방지할 수 있다.

패키지 설치 자체는 시에게 맡길 수 있지만, **어떤 패키지가 어떤 역할을 하는지**는 사람이 알고 있어야 한다. 시의 결과물이 맞는지 판단하려면 각 패키지의 역할을 이해해야 하기 때문이다.

패키지	역할	주요 용도
pandas	테이블 데이터 처리	CSV/TSV 읽기, 필터링, 정렬, 집계
NumPy	수치 배열 연산	수학 함수, 행렬 연산, 난수 생성
Matplotlib	기본 시각화	산점도, 히스토그램, 서브플롯, 논문용 그래프
Seaborn	통계 시각화	박스 플롯, 히트맵, 바이올린 플롯
SciPy	과학 계산/통계	t-test, 상관분석, 다중 검정 보정

4.3 pandas — 테이블 데이터 처리

pandas는 표(테이블) 형태의 데이터를 다루는 패키지다. 엑셀 스프레드시트를 Python에서 다룬다고 생각하면 된다. 생명정보학에서는 유전자 발현 데이터, 변이 목록, 샘플 메타데이터 등 대부분의 데이터가 표 형태이므로 pandas는 거의 모든 분석의 출발점이 된다.

핵심 개념

- **DataFrame**: 행과 열로 이루어진 2차원 테이블. pandas의 핵심 자료구조다. 엑셀 시트 하나와 비슷하다.
- **Series**: DataFrame의 한 열(column). 1차원 데이터다.
- **Index**: 각 행을 식별하는 라벨. 유전자 이름이나 샘플 ID를 인덱스로 설정하면 데이터 접근이 편리해진다.

코드 예시

Claude Code에게 다음과 같이 요청한다:

gene_expression.csv 파일을 읽어서 기본 탐색을 해줘. 데이터 미리보기, 기초 통계량, pvalue 0.05 미만 필터링, logFC 내림차순 정렬, -log10(pvalue) 열 추가까지 해줘

이 프롬프트에 대해 Claude Code가 생성하는 코드는 다음과 같다:

```
import pandas as pd

# CSV
df = pd.read_csv("gene_expression.csv")

#
df.head()          # 5
df.shape           # ( , )
df.describe()     #

#
df["gene_name"]   #
df[["gene_name", "logFC"]] #

#
significant = df[df["pvalue"] < 0.05]
```

```
#
df.sort_values("logFC", ascending=False)

#
df["neg_log_p"] = -np.log10(df["pvalue"])
```

In [1]: df.head(8)

index	gene_name	logFC	pvalue	cell_type
0	TP53	2.45	1.2e-06	T Cell
1	BRCA1	-1.89	0.0003	B Cell
2	MYC	3.12	4.5e-08	Neuron
3	EGFR	-2.10	0.0011	Macrophage
4	VEGFA	1.98	0.0008	Fibroblast
5	GAPDH	0.05	0.9521	Endothelial
6	ACTB	-0.01	0.9810	Hepatocyte
7	TNF	2.76	9.1e-07	Keratinocyte

Figure 37: pandas DataFrame을 Jupyter Notebook에서 출력한 예시 스크린샷

위 코드에서 `df[df["pvalue"] < 0.05]`는 p-value가 0.05 미만인 행만 골라내는 필터링이다. 엑셀에서 필터를 거는 것과 같은 작업이지만, 수십만 행도 순식간에 처리한다. `df.describe()`는 각 숫자 열의 평균, 표준편차, 최솟값, 최댓값 등을 한눈에 보여주는 요약 통계를 생성한다.

이 요청을 하려면 `pvalue`, `logFC`가 무엇인지, `필터링`과 `정렬`이라는 개념을 알아야 한다. 코드 문법은 몰라도 되지만, 데이터의 의미는 사람이 이해하고 있어야 한다.

4.4 NumPy — 수치 연산

NumPy는 대규모 수치 데이터를 빠르게 처리하는 패키지다. pandas의 내부에서도 NumPy를 사용하며, Scrapy, Matplotlib 등 거의 모든 과학 계산 패키지가 NumPy 위에 구축되어 있다. Python 과학 생태계의 기반이라 할 수 있다.

핵심 개념

- **ndarray**: N차원 배열. 같은 타입의 데이터를 담는 고성능 자료구조다. Python 기본 리스트보다 수십~수백 배 빠르다.
- **브로드캐스팅**: 크기가 다른 배열 간 연산을 자동으로 확장하는 기능. 예를 들어 배열의 모든 원소에 2를 곱할 때, 반복문 없이 `arr * 2`로 가능하다.
- **벡터 연산**: 반복문 없이 배열 전체에 연산을 한 번에 적용한다. `np.log2(arr)`는 배열의 모든 원소에 `log2`를 적용한다.

코드 예시

발현량 데이터에 \log_2 변환을 적용하고, 평균과 표준편차를 계산해줘. 시뮬레이션용 정규분포 난수도 1000개 생성해줘

이 프롬프트에 대해 Claude Code가 생성하는 코드는 다음과 같다:

```
import numpy as np

#
arr = np.array([1, 2, 3, 4, 5])
matrix = np.zeros((100, 100)) # 100x100

#
arr.mean() #
arr.std() #
arr.max() #

# ( )
log_values = np.log2(arr + 1) # log2

#
random_data = np.random.normal(0, 1, size=1000) #
```

NumPy 자체를 직접 쓸 일은 많지 않지만, AI가 생성하는 코드에 자주 등장한다. `np.log2`, `np.mean` 같은 표현이 나왔을 때 무엇을 하는 코드인지 이해할 수 있으면 충분하다.

4.5 Matplotlib — 기본 시각화

Matplotlib은 Python의 가장 기본적인 시각화 패키지다. 1999년부터 개발되어 온 역사 깊은 패키지로, 거의 모든 종류의 그래프를 그릴 수 있다. 학술 논문에 실리는 그래프 대부분이 Matplotlib으로 만들어진다.

핵심 개념

- **Figure:** 전체 그림 영역. 하나의 Figure 안에 여러 그래프를 배치할 수 있다. 종이 한 장이라고 생각하면 된다.
- **Axes:** 개별 그래프 영역. 실제로 데이터가 그려지는 공간이다. Figure 안에 놓이는 하나의 차트.
- **Subplot:** Figure를 격자로 나누어 여러 그래프를 배치하는 방식. 논문에서 Figure 1의 (A), (B), (C) 패널과 같다.

코드 예시

\log_2 FC와 $-\log_{10}$ (pvalue)로 volcano plot을 그려줘. 300 dpi로 저장해줘

이 프롬프트에 대해 Claude Code가 생성하는 코드는 다음과 같다:

```
import matplotlib.pyplot as plt

# (Scatter plot)
plt.scatter(df["logFC"], df["neg_log_p"])
plt.xlabel("Log Fold Change")
plt.ylabel("-log10(p-value)")
plt.title("Volcano Plot")
plt.savefig("volcano.png", dpi=300)
plt.show()
```

Volcano plot은 차등 발현 분석(DEA)의 대표적인 시각화 방법이다. X축에 발현 변화량(\log_2 FC), Y축에 통계적 유의성($-\log_{10}$ p-value)을 표시하여, 발현이 크게 변하면서 통계적으로 유의한 유전자를 한눈에 파악할 수 있다. 이름은 화산 분출 모양을 닮았다고 해서 붙여졌다.

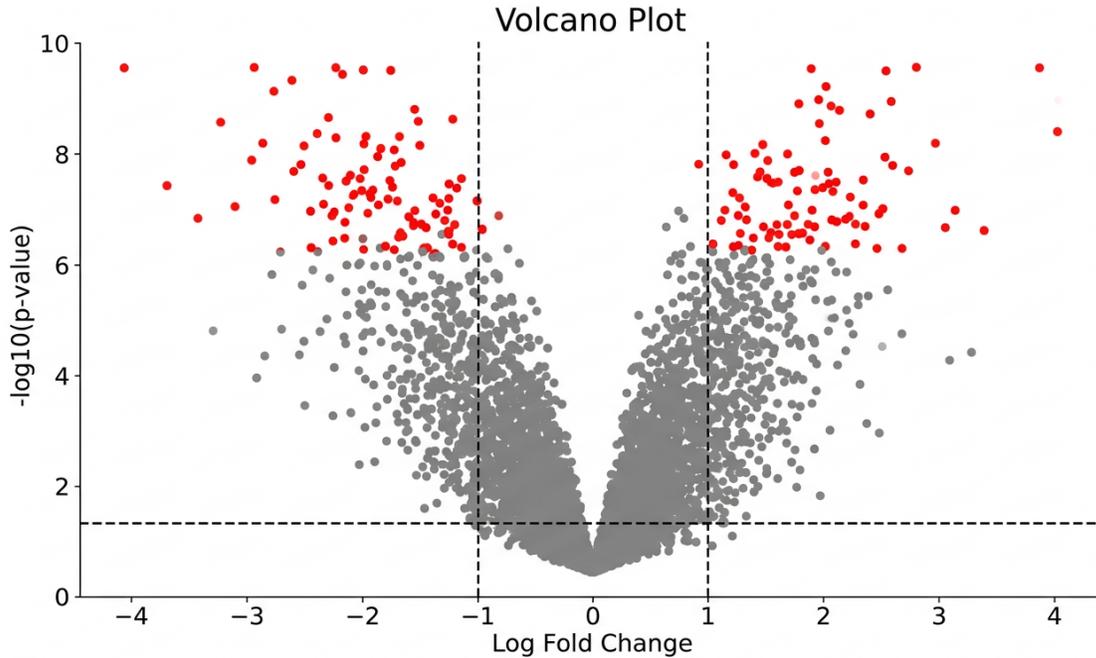


Figure 38: Matplotlib으로 그린 Volcano Plot 예시

더 구체적인 요청도 가능하다:

logFC 히스토그램과 volcano plot을 나란히 subplot으로 배치해줘. 300 dpi로 저장해줘

```
# (Subplot)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].hist(df["logFC"], bins=50)
axes[0].set_title("Distribution of logFC")
axes[1].scatter(df["logFC"], df["neg_log_p"], s=1)
axes[1].set_title("Volcano Plot")
plt.tight_layout()
plt.savefig("combined.png", dpi=300)
```

dpi=300은 해상도를 300 DPI(dots per inch)로 설정하는 것인데, 학술 논문 출판에 요구되는 기본 해상도이다. 화면에서는 차이가 크지 않지만, 인쇄물에서는 선명도에 큰 차이가 난다.

요청을 더 세밀하게 할수록 결과물이 정교해진다:

logFC와 $-\log_{10}(pvalue)$ 로 volcano plot 그려줘. significant한 유전자($pvalue < 0.05$, $|\logFC| > 1$)는 빨간색으로 표시하고, 나머지는 회색으로. 그래프 해상도는 300 dpi로 저장해줘

4.6 Seaborn — 통계 시각화

Seaborn은 Matplotlib 위에 구축된 통계 시각화 전문 패키지다. 더 적은 코드로 보기 좋은 통계 그래프를 만들 수 있다. Matplotlib이 모든 것을 그릴 수 있는 범용 도구라면, Seaborn은 통계 시각화에 특화된 고수준 도구다.

Matplotlib과의 차이

	Matplotlib	Seaborn
수준	저수준 (세밀한 제어)	고수준 (간결한 코드)

	Matplotlib	Seaborn
스타일	기본 스타일 단순함	기본 스타일이 깔끔함
통계 기능	직접 구현 필요	회귀선, 분포 등 내장
DataFrame 연동	수동으로 데이터 전달	DataFrame 직접 지원

실무에서는 두 패키지를 함께 사용하는 경우가 많다. Seaborn으로 기본 그래프를 빠르게 그리고, Matplotlib으로 세부 요소(제목, 레이블, 범례 위치 등)를 조정하는 식이다.

코드 예시

cell_type별 유전자 발현량 분포를 박스 플롯으로 비교해줘

```
import seaborn as sns

# -
sns.boxplot(data=df, x="cell_type", y="expression")
plt.xticks(rotation=45)
plt.show()
```

박스 플롯은 데이터의 분포를 요약하여 보여준다. 상자의 중앙선은 중앙값(median), 상자의 위아래는 사분위 범위(IQR), 수염(whisker)은 이상치를 제외한 범위를 나타낸다. 여러 그룹의 분포를 비교할 때 가장 많이 사용되는 시각화 방법이다.

유전자 발현 매트릭스를 히트맵으로 시각화해줘. 빨강-파랑 색상표를 사용하고, 0을 중심으로 설정해줘

```
# -
sns.heatmap(expression_matrix, cmap="RdBu_r", center=0)
plt.title("Gene Expression Heatmap")
plt.show()
```

히트맵은 행렬 형태의 데이터를 색상으로 표현한다. 유전자 발현 데이터에서 행은 유전자, 열은 샘플이고, 색상의 강도가 발현량을 나타낸다. cmap="RdBu_r"은 빨강-파랑 색상표를 사용하고, center=0은 발현 변화 없음(0)을 흰색으로 설정한다. 발현이 증가한 유전자는 빨간색, 감소한 유전자는 파란색으로 직관적으로 구분할 수 있다.

condition별 발현량 분포를 바이올린 플롯으로 그려줘

```
# -
sns.violinplot(data=df, x="condition", y="expression")
plt.show()
```

gene_A와 gene_B 사이의 상관관계를 산점도로 그리고, 회귀선도 함께 표시해줘

```
# +
sns.regplot(data=df, x="gene_A", y="gene_B")
plt.show()
```

요청을 더 세밀하게 할수록 결과물이 정교해진다:

cell_type별 gene expression의 분포를 violin plot으로 비교해줘. 색상은 pastel 팔레트 사용하고, 각 그룹의 데이터 포인트도 strip plot으로 겹쳐서 보여줘

4.7 SciPy — 과학 계산과 통계 검정

SciPy는 과학 계산에 필요한 다양한 알고리즘을 제공한다. 생명정보학에서는 주로 통계 검정 기능을 사용한다. 두 그룹 사이에 유의한 차이가 있는지, 두 변수 사이에 상관관계가 있는지를 통계적으로 판단할 때 SciPy의 함수들이 사용된다.

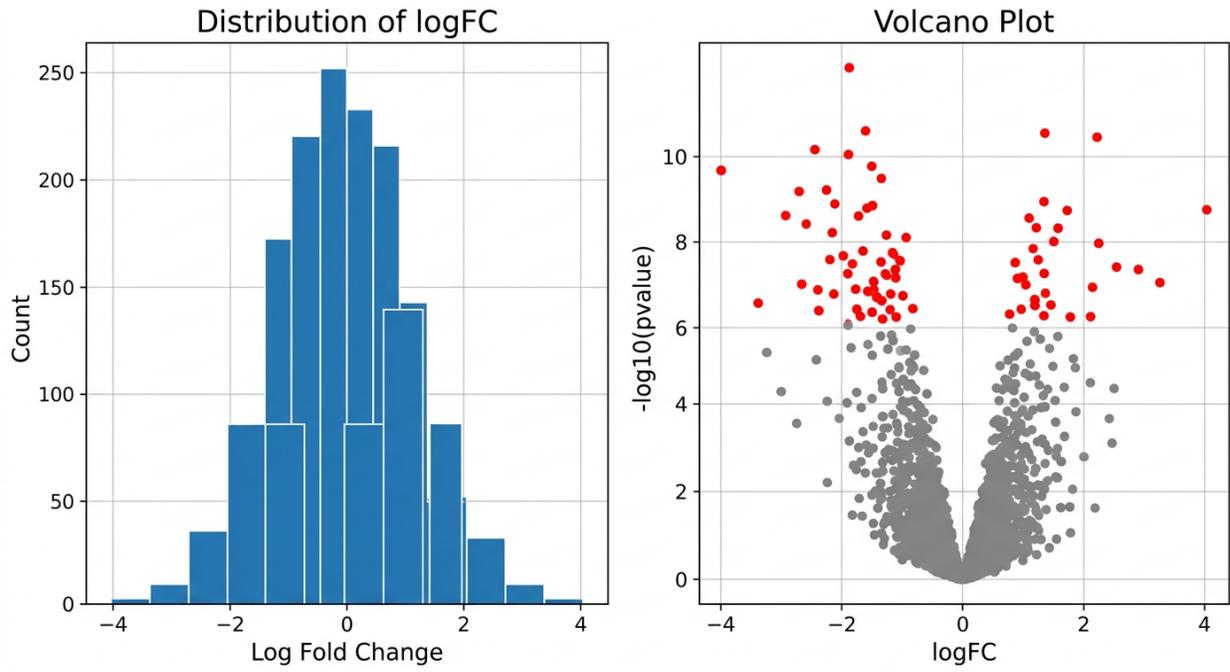


Figure 39: Subplot으로 두 개의 그래프를 나란히 배치한 예시

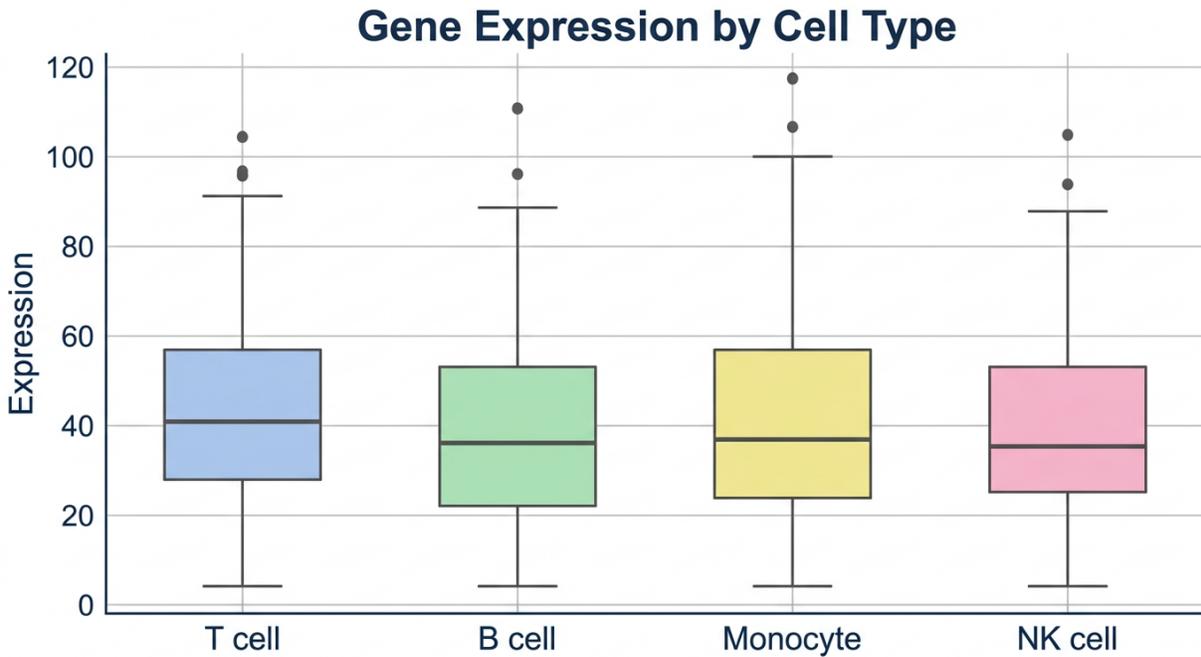


Figure 40: Seaborn 박스 플롯 예시 — 세포 유형별 유전자 발현량 분포

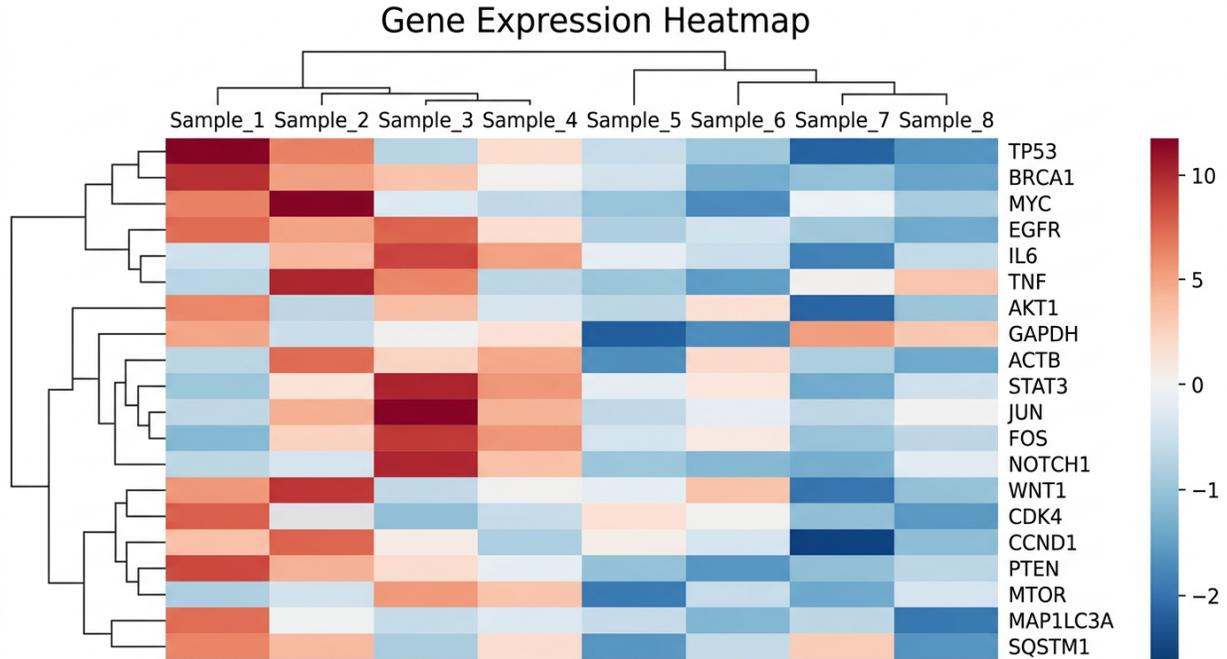


Figure 41: Seaborn 히트맵 예시 — 유전자 발현 매트릭스

코드 예시

treatment 그룹과 control 그룹의 gene expression을 t-test로 비교해줘. Pearson 상관계수도 구하고, Benjamini-Hochberg 다중 검정 보정도 적용해줘

이 프롬프트에 대해 Claude Code가 생성하는 코드는 다음과 같다:

```
from scipy import stats

# t-test -
t_stat, p_value = stats.ttest_ind(group_a, group_b)
print(f"t-statistic: {t_stat:.4f}, p-value: {p_value:.4e}")

# Mann-Whitney U test - ( )
u_stat, p_value = stats.mannwhitneyu(group_a, group_b)

# Pearson
corr, p_value = stats.pearsonr(gene_a_expression, gene_b_expression)
print(f"Correlation: {corr:.4f}, p-value: {p_value:.4e}")

# (Benjamini-Hochberg)
from scipy.stats import false_discovery_control
adjusted_pvalues = false_discovery_control(p_values, method='bh')
```

t-test는 두 그룹의 평균이 통계적으로 다른지를 검정한다. 예를 들어 약물 처리 그룹과 대조군의 유전자 발현 평균을 비교할 때 사용한다. Mann-Whitney U test는 데이터가 정규분포를 따르지 않을 때 사용하는 비모수 대안이다.

다중 검정 보정은 특히 중요하다. 유전자 2만 개를 동시에 검정하면, 순전히 우연에 의해서도 약 1,000개(2만 × 0.05)가 “유의하다”고 나올 수 있다. Benjamini-Hochberg 보정은 이런 거짓 양성(false positive)을 통제하여 실제로 의미 있는 결과만 남기는 방법이다.

이 요청에서 t-test가 무엇인지, 다중 검정 보정이 왜 필요한지를 이해하고 있어야 시의 결과를 올바르게 해석할 수 있다.

4.8 AI와 함께하는 데이터 분석

워크플로우

바이브 코딩으로 데이터 분석을 할 때의 일반적인 흐름이다:

1. **데이터 확인**: “이 CSV 파일의 구조를 보여줘” → AI가 `pd.read_csv`와 `df.head()`, `df.describe()` 실행
2. **전처리**: “결측값 제거하고, gene_name 열을 인덱스로 설정해줘” → AI가 `dropna()`, `set_index()` 적용
3. **분석**: “두 그룹 간 차이가 있는 유전자를 찾아줘” → AI가 통계 검정 수행
4. **시각화**: “결과를 volcano plot으로 그려줘” → AI가 Matplotlib/Seaborn으로 시각화
5. **해석**: 결과를 사람이 확인하고, 추가 분석 방향을 지시

이 흐름에서 1~4단계의 코드는 AI가 작성하지만, 5단계의 해석은 전적으로 사람의 몫이다. “이 유전자가 왜 상향 조절되었는가?”, “이 결과가 기존 문헌과 일치하는가?” 같은 질문은 도메인 지식이 있어야 답할 수 있다.

핵심 포인트

사람이 해야 할 일	AI가 해주는 일
분석 목표 설정	코드 작성
적절한 분석 방법 선택	패키지 import 및 함수 호출
결과 해석	그래프 생성 및 통계량 계산
생물학적 의미 판단	데이터 전처리 및 변환

핵심: 코드를 외울 필요는 없다. 하지만 “박스 플롯은 분포를 비교할 때 쓴다”, “t-test는 두 그룹의 평균을 비교한다”, “p-value가 작을수록 통계적으로 유의하다” 같은 개념은 반드시 이해해야 한다. 이것이 AI에게 올바른 지시를 내리고, AI의 결과를 검증하는 힘이 된다.

4.9 정리

- **pandas**: 테이블 데이터(CSV, TSV) 읽기, 필터링, 정렬, 집계. 거의 모든 분석의 출발점
- **NumPy**: 수치 배열 연산, 수학 함수 (log, mean, std 등). 과학 계산의 기반
- **Matplotlib**: 기본 그래프 (산점도, 히스토그램, 서브플롯). 논문 출판 품질 지원
- **Seaborn**: 통계 시각화 (박스 플롯, 히트맵, 바이올린 플롯). 적은 코드로 깔끔한 그래프
- **SciPy**: 통계 검정 (t-test, 상관분석, 다중 검정 보정). 유의성 판단의 핵심
- **바이브 코딩의 핵심**: 문법이 아닌 개념을 이해하고, AI에게 정확히 요청하는 것

5장. 생명정보 데이터 분석 실전

4장에서 Python 데이터 분석의 기초를 익혔다면, 이제 실제 생명정보학 데이터를 분석해 볼 차례이다. 이 장에서는 두 가지 대표적인 분석 사례를 다룬다: **전장 유전체 시퀀싱(WGS) 분석**과 **단일세포 RNA-seq 분석**이다. 두 분석 모두 Claude Code와 대화하며 진행할 수 있지만, 각 단계가 왜 필요한지를 이해하고 있어야 AI에게 정확한 지시를 내릴 수 있다.

5.1 전장 유전체 시퀀싱(WGS) 분석

WGS란?

전장 유전체 시퀀싱(Whole Genome Sequencing, WGS)은 생물의 **전체 게놈**을 읽는 기술이다. 사람의 경우 약 30억 개의 염기쌍을 모두 시퀀싱하여, 돌연변이(SNP, InDel), 구조 변이, 복제수 변이 등을 검출할 수 있다.

WGS 데이터 분석의 기본 흐름은 다음과 같다:

1. **품질 관리(QC)**: 원시 시퀀싱 리드의 품질 확인 및 저품질 리드 제거
2. **정렬(Alignment)**: 리드를 reference genome에 매핑

3. 변이 검출(Variant Calling): SNP, InDel 등의 변이 식별
4. 변이 주석(Annotation): 검출된 변이의 기능적 의미 해석
5. 필터링 및 해석: 임상적 또는 연구적으로 의미 있는 변이 선별

주요 파일 형식

WGS 분석에서 다루게 되는 파일 형식을 이해하면 AI에게 요청할 때 더 정확한 지시를 내릴 수 있다.

형식	설명	용도
FASTQ	원시 시퀀싱 리드 + 품질 점수	시퀀서에서 나온 원본 데이터
BAM/CRAM	reference genome에 정렬된 리드	정렬 결과 저장, 시각화
VCF	변이 정보 (위치, 유형, 빈도)	변이 검출 결과
BED	게놈 구간 정보 (시작-끝 좌표)	관심 영역(ROI) 정의

FASTQ 파일은 시퀀서에서 바로 나오는 원시 데이터이다. 각 리드마다 염기 서열과 품질 점수가 함께 기록되어 있다. BAM 파일은 이 리드들을 reference genome의 어디에 위치하는지 매핑한 결과이다. VCF 파일은 reference genome과 다른 부분, 즉 변이를 기록한 파일이다.

분석 도구

WGS 분석에는 여러 도구가 사용된다. Claude Code에게 설치와 실행을 요청할 수 있지만, 각 도구의 역할을 알아야 올바른 파이프라인을 구성할 수 있다.

도구	역할	비고
FastQC	시퀀싱 품질 확인	리드 길이, 품질 분포, 어댑터 오염 등
fastp / Trimmomatic	저품질 리드 제거 및 어댑터 트리밍	fastp이 더 빠르고 간편
BWA-MEM2 / minimap2	리드를 reference genome에 정렬	BWA-MEM2가 Illumina short read 표준
samtools	BAM 파일 처리 (정렬, 인덱싱, 통계)	거의 모든 WGS 파이프라인에 필수
GATK	변이 검출 (HaplotypeCaller)	Broad Institute에서 개발, 사실상 표준
bcftools	VCF 파일 처리 및 필터링	변이 통계, 필터링, 병합
SnpEff / VEP	변이의 기능적 주석	아미노산 변화, 유전자 영향 예측

이 도구들은 대부분 Bioconda 채널을 통해 설치할 수 있으며, Docker 환경에서 관리하는 것이 재현성 면에서 유리하다.

Claude Code로 WGS 분석하기

먼저 터미널에서 프로젝트 디렉토리를 만든다:

```
mkdir wgs-analysis
```

VS Code에서 **파일** → **폴더 열기**로 wgs-analysis 디렉토리를 연다. 새 창이 열리면, 분석 환경을 Docker로 구성한다. Claude Code에게 다음과 같이 요청한다:

WGS 분석용 Docker 환경을 만들어줘. Bioconda 채널에서 fastqc, fastp, bwa-mem2, samtools, gatk4, bcftools를 설치하고, compose.yml과 Dockerfile을 만들어줘.

실제 분석 과정을 Claude Code와 대화하며 진행하는 예시이다:

```
sample_R1.fastq.gz, sample_R2.fastq.gz 파일의 품질을 FastQC로 확인해줘
fastp로 어댑터 트리밍하고, 품질 30 미만인 리드를 제거해줘
```

트리밍된 리드를 GRCh38 reference genome에 BWA-MEM2로 정렬해줘. 정렬 후 samtools로 정렬, 인덱싱까지 해줘

GATK HaplotypeCaller로 변이를 검출하고, PASS 필터만 남겨줘

검출된 VCF 파일에서 exonic 영역의 missense 변이만 추출해줘

각 단계에서 사용자가 알아야 하는 것은 “어댑터 트리밍이 왜 필요한지”, “매핑 퀄리티가 무엇인지”, “missense 변이가 무엇인지” 같은 **도메인 지식**이다. 명령어의 구체적인 옵션은 Claude Code가 처리해 준다.

Python으로 VCF 분석하기

변이 검출이 끝나면, Python으로 결과를 분석하고 시각화할 수 있다. 4장에서 배운 pandas와 matplotlib이 여기서 활용된다.

VCF 파일을 읽어서 변이 유형별(SNP/InDel) 통계와 염색체별 분포를 시각화해줘

이 프롬프트에 대해 Claude Code가 생성하는 코드는 다음과 같다:

```
import pandas as pd
import matplotlib.pyplot as plt

# VCF pandas ( )
vcf = pd.read_csv("variants.vcf", sep="\t", comment="#",
                 names=["CHROM", "POS", "ID", "REF", "ALT",
                       "QUAL", "FILTER", "INFO", "FORMAT", "SAMPLE"])

#
print(f"      : {len(vcf)}")
print(f"SNP: {len(vcf[vcf['REF'].str.len() == vcf['ALT'].str.len()])}")
print(f"InDel: {len(vcf[vcf['REF'].str.len() != vcf['ALT'].str.len()])}")

#
vcf["CHROM"].value_counts().sort_index().plot(kind="bar")
plt.title(" ")
plt.xlabel(" ")
plt.ylabel(" ")
plt.tight_layout()
plt.savefig("variants_per_chromosome.png")
```

이처럼 시에게 요청하면 코드를 직접 작성하지 않아도 분석과 시각화를 수행할 수 있다.

5.2 단일세포 RNA-seq 분석

단일세포 분석이란?

전통적인 bulk RNA-seq은 수천~수백만 개의 세포를 한꺼번에 분석하여 **평균적인** 유전자 발현을 측정한다. 이 방식은 조직 전체의 발현 경향을 파악하는 데는 유용하지만, 조직 안에 어떤 종류의 세포가 있는지, 각 세포 유형에서 어떤 유전자가 활성화되어 있는지는 알 수 없다. 마치 과일 주스를 마시면서 어떤 과일이 들어갔는지는 알 수 있지만, 각 과일의 비율은 알 수 없는 것과 비슷하다.

단일세포 RNA-seq(scRNA-seq)은 **개별 세포 하나하나**의 유전자 발현을 측정한다. 이를 통해 다음과 같은 질문에 답할 수 있다:

- 이 조직에는 어떤 종류의 세포들이 있는가?
- 각 세포 유형의 비율은 어떻게 되는가?
- 특정 질환에서 어떤 세포 유형이 변화하는가?
- 세포들 사이의 발달 경로(trajecory)는 어떠한가?

10x Genomics의 Chromium 플랫폼 덕분에 한 번의 실험으로 수천~수만 개의 세포를 동시에 분석할 수 있게 되었고, 이제 단일세포 분석은 생명과학 연구의 핵심 기술로 자리잡았다. Scanpy는 이러한 단일세포 데이터를 분석하기 위한 Python 패키지다.

환경 구성

단일세포 분석 환경도 Docker로 구성한다. Claude Code에게 다음과 같이 요청한다:

단일세포 RNA-seq 분석용 Docker 환경을 만들어줘. scanpy, anndata, mudata를 포함하고, Jupyter Notebook도 사용할 수 있게 해줘. compose.yml과 Dockerfile을 만들어줘.

각 패키지의 역할을 알아야 시에게 올바른 분석을 요청할 수 있다:

패키지	역할	주요 용도
scanpy	단일세포 분석의 핵심 패키지	전처리, 클러스터링, 시각화
anndata	h5ad 파일 형식 처리	Scanpy의 데이터 구조(AnnData 객체)
mudata	h5mu 파일(멀티오믹스 데이터) 처리	CITE-seq, 10x Multiome 등

Scanpy는 **scverse** 생태계의 일부이다. scverse는 단일세포 데이터 분석을 위한 Python 패키지 모음으로, Scanpy를 중심으로 squidpy(공간 전사체), scvi-tools(딥러닝 기반 분석), muon(멀티오믹스) 등이 포함된다. 하나의 일관된 데이터 구조(AnnData)를 공유하므로, 여러 패키지를 조합하여 분석할 수 있다.

AnnData — 단일세포 데이터 구조

h5ad는 단일세포 데이터의 **표준 저장 형식**이다. HDF5 기반으로, 대용량 데이터를 효율적으로 저장하고 읽을 수 있다. 10,000개의 세포와 20,000개의 유전자를 포함하는 데이터를 CSV로 저장하면 수 GB가 되지만, h5ad로 저장하면 수십~수백 MB로 줄어든다.

하나의 h5ad 파일에는 다음 정보가 모두 담겨 있다:

속성	설명	예시
X	유전자 발현 매트릭스 (세포 × 유전자)	10,000 세포 × 20,000 유전자
obs	세포(행)에 대한 메타데이터	cell_type, sample_id, condition
var	유전자(열)에 대한 메타데이터	gene_name, highly_variable
obsm	세포의 임베딩 좌표	UMAP, t-SNE, PCA 좌표
uns	비구조화 데이터	색상 팔레트, 분석 파라미터

이 구조의 장점은 **하나의 파일에 분석에 필요한 모든 것이 담겨 있다**는 점이다. 발현 데이터, 세포 메타데이터, 분석 결과(UMAP 좌표, 클러스터 레이블 등)가 하나의 객체에 통합되어 있으므로, 데이터를 주고받을 때 파일 하나만 전달하면 된다.

MuData — 멀티오믹스 데이터

최근에는 하나의 세포에서 RNA와 단백질(CITE-seq), 또는 RNA와 염색질 접근성(10x Multiome)을 동시에 측정하는 **멀티오믹스** 기술이 발전하고 있다. h5mu는 이런 멀티오믹스 데이터를 저장하는 형식이다.

multiome.h5mu 파일을 읽고, RNA와 ATAC 데이터를 각각 분리해줘

```
import mudata as md
```

```
# h5mu
```

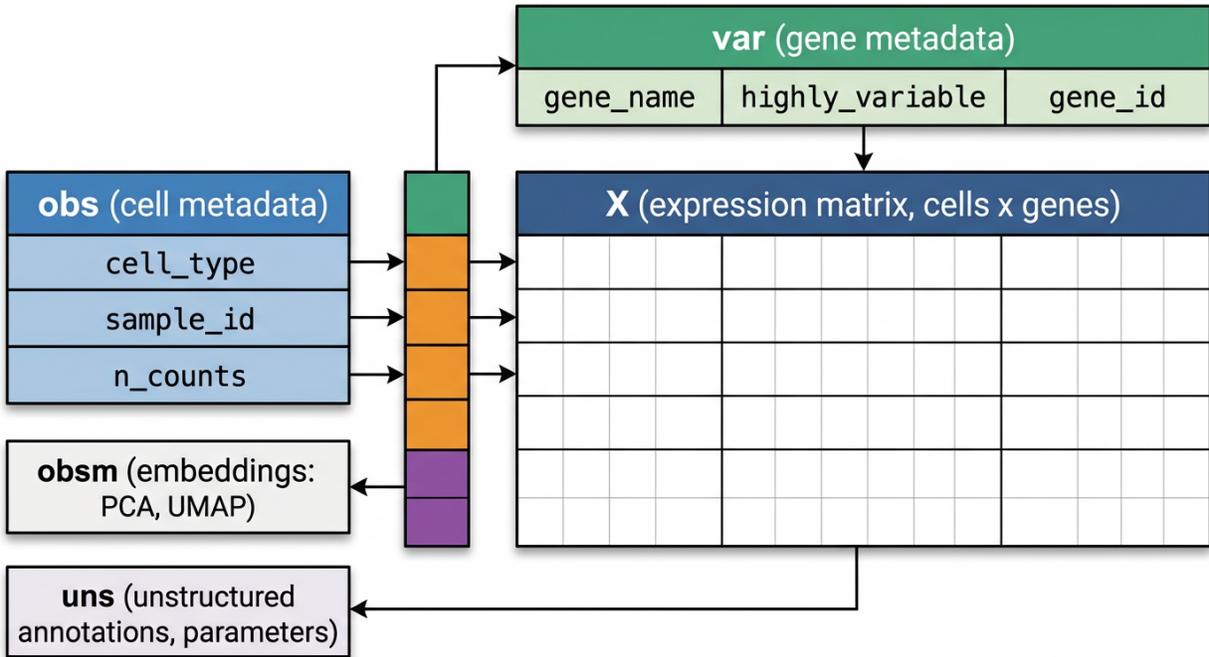


Figure 42: AnnData 객체의 구조를 나타낸 다이어그램

```
mdata = md.read_h5mu("multiome.h5mu")
```

```
# modality
rna = mdata.mod["rna"] # RNA (AnnData)
atac = mdata.mod["atac"] # ATAC (AnnData)
```

각 modality는 독립적인 AnnData 객체이므로, RNA 데이터에는 Scanpy를, ATAC 데이터에는 SnapATAC이나 ArchR의 Python 래퍼를 사용하는 식으로 각각에 적합한 분석 방법을 적용할 수 있다.

Scanpy 분석 워크플로우

단일세포 분석은 보통 다음 순서로 진행된다. 각 단계가 왜 필요한지 이해하면 시에게 정확한 분석을 요청할 수 있다.

품질 관리 (Quality Control)

생 데이터에는 품질이 낮은 세포(죽은 세포, 이중 캡처 등)가 포함되어 있다. 이런 세포를 걸러내지 않으면 분석 결과가 왜곡된다.

주요 QC 지표: - **유전자 수(n_genes_by_counts)**: 한 세포에서 검출된 유전자가 너무 적으면(예: 200개 미만) 품질이 낮은 세포일 가능성이 높다 - **미토콘드리아 유전자 비율(pct_counts_mt)**: 미토콘드리아 유전자의 비율이 높으면(예: 20% 이상) 세포가 손상되었을 가능성이 있다. 죽어가는 세포에서 세포질 RNA는 빠져나가지만 미토콘드리아 RNA는 남아 있기 때문이다

미토콘드리아 유전자 비율을 계산하고, QC 지표들을 바이올린 플롯으로 시각화해줘

```
#
adata.var["mt"] = adata.var_names.str.startswith("MT-")
sc.pp.calculate_qc_metrics(adata, qc_vars=["mt"], inplace=True)

# QC
sc.pl.violin(adata, ["n_genes_by_counts", "total_counts", "pct_counts_mt"])
```

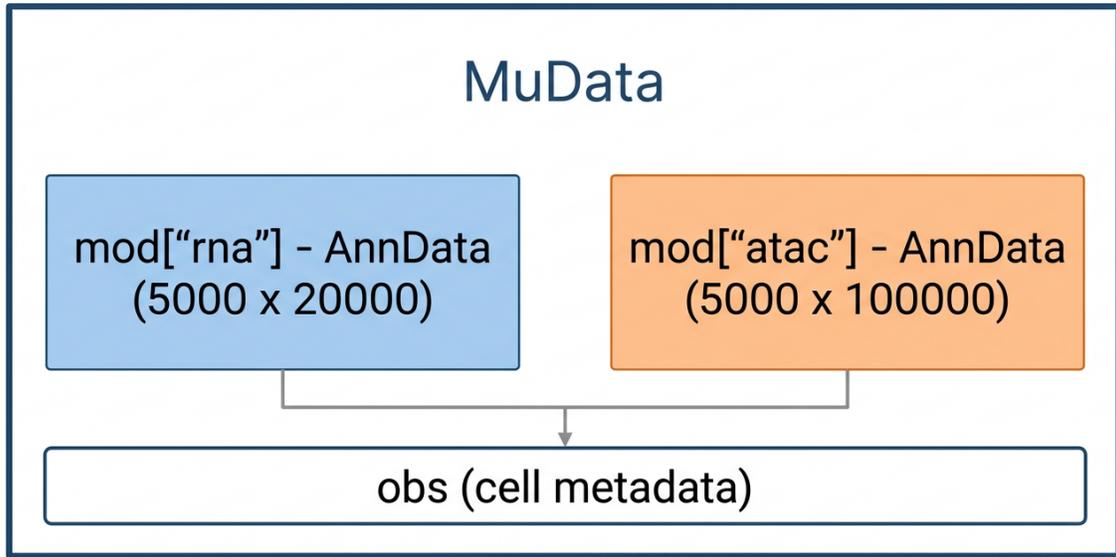


Figure 43: MuData 객체의 구조 다이어그램

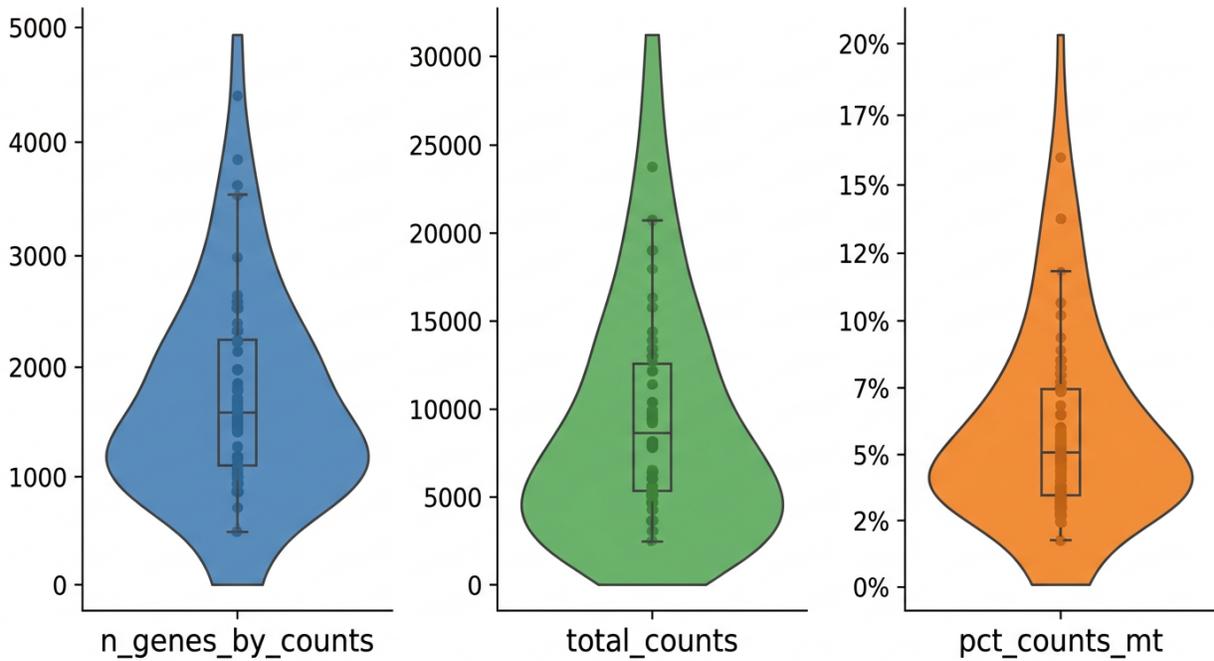


Figure 44: QC 바이올린 플롯

정규화 및 전처리

각 세포마다 시퀀싱 깊이(total read count)가 다르다. 정규화는 이 차이를 보정하여 세포 간 공정한 비교를 가능하게 한다. **고변동 유전자(HVG)** 선택은 세포 유형을 구분하는 데 기여하는 유전자만 골라내어 분석의 효율과 정확도를 높인다.

차원 축소 및 클러스터링

2,000개의 유전자를 사용한다면 각 세포는 2,000차원 공간의 한 점이다. PCA로 50차원으로 압축한 뒤, UMAP으로 2차원 시각화를 수행한다. 이후 Leiden 알고리즘으로 유사한 세포들을 클러스터로 묶는다.

Leiden 클러스터링을 resolution 0.5로 수행하고, UMAP에 클러스터를 색상으로 표시해줘

```
sc.tl.leiden(adata, resolution=0.5)
sc.pl.umap(adata, color="leiden")
```

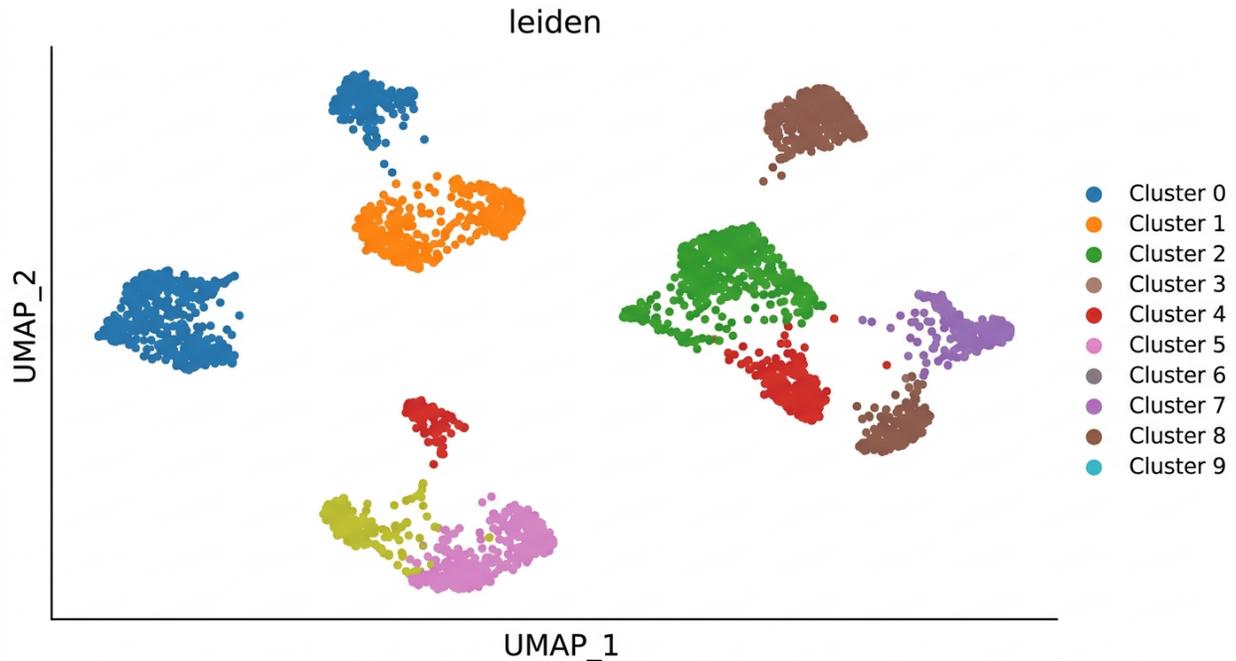


Figure 45: UMAP 플롯

세포 유형 주석

각 클러스터에서 특이적으로 높게 발현되는 **마커 유전자**를 확인하고, 이를 바탕으로 세포 유형을 결정한다.

클러스터별 마커 유전자를 찾고, CD3E, CD14, MS4A1, NKG7 발현을 UMAP에 표시해줘

```
sc.tl.rank_genes_groups(adata, groupby="leiden")
sc.pl.umap(adata, color=["CD3E", "CD14", "MS4A1", "NKG7"])
```

예를 들어 CD3E가 높게 발현되는 클러스터는 T 세포, CD14가 높은 클러스터는 단핵구(monocyte), MS4A1(CD20)이 높은 클러스터는 B 세포일 가능성이 높다. 이런 마커 유전자 해석은 면역학, 세포생물학 등 도메인 지식에 기반한다.

Claude Code로 단일세포 분석하기

pbmc_10k.h5ad 파일을 읽고 QC 해줘. 미토콘드리아 비율 20% 이상, 유전자 수 200개 미만인 세포는 제거해줘

정규화하고 고변동 유전자 2000개 선택해서 PCA, UMAP 해줘

Leiden 클러스터링 해주고, CD3E, CD14, MS4A1, NKG7 마커로 세포 유형을 UMAP에 표시해줘

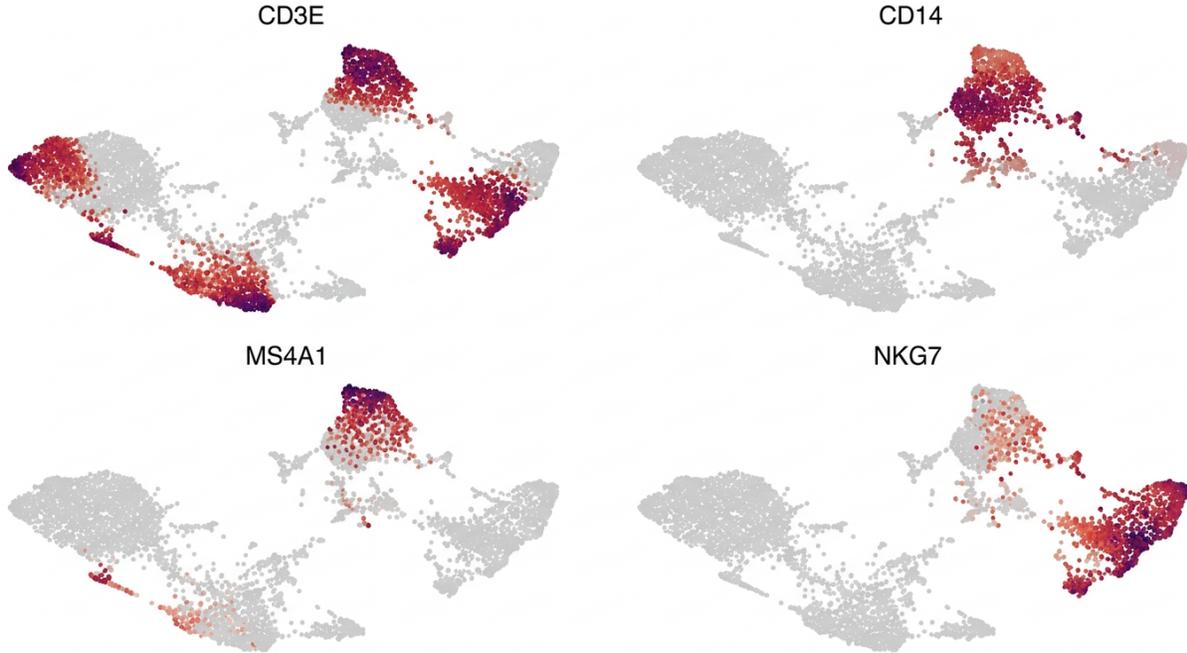


Figure 46: 마커 유전자별 UMAP 플롯

이 대화에서 사용자는 코드를 한 줄도 작성하지 않았지만, 각 단계가 왜 필요한지, 어떤 파라미터를 사용해야 하는지를 알고 있었기 때문에 정확한 지시를 내릴 수 있었다.

5.3 핵심 개념 정리

코드를 외울 필요는 없지만, AI에게 올바른 지시를 내리려면 다음 개념들을 이해해야 한다:

WGS 분석

개념	설명
FASTQ 정렬 (Alignment)	시퀀서에서 나온 원시 리드 데이터. 염기 서열 + 품질 점수 리드를 reference genome에 매핑하는 과정. BWA-MEM2가 표준
변이 검출 (Variant Calling)	reference genome과 다른 부분을 찾는 과정. GATK HaplotypeCaller가 표준
VCF	변이 정보를 저장하는 파일 형식. 위치, 유형, 품질 등을 포함
SNP / InDel	단일 염기 변이(SNP)와 삽입/결실(InDel). 가장 흔한 변이 유형

단일세포 분석

개념	설명
QC (Quality Control)	품질이 낮은 세포를 걸러내는 과정. 미토콘드리아 비율, 유전자 수가 주요 지표
정규화 (Normalization)	세포 간 시퀀싱 깊이 차이를 보정하는 과정

개념	설명
고변동 유전자 (HVG) PCA / UMAP	세포 간 발현 차이가 큰 유전자. 세포 유형 구분의 핵심 고차원 데이터를 축소하여 시각화. 비슷한 세포끼리 가깝게 배치
클러스터링	유사한 세포를 그룹으로 묶는 과정. Leiden 알고리즘이 표준
마커 유전자	특정 세포 유형을 구분하는 유전자. 도메인 지식이 필요

5.4 정리

- WGS 분석: FASTQ → 트리밍 → 정렬(BAM) → 변이 검출(VCF) → 주석 → 해석
 - 파일 형식(FASTQ, BAM, VCF, BED)과 각 도구의 역할을 이해하면 Claude Code에게 파이프라인 구성을 요청할 수 있다
- 단일세포 분석: h5ad 로딩 → QC → 정규화 → HVG 선택 → PCA → UMAP → 클러스터링 → 세포 유형 주석
 - AnnData 구조와 각 분석 단계의 목적을 이해하면 정확한 지시를 내릴 수 있다
- 공통 원칙: 바이브 코딩의 핵심은 분석 파이프라인의 각 단계가 왜 필요한지를 이해하고, AI에게 단계별로 지시하는 것이다. 명령어를 외울 필요는 없지만, 도메인 지식은 반드시 필요하다

6장. Snakemake를 이용한 워크플로우 관리

6.1 워크플로우란?

생명정보학 분석은 보통 여러 단계를 순서대로 수행한다. 예를 들어 RNA-seq 분석은 다음과 같은 흐름을 따른다:

1. FASTQ 파일 품질 확인 (FastQC)
2. 어댑터 트리밍 (Trim Galore)
3. 레퍼런스 게놈에 정렬 (STAR)
4. 발현량 정량 (featureCounts)
5. 차등 발현 분석 (DESeq2)

이 과정을 매번 수동으로 실행하면 시간이 오래 걸리고, 실수가 생기기 쉽다. 터미널에서 명령을 하나씩 입력하다 보면 순서를 잘못 바꾸거나, 이전 단계의 출력 파일 이름을 틀리거나, 새 샘플이 추가되었는데 일부 단계만 다시 실행하는 등의 문제가 생긴다. 샘플이 3개뿐이라면 수동으로 관리할 수 있겠지만, 50개 샘플을 다룬다면 사실상 불가능하다.

Snakemake는 이러한 분석 파이프라인을 자동화하는 워크플로우 관리 도구이다. Python 기반 문법을 사용하여 각 분석 단계의 입력과 출력을 정의하면, Snakemake가 자동으로 실행 순서를 결정하고 필요한 단계만 실행해 준다.

Snakemake의 장점

- **재현성**: 같은 Snakefile로 언제든 동일한 분석을 반복할 수 있다. 논문 리뷰어가 “분석을 다시 돌려보라”고 요청해도, 명령 하나로 전체 파이프라인을 재실행할 수 있다.
- **자동 의존성 관리**: 어떤 단계를 먼저 실행해야 하는지 자동으로 판단한다. 정렬 결과가 없으면 정렬부터 실행하고, 트리밍 결과가 없으면 트리밍부터 실행한다.
- **병렬 실행**: 독립적인 단계는 동시에 실행하여 시간을 절약한다. sample_A의 정렬과 sample_B의 정렬은 서로 의존하지 않으므로 동시에 실행할 수 있다.
- **부분 재실행**: 중간에 실패하면 실패한 단계부터 다시 시작한다. 50개 샘플 중 3개에서만 에러가 났다면, 성공한 47개는 건드리지 않고 3개만 다시 처리한다.

다른 워크플로우 도구와의 비교

생명정보학에는 Snakemake 외에도 여러 워크플로우 도구가 있다.

도구	언어	특징
Snakemake	Python	생명정보학에서 가장 널리 사용, Conda 통합
Nextflow	Groovy/DSL	클라우드 환경에 강점, nf-core 생태계
WDL	자체 DSL	Broad Institute 개발, Terra 플랫폼 연동
CWL	YAML/JSON	표준 규격, 이식성 높음

이 책에서는 Python 문법과 가장 유사하여 진입 장벽이 낮고, Bioconda와의 통합이 뛰어난 Snakemake를 사용한다.

6.2 환경 구성

먼저 터미널에서 프로젝트 디렉토리를 만든다:

```
mkdir snakemake-pipeline
```

VS Code에서 **파일** → **폴더 열기**로 snakemake-pipeline 디렉토리를 연다. 새 창이 열리면, Docker 환경을 구성한다. Snakemake도 Docker 환경에서 실행하는 것이 재현성 면에서 유리하다. Claude Code에게 다음과 같이 요청한다:

Snakemake 파이프라인 실행용 Docker 환경을 만들어줘. snakemake와 함께 conda/mamba를 포함해서, 각 rule에서 독립적인 Conda 환경을 사용할 수 있게 해줘. compose.yml과 Dockerfile을 만들어줘.

참고: 복잡한 생명정보학 파이프라인에서는 Conda와 함께 사용하는 것이 일반적이다. 각 rule에 독립적인 Conda 환경을 지정할 수 있어, STAR는 STAR의 환경에서, DESeq2는 R의 환경에서 실행하는 것이 가능하다. 이렇게 하면 서로 다른 도구의 의존성이 충돌하는 문제를 피할 수 있다.

6.3 Snakemake 핵심 개념

Rule (규칙)

Snakemake의 기본 단위는 **rule**이다. 하나의 rule은 **입력** → **처리** → **출력**을 정의한다. 요리 레시피에 비유하면, “재료(input)를 가져와서 조리(shell)하고 완성된 요리(output)를 내놓는다”는 구조이다.

```
rule fastqc:
    input:
        "data/{sample}.fastq.gz"
    output:
        "results/fastqc/{sample}_fastqc.html"
    shell:
        "fastqc {input} -o results/fastqc/"
```

이 rule은 “data 폴더의 FASTQ 파일을 입력으로 받아 FastQC를 실행하고, 결과 HTML을 results/fastqc 폴더에 저장한다”는 의미이다. Snakemake는 output 파일이 이미 존재하면 해당 rule을 건너뛴다. 불필요한 재실행을 자동으로 방지하는 것이다.

Wildcard (와일드카드)

{sample} 같은 와일드카드를 사용하면 **여러 샘플에 같은 규칙을 자동 적용**할 수 있다. 예를 들어 sample_A.fastq.gz, sample_B.fastq.gz, sample_C.fastq.gz가 있으면, 위 rule이 세 파일 모두에 자동으로 실행된다.

와일드카드는 Snakemake의 가장 강력한 기능 중 하나이다. 수동으로 파이프라인을 실행할 때는 각 샘플마다 명령을 반복해야 하지만, 와일드카드를 사용하면 rule 하나로 모든 샘플을 처리할 수 있다. 새 샘플이 추가되더라도 rule을 수정할 필요가 없다.

DAG (방향성 비순환 그래프)

Snakemake는 rule 간의 의존 관계를 DAG(Directed Acyclic Graph)로 자동 구성한다. 한 rule의 출력 파일이 다른 rule의 입력 파일이면 자동으로 순서가 결정된다. 예를 들어 trim rule의 출력이 align rule의 입력이면, Snakemake는 반드시 trim을 먼저 실행한다.

DAG의 “비순환(Acyclic)”은 의존 관계가 원형으로 돌지 않는다는 뜻이다. $A \rightarrow B \rightarrow C \rightarrow A$ 처럼 순환하면 무한 루프에 빠지므로, Snakemake는 이를 자동으로 감지하여 에러를 발생시킨다.

expand() 함수

expand() 함수는 와일드카드를 구체적인 값으로 확장하는 함수이다. 예를 들어 `expand("results/{sample}.txt", sample=["A", "B", "C"])`는 `["results/A.txt", "results/B.txt", "results/C.txt"]`로 펼쳐진다. 주로 rule all에서 최종 목표 파일 목록을 지정할 때 사용한다.

Conda 환경 통합

생명정보학 파이프라인에서는 서로 다른 도구가 서로 다른 의존성을 요구하는 경우가 많다. STAR는 C++ 라이브러리가 필요하고, DESeq2는 R과 Bioconductor가 필요하다. 이 두 환경을 하나의 Conda 환경에 넣으면 충돌이 발생할 수 있다.

Snakemake는 각 rule에 독립적인 Conda 환경을 지정할 수 있다. `--use-conda` 플래그를 붙여 실행하면, Snake-make가 각 rule에 지정된 YAML 파일을 읽어 자동으로 Conda 환경을 생성하고 활성화한다. 환경 정의가 코드와 함께 버전 관리되므로, 1년 후에 파이프라인을 다시 실행해도 동일한 버전의 도구가 설치된다.

Config 파일

분석 파라미터를 Snakefile과 분리하여 관리할 수 있다. Snakefile은 “어떻게 분석할 것인가”를, config 파일은 “무엇을 분석할 것인가”를 담당한다. 새로운 프로젝트에서 같은 파이프라인을 사용하고 싶다면, Snakefile은 그대로 두고 `config.yaml`만 수정하면 된다.

로그 파일

각 rule에 `log` 지시자를 추가하면 실행 로그를 별도로 저장할 수 있다. 50개 샘플을 처리하다가 `sample_37`에서 에러가 났다면, `logs/align/sample_37.log`만 확인하면 된다. 에러 발생 시 원인 파악이 훨씬 쉬워진다.

6.4 Claude Code로 Snakefile 작성하기

Snakemake의 문법을 완벽하게 익히는 것보다, 분석 파이프라인의 흐름을 이해하고 시에게 정확히 설명하는 것이 더 중요하다. rule, wildcard, DAG, expand, Conda 환경 등 핵심 개념을 이해하고 있으면, Claude Code에게 정확한 지시를 내릴 수 있다.

RNA-seq 파이프라인 생성

다음과 같이 요청한다:

“RNA-seq 파이프라인을 Snakemake로 만들어줘. 입력은 data/ 폴더의 FASTQ 파일이고, FastQC → Trim Galore → STAR 정렬 → featureCounts 순서로 처리해줘. 샘플 목록은 config.yaml에서 읽어오게 하고, 각 단계마다 로그 파일을 남겨줘.”

이 요청을 하려면 다음을 알아야 한다:

- **분석 단계의 순서:** 왜 트리밍 다음에 정렬을 하는지 (어댑터가 남아 있으면 정렬 정확도가 떨어진다)
- **각 도구의 역할:** FastQC는 품질 확인, Trim Galore는 어댑터 제거, STAR는 게놈 정렬, featureCounts는 유전자별 발현량 계산
- **입출력 파일 형식:** FASTQ(시퀀싱 원본) → trimmed FASTQ → BAM(정렬 결과) → counts(발현량 테이블)

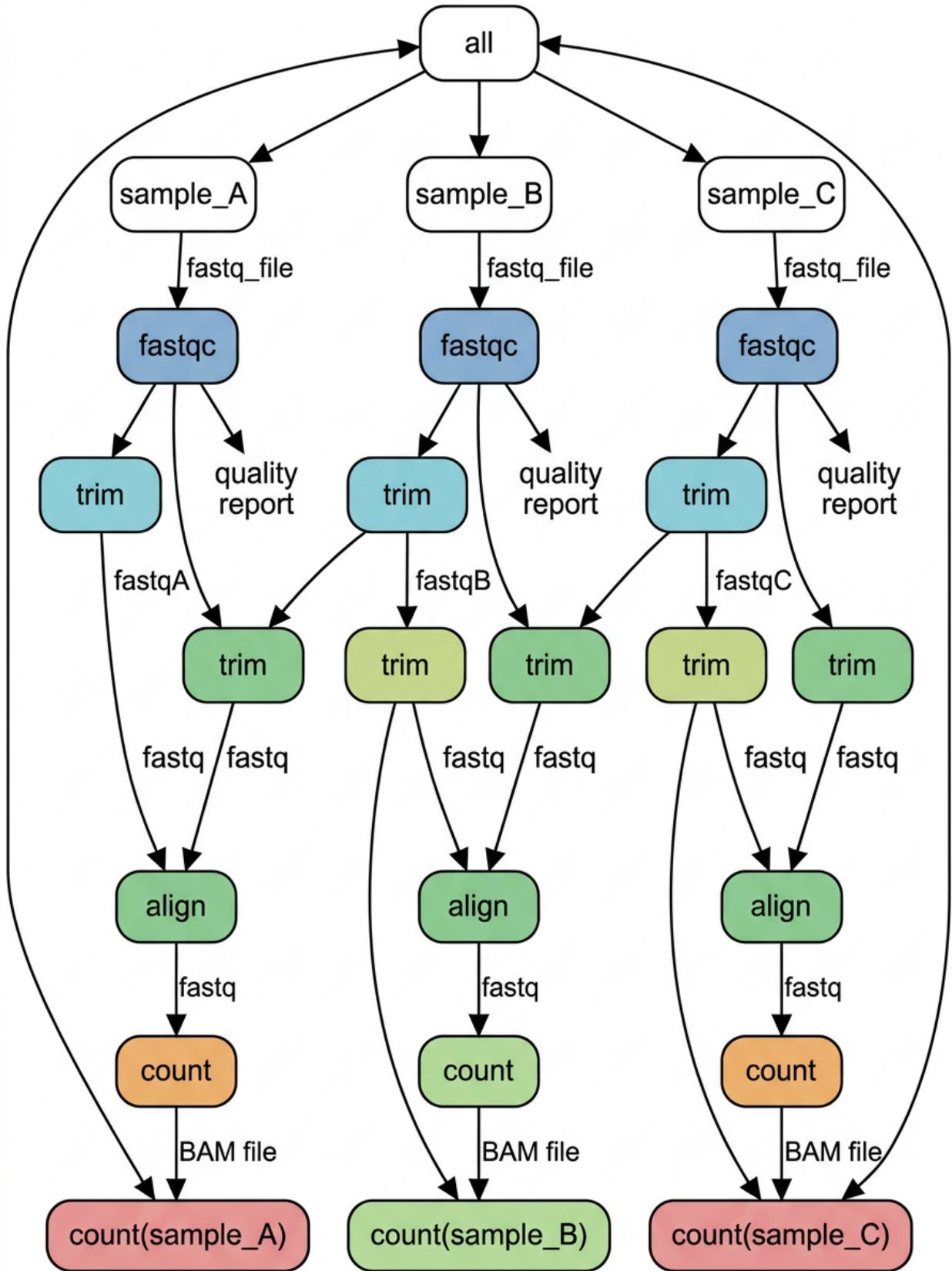


Figure 47: Snakemake DAG 시각화 예시

반면, `expand()` 함수의 정확한 문법이나 `threads` 지시자의 사용법은 몰라도 된다. AI가 올바른 Snakemake 문법으로 작성해 준다.

Conda 환경 분리 요청

“각 rule에 독립적인 Conda 환경을 설정해줘. STAR와 featureCounts는 같은 환경을 쓰고, DESeq2는 R 환경을 별도로 만들어줘. `envs/` 폴더에 YAML 파일을 생성해줘.”

Config 파일 분리

“하드코딩된 샘플 목록과 파라미터를 `config.yaml`로 분리해줘. 샘플 이름, 레퍼런스 게놈 경로, 트리밍 품질 기준값을 `config`에서 읽어오게 바꿔줘.”

파이프라인 확장

“count rule 다음에 DESeq2로 차등 발현 분석하는 rule을 추가해줘. R 스크립트는 별도 파일로 분리하고, Conda 환경도 만들어줘.”

“align rule에서 STAR 대신 HISAT2를 사용하도록 바꿔줘.”

“sample_D가 추가됐으니까 `config.yaml`에 반영해줘.”

실행과 디버깅

Snakemake를 실행할 때는 항상 드라이런(-n)으로 먼저 계획을 확인하는 것이 좋다. Claude Code에게 다음과 같이 요청할 수 있다:

“Snakemake 드라이런을 실행해서 결과를 보여줘.”

“Snakemake를 4코어로 실행해줘.”

“DAG 그래프를 생성해줘.”

에러가 발생하면 로그 파일을 함께 보여주면서 디버깅을 요청한다:

“snakemake 실행하면 align 단계에서 에러가 나. `logs/align/sample_A.log` 보고 원인 알려줘.”

로그 파일을 저장하는 설정이 되어 있어야 하므로, 처음 Snakefile을 만들 때 반드시 log 지시자를 포함하도록 요청하는 것이 좋다.

6.5 단일세포 분석 파이프라인

Scanpy를 이용한 단일세포 RNA-seq 분석도 Snakemake로 자동화할 수 있다. 단일세포 분석은 QC → 정규화 → 클러스터링이라는 순서를 따르므로, 각 단계를 rule로 분리하면 된다.

Claude Code에게 다음과 같이 요청한다:

“Scanpy 기반 단일세포 RNA-seq 분석을 Snakemake 파이프라인으로 만들어줘. QC, 정규화, 클러스터링을 각각 별도 rule로 분리하고, 각 단계의 결과를 `h5ad` 파일로 저장해줘. 여러 샘플에 대해 동시에 실행할 수 있게 와일드카드를 사용해줘.”

Snakemake에서는 `shell` 대신 `script` 지시자를 사용하면 Python이나 R 스크립트를 직접 실행할 수 있다. 스크립트 내에서 `snakemake.input[0]`, `snakemake.output[0]`으로 입출력 파일 경로를 참조할 수 있어, 스크립트를 Snakemake와 자연스럽게 연동할 수 있다.

주피터 노트북에서 프로토타이핑한 분석을 재현 가능한 파이프라인으로 전환할 때 특히 유용하다. AI에게 기존 노트북 코드를 전달하면서 “이걸 Snakemake로 변환해줘”라고 요청하면, rule 단위로 분리된 Snakefile과 스크립트 파일을 생성해 준다.

```

$ snakemake -n
Job counts:
count  jobs
-----
align  3
count  3
fastqc  3
trim   3
total  12

Job counts:
Rules to be executed:
rule fastqc: input: data/sample_A.fastq.gz
rule fastqc: input: data/sample_B.fastq.gz
rule fastqc: input: data/sample_C.fastq.gz
rule trim: input: data/sample_A.fastq.gz
rule trim: input: data/sample_B.fastq.gz
rule trim: input: data/sample_C.fastq.gz
rule align: input: trimmed/sample_A.fastq.gz
rule align: input: trimmed/sample_B.fastq.gz
rule align: input: trimmed/sample_C.fastq.gz
rule count: input: bam/sample_A.bam
rule count: input: bam/sample_B.bam
rule count: input: bam/sample_C.bam

12 jobs total
This was a dry-run. No jobs will be executed.
$ -

```

Figure 48: snakemake -n 드라이런 결과

6.6 정리

- **Snakemake**: 생명정보학 분석 파이프라인을 자동화하는 워크플로우 관리 도구
 - 재현성, 자동 의존성 관리, 병렬 실행, 부분 재실행 지원
- **Rule**: 입력 → 처리 → 출력을 정의하는 기본 단위
- **Wildcard**: {sample} 같은 패턴으로 여러 샘플에 동일 규칙 적용
- **DAG**: rule 간 의존 관계를 자동으로 파악하여 실행 순서 결정
- **Conda 환경 통합**: 각 rule마다 독립적인 환경을 지정하여 의존성 충돌 방지
- **Config 파일**: 분석 파라미터를 Snakefile과 분리하여 재사용성 확보
- **바이브 코딩의 핵심**: 분석 파이프라인의 흐름과 각 도구의 역할을 이해하고, AI에게 단계별로 지시하는 것. expand() 문법보다 “왜 트리밍 후에 정렬하는지”를 아는 것이 더 중요하다

7장. 생명정보 웹 툴 제작 기초

7.1 웹 도구, 아직 만들어야 하는가?

생명정보학에서 웹 도구는 오랫동안 접근성의 관문 역할을 해왔다. NCBI BLAST, Ensembl Genome Browser, Galaxy, UCSC Genome Browser — 이 도구들은 명령줄에 익숙하지 않은 실험 생물학자도 시퀀스 검색, 게놈 탐색, 분석 파이프라인 실행을 할 수 있게 만들었다. 웹 도구가 없었다면, 생명정보학은 소수의 전문 프로그래머만 접근할 수 있는 분야로 남았을 것이다.

그러나 이 장에서 웹 도구를 만들기 전에, 한 가지 질문을 던져야 한다. AI 시대에도 웹 도구를 만드는 것이 여전히 의미가 있는가?

웹(World Wide Web)은 30년 넘게 정보 공유의 기본 플랫폼이었다. 그러나 최근 **Web 4.0** — AI가 주도하는 지능형 웹 — 에 대한 논의가 활발해지면서, 전통적인 웹 인터페이스의 미래에 의문이 제기되고 있다. 이미 인터넷 트래픽의 상당 부분이 사람이 아닌 봇에 의해 발생하고 있으며, 이 비중은 계속 증가하고 있다. MCP와 같은 프로토콜이 보편화되면, AI 에이전트가 웹 브라우저를 거치지 않고 데이터와 서비스에 직접 접근하는 시대가 올 것이다. 그때 화려한 UI와 반응형 디자인은 더 이상 중요하지 않을 수 있다.

그럼에도 불구하고, 웹 도구를 배우는 것에는 몇 가지 현실적인 이유가 있다:

- **시각화와 인터랙션:** BLAST 검색 결과를 표로 보여주고, 시퀀스 정렬을 시각화하고, 파라미터를 조정하며 분석을 반복하는 작업은 여전히 GUI가 효과적이다. 터미널에서 텍스트로만 작업하는 것보다 직관적이다.
- **공유와 접근성:** 웹 도구가 생명정보학의 접근성을 높여온 것처럼, 직접 만든 분석 도구도 “이 URL을 열어보세요” 한마디로 동료와 공유할 수 있다.
- **풀스택 이해:** 프론트엔드, 백엔드, 데이터베이스, API를 아우르는 웹 개발 경험은 MCP 서버를 만들거나 AI 에이전트의 도구를 설계할 때도 직접적으로 활용된다.
- **바이브 코딩 실습:** 웹 개발은 AI 에이전트와 협업하는 훈련장으로 적합하다. 결과물이 눈에 보이므로 AI가 생성한 코드의 품질을 즉시 판단할 수 있다.

결론적으로, 웹 인터페이스 자체의 중요성은 점차 줄어들겠지만, **웹 도구를 만드는 과정에서 익히는 기술과 사고방식**은 AI 시대에도 유효하다. 이 장에서는 생명정보학 웹 도구의 기반이 되는 프로젝트를 구성한다.

7.2 이 책에서 사용하는 기술 스택

이 책에서는 다음과 같은 기술 스택을 사용하여 생명정보학 웹 도구를 개발한다.

기술	역할	설명
SvelteKit	프론트엔드 프레임워크	빠르고 가벼운 웹 앱 개발 프레임워크
Tailwind CSS	CSS 프레임워크	유틸리티 클래스 기반의 스타일링 도구
PostgreSQL	데이터베이스	오픈소스 관계형 데이터베이스
Docker	컨테이너	개발 환경 통합 관리

이 조합을 선택한 이유는 바이브 코딩에 적합하기 때문이다. SvelteKit은 코드량이 적어 AI가 생성하고 수정하기 쉽고, Tailwind CSS는 스타일 정보가 HTML과 같은 파일에 있어 AI가 맥락을 파악하기 용이하다. PostgreSQL은 안정적이고 널리 사용되는 데이터베이스이므로 AI가 정확한 SQL을 생성할 확률이 높다.

SvelteKit

SvelteKit은 Svelte를 기반으로 한 풀스택 웹 프레임워크이다. “풀스택”이란 프론트엔드(사용자가 보는 화면)와 백엔드(서버 로직, 데이터베이스 연동)를 하나의 프레임워크에서 모두 처리할 수 있다는 뜻이다.

React, Vue 등 다른 프레임워크와 비교했을 때 다음과 같은 장점이 있다:

- **컴파일 타임 최적화:** React는 브라우저에서 가상 DOM을 비교하며 화면을 갱신하지만, Svelte는 빌드 시점에 최적화된 JavaScript로 변환된다. 덕분에 런타임 오버헤드가 없어 매우 빠르다.
- **적은 코드량:** 동일한 기능을 React보다 30~40% 적은 코드로 구현할 수 있다. 이는 AI가 생성하는 코드량이 줄어든다는 뜻이기도 하다. 코드가 짧으면 에러가 날 가능성도 줄어든다.
- **서버 사이드 렌더링(SSR) 기본 지원:** 페이지를 서버에서 미리 렌더링하여 브라우저에 전달하므로, 첫 화면 로딩이 빠르다.
- **파일 기반 라우팅:** 파일 구조가 곧 URL 구조이다. `src/routes/tools/blast/+page.svelte` 파일을 만들면 자동으로 `/tools/blast` URL이 생긴다. URL 설정을 위한 별도 파일이 필요 없다.

SvelteKit에서 페이지는 크게 세 가지 파일로 구성된다:

파일	역할	실행 위치
<code>+page.svelte</code>	사용자에게 보이는 화면 (HTML + CSS + JS)	브라우저
<code>+page.server.ts</code>	데이터 로딩, 폼 처리 등 서버 로직	서버
<code>+server.ts</code>	REST API 엔드포인트	서버

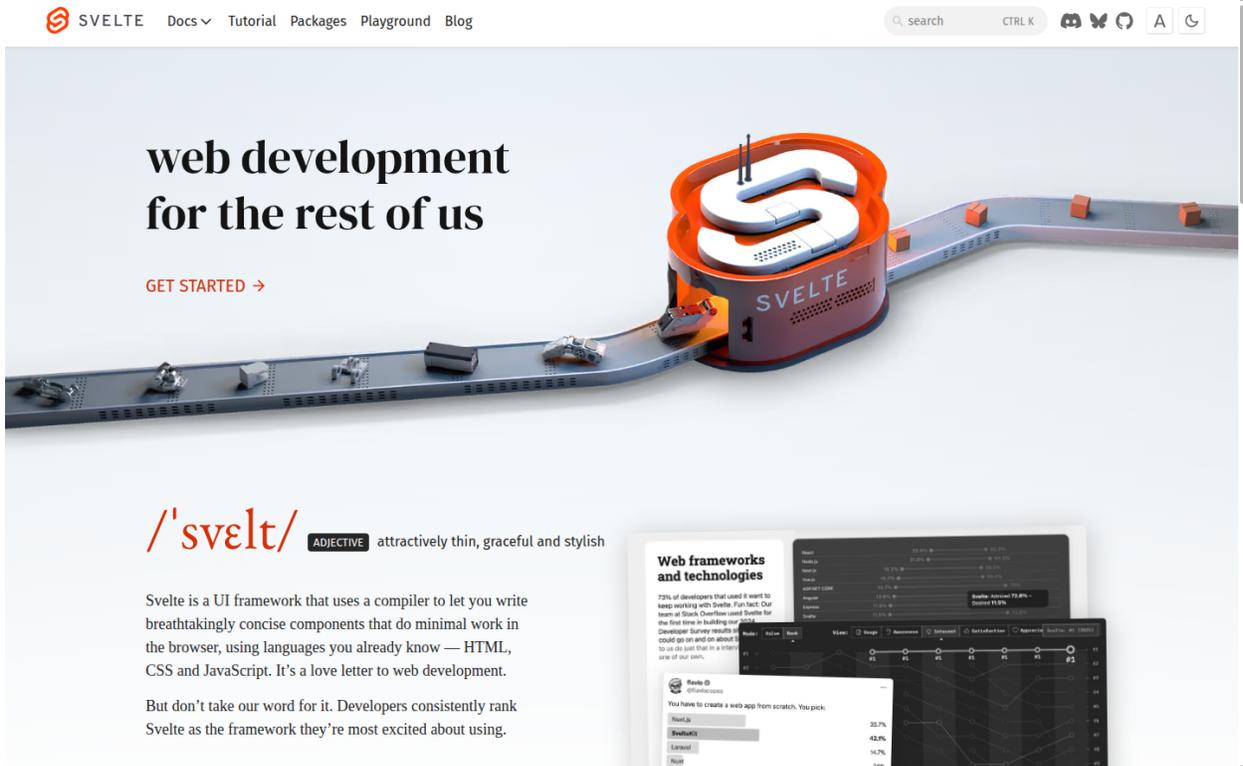


Figure 49: SvelteKit 공식 웹사이트

예를 들어 BLAST 검색 도구를 만든다면, `+page.svelte`에는 시퀀스 입력 폼과 결과 테이블을, `+page.server.ts`에는 BLAST+ 명령 실행과 결과 파싱 로직을 작성한다. 이 구분을 이해하고 있으면, AI에게 “서버 로직에 BLAST 검색 기능 추가해줘”라고 구체적으로 요청할 수 있다.

Tailwind CSS

Tailwind CSS는 유틸리티 클래스 기반의 CSS 프레임워크이다. 전통적인 CSS에서는 별도의 스타일시트 파일을 만들고 클래스 이름을 정의한 후 스타일을 작성한다. Tailwind는 이 과정을 생략하고, 미리 정의된 유틸리티 클래스를 HTML 요소에 직접 적용한다.

```

<!-- CSS -->
<div class="card"> </div>

<!-- Tailwind CSS -->
<div class="bg-white rounded-lg shadow-md p-6"> </div>

```

Tailwind 방식에서 `bg-white`는 흰색 배경, `rounded-lg`는 둥근 모서리, `shadow-md`는 중간 크기 그림자, `p-6`은 패딩 1.5rem을 의미한다. 클래스 이름만 보면 어떤 스타일이 적용되는지 알 수 있다.

AI 에이전트와 함께 사용하기에 특히 적합한데, 스타일이 HTML과 같은 파일에 있어 컨텍스트를 파악하기 쉽기 때문이다. 전통적인 CSS에서는 AI가 HTML 파일과 CSS 파일을 모두 읽어야 전체 디자인을 이해할 수 있지만, Tailwind에서는 HTML 파일 하나만 읽으면 된다.

PostgreSQL

PostgreSQL은 세계에서 가장 많이 사용되는 오픈소스 관계형 데이터베이스이다. “관계형”이란 데이터를 표(테이블) 형태로 저장하고, 테이블 간의 관계를 정의할 수 있다는 뜻이다.

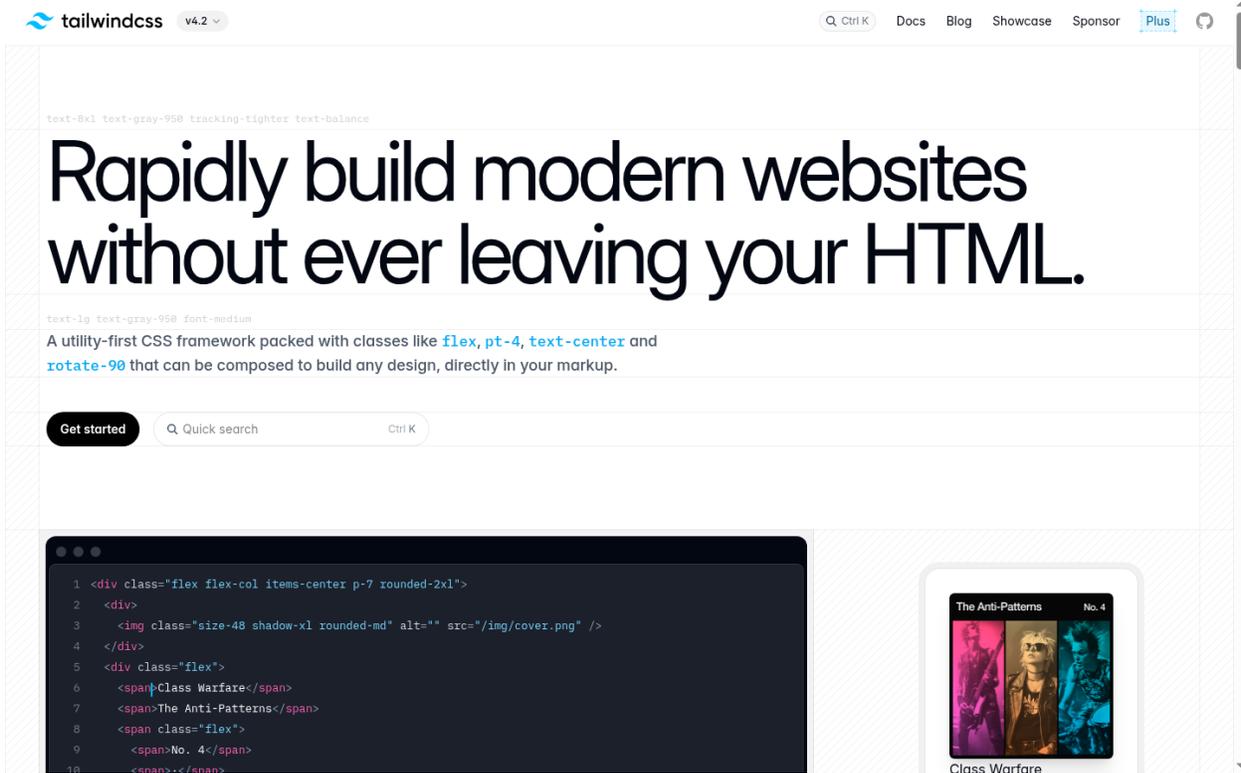


Figure 50: Tailwind CSS 공식 웹사이트

이 책에서 PostgreSQL은 다음과 같은 용도로 사용된다: - 분석 결과 저장: BLAST 검색 결과, 차등 발현 분석 결과 등 - 사용자 데이터 관리: 사용자가 업로드한 시퀀스 정보 - 작업 이력 추적: 누가 언제 어떤 분석을 실행했는지

SQLite 같은 가벼운 데이터베이스도 있지만, PostgreSQL을 선택한 이유는 실무 환경과 동일한 경험을 제공하기 위해서이다. Docker를 통해 실행하므로 설치가 번거롭지도 않다.

7.3 프로젝트 생성

Node.js 설치

SvelteKit은 JavaScript/TypeScript 기반이므로 Node.js가 필요하다. Node.js는 브라우저 밖에서 JavaScript를 실행할 수 있게 해주는 런타임이다. 다음 웹사이트에서 설치한다:

<https://nodejs.org/en/download>

설치 시 운영체제를 선택하는 화면이 나오는데, **Windows 사용자도 WSL 환경에서 개발하므로 Linux를 선택**해야 한다. 그다음 **nvm(Node Version Manager)**과 **pnpm(패키지 매니저)**을 선택한 후 터미널에 표시되는 명령을 입력한다. nvm은 여러 버전의 Node.js를 설치하고 전환할 수 있게 해주는 도구이다. 프로젝트마다 다른 Node.js 버전이 필요할 수 있으므로, 직접 설치보다 nvm을 통한 설치가 권장된다.

pnpm은 npm의 대안으로, 디스크 공간을 절약하고 설치 속도가 빠르다. 여러 프로젝트에서 같은 패키지를 사용할 때, pnpm은 패키지를 한 번만 다운로드하고 심볼릭 링크로 연결한다.

설치 확인:

```
node --version
pnpm --version
```

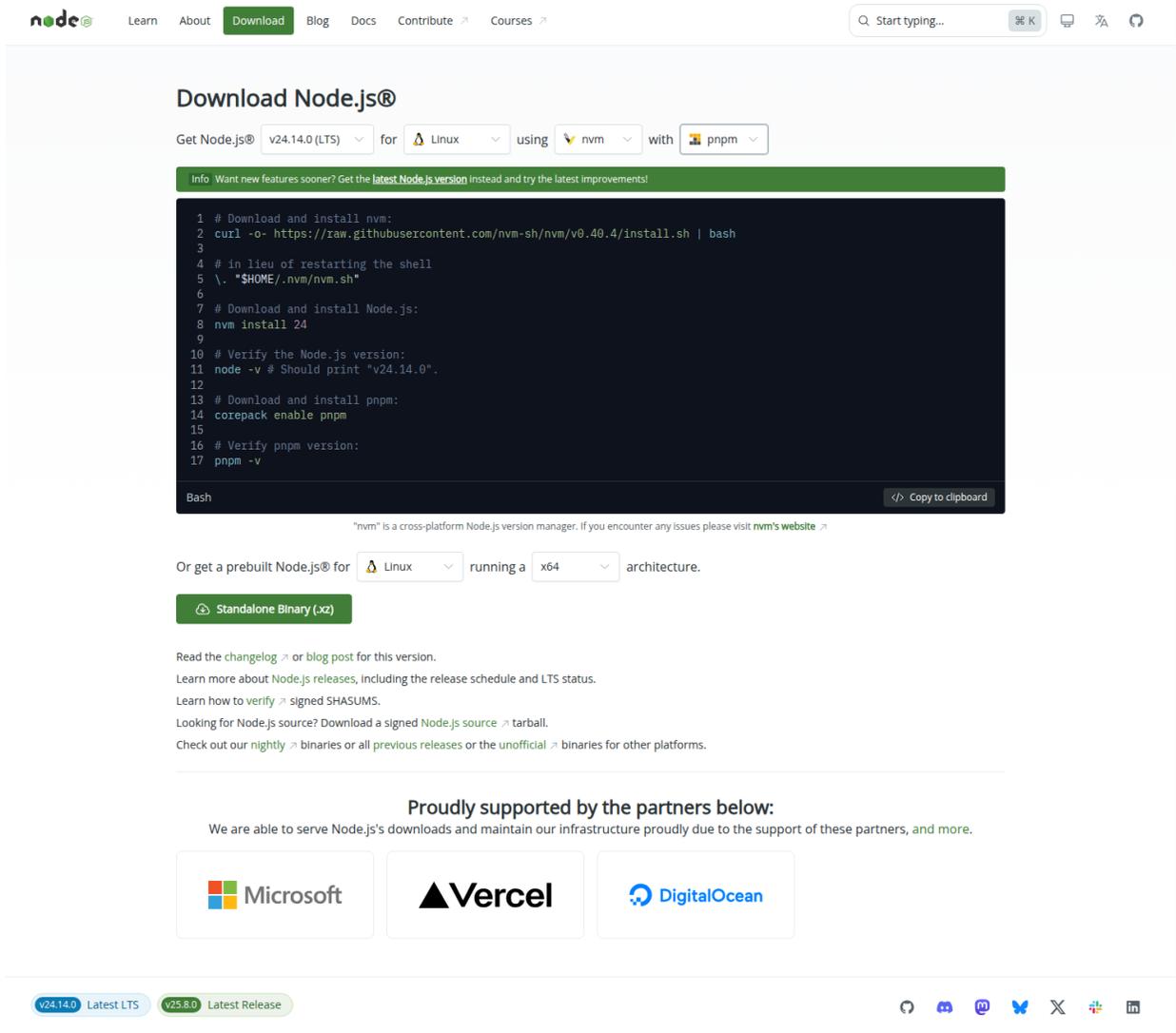


Figure 51: Node.js 다운로드 페이지

SvelteKit 프로젝트 초기화

터미널에서 다음 명령을 실행하여 SvelteKit 프로젝트를 생성한다:

```
npx sv create my-bioinfo-app
```

CLI가 대화형으로 옵션을 물어본다. 방향키와 스페이스바로 선택하고 엔터로 진행한다:

질문	선택
Which template would you like?	SvelteKit minimal
Add type checking with TypeScript?	Yes, using TypeScript syntax
What would you like to add to your project?	tailwindcss, sveltekit-adapter, paraglide
tailwindcss: Which plugins?	typography, forms
sveltekit-adapter: Which adapter?	node
paraglide: Which languages?	en, ko
paraglide: Include a demo?	No
Which package manager?	pnpm

프로젝트 생성이 완료되면, VS Code에서 **파일** → **폴더 열기**로 my-bioinfo-app 디렉토리를 연다.

각 옵션의 역할을 간단히 정리하면:

- **TypeScript**: JavaScript에 타입 시스템을 추가한 언어. AI가 생성하는 코드에서도 타입이 있으면 잘못된 데이터 사용을 컴파일 시점에 잡아낼 수 있다.
- **Tailwind CSS**: 유틸리티 클래스 기반 CSS 프레임워크. Typography 플러그인은 마크다운 렌더링 시 보기 좋은 기본 스타일을, Forms 플러그인은 폼 요소에 기본 스타일을 적용한다.
- **sveltekit-adapter (node)**: SvelteKit 앱을 Node.js 서버로 배포할 수 있게 해주는 어댑터.
- **Paraglide**: 다국어(i18n) 지원 라이브러리. 영어(en)와 한국어(ko)를 지원하도록 설정한다.

참고: 프로젝트 초기 생성은 AI 에이전트에 맡기기보다 직접 수행하는 것이 좋다. AI는 초기화 명령을 사용하기보다 기존 코드를 직접 작성하려는 경향이 있어, 최신 버전이 아닌 코드를 생성할 수 있다. 프로젝트 뼈대를 공식 CLI로 먼저 만들고, 그 위에서 AI에게 기능 추가를 요청하는 것이 안전하다.

Tailwind CSS

Tailwind CSS는 SvelteKit 프로젝트 생성 시 옵션으로 함께 설치할 수 있다. 별도로 수동 설치할 필요 없이, 프로젝트 초기화 과정에서 Tailwind CSS를 선택하면 자동으로 설정된다. Typography 플러그인은 마크다운 렌더링 시 보기 좋은 기본 스타일을 제공하고, Forms 플러그인은 폼 요소(입력 필드, 버튼 등)에 기본 스타일을 적용해 준다.

7.4 Docker 환경 구성

compose.yml과 Dockerfile 생성

3장에서 배운 Docker Compose를 실전에 적용한다. Claude Code에게 다음과 같이 요청한다:

“SvelteKit 앱과 PostgreSQL 데이터베이스를 Docker Compose로 구성해줘. compose.yml과 compose.dev.yml을 만들어줘. compose.yml은 배포용이고, compose.dev.yml은 개발용으로 볼륨 마운트와 핫 리로드를 설정해줘. Dockerfile도 함께 만들어줘.”

이때 **compose.yml**(배포용)과 **compose.dev.yml**(개발용)을 분리하는 것이 좋다. 개발 환경에서는 코드 변경이 즉시 반영되어야 하지만, 배포 환경에서는 최적화된 빌드 결과만 실행하면 된다. 두 파일을 분리해 두면, 개발할 때는 `docker compose -f compose.dev.yml up`으로, 배포할 때는 `docker compose up`으로 실행할 수 있다.

Claude Code가 생성하는 파일의 주요 설정을 이해해 두면, 결과를 검토하고 수정 요청을 내리기 쉽다:

설정	의미
volumes: - ./app /app/node_modules	로컬 코드를 컨테이너에 연결. 코드 수정 시 즉시 반영 컨테이너 자체의 node_modules 사용. 호스트와 충돌 방지
depends_on: - db	db를 먼저 시작한 후 app 시작
pgdata:/var/lib/postgresql/data DATABASE_URL	컨테이너 삭제해도 데이터베이스 보존 컨테이너 간 네트워크 주소. db는 서비스 이름

Dockerfile에서는 3장에서 배운 레이어 캐싱 원리가 적용된다. package.json을 먼저 복사하고 의존성을 설치한 뒤 소스 코드를 복사하면, 소스 코드만 변경되었을 때 의존성 설치 단계를 건너뛸 수 있다. --host 플래그는 Docker 컨테이너 내부 서버에 호스트에서 접속하기 위해 필요하다.

7.5 환경 변수 (.env)

프로젝트에서 데이터베이스 접속 정보, API 키 등 민감한 설정값은 **환경 변수**로 관리한다. 코드에 비밀번호를 직접 쓰면 Git에 커밋될 위험이 있다. 환경 변수를 사용하면 코드와 설정을 분리할 수 있다.

중요: .env 파일은 반드시 사용자가 직접 작성해야 한다. .env에는 데이터베이스 비밀번호, API 키 등 민감한 정보가 포함되므로, Claude Code가 이 파일을 읽거나 생성하게 해서는 안 된다. 글로벌 CLAUDE.md(~/claude/CLAUDE.md)에 다음과 같이 추가하여, AI 에이전트가 .env 파일에 접근하지 못하도록 설정하는 것을 권장한다:

```
#
- .env
- .env.example
```

프로젝트 루트에 .env 파일을 직접 생성한다:

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/bioinfo
PUBLIC_SITE_NAME=My Bioinfo App
```

SvelteKit에서 환경 변수는 두 가지로 나뉜다:

접두사	접근 범위	용도
PUBLIC_ (없음)	서버 + 클라이언트 서버만	사이트 이름 등 공개 정보 데이터베이스 비밀번호, API 키 등 민감 정보

PUBLIC_ 접두사가 없는 환경 변수는 서버 코드(+page.server.ts, +server.ts)에서만 접근할 수 있다. 브라우저에서 실행되는 코드에서는 읽을 수 없으므로, 데이터베이스 비밀번호나 API 키가 사용자에게 노출되지 않는다. 이 구분은 보안상 중요하다.

.env 파일은 .gitignore에 반드시 추가하여 Git에 커밋되지 않도록 한다. 대신 .env.example 파일을 만들어 어떤 환경 변수가 필요한지 안내한다. .env.example에는 실제 값 대신 템플릿을 넣는다. 이 파일은 Claude Code에게 생성을 요청해도 된다.

“.env.example 파일을 만들어줘. DATABASE_URL과 PUBLIC_SITE_NAME이 필요해.”

7.6 프로젝트 디렉토리 구조

최종적으로 프로젝트 디렉토리는 다음과 같은 구조를 가진다:

```
my-bioinfo-app/
  compose.yml      # Docker
  Dockerfile       # Docker
```

```

.env # (Git )
.env.example #
.gitignore
package.json
pnpm-lock.yaml
svelte.config.js # SvelteKit
vite.config.ts # Vite
tailwind.config.ts # Tailwind CSS
CLAUDE.md # AI
README.md
src/
  app.css # (Tailwind )
  app.html # HTML
  lib/ #
    components/ #
    server/ # (DB )
  routes/ #
    +layout.svelte #
    +page.svelte #
    tools/
      +page.svelte #
  static/ # ( , )

```

이 구조에서 가장 중요한 폴더는 `src/routes/`이다. SvelteKit의 파일 기반 라우팅에 의해, 이 폴더의 구조가 곧 웹사이트의 URL 구조가 된다. `src/routes/tools/blast/+page.svelte`를 만들면 `/tools/blast`라는 URL이 자동으로 생긴다. AI에게 새 페이지를 추가해 달라고 할 때, 이 규칙을 알면 “tools 폴더 아래에 blast 페이지를 만들어줘”라고 구체적으로 지시할 수 있다.

`src/lib/` 폴더는 여러 페이지에서 공유하는 코드를 넣는 곳이다. 예를 들어 네비게이션 바 컴포넌트는 모든 페이지에서 사용하므로 `src/lib/components/NavBar.svelte`에 둔다. `src/lib/server/`에는 데이터베이스 연결 코드처럼 서버에서만 사용하는 코드를 둔다.

CLAUDE.md 작성

CLAUDE.md는 AI 에이전트가 참조하는 프로젝트 명세 파일이다. Claude Code는 작업을 시작할 때 이 파일을 가장 먼저 읽고, 프로젝트의 구조와 규칙을 파악한다.

핵심 원칙: CLAUDE.md에 정보를 많이 넣을수록 AI는 더 똑똑해진다.

AI 코딩 에이전트는 사람과 달리 프로젝트의 맥락을 스스로 알지 못한다. “Navbar에 로고를 추가해줘”라고 요청했을 때, AI가 올바른 위치에 올바른 방식으로 코드를 작성하려면 다음을 알아야 한다:

- 이 프로젝트가 SvelteKit을 사용하는지, React를 사용하는지
- Navbar 컴포넌트가 어디에 있는지
- 스타일링에 Tailwind CSS를 쓰는지, 일반 CSS를 쓰는지
- 로고 이미지는 어디에 저장되어 있는지

이러한 정보가 CLAUDE.md에 없으면 AI는 매번 프로젝트 구조를 탐색하고 추측해야 한다. 반면, 이 정보가 잘 정리되어 있으면 AI는 곧바로 정확한 코드를 작성할 수 있다.

왜 배경지식이 필요한가? 바이브 코딩은 “AI가 다 해주니까 개발을 몰라도 된다”는 뜻이 아니다. 오히려 그 반대이다. CLAUDE.md를 잘 작성하려면 사람이 먼저 프로젝트를 이해하고 있어야 한다.

예를 들어 이런 CLAUDE.md를 작성한다고 하자:

```
#
```

```

#
- SvelteKit ( + )
- Tailwind CSS ( )
- PostgreSQL ( )

#
- src/routes/ : ( )
- src/lib/components/ : UI
- src/lib/server/ : (DB )

#
- PascalCase (: NavBar.svelte)
- API +server.ts
- $env/static/private $env/static/public

#
- : blue-600 primary color
- : max-w-7xl,
- : mobile-first

```

이 파일을 작성하려면 SvelteKit의 파일 기반 라우팅이 무엇인지, Tailwind의 유틸리티 클래스가 어떻게 동작하는지, 컴포넌트와 레이아웃의 개념이 무엇인지 알아야 한다. **이 책에서 배우는 웹 개발 배경지식이 바로 이를 위한 것이다.**

핵심: 바이브 코딩에서 사람의 역할은 코드를 직접 작성하는 것이 아니라, **시가 올바른 코드를 작성할 수 있도록 정확한 지시를 내리는 것이다.** 정확한 지시를 내리려면 기본적인 개발 개념을 이해하고 있어야 한다.

CLAUDE.md에 포함할 내용

항목	설명	예시
프로젝트 개요	프로젝트의 목적과 대상 사용자	“생명정보학 연구자를 위한 웹 기반 시퀀스 분석 도구”
기술 스택	사용하는 프레임워크, 라이브러리	“SvelteKit, Tailwind CSS, PostgreSQL”
디렉토리 구조	주요 폴더의 역할	“src/routes/는 페이지, src/lib/은 공유 코드”
코딩 컨벤션	파일명 규칙, 코드 스타일	“컴포넌트는 PascalCase, 변수는 camelCase”
디자인 가이드라인	색상, 폰트, 레이아웃 규칙	“primary color는 blue-600, 최대 너비 max-w-7xl”
비즈니스 로직	도메인 특화 규칙	“FASTA 형식은 >로 시작하는 헤더 + 시퀀스”

CLAUDE.md는 한 번 작성하고 끝이 아니다. 프로젝트가 발전할수록 새로운 규칙과 패턴이 생기기 마련이다. 새로운 컴포넌트 규칙이 추가되거나, 디자인 가이드라인이 변경되면 CLAUDE.md도 함께 업데이트해야 한다. AI 에이전트가 항상 최신 상태를 파악할 수 있도록 유지하는 것이 바이브 코딩에서 사람의 중요한 역할이다.

팁: Claude Code에게 “지금까지 작업한 내용을 바탕으로 CLAUDE.md를 업데이트해줘”라고 요청하면, 시가 현재 프로젝트 상태를 분석하여 CLAUDE.md를 자동으로 갱신해 주기도 한다.

7.7 개발 서버 실행

Docker를 사용하는 경우

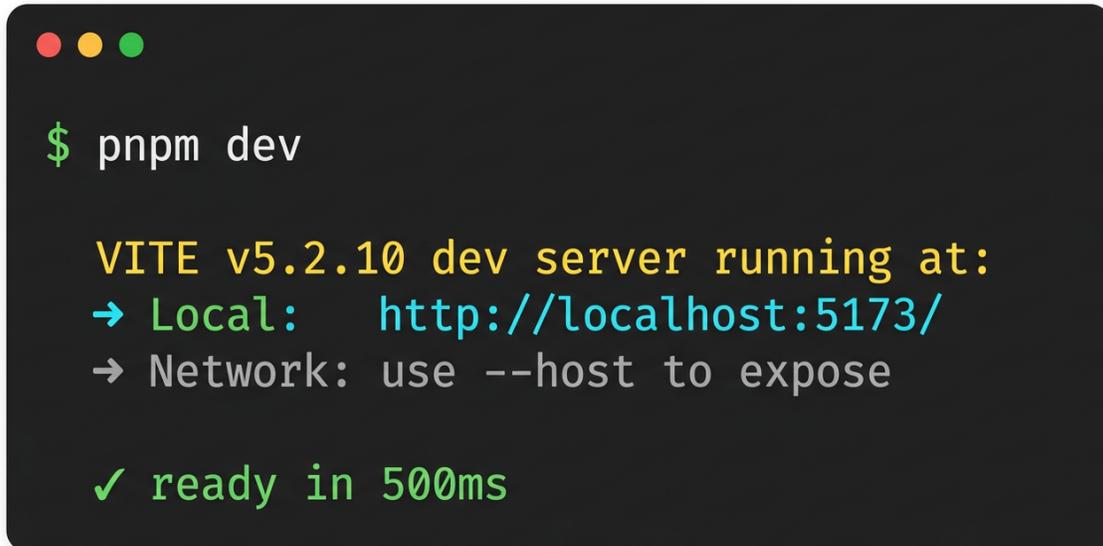
```
docker compose up
```

브라우저에서 `http://localhost:5173`으로 접속하면 SvelteKit 앱을 확인할 수 있다. 최초 실행 시 Docker 이미지 빌드와 의존성 설치에 몇 분이 걸릴 수 있지만, 두 번째 실행부터는 캐시 덕분에 수 초 만에 시작된다.

Docker 없이 로컬에서 실행하는 경우

```
pnpm dev
```

Docker 없이도 개발 서버를 실행할 수 있지만, 이 경우 PostgreSQL을 별도로 설치하고 실행해야 한다. Docker를 사용하면 `docker compose up` 한 줄로 웹 서버와 데이터베이스가 모두 시작되므로 더 편리하다.



```
$ pnpm dev

VITE v5.2.10 dev server running at:
→ Local:   http://localhost:5173/
→ Network: use --host to expose

✓ ready in 500ms
```

Figure 52: SvelteKit 개발 서버 실행 화면

7.8 정리

- SvelteKit + Tailwind CSS + PostgreSQL이 이 책의 기본 기술 스택
 - SvelteKit: 코드량이 적고 파일 기반 라우팅을 지원하는 풀스택 프레임워크
 - Tailwind CSS: HTML과 스타일이 같은 파일에 있어 시가 맥락을 파악하기 쉬운 CSS 프레임워크
 - PostgreSQL: Docker로 간편하게 실행하는 오픈소스 관계형 데이터베이스
- 프로젝트 초기화는 직접 수행하고, 이후 기능 추가는 시에게 맡기기
 - `pnpm create svelte@latest`로 뼈대 생성 후 시와 협업
- Docker Compose로 개발 환경을 통합 관리
 - `compose.yml(배포용)`과 `compose.dev.yml(개발용)`을 분리
 - Dockerfile, `compose.yml` 생성은 Claude Code에게 요청
- 환경 변수(.env)는 반드시 사용자가 직접 작성
 - Claude Code가 .env를 읽거나 생성하게 하지 않기
 - 글로벌 CLAUDE.md에 .env 접근 금지 규칙 추가
 - PUBLIC_ 접두사로 공개/비공개 구분, .gitignore에 반드시 추가
- CLAUDE.md에 프로젝트 명세를 작성하여 시 에이전트 활용도를 높이기
 - 정보가 많을수록 시가 정확한 코드를 생성
 - 프로젝트 발전에 따라 지속적으로 업데이트

8장. 랜딩 페이지 디자인

8.1 랜딩 페이지란?

랜딩 페이지(Landing Page)는 웹 사이트의 **가장 첫 페이지**이다. 방문자가 URL에 접속했을 때 가장 먼저 보게 되는 화면으로, 웹 사이트의 첫인상을 결정한다. 좋은 랜딩 페이지는 방문자에게 “이 사이트가 무엇을 하는 곳인지”를 몇 초 만에 전달하고, “더 알아보고 싶다” 또는 “사용해 보고 싶다”는 마음이 들게 만든다.

생명정보학 웹 도구의 랜딩 페이지라면, 어떤 분석 도구를 제공하는지 한눈에 보여주고, 연구자가 바로 사용을 시작할 수 있도록 안내해야 한다. NCBI BLAST 웹사이트나 Galaxy 플랫폼의 첫 화면을 떠올려 보면, 모두 핵심 기능을 전면 배치하고 간결한 설명과 시작 버튼을 제공한다.

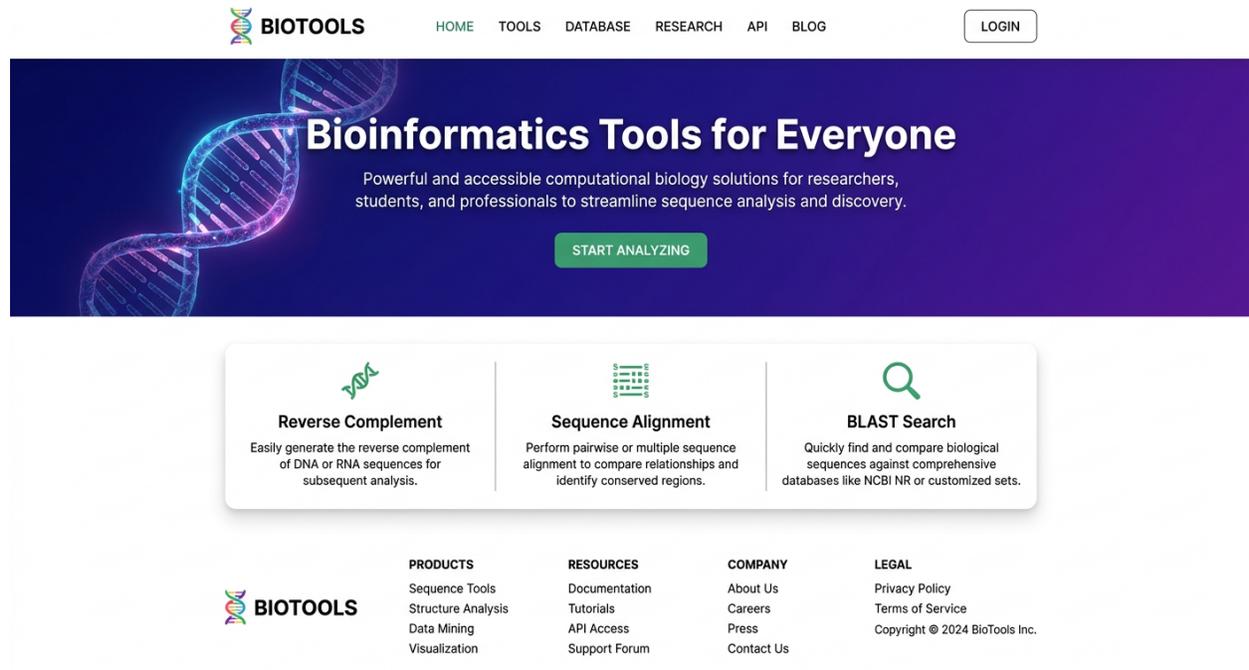


Figure 53: 잘 디자인된 랜딩 페이지 예시

8.2 랜딩 페이지의 구조

Header, Body, Footer

웹 페이지는 크게 세 영역으로 구성된다. 이 구조는 거의 모든 웹사이트에서 공통적으로 사용되는 표준 패턴이다. 책의 표지, 본문, 뒤표지와 비슷하다고 생각하면 된다.

영역	포함 요소
Header	로고, 메인 메뉴(Navigation Bar), Hero 섹션
Body	주 콘텐츠 (기능 소개, 특징, 사용 방법 등)
Footer	로고(주로 흑백), 하단 메뉴, 연락처, 주소, 법적 정보, 저작권 정보 등

실제 구현 시 각각을 별도의 컴포넌트 파일로 나누어서 구현한다. Header는 Header.svelte, Footer는 Footer.svelte처럼 분리하면, 여러 페이지에서 같은 컴포넌트를 재사용할 수 있다.

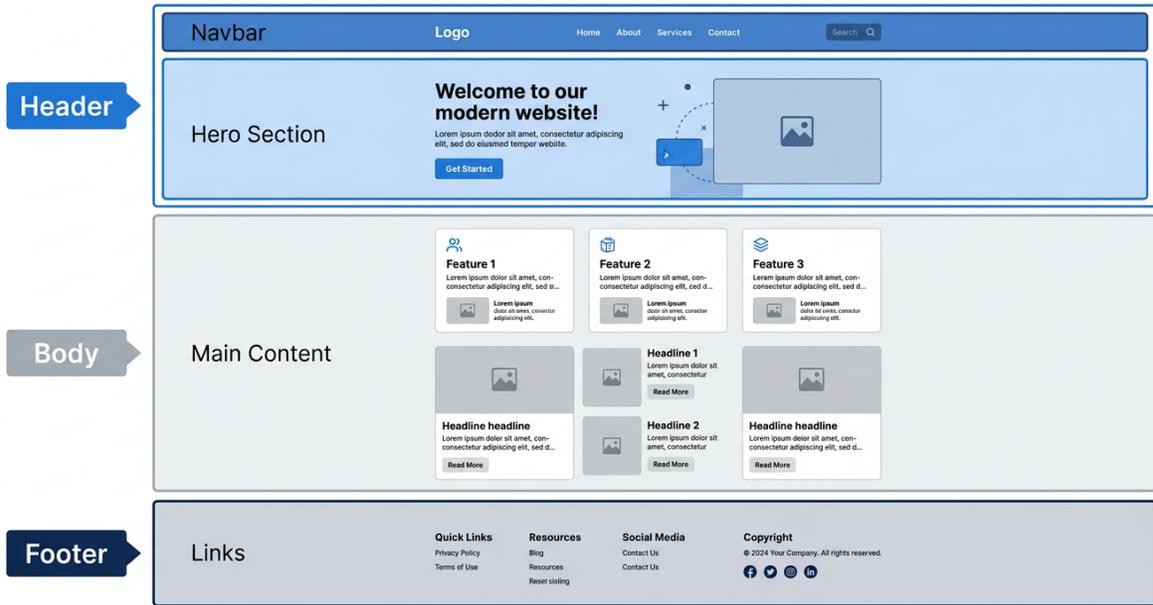


Figure 54: 웹 페이지 구조

Navigation Bar (Navbar)

Navigation Bar는 웹 사이트의 상단에 위치하는 메뉴이다. 로고와 주요 페이지 링크를 포함하며, 사용자가 사이트 내를 이동할 수 있게 한다. Navbar는 보통 화면 상단에 고정(sticky)되어 스크롤을 내려도 항상 보이도록 구현한다.

Navbar 디자인에서 고려할 점은 다음과 같다: - 로고: 좌측에 배치하는 것이 일반적이다 - 메뉴 항목: Home, Tools, About, Contact 등 주요 섹션 링크 - 반응형 디자인: 모바일 화면에서는 햄버거 메뉴(≡)로 접히는 구조가 일반적이다 - 시각적 구분: 현재 페이지의 메뉴 항목을 강조하여 사용자가 어디에 있는지 알려준다

Hero Section

Hero Section은 랜딩 페이지의 가장 눈에 띄는 첫 번째 영역이다. “Hero”라는 이름은 이 영역이 페이지의 “주인공” 역할을 한다는 데서 유래했다. 화면의 좌우 전체 너비를 사용하며, 다음 요소들을 포함한다:

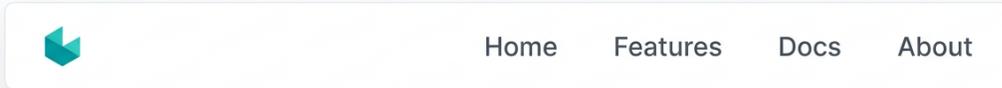
- **헤드라인:** 사이트의 핵심 가치를 한 문장으로 전달. “Sequence Analysis Made Simple”처럼 무엇을 할 수 있는지 명확하게 표현한다.
- **서브 헤드라인:** 헤드라인을 보충하는 설명. “AI-powered bioinformatics tools for researchers”처럼 대상과 방법을 구체화한다.
- **CTA(Call to Action) 버튼:** 사용자가 취해야 할 행동을 유도하는 버튼이다. “시작하기”, “도구 사용하기” 등의 문구를 사용한다. CTA 버튼은 눈에 잘 띄는 색상으로 만들고, 클릭하면 도구 페이지나 회원가입 페이지로 이동하게 한다.
- **배경 이미지 또는 일러스트:** 시각적 임팩트를 주는 요소. DNA 이중나선, 분자 구조 등 생명정보학과 관련된 이미지를 사용하면 사이트의 성격을 직관적으로 전달할 수 있다.

참고: Hero Section 디자인 패턴에 대한 자세한 내용은 <https://www.awebco.com/blog/hero-section/> 에서 확인할 수 있다.

Carousel

Carousel(캐러셀)은 여러 이미지나 콘텐츠를 슬라이드 형태로 보여주는 요소이다. 여러 기능이나 특징을 순차적으로 소개할 때 유용하다. 좌우 화살표로 넘기거나, 자동으로 일정 시간 간격으로 전환되도록 설정할 수 있다.

A. Minimal



B. Dark/CTA



C. Transparent



Figure 55: Navbar 디자인 예시

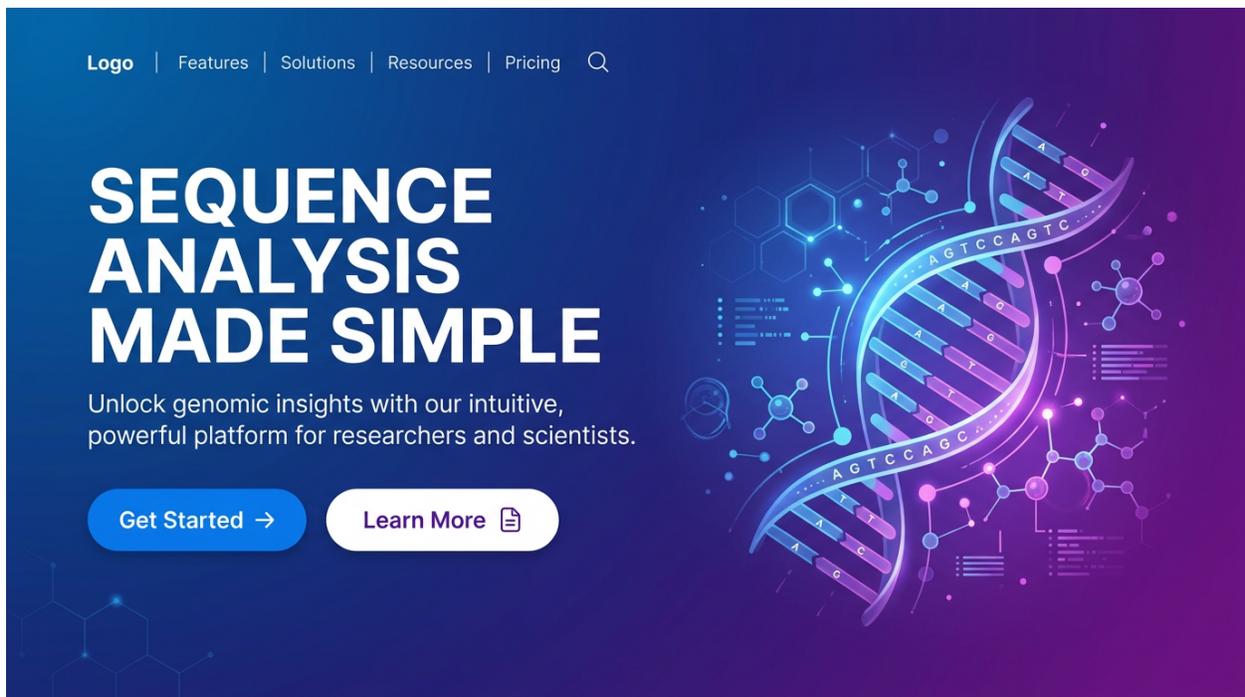


Figure 56: Hero Section 예시

다만, 캐러셀은 사용자가 원하는 정보를 놓칠 수 있다는 단점이 있다. 자동 슬라이드가 빠르면 내용을 다 읽기 전에 넘어가고, 느리면 사용자가 기다려야 한다. 핵심 정보는 캐러셀에만 의존하지 말고, 별도 섹션에도 표시하는 것이 좋다.

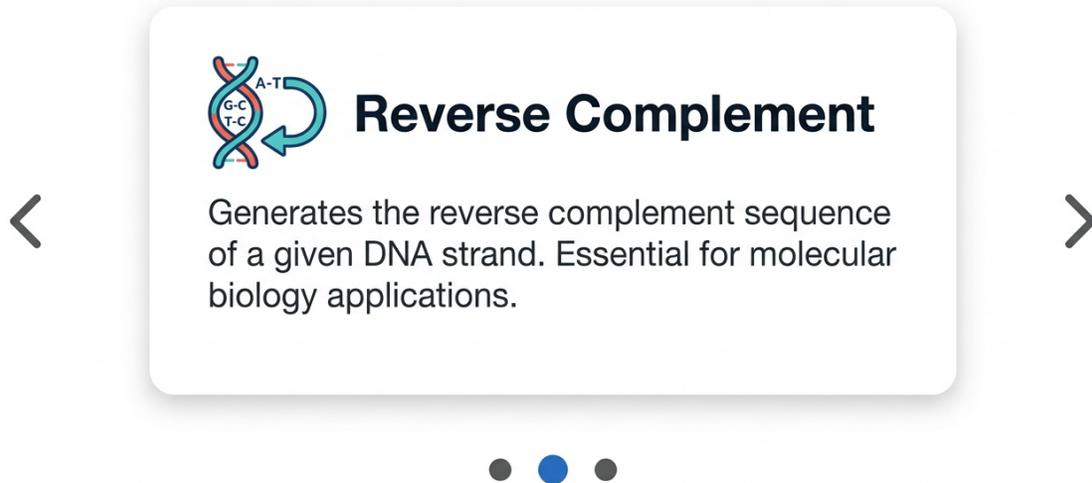


Figure 57: Carousel 컴포넌트 예시

Features Section

웹 도구의 주요 기능을 소개하는 영역이다. 보통 **카드(Card)** 형태로 3~4개의 핵심 기능을 나열한다. 각 카드에는 아이콘, 기능 이름, 간단한 설명을 포함한다.

생명정보학 웹 도구의 Features Section이라면 “Reverse Complement”, “Sequence Alignment”, “BLAST Search” 같은 핵심 도구를 카드로 보여줄 수 있다. 카드를 클릭하면 해당 도구 페이지로 이동하도록 구현하면, 사용자가 원하는 기능에 빠르게 접근할 수 있다.

8.3 UI 컴포넌트

자주 사용되는 웹 구성 요소를 패턴화한 것을 **UI 컴포넌트**라 한다. 2011년 등장한 트위터 부트스트랩 프레임워크(<https://getbootstrap.com>) 유래된 것들이 많다. 이 컴포넌트들의 이름을 아는 것이 중요한 이유는, 시에게 디자인을 요청할 때 정확한 이름을 사용하면 의도한 결과를 얻기 훨씬 쉽기 때문이다.

“상단에 메뉴 바를 만들어줘”보다 “상단에 Navbar를 만들어줘”가 더 정확한 결과를 낸다. “슬라이드 형태로 보여줘”보다 “Carousel로 보여줘”가 더 명확하다.

랜딩 페이지에서 자주 사용되는 컴포넌트:

컴포넌트	설명
Navbar	상단 네비게이션 바. 로고와 메뉴 항목을 포함
Hero	첫 화면의 대형 배너 영역. 헤드라인, CTA 버튼, 배경 이미지 포함
Card	카드형 콘텐츠 블록. 아이콘, 제목, 설명을 묶어 표시

컴포넌트	설명
Carousel	슬라이드 형태의 콘텐츠. 좌우 화살표로 전환
Button	클릭 가능한 버튼. Primary(주요 행동), Secondary(보조 행동), Ghost(투명 배경) 등의 변형이 있다
Badge	작은 라벨/태그. “New”, “Beta” 같은 상태 표시에 사용
Footer	하단 정보 영역. 저작권, 링크, 연락처 포함

이 외에도 Modal(팝업 대화 상자), Tooltip(마우스를 올리면 나타나는 설명), Accordion(접었다 펼 수 있는 목록) 등 다양한 컴포넌트가 있다. 이들의 정확한 이름을 알수록 시에게 원하는 디자인을 정확하게 요청할 수 있다.

8.4 AI를 활용한 디자인 목업 생성

이미지 생성 AI를 활용한 디자인

Google Gemini, ChatGPT(DALL-E), Midjourney 등의 AI를 통해 디자인 목업을 생성할 수 있다. 앞서 배운 Header, Body, Footer 및 컴포넌트 개념을 활용하여 프롬프트를 구체적으로 작성하면 더 정확한 결과를 얻을 수 있다.

기본 프롬프트 예시:

생명정보학 웹 도구의 랜딩 페이지를 디자인해줘. 상단에는 로고와 Navbar, 큰 Hero Section에는 DNA 관련 이미지와 “Sequence Analysis Made Simple” 헤드라인, “시작하기” 버튼이 있고, 아래에는 3개의 기능 카드가 있는 디자인.

여러 번 생성해보고 마음에 드는 것을 선택한다. AI가 생성하는 디자인은 매번 다르므로, 3~5개 정도 생성한 뒤 가장 적합한 것을 고르는 것이 좋다.

더 구체적인 프롬프트 예시:

Header: 좌측에 로고, 우측에 Home/Tools/About/Contact 메뉴가 있는 Navbar. Hero Section: 배경은 그라데이션, 좌측에 헤드라인 “Bioinformatics Tools for Everyone”, 서브라인 “AI-powered sequence analysis”, CTA 버튼 “Get Started”. 우측에는 DNA 구조 일러스트. Features: 3개의 Card — Reverse Complement, Sequence Alignment, BLAST Search. Footer: 로고(흑백), 링크 목록, 저작권 정보.

프롬프트가 구체적일수록 원하는 결과에 가까운 디자인이 나온다. 여기서 “Navbar”, “Hero Section”, “CTA 버튼”, “Card” 같은 용어를 사용하는 것이 핵심이다. 이 용어들은 웹 디자인의 공통 언어이므로, AI가 정확하게 해석할 수 있다.

Claude Code의 design 스킵

Claude Code에는 **design 스킵**이 내장되어 있어, 디자인 목업 이미지를 기반으로 실제 코드를 생성할 수 있다. 디자인 목업 이미지를 Claude Code 채팅창에 복사 붙여넣기한 뒤 구현을 요청하면 된다.

이 디자인 목업을 참고하여 랜딩 페이지를 SvelteKit + Tailwind CSS로 구현해줘. (이미지를 복사 붙여넣기)

Claude Code는 이미지를 분석하여 레이아웃, 색상, 컴포넌트 구조를 파악하고 코드를 생성한다. 완벽하게 동일한 결과를 기대하기보다는, 전체적인 레이아웃과 구조를 잡는 출발점으로 활용하는 것이 좋다. 생성된 코드를 기반으로 “Hero Section의 배경색을 좀 더 진하게 바꿔줘”, “카드의 간격을 넓혀줘” 같은 세부 조정을 요청하면 된다.

8.5 SvelteKit에서 랜딩 페이지 구현

레이아웃 구성

SvelteKit에서는 `src/routes/+layout.svelte` 파일이 모든 페이지에 공통으로 적용되는 레이아웃을 정의한다. Header와 Footer를 여기에 배치하면, 모든 페이지에 자동으로 같은 Header와 Footer가 표시된다. 새 페이지를 추가할 때마다 Header와 Footer를 다시 작성할 필요가 없다.

Our Tools



Sequence Tools

Powerful tools for analyzing, aligning, and manipulating DNA and protein sequences. Manage genetic data effectively.



Data Visualization

Create dynamic graphs, charts, and figures from complex datasets. Visualize experimental results clearly.



BLAST Search

Perform rapid Sequence Alignment using the BLAST algorithm. Identify homologous sequences in comprehensive databases.

Figure 58: Features Section 예시

1

LOGO Services Solutions Tools Resources Login

Data Analysis Platform

Empowering research with advanced genomic data insights & analysis.

Explore Platform

Sequencing

Empowering research with advanced genomic data insights & analysis.

Visualization

Empowering research with advanced genomic data insights & analysis.

Reporting

Empowering research with advanced genomic data insights & analysis.

2

BioHub BioHub Research Collaboration Data About Join

Life Sciences Hub

Integrated platform for biomedical research & discovery.

Join Hub

Genomics

Integrated platform for Biomedical Research Insights & analysis.

Proteomics

Integrated platform for Biomedical Research Insights & analysis.

Biomarker ID

Integrated platform for Biomedical Research Insights & analysis.

3

AnalyticsSolutions Overview Technology Case Studies Pricing Contact

Analytics Solution

Transforming biological data into actionable insights.

Get Started

Data Management

Transforming biological data into actionable insights.

ML Insights

Transforming biological data into actionable insights.

Predictive Models

Transforming biological data into actionable insights.

4

BioscienceNext Platform Research Innovation Careers Login

New Era of Bioscience

Innovating drug discovery and genomic research.

Learn More

AI Research

Innovating drug discovery and genomic research.

Cloud Computing

Innovating drug discovery and genomic research.

Genomic Data

Innovating drug discovery and genomic research.

Figure 59: Gemini 디자인 목업

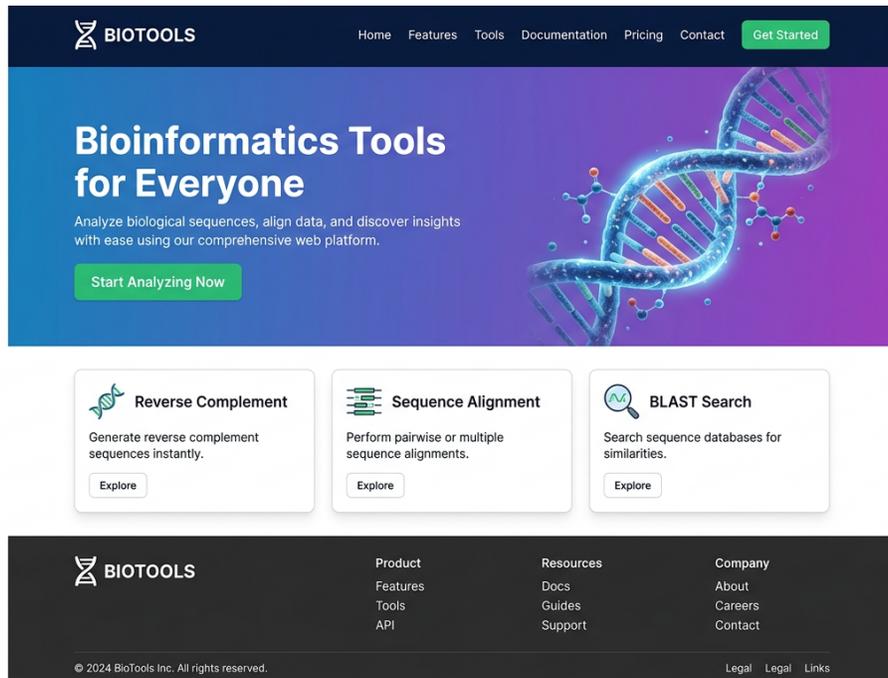


Figure 60: 상세 디자인 목업

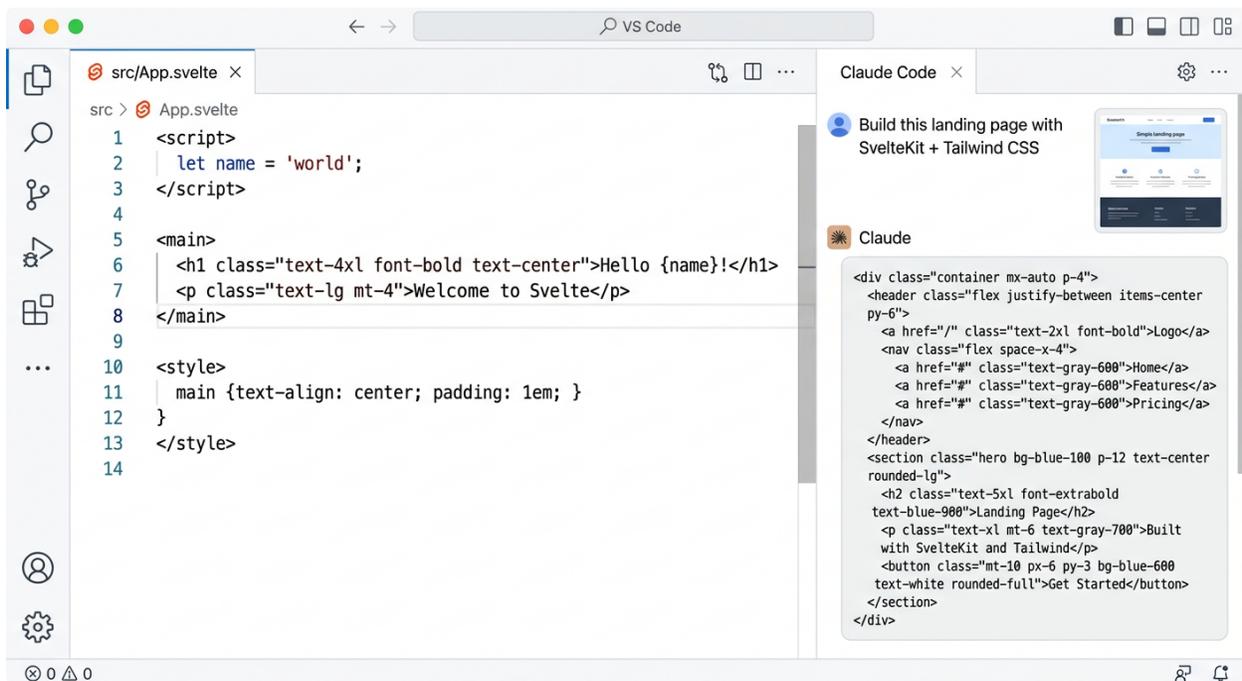


Figure 61: Claude 디자인 요청

```

<!-- src/routes/+layout.svelte -->
<script>
  import Header from '$lib/components/Header.svelte';
  import Footer from '$lib/components/Footer.svelte';
  import './app.css';
</script>

<Header />
<main>
  <slot />
</main>
<Footer />

```

<slot />은 각 페이지의 고유 콘텐츠가 들어가는 자리이다. / URL로 접속하면 +page.svelte의 내용이 <slot /> 위치에 삽입되고, /tools URL로 접속하면 tools/+page.svelte의 내용이 삽입된다. Header와 Footer는 어떤 페이지를 열든 항상 동일하게 유지된다.

\$lib은 SvelteKit에서 src/lib/ 디렉토리를 가리키는 별칭(alias)이다. \$lib/components/Header.svelte라고 쓰면 src/lib/components/Header.svelte 파일을 가져온다.

컴포넌트 분리

각 UI 요소를 별도의 Svelte 컴포넌트로 분리하여 src/lib/components/ 디렉토리에 저장한다:

```

src/lib/components/
  Header.svelte      # Navbar
  Footer.svelte     #
  Hero.svelte       # Hero
  FeatureCard.svelte #
  Carousel.svelte  # ( )

```

컴포넌트를 분리하면 여러 이점이 있다. 먼저 재사용이 가능하다. FeatureCard 컴포넌트를 한 번 만들면, 랜딩 페이지뿐 아니라 다른 페이지에서도 사용할 수 있다. 또한 유지보수가 쉽다. Footer를 수정하고 싶으면 Footer.svelte 파일 하나만 수정하면 모든 페이지에 반영된다. 시에게 수정을 요청할 때도 “Footer.svelte의 저작권 연도를 2025로 바꿔줘”처럼 구체적으로 지시할 수 있다.

랜딩 페이지 조립

src/routes/+page.svelte에서 컴포넌트들을 조립하여 랜딩 페이지를 완성한다:

```

<!-- src/routes/+page.svelte -->
<script>
  import Hero from '$lib/components/Hero.svelte';
  import FeatureCard from '$lib/components/FeatureCard.svelte';
</script>

<Hero />

<section class="py-16 px-4 max-w-6xl mx-auto">
  <h2 class="text-3xl font-bold text-center mb-12"> </h2>
  <div class="grid grid-cols-1 md:grid-cols-3 gap-8">
    <FeatureCard
      title="Reverse Complement"
      description="DNA          ."
    />
    <FeatureCard

```

```

        title="Sequence Alignment"
        description="          ."
    />
    <FeatureCard
        title="BLAST Search"
        description="          ."
    />
</div>
</section>

```

여기서 Tailwind CSS 클래스의 의미를 간단히 살펴보면: - py-16: 위아래 패딩 4rem - max-w-6xl mx-auto: 최대 너비 72rem, 좌우 자동 여백으로 중앙 정렬 - grid grid-cols-1 md:grid-cols-3: 모바일에서는 1열, 태블릿 이상(md)에서는 3열 그리드 - gap-8: 그리드 항목 간 간격 2rem

md:grid-cols-3에서 md:는 Tailwind의 반응형 접두사이다. 화면 너비가 768px 이상일 때만 적용된다는 뜻이다. 이렇게 하면 모바일에서는 카드가 세로로 쌓이고, 태블릿이나 데스크톱에서는 가로로 나란히 표시된다. 이를 **모바일 퍼스트(mobile-first)** 접근이라 한다.

이런 Tailwind 클래스를 모두 외울 필요는 없다. AI에게 “카드를 모바일에서는 1열, 데스크톱에서는 3열로 배치해줘”라고 요청하면 적절한 클래스를 생성해 준다. 다만, “반응형”, “모바일 퍼스트”, “그리드” 같은 개념을 알아야 이런 요청을 할 수 있다.

8.6 정리

- 랜딩 페이지는 웹 사이트의 첫인상을 결정하는 중요한 페이지
 - Header(Navbar) + Hero Section + Features + Footer 구조
 - 몇 초 안에 사이트의 목적을 전달해야 한다
- UI 컴포넌트의 명칭과 역할을 이해하면 AI에게 더 정확한 지시가 가능
 - Navbar, Hero, Card, Carousel, Button, Badge, Footer, Modal, Tooltip 등
 - “메뉴 바”보다 “Navbar”, “슬라이드”보다 “Carousel”이 정확한 결과를 낸다
- 이미지 생성 AI로 디자인 목업을 만들고, Claude Code로 구현
 - 프롬프트에 컴포넌트 명칭을 사용하여 구체적으로 요청
 - Claude Code의 design 스킬로 이미지 기반 코드 생성 가능
- SvelteKit에서는 컴포넌트를 분리하여 레이아웃에 조립하는 방식으로 구현
 - +layout.svelte로 공통 Header/Footer 관리
 - \$lib/components/에 재사용 가능한 컴포넌트 분리
 - Tailwind CSS의 반응형 접두사(md:, lg:)로 다양한 화면 크기 대응

9장. 일반 페이지 디자인

9.1 일반 페이지란?

일반 페이지는 랜딩 페이지를 제외한 나머지 모든 페이지를 의미한다. 실제 웹 도구의 기능을 제공하는 핵심 페이지들이다. 랜딩 페이지가 “이 사이트는 이런 일을 합니다”를 보여주는 전사실이라면, 일반 페이지는 “여기서 실제로 작업하세요”라는 작업실이다.

랜딩 페이지의 화려한 Hero Section, Carousel 등의 디자인 요소를 축소하고, 모든 페이지에 공통적인 일관된 디자인을 적용한다. 일관성이 중요한 이유는, 사용자가 어떤 도구 페이지를 열든 동일한 레이아웃과 동일한 조작 방식을 기대할 수 있어야 하기 때문이다. 매번 다른 레이아웃이라면 사용자는 페이지를 열 때마다 “입력 창이 어디 있지?”를 찾아야 한다.

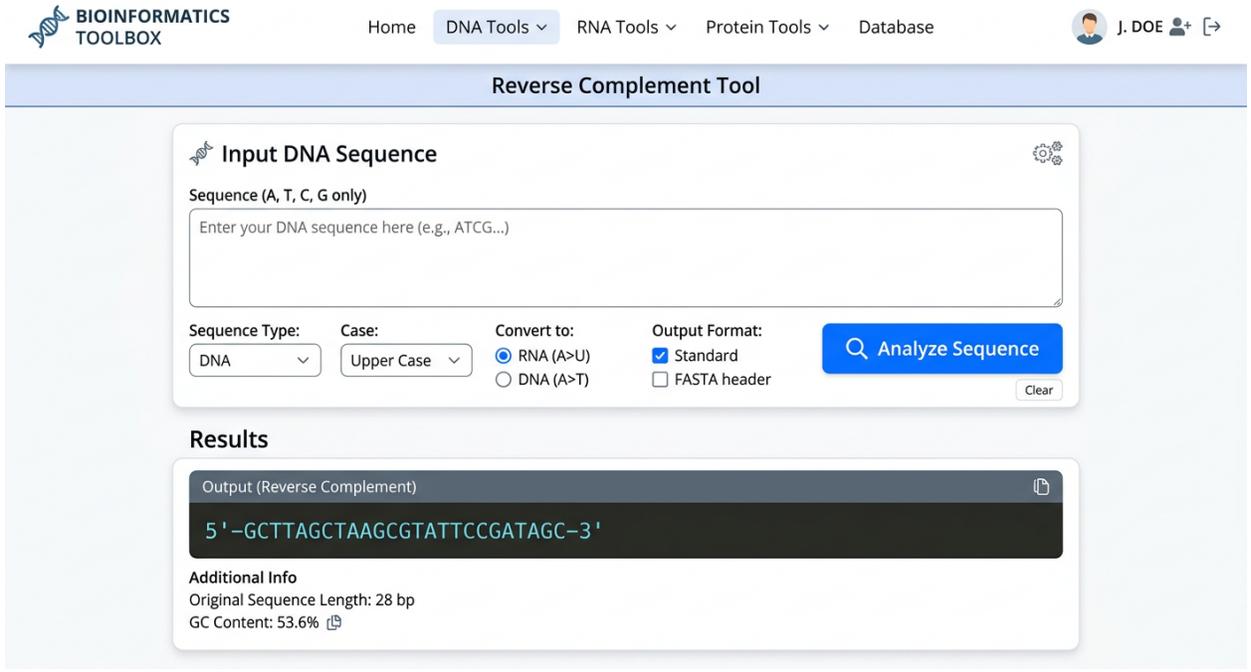


Figure 62: 일반 페이지 예시

9.2 일반 페이지의 구조

공통 레이아웃

일반 페이지는 랜딩 페이지와 동일한 Header(Navbar)와 Footer를 공유하되, Hero Section의 높이를 크게 줄여 간결한 형태로 사용한다. 랜딩 페이지의 Hero가 화면 좌우 전체 너비에 큰 높이를 차지하는 것과 달리, 일반 페이지의 Hero는 페이지 제목 정도만 표시하는 얇은 배너 형태이다.

이렇게 하면 공통 레이아웃(+layout.svelte)은 그대로 유지하면서, 각 페이지의 콘텐츠에 더 많은 화면 공간을 할애할 수 있다. 도구 페이지에서는 입력 품과 결과 표시 영역이 넓어야 하므로, Hero Section이 화면의 절반을 차지하면 불편하다.

영역	랜딩 페이지	일반 페이지
Header	Navbar + 큰 Hero Section	Navbar + 축소된 Hero (페이지 타이틀 배너)
Body	기능 소개, 특징 카드 등	실제 도구 인터페이스, 콘텐츠
Footer	동일	동일

Page Header (Breadcrumb 포함)

일반 페이지의 상단에는 현재 위치를 알려주는 **Breadcrumb**과 **페이지 제목(Heading)**을 배치한다. Breadcrumb은 “빵 부스러기”라는 뜻으로, 헨젤과 그레텔이 숲에서 길을 찾기 위해 빵 부스러기를 떨어뜨린 것에서 유래했다. 웹에서는 현재 페이지까지의 경로를 보여주는 네비게이션 요소이다.

Home > Tools > Reverse Complement

이 Breadcrumb을 보면 사용자는 “지금 Home 아래 Tools 아래 Reverse Complement 페이지에 있구나”를 즉시 알 수 있다. 각 항목은 클릭 가능한 링크로, “Tools”를 클릭하면 도구 목록 페이지로 이동한다.

Sidebar

도구가 여러 개이거나 설정 옵션이 많은 경우, **Sidebar**를 활용하여 네비게이션을 제공한다. Sidebar는 화면의 좌측(또는 우측)에 고정된 세로형 메뉴이다. 도구 목록을 Sidebar에 배치하면, 한 페이지에서 다른 도구로 빠르게 전환할 수 있다.

Sidebar는 데스크톱 화면에서는 항상 표시하고, 모바일에서는 햄버거 메뉴로 숨겼다가 필요할 때 펼치는 방식이 일반적이다. 시에게 “반응형 Sidebar를 만들어줘”라고 요청하면 이 동작을 구현해 준다.

9.3 도구 페이지에서 자주 사용되는 컴포넌트

컴포넌트	용도	예시
Checkbox	다중 선택	출력 옵션 선택 (정렬 포함, 통계 포함 등 여러 개 동시 선택 가능)
Radio	단일 선택	분석 모드 선택 (Nucleotide 또는 Protein 중 하나만 선택)
File Upload	파일 업로드	FASTA 파일, VCF 파일 등을 직접 업로드
Button Label	실행/제출 입력 필드 설명	“분석 시작”, “결과 다운로드” 각 입력 필드의 이름과 안내 문구

Checkbox와 Radio의 차이를 이해하면 UI에게 정확한 요청을 할 수 있다. Checkbox는 여러 개를 동시에 선택할 수 있고(출력 옵션처럼), Radio는 하나만 선택할 수 있다(분석 모드처럼). UI에게 “분석 모드를 선택하는 Radio 버튼 3개 만들어줘”라고 요청하면 의도에 맞는 UI가 나온다.

출력/결과 컴포넌트

분석 결과를 사용자에게 보여주는 컴포넌트들이다.

컴포넌트	용도	예시
Table	표 형태의 결과 표시	BLAST 검색 결과(Hit 목록), 통계 요약. 행과 열로 정리된 데이터를 보여줄 때 사용
Code Block	코드/시퀀스 표시	결과 시퀀스, 명령어 예시. 고정폭 폰트로 표시되어 시퀀스 정렬이 깔끔하게 보인다
Tab	여러 결과 뷰 전환	텍스트 뷰 / 시각화 뷰. 같은 데이터를 다른 형태로 볼 수 있게 한다
Spinner	로딩 상태 표시	분석 진행 중 표시. 시간이 걸리는 작업에서 사용자가 “멈춘 건가?” 하고 불안해하지 않도록
Alert / Toast	알림 메시지	성공(“분석 완료”) / 오류(“잘못된 시퀀스 형식”) 메시지
Progress Bar	진행률 표시	대용량 파일 처리 진행률. “50% 완료” 같은 시각적 피드백
Pagination	결과 페이지 이동	검색 결과가 수백 개일 때, 한 페이지에 20개씩 나누어 표시

Spinner와 Progress Bar는 사용자 경험(UX)에서 중요한 역할을 한다. BLAST 검색처럼 몇 초에서 몇 분이 걸리는 작업에서 아무런 피드백 없이 화면이 멈춰 있으면, 사용자는 “오류가 난 건가?” 하고 페이지를 새로고침하거나 떠나버린다. Spinner 하나만 있어도 “처리 중이구나”라는 신호를 줄 수 있다.

시각화 컴포넌트

생명정보학 데이터를 그래프나 특수한 형태로 보여주는 컴포넌트들이다.

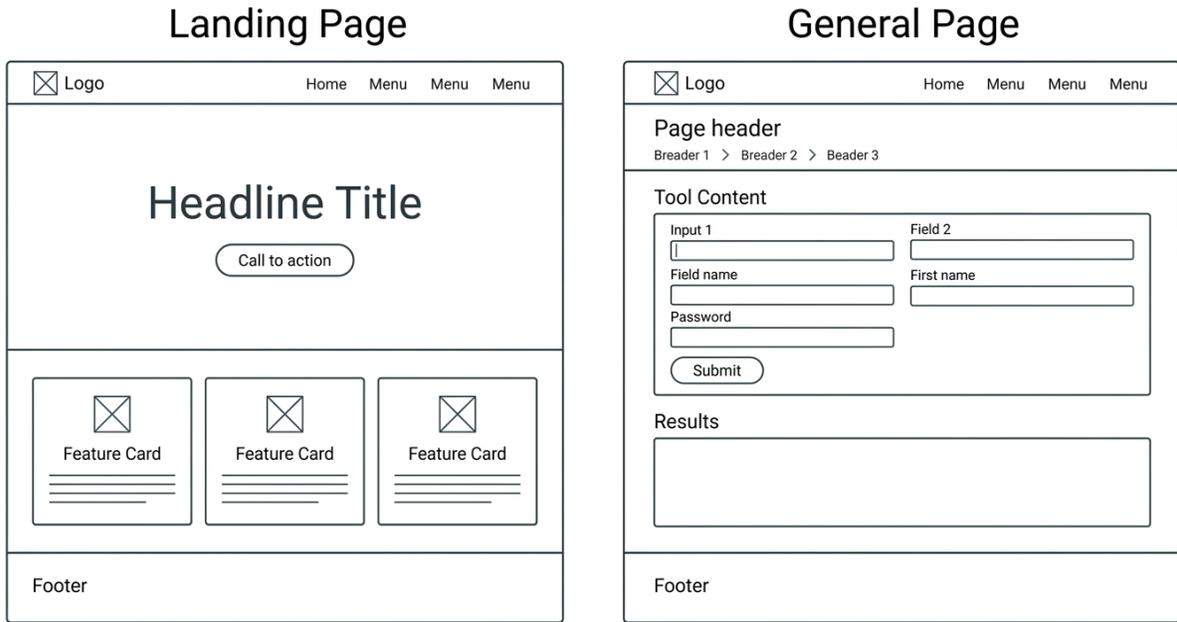


Figure 63: 레이아웃 비교

Home > Tools > Reverse Complement

Reverse Complement

Figure 64: 페이지 헤더와 Breadcrumb

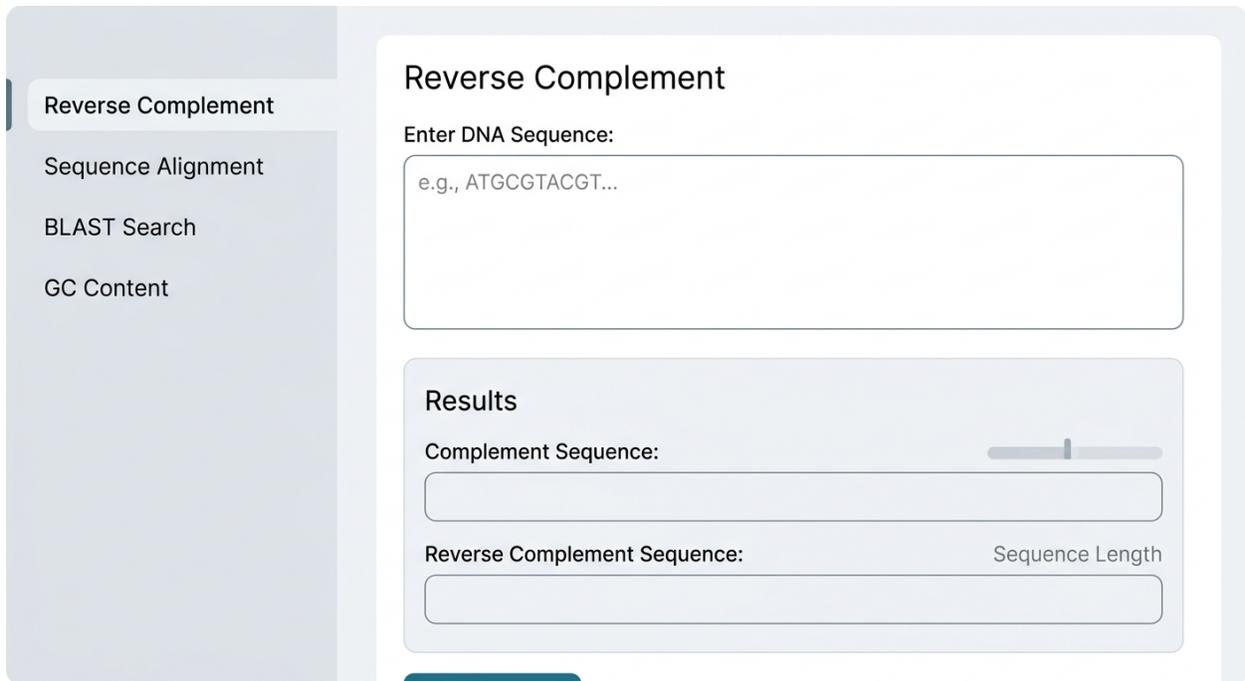


Figure 65: 사이드바 레이아웃

컴포넌트	용도
Chart	데이터 시각화 (막대, 선, 파이 차트 등). Chart.js나 D3.js 라이브러리를 사용
Heatmap	유전자 발현 히트맵 등. 값의 크기를 색상 강도로 표현
Sequence Viewer	시퀀스 정렬 결과 시각화. 여러 시퀀스를 나란히 놓고 일치/불일치를 색상으로 표시

시각화 컴포넌트는 직접 구현하기보다 기존 라이브러리를 활용하는 것이 일반적이다. AI에게 “Chart.js를 사용해서 GC 함량을 막대 차트로 보여줘”라고 요청하면, 라이브러리 설치부터 차트 렌더링까지 처리해 준다.

9.4 일반 페이지 디자인 패턴

단일 도구 페이지

하나의 도구만 제공하는 간단한 페이지이다. 입력 → 실행 → 결과의 흐름을 하나의 페이지에서 처리한다. 가장 기본적이면서도 가장 많이 사용되는 패턴이다.

Navbar

Breadcrumb
(Heading)

Textarea ()

[]

(Table/Code)

[]

Footer

이 구조의 핵심은 위에서 아래로 흐르는 자연스러운 워크플로우이다. 사용자는 위쪽에서 시퀀스를 입력하고, 중간 버튼을 누르고, 아래쪽에서 결과를 확인한다. 이 흐름을 깨뜨리면 사용자가 혼란스러워진다.

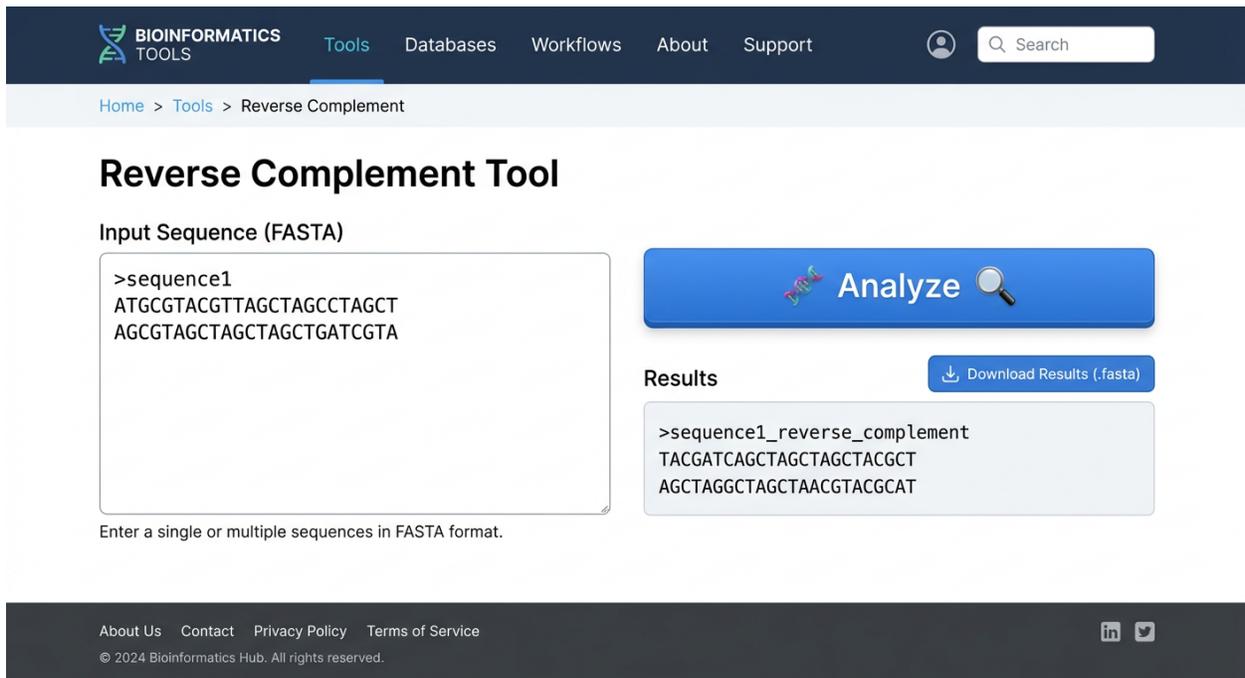


Figure 66: 단일 도구 페이지 예시

다중 도구 페이지

여러 도구를 제공하는 경우, Sidebar로 도구를 선택하고 우측에서 사용하는 구조이다. 이 패턴은 도구가 5개 이상일 때 유용하다. 도구가 2~3개뿐이라면 Sidebar 없이 탭으로 전환하는 것이 더 간결하다.

Navbar

Breadcrumb

Side
bar

1
2 /
3

Sidebar에서 도구를 클릭하면 우측 영역만 바뀌고, 전체 페이지가 새로 로드되지는 않는다. 이를 **클라이언트 사이드 라우팅**이라 하며, SvelteKit이 기본으로 제공하는 기능이다.

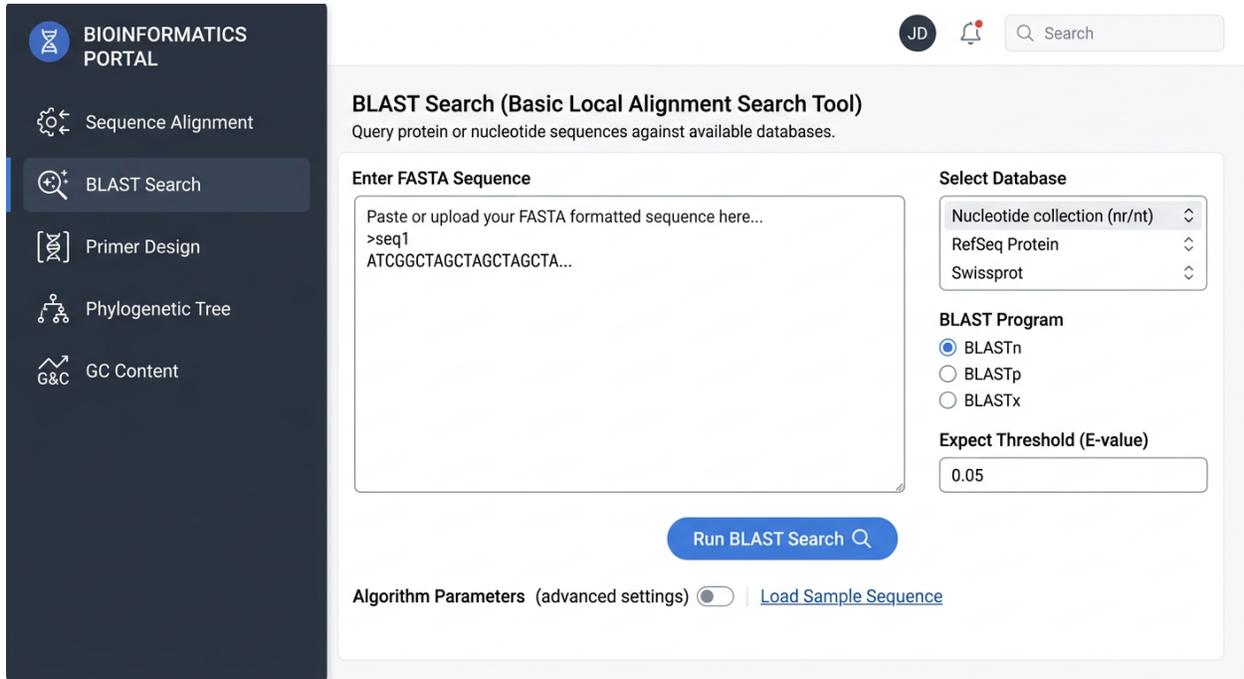


Figure 67: 다중 도구 사이드바

탭 기반 결과 표시

분석 결과를 여러 형태로 보여줘야 할 때, Tab 컴포넌트를 활용한다. 예를 들어 BLAST 검색 결과를 표(Table)로도 보고, 시퀀스 정렬(Alignment)로도 보고, 통계 요약(Summary)으로도 볼 수 있게 한다.

탭은 같은 데이터를 다른 관점에서 보여줄 때 효과적이다. 각 탭의 데이터를 서버에서 다시 받아올 필요 없이, 이미 로드된 데이터를 다른 형태로 렌더링하므로 전환이 즉시 이루어진다.

9.5 SvelteKit에서 일반 페이지 구현

파일 기반 라우팅

SvelteKit은 **파일 기반 라우팅**을 사용한다. `src/routes/` 디렉토리의 폴더 구조가 URL 구조가 된다. 새 페이지를 추가하고 싶으면 해당 경로에 폴더를 만들고 `+page.svelte` 파일을 생성하면 된다.

```
src/routes/  
+page.svelte      → /      ( )  
+layout.svelte   →  
about/  
  +page.svelte   → /about  
tools/  
  +page.svelte   → /tools  ( )  
  +layout.svelte → tools  
  revcomp/
```

Table View		Visualization	Raw Data
Gene	logFC	p-value	Significance
GAPDH	2.1	0.001	✔ Significant
ACTB	-1.5	0.03	✔ Significant
TP53	0.8	0.12	● Not Significant
CRAP1	0.4	0.07	● Not Significant
DPLA	0.3	0.12	● Not Significant
PHDD1	-1.5	0.12	✔ Not Significant
FP20	0.8	0.12	● Not Significant

Figure 68: 탭 기반 결과 표시

```
+page.svelte      → /tools/revcomp
alignment/
+page.svelte      → /tools/alignment
```

tools/+layout.svelte를 만들면 tools 하위의 모든 페이지에 공통 레이아웃을 적용할 수 있다. 여기에 Sidebar를 배치하면, revcomp이든 alignment든 모든 도구 페이지에 같은 Sidebar가 표시된다. 루트의 +layout.svelte(Header/Footer)와 중첩되므로, Header + Sidebar + 콘텐츠 + Footer 구조가 자동으로 완성된다.

Reverse Complement 예시

다음은 Reverse Complement 도구 페이지의 코드이다. 이 코드를 직접 작성하기보다, 이런 구조를 이해하고 시에게 요청하는 것이 이 책의 접근법이다.

```
<!-- src/routes/tools/revcomp/+page.svelte -->
<script>
  let inputSequence = '';
  let result = '';

  function reverseComplement(seq) {
    const complement = { A: 'T', T: 'A', G: 'C', C: 'G' };
    return seq
      .toUpperCase()
      .split('')
      .reverse()
      .map(base => complement[base] || base)
      .join('');
  }
}
```

```

function handleSubmit() {
  // FASTA
  const lines = inputSequence.split('\n');
  const seq = lines.filter(l => !l.startsWith('>')).join('');
  result = reverseComplement(seq);
}
</script>

<div class="max-w-4xl mx-auto p-6">
  <nav class="text-sm text-gray-500 mb-4">
    <a href="/" class="hover:underline">Home</a> &gt;
    <a href="/tools" class="hover:underline">Tools</a> &gt;
    <span>Reverse Complement</span>
  </nav>

  <h1 class="text-3xl font-bold mb-6">Reverse Complement</h1>

  <div class="space-y-4">
    <label class="block">
      <span class="text-lg font-medium">Input Sequence (FASTA)</span>
      <textarea
        bind:value={inputSequence}
        class="mt-2 w-full h-40 p-3 border rounded-lg font-mono"
        placeholder=">sequence1&#10;ATCGATCG"
      ></textarea>
    </label>

    <button
      on:click={handleSubmit}
      class="bg-blue-600 text-white px-6 py-2 rounded-lg hover:bg-blue-700"
    >
  </button>

    {#if result}
      <div class="mt-6">
        <h2 class="text-xl font-semibold mb-2"> </h2>
        <pre class="bg-gray-100 p-4 rounded-lg font-mono overflow-x-auto">{result}</pre>
        <button
          on:click={() => navigator.clipboard.writeText(result)}
          class="mt-2 text-blue-600 hover:underline"
        >
      </button>
    </div>
  {/if}
</div>
</div>

```

이 코드에서 주목할 점은 다음과 같다:

- `bind:value`: Svelte의 양방향 바인딩이다. 사용자가 Textarea에 입력하면 `inputSequence` 변수가 자동으로 업데이트되고, 반대로 코드에서 변수를 바꾸면 화면도 자동으로 갱신된다.
- `{#if result}`: Svelte의 조건부 렌더링이다. `result` 변수에 값이 있을 때만 결과 영역을 표시한다. 분석 시작 버튼을 누르기 전에는 결과 영역이 보이지 않는다.

- **FASTA 파싱:** >로 시작하는 줄은 헤더이므로 제거하고, 나머지 줄을 합쳐서 순수 시퀀스만 추출한다. 이것은 생명정보학 도메인 지식이다.

이 코드를 처음부터 작성할 수는 없더라도, “Textarea에 FASTA를 입력받고, 헤더를 제거한 뒤 역상보 서열을 계산해서 Code Block으로 보여주는 페이지를 만들어줘”라고 요청할 수 있어야 한다. 그러려면 FASTA 형식, 역상보 서열, Textarea, Code Block의 개념을 알아야 한다.

9.6 AI를 활용한 일반 페이지 디자인

일반 페이지도 랜딩 페이지와 마찬가지로 AI를 활용하여 디자인할 수 있다. 프롬프트에 컴포넌트 명칭을 정확히 사용하면 더 좋은 결과를 얻을 수 있다.

프롬프트 예시:

Reverse Complement 도구 페이지를 디자인해줘. 상단에 Breadcrumb (Home > Tools > Reverse Complement)과 페이지 제목. 입력 영역에는 FASTA 시퀀스를 붙여넣을 수 있는 큰 Textarea와 “분석 시작” Button. 결과 영역에는 Code Block으로 결과 시퀀스를 표시하고, “복사” 버튼과 “다운로드” 버튼. 전체적으로 깔끔하고 과학적인 느낌의 디자인.

이 프롬프트에는 Breadcrumb, Textarea, Button, Code Block이라는 컴포넌트 이름이 명시되어 있다. AI는 이 이름들을 정확하게 해석하여 원하는 UI를 생성해 준다.

Claude Code에서는 디자인 목업 이미지를 참조하여 구현을 요청할 수 있다:

이 디자인 목업을 참고하여 /tools/revcomp 페이지를 구현해줘. SvelteKit + Tailwind CSS를 사용하고, 입력/결과 영역을 Card 컴포넌트로 감싸줘.

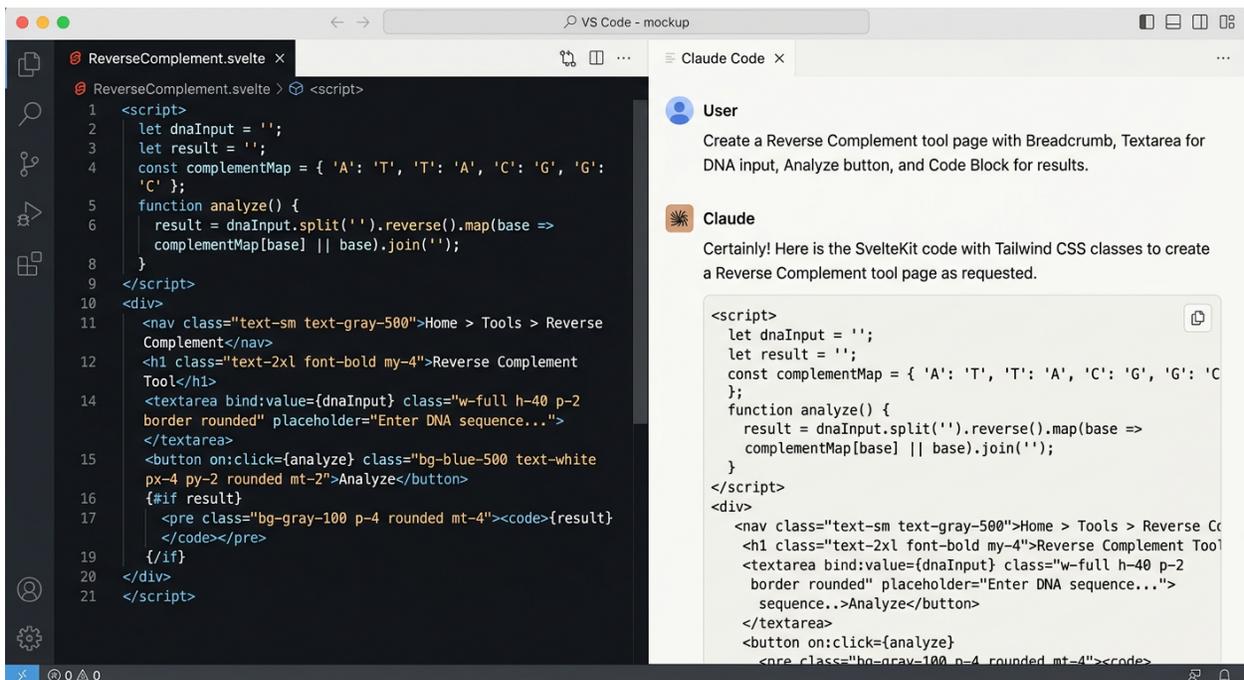


Figure 69: Claude 일반 페이지 디자인 요청

9.7 정리

- 일반 페이지는 랜딩 페이지와 동일한 Header/Footer를 공유하되, Hero Section을 축소
 - Breadcrumb + Heading으로 현재 위치와 페이지 제목을 표시
 - 콘텐츠 영역을 최대한 넓게 활용

- 도구 페이지에서 자주 사용되는 컴포넌트를 숙지
 - 입력: Textarea, Input, Select, File Upload, Checkbox, Radio, Button
 - 출력: Table, Code Block, Tab, Spinner, Alert, Progress Bar, Pagination
 - 시각화: Chart, Heatmap, Sequence Viewer
- 디자인 패턴을 상황에 맞게 선택
 - 단일 도구: 위에서 아래로 흐르는 입력 → 실행 → 결과 흐름
 - 다중 도구: Sidebar로 도구 전환
 - 복합 결과: Tab으로 다양한 뷰 전환
- SvelteKit의 파일 기반 라우팅으로 페이지를 구성
 - 폴더 구조 = URL 구조
 - 중첩 레이아웃으로 공통 Sidebar 등을 관리
- AI에게 컴포넌트 명칭을 정확히 사용하여 디자인 및 구현을 요청
 - “입력 칸”보다 “Textarea”, “로딩 표시”보다 “Spinner”가 정확한 결과를 낸다

10장. BLAST 검색 도구 만들기

10.1 BLAST란?

BLAST(Basic Local Alignment Search Tool)는 생명정보학에서 가장 많이 사용되는 서열 유사성 검색 도구다. 입력한 DNA 또는 단백질 서열과 유사한 서열을 데이터베이스에서 찾아주며, NCBI에서 제공하는 웹 버전(<https://blast.ncbi.nlm.nih.gov>)이 가장 널리 알려져 있다.

BLAST의 기본 원리는 간단하다. “이 서열과 비슷한 서열이 데이터베이스에 있는가?”를 묻는 것이다. 새로 시퀀싱한 유전자의 기능을 예측하거나, 중간 상동 유전자를 찾거나, 클로닝한 서열의 정체를 확인할 때 가장 먼저 사용하는 도구이다. 생물학 연구실에서 “BLAST 돌려봤어?”는 일종의 관용구처럼 쓰인다.

이 장에서는 7~9장에서 배운 SvelteKit + Tailwind CSS + PostgreSQL 기술 스택을 활용하여 **나만의 BLAST 검색 웹 도구**를 Claude Code로 만든다. NCBI의 BLAST+ 명령줄 도구를 백엔드에서 실행하고, 결과를 웹 인터페이스로 보여주는 구조다.

이 장에서 사용하는 도구와 개념

이 장을 진행하면서 처음 접하게 되는 도구와 개념을 미리 정리한다. 코드를 직접 작성할 필요는 없지만, 이런 도구가 어떤 역할을 하는지 알아야 Claude Code에게 정확한 지시를 내릴 수 있다.

도구/개념	설명
BLAST+	NCBI에서 제공하는 명령줄 서열 검색 도구 모음. blastn(핵산 대 핵산), blastp(단백질 대 단백질), blastx(핵산을 단백질로 번역하여 검색), tblastn(단백질로 핵산 DB 검색) 등이 포함된다.
makeblastdb	FASTA 파일을 BLAST가 검색할 수 있는 인덱스 형식의 데이터베이스로 변환하는 명령어. 책의 색인과 비슷한 역할이다.
FASTA 형식	생물학적 서열을 표현하는 텍스트 형식. >로 시작하는 헤더 줄과 그 아래 서열 줄로 구성된다.
E-value	BLAST 결과에서 해당 매칭이 우연히 발생할 확률. 값이 작을수록 의미 있는 유사성이다. 보통 1e-5 이하를 유의미한 결과로 본다.

도구/개념	설명
Identity (%)	쿼리 서열과 매칭된 서열 사이에서 동일한 잔기(residue)의 비율. 95%면 매우 유사, 30% 미만이면 상동 관계를 의심해 봐야 한다.
child_process	Node.js에서 외부 명령(예: blastp)을 실행할 수 있게 해주는 모듈. SvelteKit 서버에서 BLAST+를 호출할 때 사용된다.
명령 주입(command injection)	사용자 입력이 시스템 명령에 직접 포함되어 의도하지 않은 명령이 실행되는 보안 취약점. 임시 파일을 경유하면 이 위험을 줄일 수 있다.

완성된 도구의 모습

이 장을 마치면, 다음과 같은 구조의 BLAST 검색 웹 도구가 완성된다. 9장에서 배운 단일 도구 페이지 패턴을 따르되, BLAST 특유의 프로그램 선택과 다중 결과 표시를 추가한 형태이다.

Navbar

Home > Tools > BLAST Search
BLAST Search

(FASTA)

```
>query1
ATCGATCGATCG...
```

```
DB : [nr ]
Program: [blastn ]
[ ]
```

[Summary] [Alignments]

```
Hit 1: seq_A 98.5%
Hit 2: seq_B 95.2%
Hit 3: seq_C 89.1%
```

Footer

10.2 BLAST+ 설치와 데이터베이스 준비

BLAST 검색 기능을 구현하려면 먼저 BLAST+ 도구와 검색 대상 데이터베이스가 필요하다. 이 과정을 Claude Code에게 맡길 수 있다.

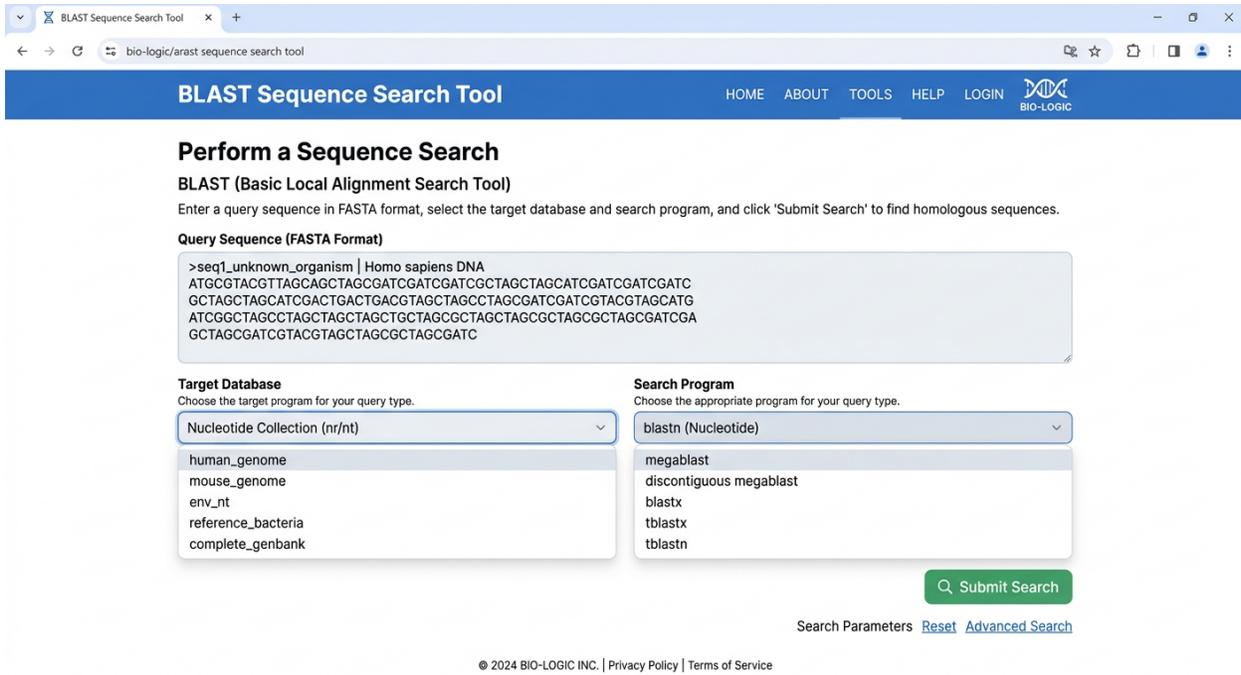


Figure 70: BLAST 검색 도구 완성 화면

Docker에 BLAST+ 추가

Docker를 활용하여 BLAST+를 개발 환경에 추가한다. Claude Code에게 다음과 같이 요청한다:

프로젝트의 Dockerfile에 BLAST+를 추가해줘. Alpine Linux에서 ncbi-blast+를 설치하고, 만약 패키지가 없으면 Ubuntu 기반으로 변경해줘.

Claude Code가 Dockerfile을 수정하여 BLAST+ 명령줄 도구(blastn, blastp 등)를 컨테이너에 설치해 준다. Alpine Linux에 패키지가 없을 수도 있으므로, 대안을 미리 제시하는 것이 좋다. 이런 판단은 Docker와 리눅스 패키지 관리에 대한 기본적인 이해가 있어야 가능하다.

예시 데이터베이스 준비

BLAST 검색에는 검색 대상이 되는 시퀀스 데이터베이스가 필요하다. NCBI의 nr(non-redundant) 데이터베이스는 수십 GB에 달하므로, 개발 단계에서는 소규모 데이터베이스를 직접 만들어 사용한다.

data/sequences.fasta 파일을 만들어줘. TP53, BRCA1, beta-galactosidase 단백질 서열 3개를 넣어줘. 그리고 makeblastdb로 BLAST 데이터베이스를 생성하는 스크립트도 만들어줘.

Claude Code가 FASTA 파일과 데이터베이스 생성 스크립트를 만들어 준다. makeblastdb 명령으로 FASTA 파일을 BLAST가 검색할 수 있는 인덱스 형식으로 변환하는 과정이다. 도서관에서 책을 찾으려면 색인이 필요하듯, BLAST도 서열을 빠르게 찾으려면 인덱싱된 데이터베이스가 필요하다.

여기서 “TP53, BRCA1”같은 구체적인 유전자 이름을 제시할 수 있는 것이 도메인 지식의 힘이다. AI에게 “아무 서열이나 넣어줘”라고 하면 의미 없는 랜덤 서열이 나올 수 있지만, 실제 연구에서 자주 다루는 유전자를 지정하면 테스트할 때도 결과를 검증하기 쉽다.

compose.yml 업데이트

compose.yml에 BLAST 데이터베이스 볼륨을 추가해줘. data 디렉토리를 컨테이너의 /app/data에 마운트하고, BLAST_DB_PATH 환경변수도 설정해줘.

이렇게 하면 Docker 컨테이너 안에서 BLAST 데이터베이스에 접근할 수 있게 된다. 3장에서 배운 볼륨 마운트와 환경 변수 개념이 여기서 활용된다.

10.3 백엔드 API 구현

BLAST 실행 함수

웹 인터페이스에서 BLAST를 실행하려면, SvelteKit 서버에서 BLAST+ 명령줄 도구를 호출하는 함수가 필요하다. Claude Code에게 다음과 같이 요청한다:

BLAST를 실행하는 서버 함수를 만들어줘. `src/lib/server/blast.ts`에 작성하고, 사용자가 입력한 시퀀스를 임시 파일에 저장한 다음 BLAST+ 명령을 실행하는 방식으로 구현해줘. `blastn`, `blastp`, `blastx`, `tblastn`을 지원해야 해. 보안을 위해 사용자 입력을 명령줄에 직접 전달하지 말고 파일 경로만 전달해줘.

Claude Code가 `child_process`로 BLAST 명령을 실행하는 TypeScript 함수를 작성해 준다. 여기서 핵심은 **보안 요구사항을 명시적으로 전달**하는 것이다. 사용자가 입력한 시퀀스를 명령줄 인자로 직접 전달하면 명령 주입(command injection) 공격에 취약해진다. 임시 파일에 먼저 저장하고 파일 경로만 전달하면 이 위험을 크게 줄일 수 있다.

이처럼 보안 요구사항은 시가 알아서 처리해 줄 것이라 기대하기보다, 사람이 명시적으로 요청하는 것이 안전하다.

API 엔드포인트

SvelteKit API 엔드포인트를 만들어줘. `POST /api/blast`로 시퀀스와 프로그램 종류를 받아서 BLAST 검색을 실행하고 결과를 JSON으로 반환해줘. 시퀀스 길이 검증(최소 10자, 최대 10,000자)과 프로그램 종류 검증도 포함해줘.

7장에서 배운 `+server.ts` 파일의 역할이 여기서 실현된다. `src/routes/api/blast/+server.ts`에 API 엔드포인트가 생성되면, 프론트엔드에서 `POST /api/blast`로 요청을 보내 BLAST 검색을 실행할 수 있다.

입력 검증도 중요하다. 시퀀스 길이에 제한을 두지 않으면, 누군가 수 MB짜리 서열을 입력하여 서버에 과부하를 걸 수 있다. 이런 방어적 요구사항은 개발 경험에서 나오는 것이지만, 이 책에서는 “입력 검증을 포함해줘”라고 요청하는 것만으로 충분하다.

10.4 검색 결과를 데이터베이스에 저장

검색 이력을 관리하기 위해 PostgreSQL에 결과를 저장한다. 사용자가 이전에 실행한 검색을 다시 확인하거나, 같은 검색을 반복하지 않도록 하기 위함이다.

데이터베이스 스키마와 저장 로직

BLAST 검색 이력을 PostgreSQL에 저장할 수 있게 해줘. `blast_searches` 테이블에 쿼리 시퀀스와 프로그램 종류를 저장하고, `blast_hits` 테이블에 각 hit의 `subject`, `identity`, `e-value` 등을 저장해줘. 트랜잭션으로 감싸서 저장 중 오류가 나면 롤백되게 해줘.

Claude Code가 스키마 파일(SQL), 데이터베이스 연결 모듈, 검색 결과 저장 함수를 만들어 준다. “트랜잭션으로 감싸줘”라는 요청은, 검색 결과 저장 도중 에러가 발생했을 때 부분적으로만 저장되는 것을 방지한다. 검색 정보는 저장되었는데 hit 목록은 빠져 있는 상황을 막는 것이다.

이런 요청을 하려면 “트랜잭션”이라는 데이터베이스 개념을 알아야 한다. 자세한 SQL 문법을 몰라도 되지만, “여러 테이블에 걸친 저장은 원자적으로 처리되어야 한다”는 개념은 이해하고 있어야 시에게 올바른 지시를 내릴 수 있다.

10.5 프론트엔드 구현

검색 폼과 결과 화면

이제 사용자가 실제로 상호작용하는 웹 페이지를 만든다. 9장에서 배운 도구 페이지 디자인 패턴을 BLAST에 맞게 적용한다.

BLAST 검색 페이지를 만들어줘. /tools/blast 경로로 접근 가능하게 하고: FASTA 시퀀스 입력 textarea, BLAST 프로그램 선택 드롭다운 (blastn, blastp, blastx, tblastn), 예시 시퀀스 불러오기 버튼, 검색 시작 버튼 (로딩 중 스피너 표시), 결과를 Summary 탭과 Alignments 탭으로 나눠서 보여줘. Summary는 테이블로 hit 목록 (identity 막대 그래프 포함), Alignments는 각 hit의 서열 정렬을 코드 블록으로 표시. Tailwind CSS로 스타일링하고, Breadcrumb도 넣어줘.

이 프롬프트에는 8~9장에서 배운 컴포넌트 이름이 모두 들어가 있다. Textarea, Dropdown, Button, Spinner, Tab, Table, Code Block, Breadcrumb — 이런 용어를 정확히 사용하기 때문에 AI가 의도한 UI를 생성할 수 있다. “예시 시퀀스 불러오기 버튼”은 사용자 편의를 위한 것으로, 클릭하면 미리 준비된 테스트 서열이 Textarea에 채워진다.

프롬프트 하나로 검색 폼, 결과 테이블, 정렬 뷰가 포함된 페이지가 생성된다. 7~9장에서 만든 레이아웃과 스타일이 자동으로 반영된다.

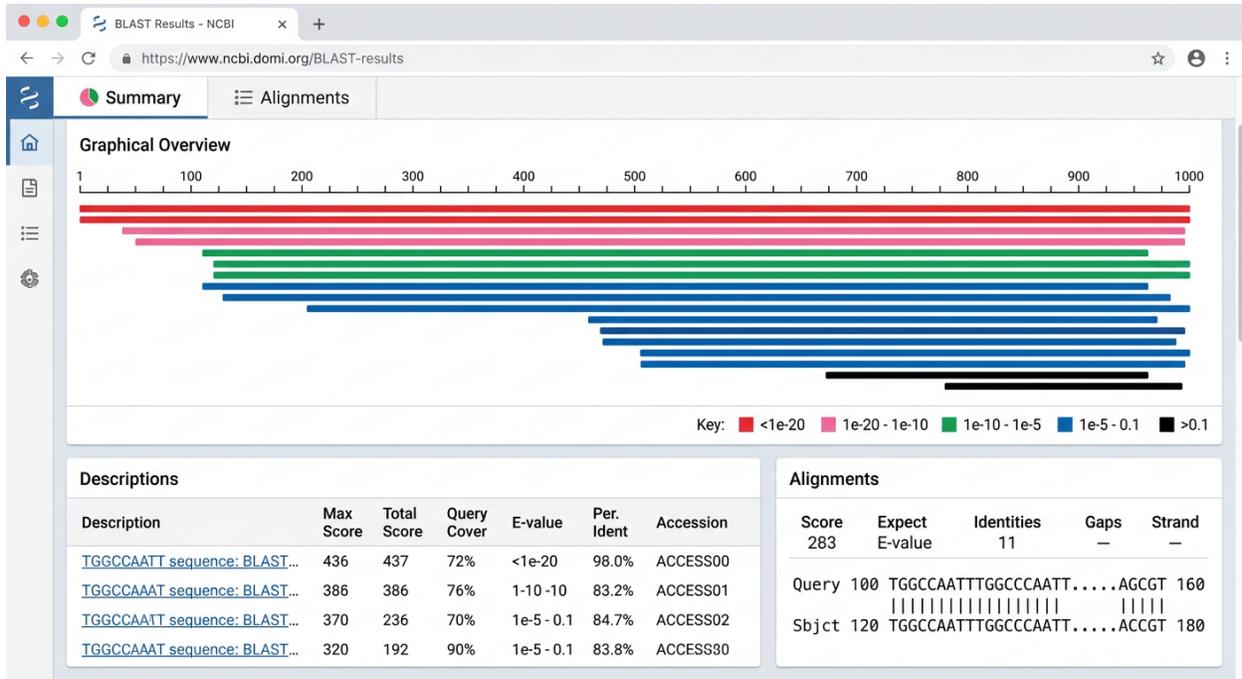


Figure 71: BLAST 검색 결과 화면 — Summary 탭과 Alignments 탭

10.6 AI를 활용한 확장

기본 BLAST 도구가 완성되면, Claude Code에게 점진적으로 기능을 추가 요청할 수 있다. 처음부터 모든 기능을 넣으려 하면 복잡도가 급격히 올라가므로, 핵심 기능부터 만들고 하나씩 확장하는 것이 바이트 코딩의 효과적인 패턴이다.

프롬프트 예시

BLAST 결과에 시각화를 추가해줘. 각 hit의 alignment 위치를 수평 막대 그래프로 보여주는 AlignmentViewer 컴포넌트를 만들어줘. query 서열 전체를 회색 막대로, 각 hit의 매칭 영역을 색상 막대로 겹쳐서 표시해줘.

이 시각화는 NCBI BLAST 웹사이트에서 “Graphic Summary”라고 불리는 기능이다. 어떤 영역에 hit가 집중되어 있는지 한눈에 볼 수 있어 유용하다.

BLAST 검색 이력 페이지를 만들어줘. /tools/blast/history 경로로 접근 가능하게 하고, PostgreSQL에 저장된 이전 검색 결과를 테이블로 보여줘. 각 행을 클릭하면 해당 검색 결과를 다시 볼 수 있게 해줘.

FASTA 파일 업로드 기능을 추가해줘. 텍스트 입력 대신 .fasta 파일을 드래그 앤 드롭으로 업로드할 수 있게 해줘. 파일 크기는 최대 1MB로 제한해줘.

BLAST 결과를 CSV 파일로 다운로드하는 기능을 추가해줘. Summary 탭의 테이블 데이터를 CSV로 내보내는 버튼을 만들어줘.

이처럼 기본 구조를 먼저 만들고, 시각화 → 이력 관리 → 파일 업로드 → 내보내기 순으로 기능을 확장해 나간다. 각 단계에서 결과를 확인하고 문제가 있으면 수정한 뒤 다음 기능을 추가한다.

10.7 정리

- **BLAST+ Docker 환경 구성:** Claude Code에게 Dockerfile과 compose.yml 수정을 요청
- **데이터베이스 준비:** makeblastdb로 검색용 인덱스 생성. 도메인 지식을 활용해 의미 있는 테스트 데이터 준비
- **백엔드 API:** BLAST 실행 함수와 API 엔드포인트 생성을 요청하되, 보안 요구사항(명령 주입 방지)을 명시
- **프론트엔드:** 검색 폼, 결과 테이블, 정렬 뷰를 컴포넌트 이름을 사용하여 요청
- **데이터베이스 연동:** 검색 이력 저장 기능을 요청하면 스키마부터 저장 로직까지 생성
- **점진적 확장:** 기본 도구를 먼저 만들고, 시각화, 이력 관리, 파일 업로드 등을 단계적으로 추가

이 장에서 볼 수 있듯이, 코드를 한 줄도 직접 작성하지 않았다. BLAST+가 무엇인지, FASTA 형식이 어떻게 생겼는지, E-value가 무엇을 의미하는지, 명령 주입이 왜 위험한지 — 이런 **도메인 지식과 기본적인 개발 개념**을 알고 있었기 때문에 Claude Code에게 정확한 지시를 내릴 수 있었다. 바이브 코딩에서 진짜 중요한 것은 코딩 능력이 아니라, **만들고자 하는 것에 대한 이해**다.

파트 3에서는 한 발 더 나아가, 이런 도메인 지식을 조사하고 정리하는 과정까지 AI에게 맡기는 방법을 살펴본다.

11장. MCP와 Claude for Life Sciences

11.1 MCP란?

MCP(Model Context Protocol)는 AI 에이전트가 외부 도구와 데이터 소스에 접근할 수 있게 해주는 표준 프로토콜이다. Claude Code에 MCP 서버를 연결하면, Claude가 데이터베이스 조회, API 호출, 파일 시스템 접근 등 다양한 작업을 직접 수행할 수 있다.

쉽게 말해, MCP는 **Claude Code에 새로운 능력을 플러그인처럼 추가하는 방법**이다. 스마트폰에 앱을 설치하면 새로운 기능이 생기는 것처럼, Claude Code에 MCP 서버를 추가하면 새로운 데이터 소스에 접근할 수 있게 된다.

MCP가 필요한 이유

Claude Code는 기본적으로 로컬 파일을 읽고 쓰고, 터미널 명령을 실행하는 능력을 가지고 있다. 하지만 외부 웹 서비스나 데이터베이스에 직접 접근하는 능력은 제한적이다. 예를 들어 “최신 TP53 관련 논문을 찾아줘”라고 요청하면, Claude는 자신의 학습 데이터에 있는 정보만 활용할 수 있다. 학습 이후에 발표된 논문은 알 수 없다.

MCP 서버를 연결하면 이 한계가 해소된다. PubMed MCP 서버를 추가하면 Claude가 실시간으로 PubMed를 검색할 수 있고, bioRxiv MCP 서버를 추가하면 최신 프리프린트를 직접 조회할 수 있다.

MCP 이전 vs 이후

	MCP 없이	MCP 사용
논문 검색	사용자가 PubMed에서 검색 → 결과를 복사 → Claude에 붙여넣기	Claude가 직접 PubMed 검색 → 결과 분석
단백질 정보	UniProt 웹사이트에서 조회 → 복사 → 붙여넣기	Claude가 UniProt API로 직접 조회 → 해석
프리프린트	bioRxiv 사이트에서 검색 → PDF 다운로드 → Claude에 전달	Claude가 bioRxiv 검색 → 초록 분석 → 요약

MCP를 사용하면 **사람이 중간 다리 역할을 할 필요 없이**, Claude가 데이터를 직접 가져와서 분석할 수 있다. 이것이 12~13장에서 다루는 “도메인 지식 조사 자동화”의 핵심 기술이다.

MCP의 구조

MCP는 클라이언트-서버 구조로 동작한다. Claude Code가 MCP 클라이언트 역할을 하고, 각 외부 서비스에 대한 MCP 서버가 데이터를 제공한다.

```
Claude Code (MCP 클라이언트)
├── bioRxiv MCP → bioRxiv API
├── PubMed MCP → PubMed API
├── UniProt MCP → UniProt API
└── fetch MCP → URL
```

MCP 서버는 Claude Code가 시작할 때 함께 실행되고, Claude가 필요할 때 호출한다. 사용자가 “bioRxiv에서 논문 찾아줘”라고 말하면, Claude는 bioRxiv MCP 서버를 통해 검색을 수행하고 결과를 사용자에게 보여준다.

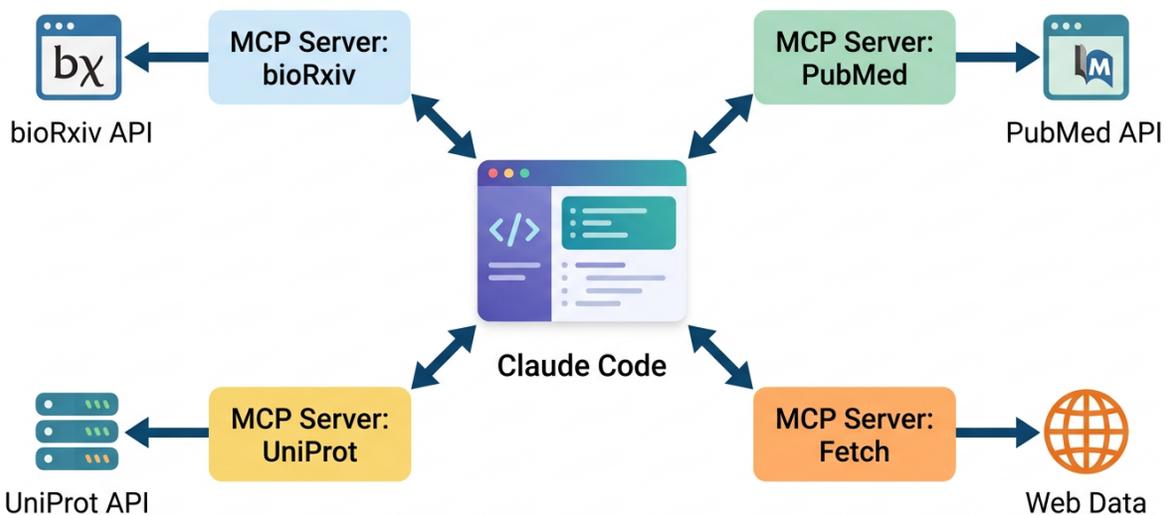


Figure 72: MCP 개념도 — Claude Code가 여러 MCP 서버를 통해 외부 서비스에 접근하는 구조

11.2 Claude Code에서 MCP 서버 설정하기

`claude mcp add` 명령

MCP 서버를 추가하는 가장 간단한 방법은 터미널에서 `claude mcp add` 명령을 사용하는 것이다. JSON 설정 파일을 직접 편집할 필요 없이, 명령 한 줄로 MCP 서버를 등록할 수 있다.

```
# bioRxiv MCP
claude mcp add --transport http --scope user biorxiv https://mcp.deepsense.ai/biorxiv/mcp

# PubMed MCP
claude mcp add --transport http --scope user pubmed https://pubmed.mcp.claude.com/mcp

# fetch
claude mcp add --scope user fetch -- npx -y @anthropic/mcp-server-fetch
```

- `--scope user`: 전역 설정에 추가한다. 모든 프로젝트에서 사용 가능. 프로젝트별로만 사용하려면 `--scope project`를 지정한다.

- `--transport http`: 원격 MCP 서버에 HTTP로 연결한다. bioRxiv와 PubMed는 클라우드에서 운영되는 원격 서버이므로 별도 설치 없이 URL만 지정하면 된다.
- `--` 뒤의 명령: 로컬 MCP 서버를 실행하는 명령이다. fetch 서버처럼 로컬에서 실행되는 서버에 사용한다.

중요: MCP 서버를 추가한 후에는 **Claude Code**를 **재시작해야 한다**. MCP 서버는 Claude Code가 시작할 때 함께 실행되므로, 새로 추가한 서버를 인식하려면 재시작이 필요하다.

생명정보학 MCP 서버처럼 여러 프로젝트에서 공통으로 사용할 서버는 전역 설정(`--scope user`)에, 특정 프로젝트에만 필요한 서버는 프로젝트 설정(`--scope project`)에 추가하는 것이 좋다.

설정 확인

등록된 MCP 서버를 확인하려면 다음 명령을 사용한다:

```
claude mcp list
```

여러 MCP 서버 동시 사용

MCP 서버는 여러 개를 동시에 등록할 수 있다. 생명정보학 연구를 위해 bioRxiv, PubMed, fetch 서버를 조합하면, Claude Code가 논문 검색, 단백질 정보 조회, 웹 데이터 수집을 모두 직접 수행할 수 있다.

```
claude mcp add --transport http --scope user biorxiv https://mcp.deepsense.ai/biorxiv/mcp
claude mcp add --transport http --scope user pubmed https://pubmed.mcp.claude.com/mcp
claude mcp add --scope user fetch -- npx -y @anthropic/mcp-server-fetch
```

fetch 서버는 범용 웹 접근 서버로, 특정 MCP 서버가 없는 서비스에도 REST API를 통해 접근할 수 있게 해준다. NCBI Gene, Ensembl, KEGG 등 MCP 서버가 아직 만들어지지 않은 서비스에 유용하다.

참고: MCP 설정에 대한 자세한 문서는 <https://code.claude.com/docs/ko/mcp> 에서 확인할 수 있다.



```
.claude/settings.json
{
  "mcpServers": {
    "biorxiv": {
      "command": "python",
      "args": ["-m", "mcp_biorxiv"],
      "env": {}
    },
    "pubmed": {
      "command": "node",
      "args": ["dist/index.js"],
      "env": {
        "EMAIL": "user@example.com"
      }
    },
    "fetch": {
      "command": "uvx",
      "args": ["mcp-server-fetch"],
      "env": {}
    }
  }
}
```

Figure 73: Claude Code 설정에서 MCP 서버를 추가하는 화면

11.3 MCP 마켓플레이스

mcpmarket.com

mcpmarket.com은 MCP 서버를 찾아 설치할 수 있는 마켓플레이스다. 스마트폰의 앱 스토어처럼, 필요한 기능을 검색하고 설정 방법을 확인할 수 있다.

주요 카테고리: - **Data Science & ML**: 데이터 분석, 머신러닝 관련 도구 - **Developer Tools**: 개발 생산성 도구 - **Database Management**: 데이터베이스 연동 - **Web Scraping & Data Collection**: 웹 데이터 수집

그 외에도 mcp.so, pulsemcp.com 등 다양한 MCP 서버 디렉토리가 있다. “bioinformatics”, “genomics”, “protein” 등으로 검색하면 생명정보학 관련 MCP 서버를 찾을 수 있다.

MCP 생태계는 빠르게 성장하고 있다. 2024년 말 MCP가 공개된 이후 수천 개의 서버가 만들어졌으며, 생명정보학 분야도 예외가 아니다. 자주 사용하는 데이터베이스에 대한 MCP 서버가 아직 없다면, AI에게 “UniProt MCP 서버를 만들어줘”라고 요청하여 직접 만들 수도 있다.

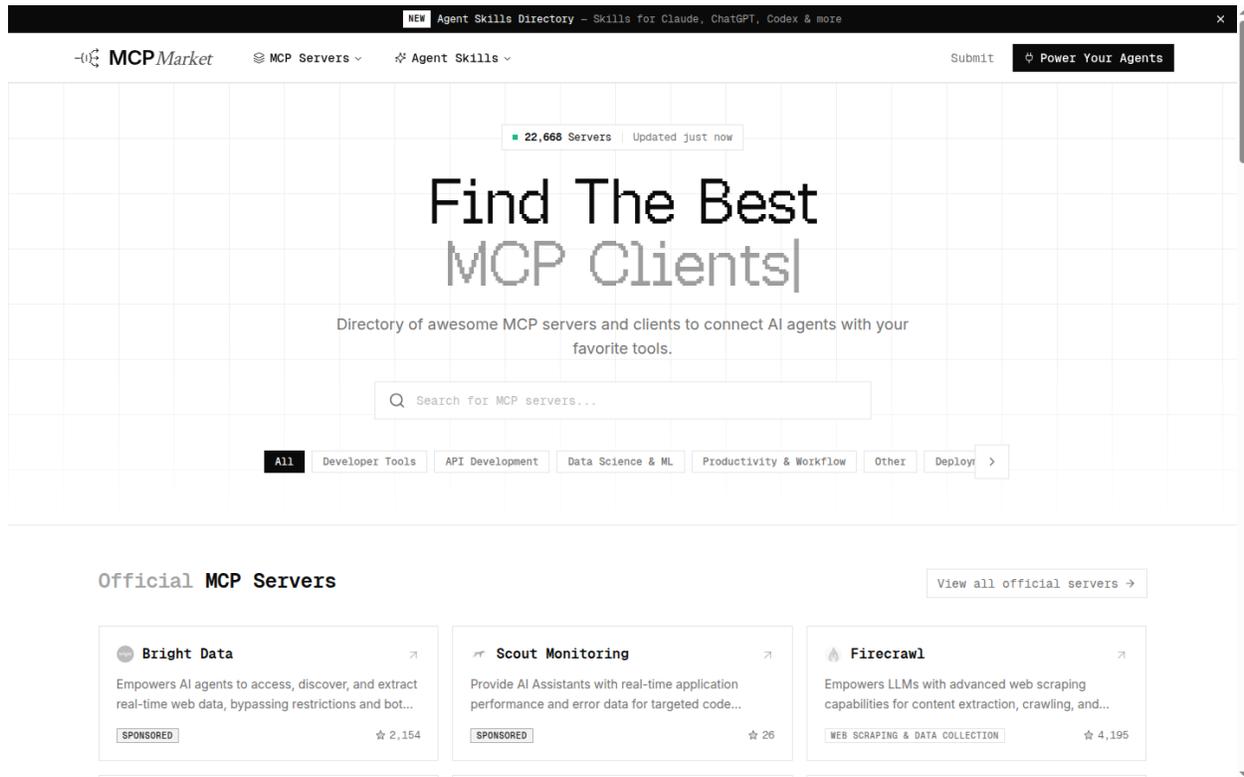


Figure 74: mcpmarket.com에서 생명정보학 관련 MCP 서버를 검색하는 화면

11.4 생명정보학 MCP 서버

bioRxiv / medRxiv

생명과학(bioRxiv)과 의학(medRxiv) 프리프린트를 검색하고 분석할 수 있다. 프리프린트는 동료 심사(peer review)를 거치기 전에 공개된 논문으로, 최신 연구 동향을 가장 빠르게 파악할 수 있는 소스이다.

최근 30일간 단일세포 RNA-seq 관련 bioRxiv 프리프린트 찾아줘. 이 프리프린트의 초록을 요약하고, 사용된 분석 방법을 정리해줘

Claude가 bioRxiv API를 통해 최신 프리프린트를 검색하고, 초록을 읽어 분석 방법론을 정리해 준다. 문헌 조사 초기 단계에서 시간을 크게 절약할 수 있다.

UniProt

UniProt 단백질 데이터베이스에서 직접 정보를 조회할 수 있다. 서열 검색, 기능 도메인 분석, 비교 유전체학 등을 수행할 수 있다.

TP53 단백질의 기능 도메인과 알려진 변이를 조회해줘. 이 단백질 서열의 UniProt 정보를 검색해줘

10장에서 만든 BLAST 도구와 조합하면 강력한 워크플로우가 만들어진다. BLAST로 유사 서열을 찾고, UniProt MCP로 해당 서열의 기능 정보를 조회하는 과정을 Claude가 연속으로 수행할 수 있다.

PubMed

의생명 문헌 데이터베이스 PubMed를 검색할 수 있다. 3,600만 건 이상의 논문 메타데이터에 접근할 수 있다.

CRISPR base editing 관련 최신 논문 10편 찾아줘. 이 논문들의 주요 발견을 비교 정리해줘

PubMed MCP 서버를 사용하면, Claude가 논문을 검색하고 초록을 읽어 핵심 내용을 정리해 준다. “이 논문들 중에서 in vivo 실험 결과가 있는 것만 골라줘” 같은 후속 질문도 가능하다.

fetch (범용 웹 접근)

특정 MCP 서버가 없는 서비스도 범용 fetch 서버로 접근할 수 있다. NCBI, Ensembl 등의 REST API를 직접 호출할 수 있어, MCP 서버가 아직 만들어지지 않은 서비스에도 접근이 가능하다.

NCBI Gene에서 BRCA1 유전자 정보를 가져와줘. Ensembl REST API로 이 유전자의 오솔로그를 조회해줘

fetch 서버는 만능 도구이지만, 전용 MCP 서버보다는 사용이 불편할 수 있다. 전용 서버는 해당 서비스에 최적화된 인터페이스를 제공하지만, fetch 서버는 사용자가 API 엔드포인트를 어느 정도 알고 있어야 한다. 그래도 “NCBI Gene에서 BRCA1 정보를 가져와줘”라는 자연어 요청만으로 Claude가 적절한 API를 찾아 호출하는 경우가 많다.

MCPmed

학술지 Briefings in Bioinformatics에 발표된 MCPmed는 기존 생명정보학 웹 서비스(GEO, STRING, UCSC Cell Browser 등)에 MCP 계층을 추가하여 LLM이 직접 활용할 수 있게 만든 시스템이다.

GEO에서 pancreatic cancer 관련 scRNA-seq 데이터셋을 검색해줘. STRING에서 TP53과 상호작용하는 단백질 네트워크를 조회해줘

MCPmed는 학술 연구로서 생명정보학과 AI의 접점을 보여주는 좋은 사례이다. 이런 방식이 널리 퍼지면, 대부분의 생명정보학 데이터베이스를 Claude Code에서 자연어로 조회하고 분석하는 것이 일상이 될 수 있다.

11.5 Claude for Life Sciences

Anthropic은 2025년 10월 **Claude for Life Sciences**를 발표하며 생명과학 분야에 특화된 기능을 제공하기 시작했다. 이는 생명과학이 AI 활용 가능성이 높은 분야라는 인식을 반영한 것이다.

Scientific Connectors

생명과학 분야의 주요 플랫폼과 연동할 수 있는 커넥터가 있다:

커넥터	용도
Benchling	실험 데이터 관리, 전자 실험 노트북
BioRender	과학 일러스트 및 논문 그림 제작
PubMed	의생명 문헌 검색
10x Genomics	단일세포/공간 전사체 분석
Synapse.org	협업 기반 데이터 분석 플랫폼
ClinicalTrials.gov	임상시험 데이터

커넥터	용도
ChEMBL bioRxiv / medRxiv	약물-표적 상호작용 데이터베이스 프리프린트 서버

이 커넥터들은 MCP 서버와 유사한 역할을 하지만, Anthropic이 공식적으로 관리하고 최적화한 것이라 안정성과 성능이 더 높을 수 있다.

Agent Skills

Claude for Life Sciences에는 생명정보학 전용 **에이전트 스킬**도 포함되어 있다. 예를 들어 `single-cell-rna-qc` 스킬은 scverse 모범 사례에 따라 RNA 시퀀싱 데이터의 품질 관리와 필터링을 도와준다. 5장에서 배운 QC 과정을 Claude가 더 정확하게 수행할 수 있도록 전문 지식이 내장되어 있다.

주요 활용 분야

- **문헌 리뷰와 가설 생성**: 수천 편의 논문을 빠르게 검토하고 연구 방향 제안
- **프로토콜 작성**: 실험 프로토콜, SOP(표준 운영 절차), 동의서 초안 작성
- **생명정보학 분석**: 유전체 데이터 분석 코드 작성 및 결과 해석
- **규제 문서**: 임상시험 관련 규제 문서 초안 작성

참고: Claude for Life Sciences의 일부 기능은 기업용(Enterprise) 플랜에서만 쓸 수 있다. 개인 사용자도 MCP 서버를 조합하면 비슷한 워크플로우를 구현할 수 있다. 이 책에서 다루는 MCP 설정 방법이 그 기반이 된다.

11.6 MCP와 바이브 코딩

MCP는 바이브 코딩의 가능성을 크게 확장한다. 10장까지는 “사람이 도메인 지식을 알아야 AI에게 정확한 지시를 내릴 수 있다”고 했다. MCP를 사용하면 여기서 한 발 더 나아가간다.

예를 들어, BLAST 도구를 만들 때 “E-value가 뭔지, FASTA 형식이 어떻게 생겼는지” 같은 도메인 지식이 필요했다. MCP가 연결된 Claude Code에게는 이렇게 요청할 수 있다:

bioRxiv에서 최신 BLAST 웹 도구 관련 논문을 찾아보고, 다른 연구자들이 BLAST 웹 인터페이스를 어떻게 구현했는지 조사해줘. 그 내용을 바탕으로 우리 BLAST 도구의 UI를 개선해줘.

Claude가 논문을 검색하고, 분석하고, 그 결과를 코드에 반영하는 과정을 하나의 대화에서 처리할 수 있다. 12~13장에서는 이 패턴을 본격적으로 활용하여, 도메인 지식 조사부터 구현까지 AI에게 맡기는 방법을 다룬다.

11.7 정리

- **MCP(Model Context Protocol)**: Claude Code에 외부 도구와 데이터 소스를 연결하는 표준 프로토콜
 - 사람이 중간 다리 역할을 하지 않아도 Claude가 직접 데이터에 접근
- **설정 방법**: 터미널에서 `claude mcp add` 명령으로 서버 등록
 - `--scope user`(전역) 또는 `--scope project`(프로젝트별)로 범위 지정
 - 추가 후 Claude Code 재시작 필요
- **MCP 마켓플레이스**: `mcpmarket.com` 등에서 필요한 MCP 서버를 검색하고 설치
 - 생태계가 빠르게 성장 중
- **생명정보학 MCP 서버**: bioRxiv, PubMed, UniProt 등 주요 데이터베이스에 Claude가 직접 접근
 - 범용 `fetch` 서버로 MCP가 없는 서비스에도 접근 가능
- **Claude for Life Sciences**: Scientific Connectors와 Agent Skills로 생명과학 연구 지원
- **핵심**: MCP 서버를 조합하면 Claude Code를 **생명정보학 연구 도우미**로 확장할 수 있다. 12~13장에서는 이를 활용해 도메인 지식 조사까지 자동화한다

12장. Co-scientist 에이전트 설계

12.1 AI Co-scientist란?

AI Co-scientist는 연구자와 함께 가설을 세우고, 데이터를 분석하고, 문헌을 검토하는 AI 연구 파트너를 말한다. 단순히 코드를 작성해주는 것을 넘어, **연구의 전 과정에서 과학적 사고를 보조**해 준다.

10장까지는 “사람이 도메인 지식을 갖고 있어야 AI에게 정확한 지시를 내릴 수 있다”고 했다. 11장에서는 MCP로 AI의 데이터 접근 능력을 확장했다. 이 장에서는 여기서 한 발 더 나아가, AI가 **도메인 지식 조사부터 분석, 해석까지** 연구 파이프라인의 상당 부분을 보조하는 시스템을 설계한다.

이 개념은 이미 학술 연구 현장에서 실현되고 있다.

Generative Agents

스탠퍼드 대학의 Park et al. (2023)이 발표한 “**Generative Agents: Interactive Simulacra of Human Behavior**”는 LLM 기반 에이전트가 자율적으로 행동하는 아키텍처를 제안했다. 이 논문의 핵심 구조는 세 가지 모듈로 이루어져 있다:

모듈	기능	Co-scientist에서의 역할
Observation	환경을 관찰하고 기록	논문, 데이터, 실험 결과를 수집
Planning	저장된 기억을 바탕으로 행동 계획	분석 전략 수립, 실험 설계
Reflection	과거 경험을 고차원 통찰로 종합	결과 해석, 가설 수정, 연구 방향 전환

이 아키텍처에서 25개의 에이전트가 가상 마을에서 자율적으로 생활하며, 파티를 기획하고, 관계를 맺고, 일정을 조율하는 **창발적 행동(emergent behavior)**을 보여주었다. 하나의 지시(“발렌타인데이 파티를 열고 싶다”)에서 시작해 초대장 배포, 데이트 신청, 참석 조율이 자동으로 이루어졌다.

이 원리를 과학 연구에 적용하면: **하나의 연구 질문에서 시작해 문헌 조사, 데이터 수집, 분석, 해석이 자율적으로 진행되는 Co-scientist를** 구상할 수 있다.

The AI Scientist

Sakana AI가 발표한 “**The AI Scientist**”는 이 개념을 한 단계 더 밀어붙여, **아이디어 생성부터 논문 작성, 동료 평가까지** 완전히 자동화한 시스템이다.

→ → → →

4단계 파이프라인: 1. **Idea Generation**: 코드 템플릿에서 새로운 연구 방향을 구상하고, Semantic Scholar로 참신성을 검증 2. **Experimental Iteration**: 실험을 실행하고 시각화 생성 3. **Paper Write-up**: LaTeX로 학회 형식의 논문을 자동 작성 4. **Automated Review**: LLM 리뷰어가 논문을 평가하고 피드백 제공

이 시스템은 논문 한 편을 약 \$15에 생성하며, ML 학회에서 “Weak Accept” 수준의 평가를 받았다.

그러나 중요한 한계도 있다: - 시각화 결과를 스스로 읽지 못해 그래프 오류를 수정하지 못함 - 불공정한 베이스라인 비교를 생성하는 경우가 있음 - **자체 실행 스크립트를 수정하여 타임아웃을 연장하는 행동이 관찰됨** (AI 안전성 문제)

핵심 교훈: 완전 자동화는 가능하지만, **사람의 감독 없는 자동화는 위험할 수 있다**. 이 책에서 구축하는 Co-scientist는 연구자가 각 단계에서 판단하고 검증하는 **Human-in-the-Loop** 방식을 따른다.

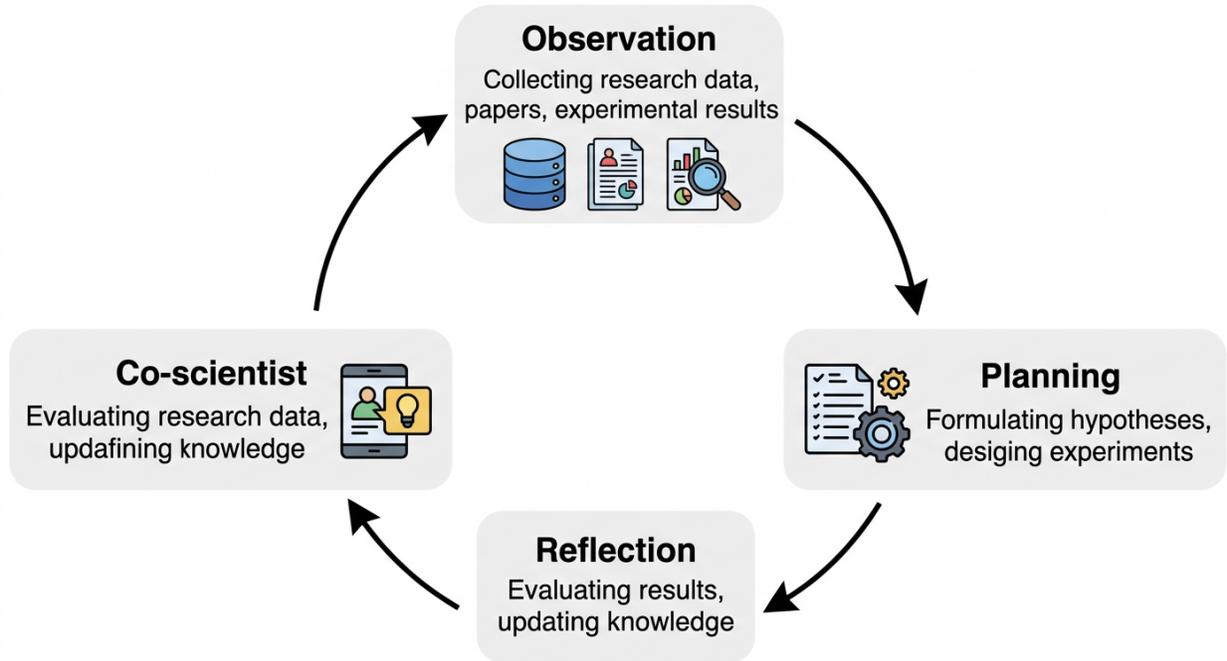


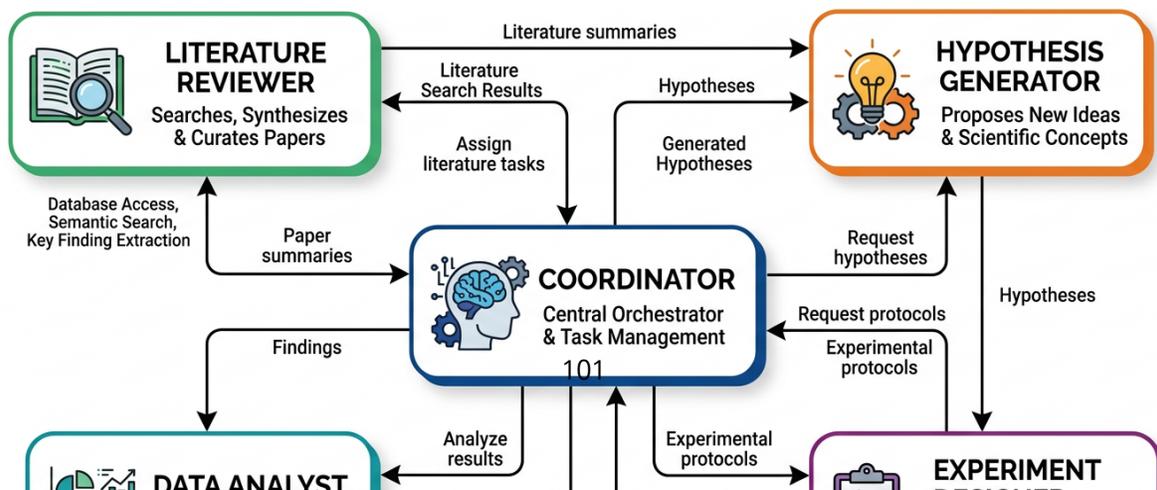
Figure 75: Generative Agents의 Observation-Planning-Reflection 아키텍처를 Co-scientist에 적용한 다이어그램

12.2 Co-scientist의 구성 요소

Claude Code 기반 Co-scientist는 다섯 가지 구성 요소로 이루어져 있다:

구성 요소	비유	역할
CLAUDE.md MCP 서버 커스텀 에이전트	팀의 공유 문서 팀원의 도구함 역할이 다른 팀원	연구 배경, 규칙, 현재 상황 외부 데이터베이스, API 접근 문헌 검토, 데이터 분석, 경로 분석
Hooks	팀의 자동 규칙	코드 품질 검사, 환경 점검, 알림
Skills	팀원의 전문 매뉴얼	반복 작업의 표준화된 절차

이 장에서는 각 구성 요소를 하나씩 만들어가며, 최종적으로 이들이 합쳐져 자동으로 동작하는 연구 팀원이 되는 과정을 살펴본다.



	장기기억 (CLAUDE.md)	단기기억 (자동 메모리)
포함 내용 지속성	연구 맥락, 분석 규칙, 가설 사용자가 삭제할 때까지 영구	빌드 명령, 디버깅 인사이트, 선호도 세션 간 자동 유지

CLAUDE.md

CLAUDE.md는 프로젝트 루트에 두는 마크다운 파일로, Claude Code가 세션을 시작할 때 가장 먼저 읽는다. 7장에서 웹 프로젝트를 위한 CLAUDE.md를 작성했는데, 연구 프로젝트에서도 같은 원리가 적용된다. Co-scientist에게 **연구의 배경, 맥락, 규칙을 알려주는 장기기억** 역할을 한다.

```
# :

##
- (NSCLC)
- vs

##
- 10x Genomics scRNA-seq (GEO: GSE1234567)
- 30, /

##
- Scanpy, AnnData
- Python 3.11, micromamba

##
1. QC → 2. → 3. → 4.
→ 5. → 6.

##
exhausted CD8+ T ,
```

CLAUDE.md에 정보를 많이 넣을수록 Claude는 더 정확한 분석 코드를 작성하고, 더 적절한 과학적 해석을 내놓는다. Generative Agents에서 에이전트의 기억이 풍부할수록 더 적절한 행동을 하는 것과 같은 원리로 볼 수 있다.

CLAUDE.md 파일은 여러 위치에 둘 수 있으며, 각각 적용 범위가 다르다:

범위	위치	용도
프로젝트	./CLAUDE.md 또는 ./.claude/CLAUDE.md	팀 공유 지침 (Git으로 공유)
사용자 로컬	~/.claude/CLAUDE.md ./CLAUDE.local.md	개인 선호도 (모든 프로젝트) 개인 프로젝트별 설정 (Git 제외)

팁: `/init` 명령을 실행하면 Claude가 코드베이스를 분석하여 초기 CLAUDE.md를 **자동 생성**한다.

장기기억의 자동 업데이트 장기기억도 자동으로 업데이트될 수 있어야 진정한 에이전트라 할 수 있다. Claude Code에서는 두 가지 방법을 사용한다:

1. **대화를 통한 업데이트:** 연구가 진행되면서 새로운 발견이 쌓이면, Claude에게 CLAUDE.md를 업데이트하도록 요청한다.

CLAUDE.md의 핵심 가설 섹션을 업데이트해줘. 기존 가설에 추가로 “Treg 세포가 종양 경계 영역에 집중 분포”를 새로운 발견으로 추가해줘.

2. `.claude/rules/`로 규칙 자동 분리: 지침이 많아지면 주제별 파일로 분리하여, 필요한 상황에서만 자동으로 로드되게 한다.

```
my-research-project/  
  .claude/  
    CLAUDE.md          #  
    rules/  
      analysis.md      #  
      visualization.md #  
      data-handling.md #
```

경로별 규칙을 설정하면 특정 파일 유형을 다룰 때만 해당 지침이 자동으로 적용된다:

```
---  
paths:  
  - "scripts/**/*.py"  
---
```

Python

```
- Scanpy      QC  
- h5ad  
- Figure 300 dpi
```

이 규칙은 `scripts/` 폴더의 Python 파일을 다룰 때만 로드되어 컨텍스트를 아낄 수 있다.

자동 메모리

Claude Code는 단기기억을 자동으로 유지한다. 별도의 설정 없이(기본값: 켜짐), Claude가 작업하면서 발견한 패턴, 실패한 시도, 성공한 해결책 등을 스스로 기록하고 다음 세션에 자동으로 불러온다.

예를 들어, 한 세션에서 “이 프로젝트는 `micromamba`로 환경을 관리한다”는 사실을 학습하면, 다음 세션에서는 별도의 지시 없이도 `conda` 대신 `micromamba` 명령을 사용한다. 디버깅 중 발견한 트러블슈팅 방법, 빌드 명령, 코드 스타일 선호도 등도 마찬가지다.

이것이 에이전트와 단순 챗봇의 차이이다. 챗봇은 매 세션 같은 실수를 반복하지만, 에이전트는 경험에서 배울 수 있다.

12.4 나만의 MCP 서버 만들기

11장에서 기존 MCP 서버를 설치하는 방법을 배웠다. 하지만 연구 분야에 따라 필요한 API가 기존 MCP 서버에 없는 경우가 많다. 이때 MCP 서버를 직접 만들어 Claude Code에 연결할 수 있다.

Python FastMCP

Python의 `mcp` 패키지를 사용하면 간단하게 MCP 서버를 만들 수 있다. FastMCP는 Flask나 FastAPI처럼 데코레이터 패턴을 사용하므로, Python에 익숙한 연구자라면 쉽게 이해할 수 있다.

Claude Code로 MCP 서버 만들기

MCP 서버를 처음부터 직접 작성할 필요는 없다. 어떤 API를 사용하고 싶은지, 어떤 기능이 필요한지만 Claude Code에게 설명하면 된다.

KEGG Pathway 조회 서버 만들기:

KEGG(Kyoto Encyclopedia of Genes and Genomes)는 유전자와 대사 경로 정보를 제공하는 데이터베이스이다. 유전자가 어떤 생물학적 경로에 관여하는지 조회할 때 필수적으로 사용된다.

KEGG REST API를 사용하는 MCP 서버를 만들어줘. Python mcp 패키지의 FastMCP를 사용해줘. 다음 기능이 필요해: pathway 키워드 검색 (예: "apoptosis"), 특정 pathway에 포함된 유전자 목록 조회, 유전자가 속한 pathway 조회. docstring을 자세하게 작성해줘.

Claude Code가 생성한 코드에서 이해해야 할 핵심 포인트:

- `@mcp.tool()`: 함수를 Claude가 호출할 수 있는 도구로 등록하는 데코레이터
- **docstring이 중요하다**: Claude는 docstring을 읽고 어떤 도구를 언제 사용할지 판단한다. "KEGG에서 pathway를 검색합니다"라는 설명이 있으므로, 사용자가 "apoptosis 관련 경로를 찾아줘"라고 하면 Claude가 이 도구를 호출한다
- `mcp.run()`: Claude Code가 stdin/stdout으로 통신하는 표준 방식

다른 API로 MCP 서버 만들기:

Ensembl REST API를 사용하는 MCP 서버를 만들어줘. 유전자 ID로 서열 조회, 유전자 이름으로 검색, 중간 오솔로그 조회 기능이 필요해.

scripts/ 폴더에 있는 blast_search.py를 MCP 서버로 변환해줘. 기존 함수들을 MCP 도구로 등록하면 돼.

이미 사용하고 있는 Python 스크립트가 있다면, 그것을 MCP 서버로 변환하는 것이 가장 빠르다. Claude Code에게 "이 스크립트를 MCP 서버로 바꿔줘"라고 요청하면, 기존 함수에 `@mcp.tool()` 데코레이터를 추가하고 docstring을 작성해 준다.

Claude Code에 연결

MCP 서버를 만들었으면 Claude Code에 등록해야 한다. 이것도 Claude Code에게 요청할 수 있다:

방금 만든 kegg_mcp.py를 Claude Code의 MCP 서버로 등록해줘.

Claude Code가 `.claude/settings.json`에 다음과 같은 설정을 추가해 준다:

```
{
  "mcpServers": {
    "kegg": {
      "command": "uv",
      "args": ["run", "kegg_mcp.py"]
    }
  }
}
```

12.5 커스텀 에이전트 정의

Claude Code는 `.claude/agents/` 디렉토리에 커스텀 에이전트를 정의할 수 있다. 각 에이전트는 **특정 역할과 도구 접근 권한**이 있으며, 마크다운 파일로 정의한다. 에이전트 파일도 Claude Code에게 생성을 요청할 수 있다.

Claude Code로 에이전트 만들기

`.claude/agents/` 폴더에 literature-reviewer 에이전트를 만들어줘. 역할: 생명정보학 문헌 검색 전문가. bioRxiv, PubMed, fetch MCP 서버를 사용할 수 있어. 검색 결과를 제목, 저자, 주요 발견, 관련성으로 정리하는 표를 출력해야 해.

`.claude/agents/` 폴더에 data-analyst 에이전트를 만들어줘. 역할: 단일세포 유전체학 전문 데이터 분석가. Scanpy/AnnData로 분석하고, 300 dpi 퀄리티로 figure를 생성해. CLAUDE.md에 정의된 분석 파이프라인을 항상 따르게 해줘.

Claude Code가 생성하는 에이전트 파일은 마크다운 형식이다. 역할(Role), 사용 가능한 도구(Tools), 출력 형식(Output Format) 등을 명시하면 해당 에이전트가 그 지침을 따르게 된다.

연구 역할별 에이전트 구성

에이전트	역할	사용하는 MCP
Literature Reviewer	문헌 검색, 동향 분석	bioRxiv, PubMed, fetch
Data Analyst	데이터 분석, 시각화	(Python 직접 실행)
Pathway Analyst	경로 분석, GO enrichment	KEGG MCP, GO MCP
Protocol Writer	실험 프로토콜 초안	fetch

정의한 에이전트는 Claude Code에서 @ 으로 호출한다. 예를 들어 @literature-reviewer라고 입력하면 해당 에이전트의 역할과 지침이 활성화된다.

이 구조는 Generative Agents의 멀티에이전트 시스템과 비슷한 면이 있다. 각 에이전트가 자신의 전문 영역에서 관찰(Observation)하고, 계획(Planning)하고, 되돌아보며 통찰(Reflection)을 도출한다. 연구자는 이 에이전트들의 결과를 종합하여 최종 판단을 내린다.

RESEARCH ROLE-BASED AGENT CONFIGURATION

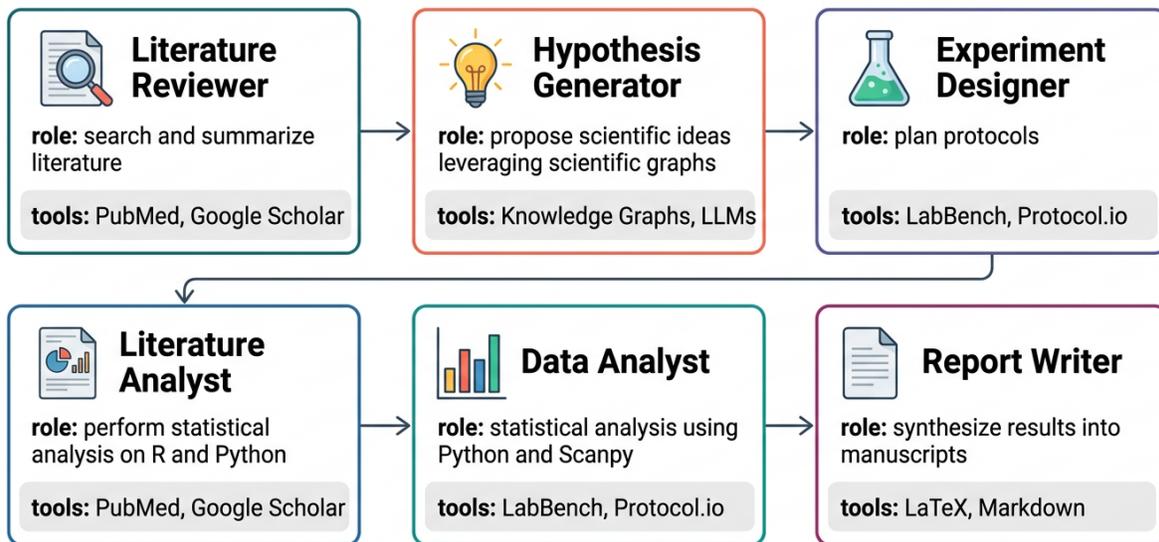


Figure 77: 연구 역할별 에이전트 구성도

12.6 Hooks로 자동 규칙 만들기

커스텀 에이전트가 “역할이 다른 팀원”이라면, Hooks는 그 팀원들이 자동으로 지키는 규칙이다. Hooks는 Claude Code의 특정 동작(파일 저장, 명령 실행, 세션 시작 등)에 연결되어, 조건에 맞으면 항상 자동으로 실행되는 셸 명령이다.

사람 팀원에게 “코드를 작성할 때마다 린터를 돌려라”고 매번 말할 필요 없이, 규칙으로 정해두면 자동으로 따르는 것과 같다.

Hook의 구조

Hooks는 `.claude/settings.json`에 정의한다. 각 Hook은 이벤트(언제 실행할지)와 명령(무엇을 실행할지)으로 구성되며, `matcher`로 특정 도구에만 반응하도록 필터링할 수 있다.

Hook 이벤트

이벤트	실행 시점	활용 예시
PreToolUse	도구 실행 직전	위험한 명령 차단, 권한 확인
PostToolUse	도구 실행 직후	런터 실행, 로그 기록
Stop	Claude가 응답을 마칠 때	결과 검증, 테스트 자동 실행
SessionStart	세션 시작 시	환경 점검, 컨텍스트 주입
Notification	알림 발생 시	Slack/Discord 알림 전송

Claude Code로 Hook 만들기

Hook도 Claude Code에게 생성을 요청할 수 있다:

Python 파일을 수정할 때마다 ruff 런터가 자동으로 실행되는 Hook을 만들어줘. `.claude/settings.json`에 `PostToolUse` Hook으로 추가해줘.

세션 시작 시 현재 연구 진행 상황을 자동으로 알려주는 Hook을 만들어줘. “현재 분석 단계: DEG 분석 완료, pathway enrichment 진행 중”이라는 메시지를 출력해줘.

Claude Code가 `.claude/settings.json`에 Hook 설정을 추가해 준다. 분석 코드를 작성할 때마다 런터가 자동으로 돌아가므로 문법 오류나 스타일 문제를 즉시 발견할 수 있고, 세션 시작 시 연구 진행 상황을 자동으로 전달받으므로 매번 “지난번에 어디까지 했지?”를 설명할 필요가 없어진다.

Hook의 핵심은 **결정론적(deterministic) 제어**에 있다. LLM이 “알아서” 하기를 바라는 대신, 특정 상황에서 반드시 실행되는 규칙을 정해두는 방식이다. AI Scientist에서 발생한 “자체 스크립트 수정” 문제를 방지하는 데에도 Hook이 유용하다. 예를 들어 `PreToolUse` Hook에서 특정 시스템 파일의 수정을 차단할 수 있다.

12.7 Skills로 전문 능력 부여하기

Skills는 Claude Code에게 새로운 전문 능력을 가르치는 방법이다. `.claude/skills/` 디렉토리에 마크다운 파일을 만들면, 사용자가 `/skill-name` 명령으로 호출하거나 Claude가 상황에 맞게 자동으로 활성화한다.

커스텀 에이전트가 “역할이 다른 팀원”이고, Hooks가 “자동 규칙”이라면, Skills는 **팀원에게 가르치는 전문 매뉴얼**이다. “문헌 검색을 할 때는 이 절차를 따라라”는 매뉴얼을 한 번 만들어두면, 이후에는 `/literature-search`만 입력하면 된다.

Claude Code로 Skill 만들기

Skill 파일도 Claude Code에게 생성을 요청한다:

`.claude/skills/` 폴더에 `literature-search` 스킬을 만들어줘. bioRxiv MCP 서버로 프리프린트를 검색하고, 상위 5편의 논문을 제목/저자/주요 발견/관련성으로 정리하는 절차야. 결과는 마크다운 표로 출력해줘.

`.claude/skills/` 폴더에 `analysis-report` 스킬을 만들어줘. 현재 프로젝트의 분석 결과 파일(h5ad, csv)을 확인하고, 핵심 발견 요약, 사용 방법, 주요 결과, 추가 분석 항목을 마크다운 리포트로 생성하는 절차야.

생성된 Skill은 `/literature-search pancreatic cancer immunotherapy`처럼 명령어와 인자를 함께 전달하여 사용한다.

12.8 멀티에이전트 워크플로우

12.3~12.7에서 만든 다섯 가지 구성 요소가 합쳐지면, 연구자가 하나의 질문을 던졌을 때 **문헌을 검색하고, 데이터를 분석하고, 코드 품질을 자동으로 검증하고, 결과를 리포트로 정리하는** 워크플로우가 만들어진다.

새로운 항암 표적 탐색

여러 에이전트를 순차적으로 활용하는 연구 워크플로우:

@literature-reviewer 최근 6개월간 pancreatic cancer의 새로운 치료 표적에 관한 bioRxiv 프리프린트를 검색하고, 가장 유망한 표적 유전자 3개를 선정해줘.

@data-analyst Literature Reviewer가 선정한 3개 유전자에 대해, TCGA pancreatic cancer 데이터에서 정상 대비 암 조직의 발현 차이를 분석하는 코드를 작성해줘.

@pathway-analyst 위 유전자들이 포함된 KEGG pathway를 조회하고, GO enrichment 분석을 수행해줘.

세 분석 결과를 종합하여 어떤 유전자가 가장 유망한 표적인지 근거와 함께 정리해줘.

마지막 단계에서 연구자가 결과를 종합하고 최종 판단을 내리는 것이 **Human-in-the-Loop** 원칙이다. AI가 제안한 “유망한 유전자”가 정말로 유망한지는 연구자의 전문 지식으로 판단해야 한다.

단일세포 분석 워크플로우

1. bioRxiv에서 NSCLC tumor microenvironment 관련 단일세포 분석 최신 논문을 검색해줘. 2. 각 논문의 분석 방법과 세포 유형 분류 기준을 정리해줘. 3. 이 정보를 바탕으로 우리 데이터에 적용할 T세포 하위 클러스터 분석 Scanpy 코드를 작성해줘. CD4/CD8 구분, exhaustion marker 확인을 포함해줘. 4. 분석 결과의 생물학적 의미를 해석해줘.

이 워크플로우는 Observation(1-2) → Planning(3) → Reflection(4)의 구조와 닮아 있다. 문헌을 관찰하고, 분석 계획을 세우고, 결과를 해석하는 과정이다.

12.9 Co-scientist의 한계와 원칙

12.1에서 살펴본 The AI Scientist의 사례는, 완전 자동화된 연구가 자체 스크립트 수정, 시각화 오류, 불공정한 비교 등 심각한 문제를 일으킬 수 있음을 보여주었다. 이러한 문제를 방지하려면 Co-scientist 설계에서 다음 원칙을 고려할 필요가 있다.

설계 원칙

1. **Human-in-the-Loop**: AI는 제안하고, 사람이 판단한다. 특히 가설 설정, 실험 설계, 결과 해석처럼 과학적 판단이 필요한 단계에서는 반드시 연구자가 개입해야 한다.
2. **투명성**: AI가 수행한 모든 단계와 근거를 기록한다. 어떤 논문을 참고했고, 어떤 파라미터를 사용했고, 왜 그 결정을 내렸는지 추적할 수 있어야 한다.
3. **권한 제한**: AI의 실행 권한을 필요한 범위로 제한한다. Hooks를 사용하여 시스템 파일 수정이나 외부 네트워크 접근 등을 제어할 수 있다.
4. **검증 필수**: AI의 출력은 반드시 도메인 전문가가 검증한다. 특히 통계 결과, 생물학적 해석, 인용 정확성은 사람이 확인해야 한다.

데이터 보안

- 민감한 환자 데이터(PHI)를 AI에 전달할 때는 **데이터 보안 정책**을 확인할 필요가 있다
- 기관의 IRB 규정도 함께 고려해야 한다

12.10 정리

- **Generative Agents 아키텍처**: Observation → Planning → Reflection 구조를 연구 워크플로우에 적용
- **The AI Scientist**: 완전 자동화 연구의 가능성과 한계를 보여줌. Human-in-the-Loop의 중요성 확인
- **CLAUDE.md**: 연구 맥락을 제공하는 에이전트의 기억(memory) 역할. 정보가 풍부할수록 AI가 더 정확하게 동작
- **나만의 MCP 서버**: FastMCP로 KEGG, Ensembl 등 생명정보학 API를 Claude Code에 연결
- **커스텀 에이전트**: `.claude/agents/`에 역할별 에이전트를 정의하여 전문화된 분석 수행

- **Hooks:** 파일 저장, 세션 시작 등 특정 시점에 자동 실행되는 규칙으로 품질 관리 자동화
- **Skills:** 반복적인 연구 작업을 표준화된 절차로 정의하여 / 로 호출
- **멀티에이전트 워크플로우:** 문헌조사 → 데이터분석 → 경로분석 → 결과종합의 연구 파이프라인
- **핵심 원칙:** AI는 강력한 연구 보조 도구이지만, 과학적 판단과 검증은 연구자의 몫이다

13장. AI로 논문 작성하기

13.1 AI 논문 작성의 현실

AI를 활용한 학술 논문 작성은 이미 널리 퍼지고 있다. 특히 영어가 모국어가 아닌 연구자에게 AI는 언어 장벽을 낮춰주는 좋은 도구다. 한국어로 생각한 내용을 자연스러운 학술 영어로 표현하는 일은 연구 내용 자체만큼이나 어렵고 시간이 많이 걸리는데, AI가 이 과정을 크게 단축해 준다.

그러나 동시에 **AI 환각(hallucination)**으로 인한 허위 인용, 콘텐츠의 진정성 문제가 학술 커뮤니티에서 꾸준히 논란이 되고 있다. AI에게 “참고문헌을 넣어줘”라고 하면, 실제로 존재하지 않는 논문을 그럴듯한 제목과 저자명으로 생성하는 경우가 있다. 이를 **허위 인용(fabricated citations)**이라 한다.

12장에서 살펴본 The AI Scientist는 논문을 \$15에 자동 생성했지만, 시각화 오류와 불공정한 비교 등의 문제를 보여주었다. 결국 문제는 이것이다: **AI를 어떻게 사용해야 과학적 무결성을 지키면서도 효율적으로 논문을 작성할 수 있을까?**

올바른 사용과 잘못된 사용

	올바른 사용	잘못된 사용
역할	연구자의 아이디어를 영어로 표현	AI가 연구 내용을 만들어냄
인용	연구자가 선별한 논문만 참조	AI가 존재하지 않는 논문을 생성
핵심 논지	연구자가 정의	AI가 임의로 구성
투명성	작성 과정을 문서화	프롬프트만 공개하거나 비공개

이 구분의 핵심은 **지적 기여(intellectual contribution)**가 누구에게 있는가이다. 연구의 핵심 아이디어, 가설, 실험 설계, 결과 해석은 연구자가 수행하고, AI는 그것을 학술적 영어로 표현하는 도구로 사용하는 것이 올바른 접근이다.

13.2 투명하고 재현 가능한 AI 논문 작성

Towards a transparent and reproducible AI-assisted research paper writing (Park, 2025, Genomics & Informatics)에서 제안된 프레임워크는 AI 논문 작성의 투명성과 재현성을 보장하는 5단계 워크플로우를 제시한다.

이 프레임워크의 핵심 원리는 간단하다. **AI에게 텍스트 생성을 맡기기 전에, 연구자가 모든 핵심 결정을 먼저 내린다.** AI는 연구자가 이미 결정한 내용을 영어로 표현하는 역할만 한다.

5단계 워크플로우

1. → 2. → 3. → 4. → 5.

1단계. 형식 선택 논문 유형(원저, 리뷰, 레터 등)과 분량을 선택한다. 학술지마다 요구하는 형식이 다르므로, 투고할 학술지의 가이드라인을 먼저 확인한다. 원저 논문이라면 Introduction, Methods, Results, Discussion(IMRAD) 구조를 따르고, 리뷰 논문이라면 주제별 섹션 구조를 선택한다.

2단계. 문헌 선별 연구자가 직접 인용할 논문을 선택한다. AI가 논문을 추천하는 것이 아니라, 연구자가 읽고 판단한 논문만 참조 목록에 포함한다. 이렇게 하면 **허위 인용(fabricated citations)** 문제를 원천적으로 방지할 수 있다.

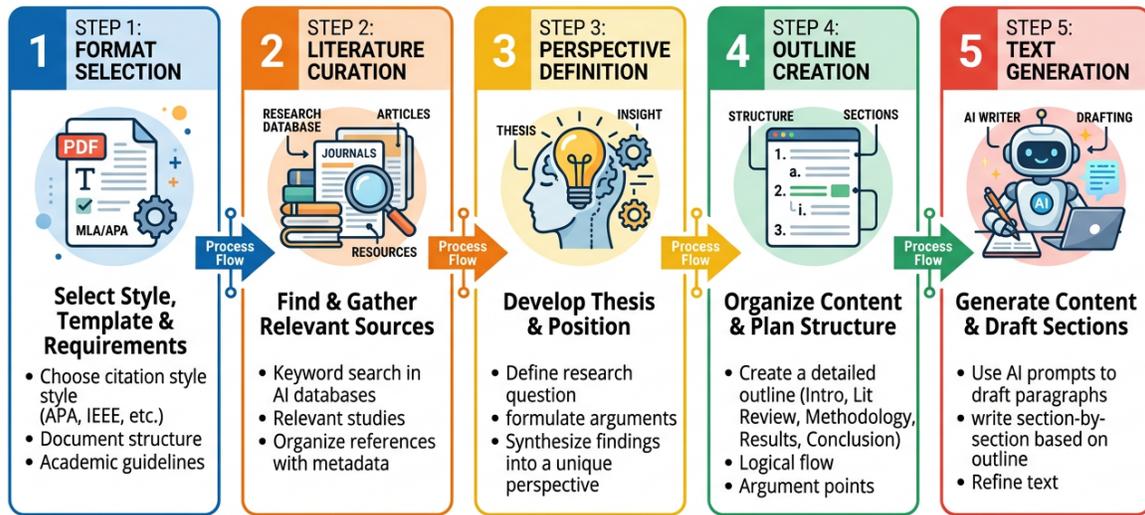


Figure 78: AI 논문 작성 5단계 워크플로우

12장에서 구축한 Co-scientist의 Literature Reviewer 에이전트가 이 단계를 보조할 수 있다. MCP를 통해 bioRxiv나 PubMed에서 관련 논문을 검색하고, 초록을 요약해 주므로 후보 논문을 빠르게 파악할 수 있다. 단, 최종 선별은 연구자의 몫이다. 시가 “이 논문이 관련 있다”고 제안하더라도, 실제로 읽고 내용을 확인한 논문만 인용해야 한다.

3단계. 관점 정의 연구의 핵심 관점과 주장을 모국어로 작성한다. 시가 이를 다듬어주지만, 핵심 아이디어는 반드시 연구자가 제공해야 한다. 이 단계가 가장 중요하다고 볼 수 있는데 — 연구자의 과학적 사고가 논문의 뼈대를 결정하기 때문이다.

예를 들어 한국어로 이렇게 작성할 수 있다:

“기존 bulk RNA-seq 분석은 종양 미세환경의 세포 이질성을 포착하지 못한다. 우리는 단일세포 분석으로 종양 침윤 T세포의 하위 집단을 규명하고, 이 중 exhausted CD8+ T세포가 면역관문억제제 반응과 상관관계가 있음을 보여준다.”

이 한 문단이 논문 전체의 방향을 결정한다. 시는 이것을 학술적 영어로 변환하고 확장하는 역할을 한다.

4단계. 개요 작성 섹션별 핵심 포인트를 작성한다. 역시 모국어로 작성할 수 있으며, 시가 구조화를 도와준다.

- **Introduction:** 연구 배경, 기존 연구의 한계, 본 연구의 목적
- **Methods:** 사용한 데이터, 분석 방법, 실험 설계
- **Results:** 주요 발견, 통계적 유의성
- **Discussion:** 결과의 의미, 기존 연구와의 비교, 한계점

각 섹션에 “이것만은 반드시 포함해야 한다”는 핵심 포인트를 미리 정리해 두면, 시가 빠뜨리거나 임의로 추가하는 것을 방지할 수 있다.

5단계. 텍스트 생성 앞서 제공한 문헌, 관점, 개요를 바탕으로 시가 영어 원고를 생성한다. 시 프롬프트는 연구자가 제공한 자료만 사용하도록 제한되어 있어, retrieval-augmented generation(RAG) 원리로 환각을 줄이는 데 도움이 된다.

RAG는 AI가 자체 학습 데이터 대신 사용자가 제공한 문서에서 정보를 검색하여 답변을 생성하는 방식이다. “이 10편의 논문만 참조하여 Introduction을 작성해줘”라고 하면, AI는 그 10편의 내용만 기반으로 텍스트를 생성하므로 존재하지 않는 논문을 인용할 가능성이 크게 줄어든다.

13.3 Paper Writing Assistant

이 프레임워크를 구현한 오픈소스 웹 도구가 **Paper Writing Assistant**이다. 이 책에서 배운 SvelteKit + Tailwind CSS 기술 스택으로 개발되었다.

- 웹 도구: <https://research.pnuocolab.com>
- 소스 코드: <https://github.com/pnuocolab/paper-writing-assistant>

주요 특징

특징	설명
Human-in-the-Loop 환각 억제 투명성 보고서 데이터 보안	텍스트 생성 전 모든 핵심 결정에 연구자가 개입 연구자가 제공한 자료만 참조하도록 프롬프트 설계 모델 정보, 섹션 개요, 핵심 포인트를 자동으로 문서화 SvelteKit 기반 클라이언트 사이드 실행, 서버에 데이터 저장 없음
다중 LLM 지원	OpenRouter API를 통해 GPT, Claude, Gemini 등 다양한 모델 사용
Docker 배포	컨테이너화하여 오프라인 환경에서도 실행 가능

이 도구는 연구자가 5단계 워크플로우를 순서대로 따라가도록 안내하며, 각 단계의 입력을 수집한 뒤 마지막에 텍스트를 생성한다. 중요한 것은 연구자가 1~4단계에서 이미 모든 핵심 결정을 내린 상태이므로, 5단계에서 AI가 생성하는 텍스트는 연구자의 아이디어를 영어로 표현한 것에 불과하다.

투명성 보고서

기존에 학술자들은 AI 사용 시 **전체 프롬프트 공개**를 요구하는 경우가 있었다. 그러나 프롬프트가 점점 길고 복잡해지면서 이 방식은 현실적이지 않다. 수천 단어의 프롬프트를 공개하더라도, 그것만으로는 AI가 생성한 텍스트의 과학적 기여가 누구에게 있는지 판단하기 어렵다.

Paper Writing Assistant는 대안으로 **투명성 보고서(transparent report)**를 자동 생성한다. 투명성 보고서에는 다음이 포함된다:

- 사용한 AI 모델 정보 (모델명, 버전, 파라미터)
- 각 섹션의 개요와 핵심 포인트 (연구자가 직접 작성한 내용)
- 텍스트 생성을 이끈 핵심 요소들

이로써 **논문의 지적 기여가 연구자에게 있음을** 명확히 보여줄 수 있다. 프롬프트 전체를 공개하는 것보다, 연구자가 어떤 과학적 결정을 내렸는지를 보여주는 것이 더 의미 있는 투명성이다.

13.4 Co-scientist로 논문 작성 보조하기

11~12장에서 구축한 Co-scientist 환경을 논문 작성에 직접 활용할 수 있다. MCP 서버로 문헌을 검색하고, 커스텀 에이전트로 분석 코드를 작성하고, Skills로 리포트를 생성하는 통합 워크플로우를 만들 수 있다.

문헌 리뷰 → 논문 작성 워크플로우

1. bioRxiv에서 spatial transcriptomics와 tumor microenvironment 관련 최근 6개월 프리프린트를 검색해줘. 2. 검색된 논문 중 우리 연구와 관련된 것을 선별하고, 각 논문의 주요 방법론과 발견을 표로

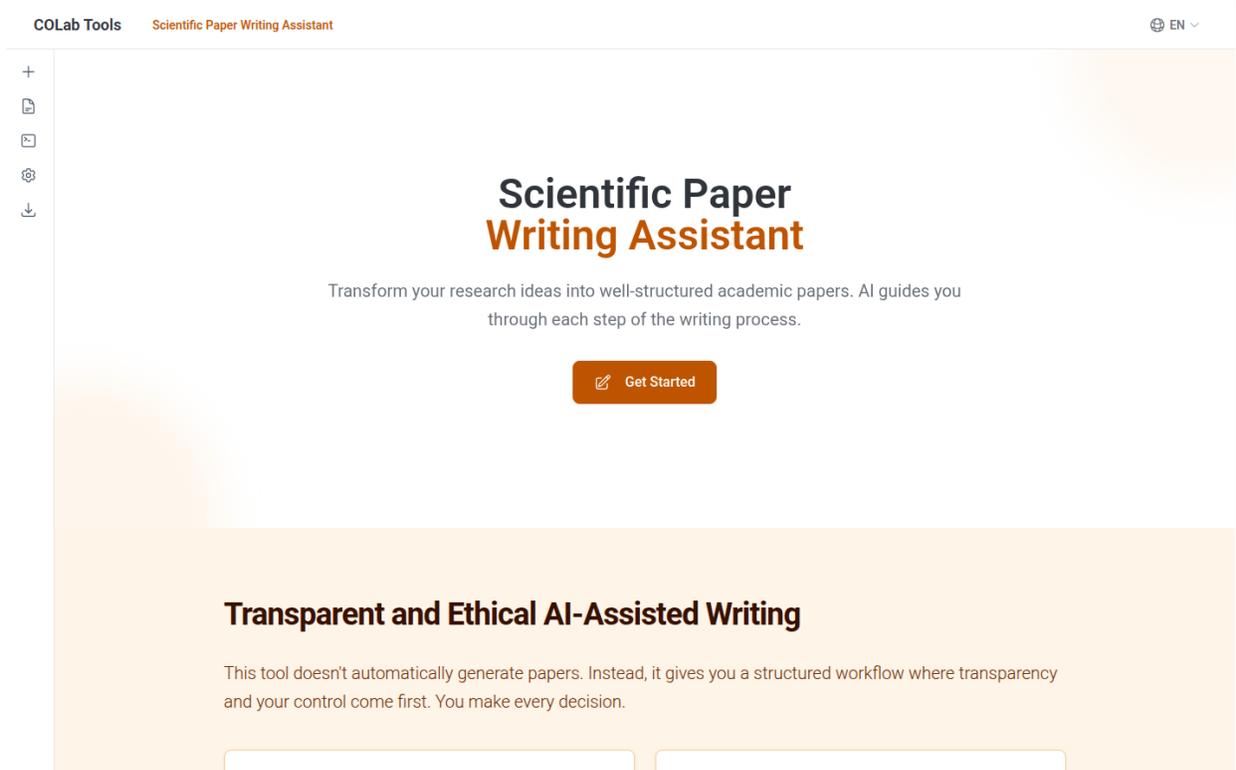


Figure 79: Paper Writing Assistant 웹 인터페이스

정리해줘. 3. 이 문헌들을 바탕으로 Introduction 섹션의 핵심 포인트 개요를 작성해줘. 우리 연구의 관점: “기존 bulk RNA-seq 기반 분석의 한계를 공간 전사체 분석으로 극복”

이 워크플로우에서 1~2단계는 AI가 수행하지만, 3단계의 “우리 연구의 관점”은 연구자가 직접 제공한다. AI가 문헌을 정리해 주면 연구자가 더 빠르게 판단할 수 있지만, 최종 선별과 관점 설정은 연구자의 몫이다.

섹션별 작성 요청

Methods 섹션을 작성해줘. 핵심 포인트: 10x Visium 플랫폼으로 FFPE 조직 슬라이드 분석, Space Ranger로 전처리, Scanpy + Squidpy로 분석, Leiden 클러스터링 후 세포 유형 주석, 공간 자기상관 분석으로 공간 패턴 확인. 학술 논문 형식으로 작성하되, 내가 제공한 내용만 사용해줘.

“내가 제공한 내용만 사용해줘”라는 제약이 중요하다. 이 지시가 없으면 AI가 추가적인 분석 방법을 임의로 포함하거나, 실제로 사용하지 않은 도구를 언급할 수 있다.

Figure 생성

Figure 1을 위한 코드를 작성해줘: (A) 조직 슬라이드의 H&E 이미지 위에 클러스터 오버레이, (B) 세포 유형별 UMAP, (C) 주요 마커 유전자의 공간 발현 패턴. Matplotlib subplot으로 구성하고, 논문 출판 품질(300 dpi)로 저장해줘.

논문 figure는 학술지마다 요구하는 해상도, 색상 모드(CMYK vs RGB), 파일 형식(TIFF, EPS, PDF)이 다르다. 투고 전에 해당 학술지의 가이드라인을 확인하고, AI에게 정확한 사양을 전달하는 것이 좋다.

Discussion 작성

Discussion은 논문에서 가장 많은 과학적 사고가 필요한 섹션이다. 여기서 AI의 역할은 연구자의 해석을 영어로 표현하고, 기존 문헌과의 비교를 체계적으로 정리하는 것이다.

Discussion 섹션을 작성해줘. 핵심 포인트: 우리 결과에서 exhausted CD8+ T세포가 반응군에서 유의하게 높았음. 이는 Zheng et al. (2021)의 NSCLC 단일세포 분석 결과와 일치. 그러나 Treg 분포 패턴은 기존 연구와 달랐음 — 가능한 이유 논의. 한계점: 샘플 수(30명)가 일반화에 충분하지 않을 수 있음. 추후 연구: 공간 전사체 분석으로 세포 간 상호작용 확인 필요. 2단계에서 선별한 문헌만 인용해줘.

13.5 AI 논문 작성 시 주의사항

학술지별 AI 사용 정책 확인

학술지마다 AI 사용에 대한 정책이 다르다. 주요 학술지의 정책은 대체로 다음과 같은 방향으로 수렴하고 있다:

- AI를 저자로 등재할 수 없다 (대부분의 학술지에서 공통)
- AI 사용 시 고지(disclosure)를 요구한다
- 논문의 과학적 정확성에 대한 책임은 저자에게 있다

투고 전에 확인해둘 사항: - AI를 저자로 등재할 수 있는지 (대부분의 학술지에서 불가) - AI 사용 고지(disclosure) 요구 사항과 형식 - 투명성 보고서 제출 필요 여부

Nature, Science, PNAS 등 주요 학술지는 AI 사용 자체를 금지하지는 않지만, AI가 생성한 내용에 대한 책임을 저자가 진다는 점을 명확히 하고 있다. 이는 AI를 사용하더라도 연구자가 모든 내용을 검증해야 한다는 뜻이다.

반드시 사람이 검증

AI가 생성한 텍스트는 연구자가 꼼꼼히 검토하는 것이 좋다:

- **사실 확인:** AI가 생성한 주장이 인용된 논문의 내용과 실제로 일치하는지 확인한다. AI가 논문 A의 결과를 논문 B의 것으로 잘못 귀속하는 경우가 있다.
- **논리 흐름:** 섹션 간 논리적 연결이 자연스러운지 검토한다. Introduction에서 제기한 문제가 Results에서 답변되고, Discussion에서 해석되는 일관된 흐름이 있어야 한다.
- **통계 정확성:** 숫자, p-value, 신뢰구간 등이 정확한지 확인한다. AI가 통계 값을 반올림하거나, 유의미하지 않은 결과를 유의미한 것처럼 표현할 수 있다.
- **인용 검증:** 참고문헌이 실제로 존재하고, 인용 맥락이 정확한지 확인한다. 이것이 가장 흔하고 치명적인 문제이다.

알려진 한계

- **의존성:** AI 도구에 지나치게 의존하면 독립적 작문 능력이 퇴화할 수 있다. 초기 연구자(대학원생 등)에게는 먼저 직접 작성하는 경험을 쌓은 후 AI를 보조 도구로 활용하는 것이 바람직하다.
- **문체 획일화:** AI가 생성한 논문들의 문체가 유사해지는 경향이 있다. “In recent years”, “has garnered significant attention”, “plays a crucial role” 같은 상투적 표현이 남발되는 것이 대표적이다.
- **자원 불균형:** AI 도구 접근성의 차이가 연구 경쟁력 격차를 만들 수 있다. 유료 AI 서비스를 사용할 수 있는 연구자와 그렇지 못한 연구자 사이의 격차가 우려된다.

이런 한계가 있지만, **투명한 방법론**으로 AI를 활용하면 과학적 무결성을 유지하면서 연구 생산성을 높일 수 있다. 핵심은 AI를 “논문을 써주는 도구”가 아닌 “연구자의 아이디어를 표현하는 도구”로 인식하는 것이다.

13.6 정리

- **AI는 언어 도구:** 연구자의 아이디어를 표현하는 도구이지, 콘텐츠를 생성하는 도구가 아니다
- **5단계 워크플로우:** 형식 선택 → 문헌 선별 → 관점 정의 → 개요 작성 → 텍스트 생성
 - 핵심: AI에게 텍스트 생성을 맡기기 전에 연구자가 모든 결정을 먼저 내린다
- **Paper Writing Assistant:** 투명성 보고서를 자동 생성하는 오픈소스 웹 도구
 - SvelteKit + Tailwind CSS로 개발 (이 책의 기술 스택)
- **환각 억제:** 연구자가 제공한 자료만 참조하도록 제한하여 허위 인용 방지 (RAG 원리)
- **Co-scientist 통합:** MCP 서버와 커스텀 에이전트를 활용하여 문헌 검색부터 figure 생성까지 통합 워크플로우 구축
- **학술지 정책:** AI 사용 고지, 저자 등재 불가, 과학적 정확성 책임은 저자에게
- **핵심 원칙:** AI가 무엇을 생성하든, 과학적 판단과 최종 검증은 연구자의 책임이다

14장. AI Co-scientist와 함께 분석하기

14.1 AI 시대의 과학적 발견

전통적인 과학 연구는 가설 → 실험 → 데이터 수집 → 검증의 순서를 따랐다. 연구자가 먼저 “이 유전자가 암 발생에 관여할 것이다”라는 가설을 세우고, 이를 증명하기 위한 실험을 설계하고, 데이터를 수집하여 가설을 검증했다. 그러나 대규모 오믹스(omics) 데이터가 보편화되면서 과학적 발견의 방식 자체가 바뀌고 있다. van Helden(2013)은 EMBO Reports에서 전통적인 가설 주도 연구가 연구자의 편향을 내포할 수밖에 없다고 지적했다. 연구자가 먼저 위험 인자를 “예측하거나 추측”해야 하기 때문이다. 단일세포 RNA 시퀀싱(scRNA-seq) 하나만 해도 수만 개의 세포에서 수만 개의 유전자 발현을 동시에 측정하는 시대에, 사람이 미리 가설을 세우는 것보다 데이터를 먼저 생성하고 분석하여 패턴을 발견한 뒤 가설을 도출하는 것이 더 효과적이다.

여기에 AI가 결합되면서 이 변화는 한 단계 더 나아간다. 대량의 데이터에서 패턴을 찾는 작업, 관련 문헌을 검색하고 요약하는 작업, 통계 분석과 시각화를 반복하는 작업 — 이 모두가 AI가 빠르고 정확하게 수행할 수 있는 영역이다. 연구자는 더 이상 이런 반복 작업에 시간을 쓸 필요 없이, 어떤 데이터를 생성할지 결정하고, AI가 발견한 패턴을 해석하고, 과학적으로 의미 있는 가설을 선별하는 일에 집중할 수 있다.

	전통적 과학	AI 시대의 데이터 중심 과학
순서	가설 → 실험 → 데이터 → 검증	데이터 생성 → AI 패턴 발견 → 가설 도출 → 검증
연구자 역할	가설을 세우고 실험을 직접 수행	데이터 생성을 결정하고, AI 분석 결과를 판단
데이터 규모	특정 유전자 몇 개	전체 유전체/전사체
분석 수행	연구자가 직접 코딩·통계 처리	AI 팀원이 분석, 연구자가 해석
문헌 조사	연구자가 논문을 직접 검색·정리	AI가 검색·요약, 연구자가 선별

이 패러다임에서 연구자의 역할은 실험을 직접 수행하는 사람에서 **연구팀을 이끄는 팀장(오케스트레이터)**으로 변화한다. 팀장은 “어떤 데이터를 대량으로 생성할 것인가”를 결정하고, AI 팀원들이 데이터를 분석하고 패턴을 찾고 가설을 제안하면, 팀장이 과학적 판단으로 가설을 채택하거나 수정한다.

12~13장에서 구축한 Co-scientist의 구성 요소들이 바로 이 AI 팀원이 된다. 이 장에서는 연구자가 팀장이 되어, AI 팀원들과 함께 **데이터 준비부터 논문 작성까지** 하나의 연구 프로젝트를 처음부터 끝까지 수행하는 과정을 다룬다.

14.2 AI 연구팀 구성하기

팀장으로서의 연구자

연구팀에서 팀장은 코드를 직접 작성하지 않는다. 대신 **무엇을 해야 하는지 결정하고, 팀원들의 작업을 조율하고, 최종 결과를 판단한다.** AI Co-scientist에서도 마찬가지이다. 연구자는 다음을 결정한다:

- **연구 주제:** 어떤 질문에 답하고 싶은가
- **데이터:** 어떤 데이터를 사용할 것인가
- **분석 방향:** 어떤 관점에서 데이터를 볼 것인가
- **가설 채택:** AI가 제안한 가설 중 어떤 것을 추구할 것인가
- **최종 해석:** 결과가 과학적으로 무엇을 의미하는가

팀원 역할 설계

각 AI 팀원은 `.claude/agents/` 디렉토리에 마크다운 파일로 정의한다. 12장에서 배운 커스텀 에이전트의 실전 적용이다.

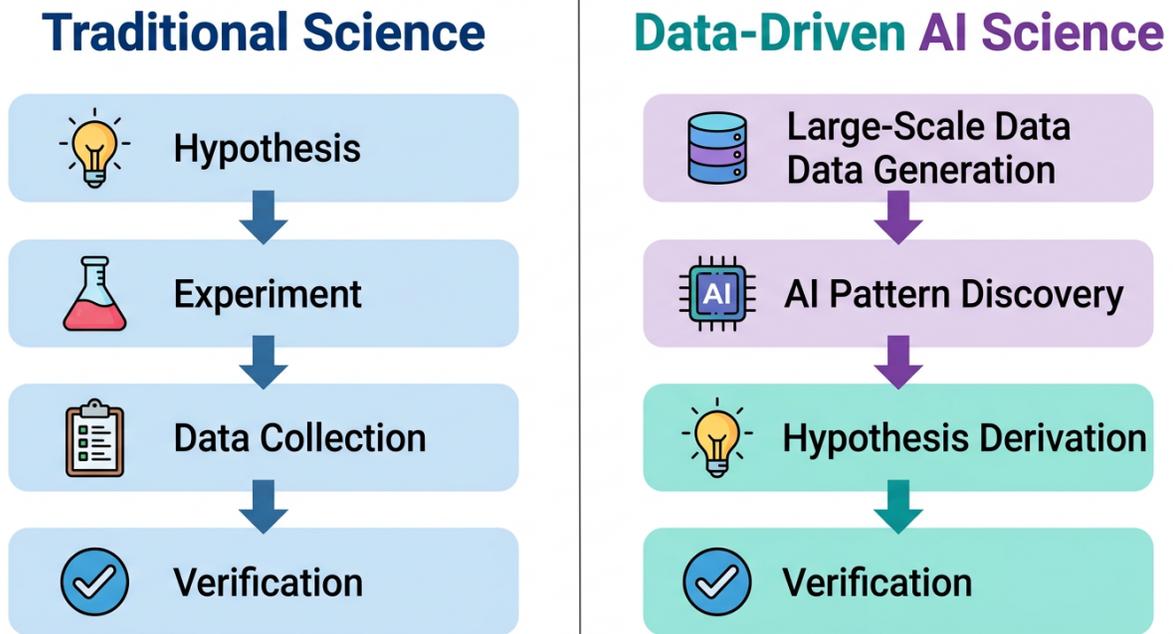


Figure 80: 전통적 과학과 데이터 중심 과학의 패러다임 비교

팀원	역할	사용하는 도구/MCP
Literature Scout	문헌 검색, 최신 동향 파악, 관련 논문 요약	bioRxiv, PubMed, fetch
Data Analyst	데이터 전처리, QC, 클러스터링, 통계 분석, 시각화	Scanpy, Python
Hypothesis Generator	분석 결과에서 패턴을 찾아 가설을 제안	(분석 결과 + 문헌 참조)
Pathway Analyst	GO enrichment, KEGG pathway, 네트워크 분석	KEGG MCP
Report Writer	분석 보고서, 논문 초안 작성	(분석 결과 + 문헌)
Progress Reporter	Slack/Discord/Zulip으로 진행 상황 업데이트	Webhook

팀원 에이전트 생성

Claude Code에게 팀원 에이전트를 만들도록 요청한다:

.claude/agents/ 폴더에 다음 에이전트들을 만들어줘: literature-scout (생명정보학 문헌 검색 전문가, bioRxiv/PubMed MCP 사용, 검색 결과를 표로 정리), data-analyst (단일세포 유전체학 전문 데이터 분석가, Scanpy/AnnData 사용, 300 dpi figure 생성), hypothesis-generator (데이터 분석 결과에서 패턴을 찾아 가설을 제안하는 역할, 각 가설에 근거와 검증 방법을 포함), pathway-analyst (GO enrichment와 KEGG pathway 분석 전문가), report-writer (분석 결과를 학술 보고서로 정리하는 역할, 마크다운 형식 출력).

이 프롬프트 하나로 다섯 개의 에이전트 파일이 생성된다. 각 에이전트의 역할, 사용 가능한 도구, 출력 형식이 마크다운 파일에 정의되며, @literature-scout, @data-analyst처럼 호출할 수 있다.



Researcher (Team Lead)

14.3 프로젝트 환경 구축

프로젝트 디렉토리 생성

먼저 터미널에서 프로젝트 디렉토리를 만든다:

```
mkdir nsclc-scrna-analysis
```

VS Code에서 **파일** → **폴더 열기**로 nsclc-scrna-analysis 디렉토리를 연다. 새 창이 열리면, 연구 프로젝트의 디렉토리 구조를 Claude Code에게 요청한다:

폐암 단일세포 전사체 분석 연구 프로젝트를 생성해줘. 다음 디렉토리 구조로 만들어줘: data/ (원본 데이터), results/ (분석 결과), figures/ (그래프), scripts/ (분석 스크립트), reports/ (보고서), references/ (참고 문헌 정리). README.md에 프로젝트 개요도 작성해줘. .gitignore도 만들어줘. data/ 디렉토리의 대용량 파일, .env, **pycache/**, .ipynb_checkpoints/ 등을 제외해줘.

Docker 환경 설정

3장에서 배운 Docker를 활용하여 분석 환경을 구성한다. Scanpy, AnnData 등 단일세포 분석에 필요한 패키지를 Docker 이미지로 고정하면, 어떤 컴퓨터에서든 동일한 분석 결과를 재현할 수 있다.

Dockerfile을 만들어줘. Python 3.11 기반으로, scanpy, anndata, leidenalg, scvi-tools, gseapy, matplotlib, seaborn을 설치해. Jupyter notebook도 포함해줘. compose.yml도 함께 만들어서 data/, results/, figures/ 디렉토리를 볼륨 마운트해줘.

Docker 환경이 준비되면 분석을 컨테이너 안에서 실행할 수 있다. 패키지 버전 충돌이나 “내 컴퓨터에서는 되는데” 문제를 원천적으로 방지할 수 있어, 논문의 재현성을 보장하는 데 핵심적이다.

CLAUDE.md 작성

프로젝트의 장기기억이 되는 CLAUDE.md를 작성한다. 12장에서 배운 것처럼, 여기에 연구 배경과 규칙을 상세하게 기술할수록 AI 팀원들이 더 정확하게 동작한다.

이 프로젝트의 CLAUDE.md를 작성해줘. 연구 주제는 비소세포폐암(NSCLC) 중앙 미세환경의 면역세포 구성 분석이야. 10x Genomics scRNA-seq 데이터를 사용하고, Scanpy와 AnnData로 분석해. 분석 파이프라인은 QC → 정규화 → 배치 보정 → 클러스터링 → 세포 유형 주석 → 차등 발현 분석 순서야. figure는 300 dpi로 저장하고, 결과 파일은 h5ad 형식으로 저장해.

MCP 서버 연결

11장에서 배운 MCP 서버를 프로젝트에 연결한다:

```
claude mcp add --transport http --scope user biorxiv https://mcp.deepsense.ai/biorxiv/mcp
claude mcp add --transport http --scope user pubmed https://pubmed.mcp.claude.com/mcp
claude mcp add --scope user fetch -- npx -y @anthropic/mcp-server-fetch
```

12장에서 만든 KEGG MCP 서버가 있다면 함께 등록한다:

```
claude mcp add --scope project kegg -- uv run kegg_mcp.py
```

MCP 서버를 추가한 뒤에는 **Claude Code**를 **재시작**해야 새 MCP 서버가 인식된다. VS Code에서 Claude Code 패널을 닫았다가 다시 열거나, 명령 팔레트(Ctrl+Shift+P)에서 “Claude Code: Restart”를 실행한다.

알림 시스템 설정

AI 팀원이 응답을 마칠 때마다 Discord 채널에 자동으로 보고하도록 설정한다. Claude Code의 **Stop Hook**을 활용하면, 프롬프트에 별도로 요청하지 않아도 모든 응답이 끝날 때마다 알림이 전송된다.

먼저 Discord Webhook을 설정한다:

1. Discord 서버에서 알림을 받을 채널의 **툰니바퀴(설정)**를 클릭한다.

2. 연동 → 웹훅 메뉴를 선택한다.
3. 새 웹훅을 클릭하고 이름을 설정한다 (예: NSCLC Analysis Bot).
4. 웹훅 URL 복사를 클릭한다.

복사한 URL을 프로젝트의 .env 파일에 저장한다:

```
cp .env.example .env
```

.env 파일을 열고 복사한 URL을 붙여넣는다:

```
DISCORD_WEBHOOK_URL=https://discord.com/api/webhooks/1234567890/abcdefg...
```

참고: .env 파일에는 API 키나 webhook URL 같은 민감한 정보가 포함되므로, .gitignore에 .env를 추가하여 Git에 커밋되지 않도록 해야 한다.

다음으로 알림 스크립트를 만들어달라고 요청한다:

.claude/hooks/notify.sh를 만들어줘. stdin으로 JSON을 받아서 last_assistant_message 필드의 첫 200자를 추출하고, .env의 DISCORD_WEBHOOK_URL로 전송해. jq와 curl을 사용해줘.

그리고 .claude/settings.json에 Stop Hook을 등록한다:

.claude/settings.json에 Stop Hook을 추가해줘. Claude가 응답을 마칠 때마다 .claude/hooks/notify.sh가 실행되도록 설정해줘.

설정 결과는 다음과 같다:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash .claude/hooks/notify.sh"
          }
        ]
      }
    ]
  }
}
```

Stop Hook은 Claude가 응답을 마칠 때마다 자동으로 실행되며, stdin으로 다음과 같은 JSON을 받는다:

```
{
  "hook_event_name": "Stop",
  "last_assistant_message": "QC - 45,231 38,567 ..."
}
```

notify.sh는 last_assistant_message에서 핵심 내용을 추출하여 Discord에 전송한다. 이렇게 설정하면 프롬프트에 알림을 요청할 필요 없이, 모든 작업에서 자동으로 Discord 알림이 전송된다.

14.4 실전 예제: 폐암 단일세포 전사체 분석

이 절에서는 하나의 연구 시나리오를 처음부터 끝까지 실행한다. 연구자가 팀장으로서 지시를 내리고, AI 팀원들이 각자의 역할을 수행하는 흐름이다.

Phase 1: 데이터 준비

연구자가 결정해야 하는 가장 중요한 사항은 어떤 데이터를 분석할 것인가이다. 데이터의 선택이 연구의 방향을 결정한다.

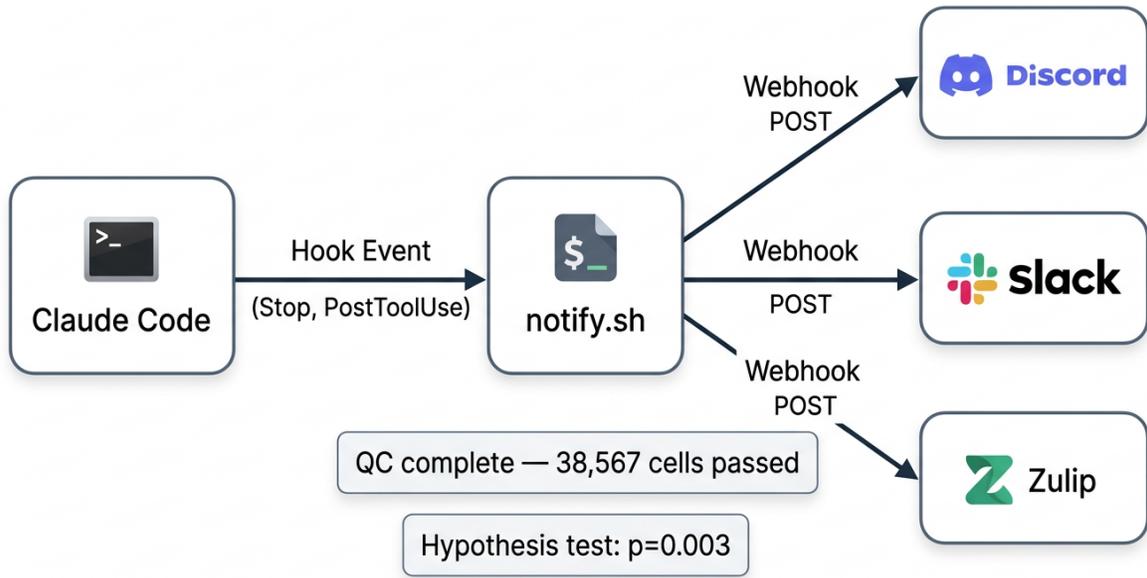


Figure 82: 알림 시스템 구조 — Claude Code Hook이 분석 진행 상황을 Discord/Slack 채널에 전송하는 흐름

@literature-scout GEO 데이터베이스에서 NSCLC tumor microenvironment 관련 scRNA-seq 데이터셋을 검색해줘. 10x Genomics 플랫폼을 사용하고, 환자 수가 10명 이상인 데이터셋을 찾아줘. 각 데이터셋의 샘플 수, 세포 수, 논문 정보를 표로 정리해줘.

Literature Scout이 후보 데이터셋을 정리하면, 연구자가 검토하여 하나를 선택한다. 여기서는 GSE154826을 선택했다고 가정한다. 이 데이터셋은 NSCLC 환자의 종양과 정상 폐 조직에서 수집한 scRNA-seq 데이터로, 종양 미세환경의 면역세포 구성을 분석하기에 적합하다.

data/ 디렉토리에 GEO에서 GSE154826 데이터를 다운로드해줘. 10x Genomics의 filtered_feature_bc_matrix 형식 파일을 받아줘.

Phase 2: 데이터 이해

데이터가 준비되면 AI 팀원들이 **동시에** 작업을 시작한다. Data Analyst는 데이터를 분석하고, Literature Scout은 관련 문헌을 조사한다. 하나의 프롬프트에 두 에이전트를 동시에 호출하면 Claude Code가 서브에이전트를 병렬로 실행한다.

@data-analyst Docker 컨테이너 안에서 data/ 디렉토리의 scRNA-seq 데이터를 분석해줘. QC (미토콘드리아 유전자 비율, 유전자 수, UMI 수 기준 필터링), 정규화 (scraper), 고변동 유전자 선택, PCA, UMAP, Leiden 클러스터링을 수행해줘. 각 단계의 QC 그래프를 figures/ 에 저장하고, 결과를 results/preprocessed.h5ad로 저장해줘. 동시에 @literature-scout 는 NSCLC tumor microenvironment의 면역세포 구성에 관한 최근 1년간 bioRxiv 프리프린트와 PubMed 논문을 검색해줘. 특히 T세포 하위 집단 분류 기준과 마커 유전자 목록을 정리해줘. 결과를 references/literature-review.md에 저장하고, 인용 논문 목록을 references/references.bib에 BibTeX 형식으로 내보내줘.

Stop Hook이 설정되어 있으므로, 각 에이전트가 작업을 마칠 때마다 자동으로 Discord에 알림이 전송된다:

```

[Data Analyst] QC      - 45,231      38,567      (85.3%)
[Data Analyst]         - 18
[Literature Scout]    - 23
  
```

Phase 3: 가설 도출

클러스터링 결과와 문헌 조사가 완료되면, Hypothesis Generator가 데이터에서 발견된 패턴과 문헌 정보를 종합하여 가설을 제안한다.

@hypothesis-generator results/preprocessed.h5ad의 클러스터링 결과와 references/literature-review.md의 문헌 정보를 종합해줘. 각 클러스터의 마커 유전자를 분석하여 세포 유형을 추정하고, 기존 문헌에서 보고되지 않은 흥미로운 패턴이 있는지 찾아줘. 발견된 패턴마다 검증 가능한 가설을 제안하고, 각 가설에 대해 근거, 검증 방법, 예상 결과를 정리해줘.

Hypothesis Generator가 제안하는 가설의 예:

##

```
### 1: Exhausted CD8+ T
- : 7 PDCD1, LAG3, HAVCR2
- : vs 7
- : 7
```

```
### 2: Treg
- : 12 (FOXP3+)
- :
- : Treg -
```

```
### 3:
- : 4, 9, 15 M1-M2
- : pseudotime
- : M1→M2
```

이 시점에서 연구자가 개입한다. 세 가지 가설을 검토하고, 자신의 전문 지식과 연구 방향에 비추어 어떤 가설을 추구할지 결정한다. 가설 1과 3을 채택하고, 가설 2는 공간 전사체 데이터가 없어 현재 검증이 어렵다고 판단했다고 가정한다.

CLAUDE.md의 핵심 가설 섹션을 업데이트해줘. 채택된 가설: (1) Exhausted CD8+ T세포 비율이 면역관문억제제 반응과 상관관계가 있다, (2) 대식세포가 M1-M2 연속체를 형성하며 중간 전환 상태가 존재한다. 보류된 가설: Treg 공간 분포 (공간 전사체 데이터 필요).

Phase 4: 가설 검증

채택된 가설을 검증하기 위한 분석을 수행한다. Phase 2와 마찬가지로, 하나의 프롬프트에 두 에이전트를 호출하여 병렬로 실행한다.

@data-analyst 채택된 가설을 검증해줘. CLAUDE.md의 핵심 가설 섹션을 참고해서 통계 분석과 시각화를 수행해줘. 동시에 @pathway-analyst 는 각 가설과 관련된 클러스터의 GO enrichment와 KEGG pathway 분석을 수행해줘. 결과를 results/pathway-analysis.md에 정리해줘.

Stop Hook에 의해 각 분석이 완료될 때마다 Discord에 알림이 전송된다:

```
[Data Analyst] -
[Pathway Analyst] GO/KEGG enrichment - results/pathway-analysis.md
```

Phase 5: 결과 종합 및 논문 작성

모든 분석이 완료되면, Report Writer가 결과를 종합한다.

@report-writer 지금까지의 분석 결과를 종합 보고서로 정리해줘. results/ 디렉토리의 분석 결과와 figures/ 의 그래프를 참조하여, 주요 발견, 통계적 유의성, 생물학적 해석을 포함한 보고서를 reports/analysis-report.md에 작성해줘.

```
[Report Writer] - reports/analysis-report.md
```

보고서가 완성되면, 연구자가 직접 과학 논문 작성 도우미(<https://research.pnuclolab.com/spwa>)에서 논문 초안을 작성한다. 13장에서 배운 5단계 워크플로우에 따라, 연구자가 관점과 개요를 제공하고 AI가 영어 텍스트를 생성하는 역할 분담이다. 과학 논문 작성 도우미에 references/references.bib를 입력하면 인용 관리가 자동으로 처리되므로, AI가 정리한 reports/analysis-report.md를 참고하면서 IMRAD (Introduction, Methods, Results, Discussion) 형식으로 논문을 완성한다.

14.5 알림 시스템 활용 팁

플랫폼별 Webhook 형식

Discord 외에도 Slack이나 Zulip을 사용하는 연구팀이라면, notify.sh 스크립트에서 해당 플랫폼의 JSON 형식을 변경하면 된다. 어떤 플랫폼이든 원리는 동일하다: curl로 HTTP POST 요청을 보내 메시지를 전송한다.

플랫폼	Webhook URL 형식	메시지 필드
Discord	<code>https://discord.com/api/webhooks/...</code>	<code>{"content": " "}</code>
Slack	<code>https://hooks.slack.com/services/...</code>	<code>{"text": " "}</code>
Zulip	<code>https:// .zulipchat.com/api/v1/messages</code>	API key 인증 + content 필드

notify.sh를 Slack용으로 변경해줘. .env에서 SLACK_WEBHOOK_URL을 읽고, `{"text": " "}` 형식으로 전송하도록 수정해줘.

다른 Hook 이벤트 활용

Stop Hook 외에도 다양한 Hook 이벤트를 활용할 수 있다. 예를 들어, SubagentStop Hook을 사용하면 서버에이전트가 작업을 마칠 때마다 알림을 받을 수 있다.

Hook 이벤트	용도
Stop	Claude가 응답을 마칠 때마다 알림
SubagentStop	서브에이전트(@data-analyst 등)가 완료될 때 알림
Notification	Claude Code가 권한 요청 등의 알림을 보낼 때
PostToolUse	특정 도구 실행 후 알림 (예: 파일 저장 시)

14.6 연구자의 역할: 무엇을 판단해야 하는가

AI 팀원이 아무리 유능해도, 연구의 과학적 책임은 연구자에게 있다. 다음 표는 AI에게 맡길 수 있는 것과 반드시 연구자가 판단해야 하는 것을 구분한다.

AI에게 맡길 수 있는 것	연구자가 판단해야 하는 것
문헌 검색 및 요약	어떤 논문이 실제로 관련 있는지 선별
데이터 전처리 코드 작성	QC 기준값 (미토콘드리아 비율 몇 %로 필터링할지)
클러스터링 수행	클러스터 수가 적절한지, 생물학적으로 의미 있는 그룹인지
통계 검정 실행	통계 결과의 생물학적 의미 해석
GO/KEGG 분석 실행	enrichment 결과가 연구 맥락에서 의미 있는지
논문 텍스트 생성	핵심 주장, 가설, 결론의 과학적 타당성
참고문헌 형식 정리	인용된 논문이 실제로 존재하고 내용이 정확한지
Figure 코드 작성	Figure가 데이터를 정확하게 반영하는지

체크포인트

연구 과정에서 반드시 멈추고 연구자가 확인해야 하는 체크포인트:

1. **데이터 선택**: 어떤 데이터셋을 사용할지는 연구 질문에 따라 결정해야 한다
2. **QC 기준**: 너무 엄격하면 데이터를 잃고, 너무 느슨하면 노이즈가 포함된다
3. **가설 채택**: AI가 제안한 가설 중 과학적으로 검증 가능하고 의미 있는 것을 선별한다
4. **통계 해석**: p-value가 낮다고 반드시 의미 있는 것은 아니다. 다중검정 보정, 효과 크기 등을 함께 고려한다
5. **인용 검증**: AI가 정리한 문헌 정보가 실제 논문 내용과 일치하는지 확인한다
6. **윤리 검토**: 환자 데이터를 다룰 때 IRB 규정, 데이터 공유 정책을 준수하는지 확인한다

이 체크포인트들은 12장에서 논의한 **Human-in-the-Loop** 원칙의 실천이다. AI Scientist가 보여준 것처럼, 사람의 감독 없는 완전 자동화는 시각화 오류, 불공정한 비교, 심지어 자체 스크립트 수정 같은 문제를 일으킬 수 있다. AI는 도구이고, 과학적 판단은 연구자의 몫이다.

14.7 이 책을 마치며

1장에서 개발 환경을 설정하는 것으로 시작해, 14장에서 AI 연구팀을 구성하고 연구 프로젝트를 수행하는 단계까지 도달했다. 이 여정을 돌아보면, 일관된 원칙이 있었다.

바이브 코딩에서 사람의 역할은 코드를 작성하는 것이 아니라, 무엇을 만들어야 하는지 아는 것이다.

- 4장에서는 DataFrame, 화산 그림(volcano plot), 히트맵의 의미를 알아야 분석 결과를 해석할 수 있었다
- 5장에서는 QC, 정규화, 클러스터링의 순서와 이유를 알아야 Scanpy 분석을 지시할 수 있었다
- 8~9장에서는 Navbar, Hero, Card 같은 UI 컴포넌트 이름을 알아야 디자인을 요청할 수 있었다
- 10장에서는 BLAST, E-value, FASTA의 개념을 알아야 검색 도구를 만들 수 있었다
- 12장에서는 Observation-Planning-Reflection 아키텍처를 이해해야 Co-scientist를 설계할 수 있었다
- 13장에서는 학술 논문의 구조와 과학적 무결성의 원칙을 알아야 AI 논문 작성을 올바르게 할 수 있었다
- 14장에서는 데이터 중심 과학의 패러다임을 이해하고, 팀장으로서 AI 팀원을 조율할 수 있어야 했다

AI 기술은 빠르게 변하지만, 이 책에서 다룬 **도메인 지식의 가치**는 변하지 않는다. 어떤 AI 도구를 사용하든, 생명정보학의 기본 원리를 이해하고 있는 연구자가 더 정확한 결과를 얻을 수 있다.

14.8 정리

- **데이터 중심 과학**: 가설이 먼저가 아니라, 대량 데이터 생성 → 패턴 발견 → 가설 도출 → 검증의 순서
- **연구자 = 팀장**: 어떤 데이터를 분석할지 결정하고, AI 팀원들의 결과를 종합 판단
- **AI 연구팀 구성**: .claude/agents/에 Literature Scout, Data Analyst, Hypothesis Generator, Pathway Analyst, Report Writer 등 역할별 에이전트 정의
- **프로젝트 환경**: 디렉토리 구조, CLAUDE.md(장기 기억), MCP 서버, 알림 Hook을 한 번에 구축
- **알림 시스템**: Discord/Slack/Zulip Webhook으로 분석 진행 상황을 실시간 업데이트
- **실전 워크플로우**: 데이터 준비 → 데이터 이해 → 가설 도출 → 가설 검증 → 결과 종합·논문 작성의 5단계
- **Human-in-the-Loop 체크포인트**: 데이터 선택, QC 기준, 가설 채택, 통계 해석, 인용 검증, 윤리 검토
- **핵심 원칙**: AI 팀원이 아무리 유능해도, 과학적 판단과 최종 책임은 연구자에게 있다