

Anthropic 엔지니어링 연구 분석

Claude의 특성 이해와 효과적인 협업 방법

문서 목적: 이 문서는 Anthropic의 6개 엔지니어링 블로그 포스트를 종합하여 Claude의 특성을 이해하고, Claude AI와 작업 할 때 효과적인 협업 방법을 파악합니다.

출처 문서:

1. Claude Code: Best Practices for Agentic Coding
2. How We Built Our Multi-Agent Research System
3. Writing Effective Tools for AI Agents
4. Effective Context Engineering for AI Agents
5. Equipping Agents for the Real World with Agent Skills
6. Claude Code Sandboxing: Security Architecture & Implementation

분석 일자: 2025년 11월

Part 1: Claude의 특성 이해

1.1 Attention Budget과 Context Window

Claude는 200,000 토큰의 컨텍스트 윈도우를 가지고 있지만, 이것이 모든 정보에 똑같이 접근 가능하다는 의미는 아닙니다. 근본적인 제약은 트랜스포머 아키텍처 자체에서 비롯됩니다.

핵심 인사이트: 트랜스포머는 n^2 쌍방향 토큰 관계를 생성합니다. 컨텍스트 길이가 증가하면 모든 토큰에 대한 attention을 유지하는 모델의 능력이 저하됩니다—이는 능력의 한계가 아니라 아키텍처적 제약 때문입니다.

실무적 함의: 토큰 예산은 유한하고 귀중한 것으로 취급되어야 합니다. 더 많은 컨텍스트가 자동으로 더 나은 성능을 의미하지 않습니다; 높은 신호의 토큰을 전략적으로 선별하는 것이 중요합니다.

1.2 Context Rot 현상

정의: "Context rot"은 컨텍스트가 확장됨에 따라 회상 정확도가 감소하는 현상을 말합니다. 이는 능력 수준에 관계없이 모든 모델에 영향을 미칩니다.

연구 결과: 연구들은 컨텍스트 시퀀스가 길어질수록 정보 검색과 장거리 추론에서 일관된 성능 저하를 문서화했습니다. 모델은 긴 컨텍스트에서도 여전히 작동하지만, 짧은 시퀀스에 비해 정밀도가 감소합니다.

설계 원칙: 이는 포괄적인 정보 포함보다 신중한 선별의 실질적 필요성을 만듭니다. 질문은 "Claude가 이만큼의 컨텍스트를 처리할 수 있는가?"가 아니라 "이 작업에 최적의 컨텍스트는 무엇인가?"입니다.

1.3 Token 소비 패턴

핵심 발견: Anthropic의 다중 에이전트 연구 시스템에서 토큰 사용량이 브라우징 작업의 성능 분산의 80%를 설명했습니다.

소비율:

- 다중 에이전트 시스템은 단일 에이전트 채팅 상호작용보다 약 15배 더 많은 토큰을 사용
- 쓰기 작업은 2배의 토큰을 소비 (메모리 + 출력)
- 도구 응답 제한: Claude Code는 기본적으로 25,000 토큰으로 제한

트레이드오프: 다중 에이전트 시스템이 상당히 더 많은 토큰을 소비하지만, 다음을 달성합니다:

- 단일 에이전트 시스템 대비 90.2% 성능 향상
- 병렬 실행을 통한 최대 90% 시간 단축

결론: 다중 에이전트 아키텍처에 대한 토큰 투자는 작업이 높은 병렬화 가능성을 포함하거나 단일 컨텍스트 윈도우를 초과할 때 측정 가능한 수익을 제공합니다.

1.4 비결정적 특성

도전: 결정적 동작을 가진 전통적인 소프트웨어와 달리, Claude의 출력은 동일한 입력에서도 실행마다 다릅니다.

프로덕션 함의:

- 전통적인 디버깅 접근법(재현 → 격리 → 수정)이 직접 적용되지 않음
- 대화 내용 모니터링 없이 전체 프로덕션 추적 필요
- 의사결정 패턴과 상호작용 구조를 통한 체계적 근본 원인 분석

해결 전략: AI 적응성과 결정적 안전장치 결합:

- 지수 백오프를 사용한 재시도 로직
- 재개 가능한 작업을 위한 체크포인트
- 장기 작업을 위한 상태 관리

Part 2: 핵심 협업 원칙

2.1 Context Engineering vs Prompt Engineering

구분:

- Prompt Engineering: 효과적인 지시사항과 쿼리 작성
- Context Engineering: LLM 추론 중 최적의 토큰 세트를 전략적으로 선별하고 유지

정의: "LLM 추론 중 최적의 토큰 세트(정보)를 선별하고 유지하기 위한 전략의 집합"

범위: Context engineering은 추론 중 LLM에 제공되는 모든 정보를 관리하는 더 광범위한 과제를 다룹니다:

- 시스템 지시사항
- 도구와 그 설명
- 외부 데이터와 리소스
- 대화 이력
- 예시 (few-shot learning)

핵심 원칙: "원하는 결과의 가능성을 최대화하는 가장 작은 고신호 토큰 세트를 찾으라"

2.2 다중 에이전트 시스템을 위한 8가지 원칙

이 원칙들은 단일 에이전트 시스템 대비 90.2% 성능 향상을 달성한 Anthropic의 다중 에이전트 Research 기능을 구축하면서 도출되었습니다.

원칙 1: 에이전트처럼 생각하기

실천: 프로덕션 이전에 정확한 시스템 프롬프트와 도구를 사용하여 시뮬레이션을 구축하고 실패 모드를 식별합니다.

발견된 일반적인 실패 모드:

- 과도한 서브에이전트 생성
- 수렴 없는 무한 웹 검색
- 모호한 설명으로 인한 도구 오용

구현: 정확한 에이전트 환경을 사용하여 프롬프트와 워크플로우를 테스트하고, 에이전트가 막히거나 잘못된 결정을 내리는 지점을 관찰합니다.

원칙 2: 위임 가르치기

핵심 인사이트: 모호한 지시사항은 다중 에이전트 시스템에서 중복과 간극을 야기합니다.

효과적인 위임을 위한 필수 요소:

- 명확한 목표: 구체적이고 측정 가능한 결과
- 출력 형식: 구조화된 기대치 (JSON, markdown, 특정 스키마)

- **도구/소스 지침:** 어떤 도구를 언제 사용할지
- **작업 경계:** 범위 내외의 것이 무엇인지

SPARK 예시: "2호 위임 프로토콜"은 implementer 에이전트에 위임할 때 프로젝트 표준, 표준 모듈, 품질 집행 컨텍스트를 제공하도록 요구합니다.

원칙 3: 복잡도에 맞춰 노력 조정

가이드 매트릭스:

- **단순 쿼리:** 1개 에이전트, 3-10번 도구 호출
- **직접 비교:** 2-4개 서브에이전트, 각 10-15번 호출
- **복잡한 연구:** 10개 이상 서브에이전트, 책임 분산

적용: 최대 병렬화를 기본으로 하지 말고 작업 복잡도에 에이전트 아키텍처를 맞춥니다.

원칙 4: 도구 설계가 중요하다

발견: 품질 높은 도구 설명은 에이전트가 잘못된 경로를 추구하는 것을 방지합니다. 부실한 설명은 다음을 초래합니다:

- 실패한 시도에 톤 낭비
- 막다른 전략 추구
- 유사 도구 간 혼란

해결책: MCP 서버는 적절한 도구 선택을 위한 명시적 휴리스틱이 필요합니다. 명확하고 모호하지 않은 도구 설명에 시간을 투자하세요.

원칙 5: 에이전트가 스스로 개선하게 하기

획기적 발견: Claude 모델은 실패 모드를 진단하고 프롬프트 개선을 제안할 수 있습니다.

사례 연구: 도구 테스트 에이전트가 인간 설계자가 놓친 도구 사용의 뉘앙스를 발견하여 40% 작업 완료 개선을 달성했습니다.

실천: 평가 기록을 Claude에 피드백하여 다음의 개선사항을 식별합니다:

- 시스템 프롬프트
- 도구 설명
- 워크플로우 패턴
- 오류 처리

원칙 6: 넓게 시작해서 좁히기

전략: 짧고 넓은 쿼리로 시작; 사용 가능한 정보 평가; 처음부터 지나치게 구체적인 검색보다 점진적으로 초점을 맞춥니다.

근거: 에이전트는 탐색하기 전까지 어떤 정보가 존재하는지 모릅니다. 조기 특정화는 에이전트가 관련 정보를 놓치거나 사용 불가능한 데이터에 집착하게 만들 수 있습니다.

패턴: 넓은 탐색 → 평가 → 집중 조사

원칙 7: 사고 과정 안내하기

기법:

- Extended Thinking Mode: 복잡한 추론을 위한 제어 가능한 스크래치패드
- Interleaved Thinking: 추론과 도구 사용 혼합

이점:

- 지시사항 준수 개선
- 더 나은 토큰 효율성
- 더 투명한 의사결정

사용법: "think", "think hard", "think harder", "ultrathink" 프롬프트로 점진적으로 더 철저한 분석을 유도합니다.

원칙 8: 병렬 도구 호출

성능 영향: 서브에이전트가 순차적이 아닌 3개 이상의 도구를 동시에 사용하면 연구 속도가 극적으로 가속됩니다.

구현: 충돌 없이 동시 호출을 지원하는 도구 인터페이스를 설계합니다. 가능한 순차 종속성을 피합니다.

결과: 복잡한 연구 작업에서 90% 시간 단축의 주요 기여 요인입니다.

2.3 도구 설계를 위한 5가지 원칙

Anthropic의 에이전트를 위한 효과적인 도구 작성 포스트에서 도출되었습니다.

원칙 1: 올바른 도구 선택

핵심 인사이트: "더 많은 도구가 더 나은 결과를 보장하지 않습니다."

가이드라인:

- 특정 고영향 워크플로우 우선순위
- 기능 통합 (예: 별도의 `list_users`, `list_events`, `create_event` 대신 `schedule_event`)
- 에이전트 어포던스를 고려하지 않고 단순히 API 엔드포인트를 래핑하는 도구 회피

안티패턴: 모든 엔티티에 대한 포괄적인 CRUD 작업 생성. 에이전트는 저수준 API가 아닌 작업 지향적 도구가 필요합니다.

원칙 2: 명확한 네임스페이스 사용

패턴: 관련 도구를 공통 접두사로 그룹화 (예: `asana_search`, `jira_search`)

이점:

- 에이전트가 적절한 도구 선택에 도움
- 컨텍스트 오버헤드 감소
- 도구 관계를 명확히 함

참고: 접두사 대 접미사 네이밍이 성능에 영향; 도메인에 맞게 두 접근법 모두 테스트하세요.

원칙 3: 의미 있는 컨텍스트 반환

회피: 저수준 기술 식별자 (**uuid** , **mime_type** , 내부 ID)

선후: 의미적 이름과 자연어 설명

구현: 에이전트가 요청할 수 있는 **response_format** enum 매개변수 구현:

- "concise": 다음 단계를 위한 최소 토큰
- "detailed": 최종 보고서를 위한 완전한 정보

이점: 다운스트림 도구 호출을 위한 유연성을 유지하면서 토큰 사용을 최적화합니다.

원칙 4: 토큰 효율성 최적화

전략:

- 페이지네이션: 관리 가능한 청크 반환 (예: 10–25개 항목)
- 필터링: 에이전트가 결과를 좁힐 수 있도록 허용
- 자르기: "..." 계속 표시기로 긴 응답 클리핑
- 범위 선택: 하위 집합 선택 지원 (예: "항목 50–75")
- 합리적 기본값: 자동으로 합리적인 제한 선택

오류 메시지: 불투명한 오류 코드를 반환하는 대신 에이전트를 더 나은 전략으로 안내하는 메시지를 작성합니다.

예시: "ERROR 413: Payload too large" 대신 "쿼리가 25K 토큰 제한을 초과하는 10,000개 결과를 반환했습니다. 필터 추가(date_range, category) 또는 페이지네이션(page_size=25) 시도하세요."

원칙 5: 도구 설명에 프롬프트 엔지니어링 적용

영향: 명확하고 구체적인 도구 설명이 에이전트 성능을 크게 향상시킵니다.

접근법: 신입 팀원에게 하듯 도구를 설명하세요:

- 암묵적 컨텍스트를 명시적으로 만들기
- 사용 예시 제공
- 사용 시점 vs 사용하지 않을 시점 설명

- 매개변수를 명확하게 명명 (`user` 가 아닌 `user_id`)

발견: 작은 개선도 오류율을 극적으로 감소시킵니다. 내부 테스트에서 Claude 최적화 도구가 Slack과 Asana MCP 서버 모두에서 인간이 작성한 구현을 능가했습니다.

2.4 점진적 공개 (Progressive Disclosure)

개념: "목차로 시작하고, 그 다음 특정 장, 마지막으로 상세한 부록으로 이어지는 잘 조직된 매뉴얼"

3단계 정보 모델 (Agent Skills에서):

1. Level 1: 시스템 프롬프트의 스킬 이름과 설명 (경량)
2. Level 2: 트리거될 때 전체 SKILL.md 콘텐츠 (중간)
3. Level 3+: 필요에 따라 로드되는 참조 파일 (상세)

이점:

- 초기 컨텍스트 부하 감소
- 적시에 정보 제공
- 파일시스템 접근을 통해 사실상 무한한 컨텍스트로 확장
- 에이전트가 관련 정보만 로드

적용: 에이전트 정의, 문서, 리소스는 모든 것을 사전 로딩하는 대신 점진적 공개를 따라야 합니다.

Part 3: 실무 워크플로우

3.1 단일 에이전트 모범 사례

"Claude Code: Best Practices for Agentic Coding"에서.

구체성이 중요하다

약함: "foo.py에 테스트 추가"

강함: "foo.py에 사용자가 로그아웃된 엣지 케이스를 다루는 새 테스트 케이스 작성, 목(mock) 회피"

영향: 상세한 지시사항이 성공률을 크게 향상시킵니다. 구체성은 다음을 제공합니다:

- 명확한 성공 기준
- 경계 정의
- 기술적 제약사항
- 기대되는 접근법

시각적 컨텍스트

Claude는 이미지에 탁월합니다. 방법:

- 스크린샷 (macOS: cmd+ctrl+shift+4로 클립보드에)
- 드래그 앤 드롭 지원
- 파일 경로 참조

팁: 디자인 충실도가 중요할 때 시각적 매력의 중요성을 명시적으로 언급하세요.

방향 설정

전략:

- 구현 전 계획 요청
- Escape를 눌러 중단 및 방향 전환
- Escape 두 번 눌러 이전 프롬프트 편집
- Claude에게 변경사항 취소 요청

철학: 단일 시도 솔루션은 드뭅니다; 반복적 개선이 일반적으로 더 빠르게 더 나은 결과를 생성합니다.

컨텍스트 관리

실천: 작업 간 **/clear** 사용으로 Claude의 주의를 재집중하고 성능 개선.

근거: 이전 작업의 컨텍스트 축적이 현재 목표에서 주의를 분산시킬 수 있습니다. 새로운 컨텍스트 윈도우는 더 깨끗한 정신적 작업 공간을 제공합니다.

복잡한 작업을 위한 체크리스트

사용 사례:

- マイグレーション
- 대량 수정
- 다단계 워크플로우

패턴: Claude가 마크다운 체크리스트를 생성하고 진행 상황을 체계적으로 추적하게 합니다.

이점: 단계를 잊는 것 방지, 재개 가능성 활성화, 진행 상황 가시성 제공.

3.2 다중 에이전트 아키텍처

Orchestrator-Worker 패턴

구조:

- Lead Researcher Agent 사용자 쿼리 분석 및 전략 개발

- 메모리 지속성 컨텍스트 윈도우 제한 전에 계획 저장
- 병렬 서브에이전트 독립적으로 연구 작업 실행
- 반복적 종합 결과 평가 및 추가 연구 필요 여부 결정
- 인용 에이전트 출처 식별 및 주장 귀속
- 사용자 전달 적절한 인용이 포함된 최종 결과

성능:

- 단일 에이전트 Opus 4 대비 90.2% 향상 (Opus 4 리드 + Sonnet 4 서브에이전트 사용)
- 단일 에이전트 대비 15배 토큰 소비
- 병렬화를 통한 90% 시간 단축

최적 사용 사례

탁월한 용도:

- 높은 병렬화 가능성
- 단일 컨텍스트 윈도우를 초과하는 정보
- 수많은 복잡한 도구
- 증가된 토큰 소비를 정당화하는 높은 작업 가치

부적합:

- 대부분의 코딩 작업 (진정한 병렬화 부족)
- 실시간 조정 요구사항
- 가치가 낮은 반복 작업
- 토큰 제약 환경

SPARK 구현 예시

아키텍처: 2호 (Director) + 21개 전문 에이전트 (6 core + 15 team)

핵심 특성:

- Orchestrator (2호)가 위임 결정
- 에이전트 온디맨드 로딩 (95.5% 토큰 감소)
- 조정을 위한 JSON 상태 관리
- 품질 게이트가 작업 표준 보장

3.3 워크플로우 패턴

Explore-Plan-Code-Commit 패턴

Phase 1: Explore (탐색)

- 코드 작성 없이 Claude에게 관련 파일/URL 읽기 요청
- 계획을 위한 extended thinking 사용
- 행동 전 컨텍스트 수집

Phase 2: Plan (계획)

- 계획을 캡처하는 문서 또는 GitHub 이슈 생성
- 구현 전 Claude가 접근법 설명하게 함
- 요구사항 이해 검증

Phase 3: Code (코딩)

- 명시적 검증과 함께 솔루션 구현
- 테스트/피드백 기반 반복적 개선 사용
- 계획 목표에 집중 유지

Phase 4: Commit (커밋)

- 커밋 및 풀 리퀘스트 생성
- 변경 로그 업데이트
- 결정사항 문서화

이점: 재작업 감소, 코드 품질 개선, 프로젝트 문서화 유지.

테스트 주도 개발 (TDD)

Step 1: 입출력 사양 기반 테스트 작성

Step 2: 테스트 실행 및 실패 확인

Step 3: 테스트 코드 커밋

Step 4: 테스트 통과할 때까지 코드 구현

Step 5: 과적합 방지를 위해 독립 서브에이전트로 검증

Step 6: 구현 커밋

장점: 사양 명확성, 회귀 방지, 실행 가능한 문서화.

시각적 반복 워크플로우

Step 1: 스크린샷 도구 제공 (Puppeteer MCP, iOS 시뮬레이터, 수동 이미지)

Step 2: 붙여넣기, 드래그, 파일 경로를 통해 디자인 목업 제공

Step 3: 일치할 때까지 반복적으로 디자인 구현

Step 4: 최종 결과 커밋

성능: 일반적으로 2-3회 반복 후 크게 개선됩니다.

적용: UI 개발, 디자인 구현, 시각적 디버깅.

다중 Claude 워크플로우

병렬 검증 패턴:

- 한 Claude가 코드 작성
- 다른 Claude가 검토
- 독립적 관점을 통해 별도 컨텍스트가 종종 더 나은 결과 산출

다중 리포지토리 체크아웃:

- 3-4개의 별도 폴더 생성
- 병렬 터미널 탭에서 다른 작업
- 기능에 걸쳐 분산 작업

Git Worktrees:

```
git worktree add .. /project-feature-a feature-a
```

- 경량 브랜칭
- 독립 기능에 대한 동시 Claude 세션
- 브랜치 간 컨텍스트 오염 없음

3.4 컨텍스트 관리 기법

Just-In-Time 검색

전략: 모든 데이터를 사전 로딩하는 대신, 에이전트가 경량 참조(파일 경로, URL, 쿼리)를 유지하고 실행 중 동적으로 정보를 검색합니다.

예시: Claude Code는 bash 명령을 사용하여 컨텍스트 윈도우를 소진하지 않고 대규모 데이터셋을 분석합니다.

이점:

- 탐색을 통한 점진적 공개
- 메타데이터 신호가 에이전트 결정에 정보 제공 (위치, 명명, 타임스탬프)

- 관련 하위 집합에 대한 자체 관리 컨텍스트 초점

트레이드오프:

- 런타임 탐색이 사전 계산보다 느림
- 막다른 곳이나 오용된 도구에 컨텍스트 낭비 방지를 위한 신중한 도구 설계 필요

장기 작업 기법

Compaction (압축):

- 컨텍스트 제한에 근접한 대화 요약
- 압축된 요약으로 재시작
- 아키텍처 결정과 미해결 버그 보존
- 중복 출력 삭제

구현 가이드:

- 회상 최대화로 시작 (모든 관련 정보 캡처)
- 정밀도를 향해 반복 (불필요한 콘텐츠 제거)
- 도구 결과 정리: "가장 안전하고 가벼운 접촉" 접근법

구조화된 노트 작성:

- 에이전트가 컨텍스트 윈도우 밖에 지속적 노트 유지
- 나중에 필요에 따라 다시 가져오기
- 복잡한 작업에 걸쳐 진행 상황 추적
- 점진적으로 이해 개발

실제 예시: Pokémon을 플레이하는 Claude는 수천 단계에 걸쳐 집계를 유지하고, 지도를 개발하고, 명시적 프롬프팅 없이 업적을 기억합니다.

서브 에이전트 아키텍처:

- 전문 서브에이전트가 깨끗한 컨텍스트 윈도우로 집중된 작업 처리
- 압축된 요약 반환 (일반적으로 1,000-2,000 토큰)
- 상세한 검색 컨텍스트를 고수준 종합에서 분리
- 조정 에이전트가 전략적 개요 유지

패턴: 리드 에이전트가 서브에이전트에 심층 조사를 위임하고, 요약을 받고, 상세한 검색 컨텍스트를 유지하지 않고 결론을 종합합니다.

Part 4: 도구와 확장

4.1 CLAUDE.md 커스터마이징

목적: Claude가 자동으로 컨텍스트에 통합하는 특수 마크다운 파일.

이상적인 콘텐츠:

- Bash 명령과 그 목적
- 핵심 유ти리티 함수와 파일
- 코드 스타일 가이드라인
- 테스팅 절차
- 리포지토리 관례 (브랜칭, 병합 전략)
- 환경 설정 요구사항
- 프로젝트별 경고 또는 동작
- 팀 지식과 제도적 컨텍스트

위치:

- 리포지토리 루트 (프로젝트별)
- 부모 디렉토리 (모노레포에 유용)
- 자식 디렉토리 (컴포넌트별)
- 홈 폴더 (`~/.claude/CLAUDE.md` 범용 접근)

모범 사례: CLAUDE.md 콘텐츠를 프로덕션 프롬프트처럼 취급—결과에 따라 정기적으로 반복. 세션 중 `#` 키를 사용하여 새 지시사항을 캡처한 다음 커밋에 업데이트 포함.

항상 기법:

- 프롬프트 개선기를 통해 파일 실행
- 준수 향상을 위해 "IMPORTANT" 같은 핵심 용어 강조
- 규칙과 함께 예시 추가
- 실패 모드와 솔루션 문서화

4.2 도구 설계 가이드라인

개발 프로세스

Phase 1: 프로토타입 구축

- Claude Code를 사용한 빠른 도구 구현으로 시작

- 로컬 MCP 서버 또는 Desktop 확장에 도구 래핑
- 가능할 때 LLM 친화적 문서 (llms.txt 파일) 사용

Phase 2: 평가 실행

- 다중 도구 호출을 요구하는 현실적 평가 작업 생성
- 예시: 첨부파일이 있는 미팅 일정, 청구 문제 조사, 유지 제안 준비
- 단순한 에이전트 루프를 사용하여 프로그래밍 방식으로 실행
- 메트릭 수집: 정확도, 런타임, 토큰 소비, 오류율

Phase 3: 에이전트와 협업

- 평가 기록을 Claude Code에 피드백
- Claude의 분석을 통해 개선사항 식별
- 도구 설명, 매개변수, 응답 형식에 반복

핵심 인사이트: "이 포스트의 조언 대부분은 Claude Code로 내부 도구 구현을 반복적으로 최적화하면서 나왔습니다."

응답 형식 고려사항

만능 솔루션은 없음: 도구 응답 형식 (XML, JSON, Markdown)이 사용 사례에 따라 성능에 다르게 영향을 미칩니다.

전략: **response_format** 매개변수 구현:

```
def search_documents(query: str, response_format: Literal["concise", "detailed"] = "concise"):  
    """  
    response_format:  
        - concise: 다음 단계를 위한 최소 토큰  
        - detailed: 최종 보고서를 위한 완전한 정보  
    """
```

이점: 에이전트가 필요에 따라 적절한 세부 수준을 선택합니다.

오류 메시지 설계

안티패턴: 불투명한 오류 코드

```
ERROR 403: Forbidden
```

모범 사례: 실행 가능한 지침

```
/admin/users에 대한 접근 거부됨. 이 리소스는 'admin' 역할이 필요합니다.  
현재 사용자는 'viewer' 역할을 가지고 있습니다. 관리자에게 접근 요청하거나  
공개 사용자 목록을 위해 /users 엔드포인트를 시도하세요.
```

원칙: 에이전트가 추측하도록 강요하는 대신 더 나은 전략으로 안내합니다.

4.3 Agent Skills

아키텍처: 에이전트가 동적으로 발견하고 로드하는 지시사항, 스크립트, 리소스의 조직화된 폴더.

핵심 컴포넌트: YAML frontmatter가 있는 **SKILL.md** 파일:

```
---
```

```
name: skill-name
description: 스킬 능력에 대한 간단한 설명
---
```

```
# 스킬 콘텐츠
상세한 지시사항, 예시, 참조 ...
```

점진적 로딩:

- 시스템 프롬프트가 모든 스킬 이름과 설명 표시 (경량)
- 관련시 에이전트가 스킬 호출 (SKILL.md 로드)
- 필요에 따라 에이전트가 참조 파일 읽기 (상세 컨텍스트)

개발 가이드라인:

평가로 시작:

- 대표 작업에 에이전트 실행
- 능력 간극 식별
- 간극 해결을 위해 점진적으로 스킬 구축

확장을 위한 구조화:

- 큰 SKILL.md를 참조 문서로 분할
- 상호 배타적 컨텍스트를 별도로 유지
- 선택적 로딩을 통해 토큰 사용 감소

Claude의 관점에서 생각:

- 실제 사용 패턴 모니터링

- 관찰에 기반한 반복
- 스킬 이름과 설명 명확성 개선

Claude와 반복:

- 성공적인 접근법을 캡처하기 위해 협업
- 임시 솔루션을 재사용 가능한 스킬로 변환
- 작업 중 나타나는 패턴 문서화

보안: 신뢰할 수 있는 소스의 스킬만 설치. 악성 코드 종속성과 신뢰할 수 없는 외부 네트워크로 Claude를 안내하는 지시사항에 대해 번들 파일을 감사하세요.

4.4 MCP 통합

Model Context Protocol (MCP): Claude를 외부 서비스와 데이터 소스에 연결하는 표준화된 방법.

구성 옵션:

- 프로젝트별 구성 (리포지토리의 `.mcp.json`)
- 전역 설정 (`~/claude/mcp.json`)
- 팀 접근성을 위한 체크인 파일

모범 사례: 커스텀 MCP 도구를 다음으로 문서화:

- 프롬프트의 사용 예시
- `--help` 문서 참조
- 자주 사용하는 도구에 대한 CLAUDE.md 항목

문제 해결: 연결 또는 도구 호출 문제 진단 시 `--mcp-debug` 플래그 사용.

이점: Claude Code 업데이트 없이 내장 도구를 넘어 Claude의 능력 확장.

Part 5: 평가와 프로덕션

5.1 평가 방법론

소규모 샘플 시작점

접근법: 대규모 평가를 기다리지 말고 ~20개의 대표 쿼리로 시작.

근거:

- 초기 변경이 30-80% 성공률 개선을 보임

- 빠른 반복 주기
- 개념의 즉각적 검증

프로세스:

1. 대표 작업 세트 생성
2. 작업을 통해 에이전트 실행
3. 성공/실패 측정
4. 패턴 식별
5. 프롬프트/도구에 반복
6. 재평가

이점: 배포 전 완벽화보다 빠르게 개선사항 출시.

LLM-as-Judge 접근법

방법: 루브릭에 대해 평가하는 단일 LLM 호출

평가 차원:

- 사실 정확도
- 인용 정확도
- 완전성
- 출처 품질
- 도구 효율성

출력: 0.0-1.0 점수 + 합격/불합격 등급

장점:

- 확장 가능한 평가
- 일관된 기준 적용
- 정량적 메트릭

제한사항:

- 엣지 케이스를 놓칠 수 있음
- 잘 정의된 루브릭 필요
- 모든 실패 모드를 포착할 수 없음

인간 테스팅

필수 용도:

- 환각 포착
- 시스템 실패
- 출처 선택 편향
- 자동화된 평가가 놓치는 엣지 케이스

프로세스:

- 에이전트 출력의 수동 검토
- 적대적 케이스 레드팀
- 사용자 수용 테스팅
- 프로덕션 모니터링

결합 전략: 규모를 위해 LLM-as-Judge 사용, 품질 보증과 엣지 케이스 발견을 위해 인간 테스팅.

최종 상태 평가

패턴: 턴에 걸쳐 지속 상태를 수정하는 에이전트의 경우, 중간 단계가 아닌 최종 결과를 평가합니다.

이점: 특정 액션 시퀀스를 규정하는 대신 목표 달성을 위한 대안적 유효 경로를 허용합니다.

예시: 코드 구현은 다른 접근법을 취할 수 있습니다; 취한 정확한 단계가 아닌 최종 기능을 평가합니다.

홀드아웃 테스트 세트

목적: 훈련 예시에 대한 평가 과정 방지.

실천:

- 테스트 케이스의 일부 예약
- 홀드아웃 세트에 대해 최적화하지 않음
- 최종 검증에 사용

발견: 내부 테스팅에서 Claude 최적화 도구가 홀드아웃 테스트 세트에서 성능 향상을 유지하여 일반화를 확인했습니다.

5.2 프로덕션 엔지니어링

상태 저장 오류 복합

도전: 초기 단계의 오류가 다단계 에이전트 워크플로우를 통해 전파되고 복합됩니다.

안티패턴: 실패 시 처음부터 재시작 (토큰과 시간 낭비)

해결책: 다음을 갖춘 재개 가능한 시스템 구축:

- **재시도 로직:** 최대 시도 횟수를 가진 지수 백오프
- **체크포인트:** 단계 경계에서 상태 저장

- **복구:** 마지막 성공 체크포인트에서 재개
- **결정적 안전장치:** AI 적응성과 폐일세이프 결합

예시: 다중 에이전트 연구 시스템은 서브에이전트를 생성하기 전에 계획을 저장; 서브에이전트가 실패해도 전략 방향을 잃지 않음.

비결정적 디버깅

도전: 전통적 디버깅은 재현 가능한 동작을 가정; AI 에이전트는 실행마다 다릅니다.

해결책: 대화 내용 모니터링 없이 전체 프로덕션 추적

추적 전략:

- 의사결정 패턴 (어떤 도구, 어떤 순서)
- 상호작용 구조 (에이전트 통신 흐름)
- 리소스 소비 (토큰, 시간, API 호출)
- 성공/실패 결과

이점: 정확한 대화 재현 없이 시스템적 문제 드러냄.

신중한 배포

Rainbow 배포:

- 버전 간 트래픽 점진적 이동
- 두 버전 모두 동시에 실행 유지
- 회귀 모니터링
- 문제 감지 시 롤백

근거: 작업 중간에 활성 에이전트 방해 회피; 세션 연속성 보존.

모범 사례: 트래픽이 적은 시간대에 배포; 전환 중 면밀히 모니터링.

동기 병목

현재 제한사항: 리드 에이전트가 진행 전 서브에이전트 완료를 기다립니다.

미래 방향: 비동기 실행이 동시 에이전트를 가능하게 할 수 있지만 다음을 도입합니다:

- 상태 일관성 문제
- 경쟁 조건
- 조정 오버헤드

현재 권장사항: 동기 제약 수용; 더 나은 작업 분해와 병렬 서브에이전트 설계를 통한 최적화.

장기 작업 관리

파일시스템 출력 패턴:

- 서브에이전트가 출력을 외부 시스템에 직접 저장
- 경량 참조 반환 (파일 경로, 식별자)
- 다단계 처리를 통한 정보 손실 방지

이점:

- 토큰 제한을 넘어 상세한 결과 보존
- 세션에 걸쳐 결과 재사용 활성화
- 감사 추적 유지

구현:

```
# 서브에이전트가 상세 분석 저장
with open("analysis_results.json", "w") as f:
    json.dump(detailed_analysis, f)

# orchestrator에 경량 참조 반환
return {"status": "complete", "results_file": "analysis_results.json", "summary": "..."}
```

5.3 보안 (샌드박싱)

보안 모델: 함께 작동하는 이중 경계

경계 1: 파일시스템 격리

- "Claude는 특정 디렉토리만 접근하거나 수정할 수 있음"
- 무단 시스템 파일 수정 방지
- 프롬프트 주입 공격으로부터 보호

경계 2: 네트워크 격리

- "Claude는 승인된 서버에만 연결할 수 있음"
- 민감한 데이터 유출 차단
- 멀웨어 다운로드 방지

중요 원칙: 두 메커니즘 모두 필요—어느 것도 단독으로 충분하지 않음.

기술 구현:

- Linux:** 프로세스 격리를 위한 bubblewrap

- macOS: 제한 집행을 위한 seatbelt
- 적용 범위: 직접 상호작용 + 생성된 서브프로세스

네트워크 아키텍처:

- 인터넷 접근이 unix 도메인 소켓을 통해 라우팅
- 외부 프록시 서버가 도메인 제한 집행
- 프록시가 새 연결에 대한 사용자 확인 처리
- 향상된 보안을 위한 커스터마이징 가능한 규칙

결과: 내부 테스팅에서 샌드박싱이 보안 가드레일을 유지하면서 "권한 프롬프트를 안전하게 84% 감소" 시켰습니다.

안전한 YOLO 모드:

- 권한 검사 우회를 위해 **--dangerously-skip-permissions** 사용
- 인터넷 접근이 없는 컨테이너화된 환경에서만
- Docker Dev Containers 참조 구현 따르기
- 안전 경계를 가진 자율 작업 활성화

5.4 최적화

토큰 효율성 전략

도구 응답 제한:

- 기본: 도구 응답당 25,000 토큰
- 더 큰 데이터셋을 위한 페이지네이션 구현
- 필터링 옵션 제공
- 범위 선택 지원

컨텍스트 최적화:

- 처리 후 도구 결과 정리
- 전체 이력 보존 대신 요약
- 인라인 콘텐츠 대신 참조 사용
- 정보의 점진적 공개

쓰기 작업 인식:

- 쓰기 작업이 2배 토큰 소비 (메모리 + 출력)
- 가능할 때 전체 재작성보다 편집 선호

- 관련 쓰기 일괄 처리
- 불필요한 파일 작업 최소화

모델 선택 트레이드오프

발견: 모델 업그레이드 (예: Sonnet 4)가 토큰 예산 두 배보다 더 큰 성능 향상을 제공합니다.

전략:

- 리드 에이전트에 최고 사용 가능 모델 사용
- 서브에이전트에 비용/성능 트레이드오프 고려
- 특정 사용 사례 벤치마크

예시: Anthropic의 다중 에이전트 연구는 최적 비용/성능을 위해 Opus 4 리드와 Sonnet 4 서브에이전트를 사용합니다.

압축 기법

필요시: 컨텍스트 제한에 근접하지만 이력 정보 필요

방법:

- 요약: 핵심 포인트로 압축
- 추상화: 특정 예시를 패턴으로 변환
- 참조: 인라인 콘텐츠를 포인터로 대체
- 정리: 중복 또는 낮은 가치 콘텐츠 제거

결과: 필수 정보 유지하면서 30-50% 토큰 감소

병렬 실행 최적화

패턴: 순차적이 아닌 독립 작업을 동시에 시작

예시:

```
# 순차 (느림)
result1 = search_codebase("authentication")
result2 = search_documentation("authentication")
result3 = check_tests("authentication")

# 병렬 (빠름)
results = await asyncio.gather(
    search_codebase("authentication"),
    search_documentation("authentication"),
    check_tests("authentication")
)
```

이점: 다단계 워크플로우에 걸쳐 시간 절약 배가

요구사항: 작업이 진정으로 독립적이어야 함 (데이터 종속성 없음)

결론: 핵심 요점

Claude의 본질 이해

1. 토큰 예산은 유한함: 컨텍스트를 귀중하게 취급; 고신호 토큰 선별
2. Context Rot은 실재함: 길이에 따라 성능 저하; 전략적 선별 필수
3. 비결정적 동작: 변동 수용; 탄력적 시스템 구축
4. Attention 아키텍처: n^2 관계가 근본적 확장 제한 생성

효과적인 협업을 위한 핵심 원칙

1. 프롬프트 엔지니어링보다 컨텍스트 엔지니어링: 지시사항만이 아닌 모든 토큰 관리
2. 점진적 공개: 사전이 아닌 적시에 정보 로드
3. 에이전트처럼 생각: 실제 도구와 프롬프트로 시뮬레이션
4. 위임 가르치기: 명확한 목표, 형식, 지침, 경계 제공
5. 복잡도에 노력 조정: 작업 요구에 아키텍처 맞추기

다중 에이전트 아키텍처 이점

1. 90.2% 성능 향상: 단일 에이전트 시스템 대비
2. 90% 시간 단축: 병렬화를 통해
3. 컨텍스트 윈도우 곱셈: 에이전트에 걸쳐 별도 컨텍스트
4. 폭 우선 탐색: 독립 방향의 병렬 탐색

트레이드오프: 15배 토큰 소비로 비용 정당화를 위한 고가치 작업 필요

도구 설계 우수성

1. 신중하게 선택: 더 많은 도구 ≠ 더 나은 결과
2. 명확하게 네임스페이스: 관련 기능 그룹화
3. 의미 있는 컨텍스트 반환: 기술적보다 의미적
4. 토큰 최적화: 페이지네이션, 필터링, 자르기
5. 설명에 프롬프트 엔지니어링: 명확성이 성능을 극적으로 향상

프로덕션 준비

- 소규모 샘플 평가: ~20 쿼리로 시작, 빠르게 반복
- LLM-as-Judge + 인간 테스팅: 규모와 품질 보증 결합
- 재개 가능한 시스템: 체크포인트, 재시도 로직, 복구
- 보안 경계: 파일시스템 + 네트워크 격리 함께
- Rainbow 배포: 룰백 능력을 가진 점진적 전환

효과적인 워크플로우 패턴

- Explore-Plan-Code-Commit: 행동 전 컨텍스트 수집
- 테스트 주도 개발: 사양 명확성, 회귀 방지
- 시각적 반복: 디자인 매치를 위한 2-3 주기
- 다중 Claude: 병렬 검증, 별도 컨텍스트
- Just-In-Time 검색: 동적 정보 로딩

최적화 전략

- 모델 업그레이드 > 토큰 두 배: 더 나은 모델이 더 많은 토큰을 이김
- 병렬 도구 호출: 3개 이상 동시 작업
- 컨텍스트 관리: 정리, 요약, 참조, 정리
- 쓰기 작업 인식: 2배 토큰 소비
- 필요시 압축: 30-50% 감소 가능

맺음말

이 분석은 프로덕션 AI 에이전트 시스템을 구축하는 Anthropic 엔지니어링 팀의 학습을 종합합니다. 6개 문서 전체에 걸친 일관된 주제: 신중한 선별과 전략적 정보 관리가 원시 능력이나 컨텍스트 길이보다 중요합니다.

핵심 메타 인사이트:

- 에이전트는 스스로 개선할 수 있음: 최적화를 위해 기록을 Claude에 피드백 (40% 개선 문서화)
- 에이전트가 생략하는 것이 중요함: 때때로 추론에서 남겨진 것이 포함된 것보다 더 중요
- 다중 유효 경로 존재: 규정된 액션 시퀀스가 아닌 최종 상태 평가
- 간단하게 시작, 측정, 반복: 배포 전 완벽을 기다리지 말 것
- 컨텍스트는 귀중함: "가장 작은 고신호 토큰 세트 찾기"가 보편적으로 적용

이러한 원칙은 모델 정교함 발전에도 불구하고 필수적으로 남아있습니다. Claude의 능력이 성장함에 따라, 효과적인 협업의 기본—명확한 소통, 전략적 정보 관리, 신중한 도구 설계—이 계속해서 성공을 결정합니다.

문서 버전: 1.0

최종 업데이트: 2025년 11월

분석: 2호 (Claude Code Agent)

종합 방법: Ultrathink 심층 분석