How Anthropic Built Multi-Agent Research System

Anthropic Engineering 문서 요약 (2025)

📋 개요

Claude의 Research 기능은 **orchestrator-worker 패턴**을 사용하는 멀티 에이전트 시스템입니다. 리드 에이전트가 서브 에이전트들을 병렬로 조정하며, 웹, Google Workspace, 통합 서비스를 검색하여 복잡한 작업을 수행합니다.

핵심 메시지:

> "Multi-agent systems work mainly because they help spend enough tokens to solve the problem."

🎯 Multi-Agent System을 선택하는 이유

1. 리서치 작업의 본질적 특성

- **예측 불가능성**: 필요한 단계를 미리 하드코딩할 수 없음
- **경로 의존성**: 발견한 내용에 따라 접근 방식을 지속적으로 업데이트
- **동적 탐색**: 조사가 진행됨에 따라 관련 연결고리를 따라가야 함

2. 압축(Compression)의 본질

검색의 본질은 방대한 코퍼스에서 인사이트를 추출하는 것입니다.

서브 에이전트의 역할:

- 각자의 컨텍스트 윈도우에서 병렬 작동
- 질문의 다른 측면을 동시에 탐색
- 가장 중요한 토큰만 리드 에이전트에게 전달
- 관심사의 분리(separation of concerns) 제공

3. 집단 지능의 스케일링

> "Even generally-intelligent agents face limits when operating as individuals; groups of agents can accomplish far more."

성능 데이터:

- Multi-agent (Opus 4 lead + Sonnet 4 subs) vs Single-agent (Opus 4)
- **90.2% 성능 향상** (내부 리서치 평가)
- 예시: S&P 500 IT 기업 이사회 멤버 식별 과제에서 단일 에이전트는 실패, 멀티 에이전트는 성공

4. 토큰 사용량이 성능을 설명한다

BrowseComp 평가 분석:

- **토큰 사용량**: 성능 분산의 80% 설명
- **도구 호출 횟수**: 추가 설명 요인
- **모델 선택**: 추가 설명 요인
- 합계: 세 가지 요인이 95% 설명

실제 토큰 사용량:

- Chat 대비 Agents: **약 4배**
- Chat 대비 Multi-agents: **약 15배**

5. Multi-Agent가 적합한 경우

✅ **적합한 도메인:**

- 높은 병렬화가 가능한 작업
- 단일 컨텍스트 윈도우를 초과하는 정보
- 다수의 복잡한 도구와 인터페이스
- 작업의 가치가 증가된 비용을 정당화

❌ **부적합한 경우:**

- 모든 에이전트가 동일한 컨텍스트를 공유해야 하는 경우
- 에이전트 간 의존성이 많은 경우
- 코딩 작업 (대부분 병렬화 가능한 작업이 리서치보다 적음)
- LLM 에이전트의 실시간 조정/위임이 아직 미숙한 영역

🏗️ Architecture Overview

Orchestrator-Worker Pattern

```
User Query
    ↓
LeadResearcher (Orchestrator)
    ├→ SubResearcher 1 (Worker)
    ├→ SubResearcher 2 (Worker)
    ├→ SubResearcher 3 (Worker)
    └→ SubResearcher N (Worker)
         ↓
    Parallel Search & Analysis
         ↓
    Findings Return to Lead
         ↓
    CitationAgent
         ↓
    Final Research Report
```

컴포넌트 설명

1. LeadResearcher (리드 에이전트)

역할:

- 쿼리 분석 및 전략 수립
- 서브 에이전트 생성 및 작업 분배
- 결과 종합 및 추가 리서치 필요성 판단
- Extended thinking으로 계획 수립

주요 책임:

- 쿼리 복잡도 평가
- 서브 에이전트 수 결정
- 각 서브 에이전트의 역할 정의
- 적절한 도구 선택

2. SubResearcher (서브 에이전트)

역할:

- 특정 측면에 대한 독립적 조사

- 검색 도구를 반복적으로 사용하여 정보 수집
- Interleaved thinking으로 결과 평가 및 개선
- 압축된 발견사항을 리드에게 반환

지능적 필터 기능:

- 관련 정보 식별
- 새로운 발견에 적응
- 결과 분석 및 품질 평가

3. CitationAgent (인용 에이전트)

역할:

- 문서와 리서치 보고서 처리
- 인용을 위한 특정 위치 식별
- 모든 주장이 출처에 적절히 귀속되도록 보장

RAG vs Multi-Step Search

특성	Traditional RAG	Multi-Agent Research
검색 방식	정적 (Static)	동적 (Dynamic)
접근법	유사도 기반 청크 검색	반복적 탐색 및 적응
깊이	단일 단계	다단계 분석
적응력	없음	새로운 발견에 따라 조정

💡 Prompt Engineering 핵심 원칙

1. 에이전트처럼 생각하기 (Think Like Your Agents)

방법:

- Anthropic Console에서 정확한 프롬프트와 도구로 시뮬레이션 구축
- 에이전트 작동을 단계별로 관찰
- 실패 모드를 즉시 파악

발견된 실패 모드:

- 충분한 결과가 있는데도 계속 진행
- 지나치게 장황한 검색 쿼리 사용
- 잘못된 도구 선택

핵심:

> "Effective prompting relies on developing an accurate mental model of the agent."

2. Orchestrator에게 위임 방법 가르치기

리드 에이전트가 서브 에이전트에게 제공해야 할 것:

- ✅ 목표 (Objective)
- ✅ 출력 형식 (Output format)
- ✅ 사용할 도구 및 소스 가이던스
- ✅ 명확한 작업 경계

초기 문제:

```
❌ 나쁜 예: "semiconductor shortage를 조사해줘"
→ 모호하여 서브 에이전트가 오해하거나 작업 중복
→ 한 에이전트는 2021 자동차 칩 위기 탐색
→ 다른 2개는 2025 공급망 중복 조사
```

개선된 접근:

```
✅ 좋은 예: "2025년 현재 semiconductor shortage의
자동차 산업 영향을 조사해줘.
주요 제조사 3곳의 생산 차질 데이터와
재고 수준을 포함해야 해."
```

3. 쿼리 복잡도에 맞춰 노력 조정하기

명시적 가이드라인:

쿼리 유형	에이전트 수	도구 호출/에이전트	예시
간단한 사실 확인	1	3-10	"애플 CEO는 누구?"
직접 비교	2-4	10-15	"테슬라 vs 리비안 판매량"
복잡한 리서치	10+	다수	"AI 반도체 시장 5년 전망"

초기 문제:

- 간단한 쿼리에 과도한 투자
- 복잡한 쿼리에 불충분한 리소스

4. 도구 설계 및 선택이 핵심

도구-에이전트 인터페이스 = 사람-컴퓨터 인터페이스

명시적 휴리스틱 제공:

```
1. 먼저 사용 가능한 모든 도구를 검토
2. 도구 사용을 사용자 의도에 맞춤
3. 광범위한 외부 탐색은 웹 검색
4. 전문 도구를 일반 도구보다 선호
```

도구 설명의 중요성:

- 나쁜 도구 설명 → 완전히 잘못된 경로
- 각 도구는 명확한 목적과 설명 필요
- MCP 서버를 통한 도구 접근 시 설명 품질이 매우 다양함

예시:

```
❌ 나쁜 도구 설명:
"search_tool: 정보를 검색합니다"

✅ 좋은 도구 설명:
"web_search: 공개 웹에서 최신 정보를 검색합니다.
사용 시기: 실시간 데이터, 뉴스, 공개 문서가 필요할 때
피할 시기: 사용자 내부 문서나 Slack 메시지 검색"
```

5. 에이전트가 스스로 개선하게 하기

Claude 4 모델의 프롬프트 엔지니어링 능력:

- 프롬프트와 실패 모드를 제공하면 원인 진단
- 개선 방안 제안
- 도구 설명 재작성 가능

Tool-Testing Agent:

- 결함 있는 MCP 도구를 받으면 사용 시도
- 도구 설명을 재작성하여 실패 방지
- 수십 번 테스트하여 뉘앙스와 버그 발견

결과:

- 새 설명 사용 시 작업 완료 시간 **40% 감소**
- 미래 에이전트가 대부분의 실수 회피

6. 넓게 시작해서 좁히기 (Start Wide, Then Narrow)

전문가 인간 리서치 전략 반영:

```
❌ 나쁜 접근:
"2025년 1월 샌프란시스코에서 열린
AI 스타트업 시리즈 A 펀딩 라운드에서
a16z가 참여한 딜"
→ 결과가 거의 없거나 없음

✅ 좋은 접근:
1. "AI 스타트업 펀딩 2025" (넓게)
2. "시리즈 A AI 스타트업" (좁히기)
3. "a16z 시리즈 A 2025" (더 좁히기)
4. 특정 위치/시기로 필터링
```

에이전트의 기본 경향:

- 지나치게 길고 구체적인 쿼리 선호
- 결과가 적거나 없음

프롬프트 대응책:

> "Start with short, broad queries, evaluate what's available, then progressively narrow focus."

7. 사고 과정 가이드하기

Extended Thinking (리드 에이전트)

용도:

- 접근 방식 계획
- 작업에 맞는 도구 평가
- 쿼리 복잡도 및 서브 에이전트 수 결정
- 각 서브 에이전트의 역할 정의

효과:

- 지시 사항 준수 개선
- 추론 능력 향상
- 효율성 증가

Interleaved Thinking (서브 에이전트)

용도:

- 도구 결과 후 품질 평가
- 격차 식별
- 다음 쿼리 개선

효과:

- 서브 에이전트가 모든 작업에 더 효과적으로 적응

8. 병렬 도구 호출의 혁명적 영향

초기 문제:

- 순차적 검색 실행
- 고통스럽게 느림

두 가지 병렬화:

(1) 리드 에이전트 레벨

```
Before: 서브 에이전트 1 완료 → 2 시작 → 3 시작
After:  서브 에이전트 1, 2, 3 동시 시작
```

- 3-5개 서브 에이전트 병렬 생성

(2) 서브 에이전트 레벨

```
Before: 도구 1 실행 → 완료 → 도구 2 실행
After:  도구 1, 2, 3+ 동시 실행
```

- 3개 이상 도구 병렬 사용

성과:

- 복잡한 쿼리의 리서치 시간 **최대 90% 단축**
- 시간 단위에서 분 단위로 변환
- 다른 시스템보다 더 많은 정보 커버

📊 Effective Evaluation of Agents

멀티 에이전트 평가의 독특한 도전

전통적 평가:

```
입력 X → 경로 Y → 출력 Z
(항상 동일한 경로 예상)
```

멀티 에이전트 평가:

```
입력 X → 경로 ??? → 출력 Z
(다양한 유효한 경로 가능)
```

차이점:

- 동일한 시작점에서도 완전히 다른 유효한 경로 선택 가능
- 한 에이전트는 3개 소스 검색, 다른 에이전트는 10개
- 다른 도구를 사용해도 동일한 답변 도달 가능

평가 원칙:

> "Judge whether agents achieved the right outcomes while also following a reasonable process."

1. 소규모 샘플로 즉시 평가 시작

초기 개발 단계의 특징:

- 변화가 극적인 영향
- 프롬프트 조정만으로 성공률 30% → 80%
- 효과 크기가 크면 소수 테스트 케이스만으로 충분

실전 접근:

- 실제 사용 패턴을 대표하는 **약 20개 쿼리**로 시작
- 변화의 영향을 명확히 확인 가능

일반적 오해:

> "Many teams delay creating evals because they believe only large evals with hundreds of test cases are useful."

올바른 접근:

> "Start with small-scale testing right away with a few examples."

2. LLM-as-Judge 평가 (잘 수행하면 스케일 가능)

리서치 출력의 특징:

- 자유 형식 텍스트
- 단일 정답이 드묾
- 프로그래밍 방식 평가 어려움

평가 기준 (Rubric):

기준	설명	예시 질문
사실 정확성	주장이 출처와 일치하는가?	"이 통계가 실제 출처에 있는가?"
인용 정확성	인용된 출처가 주장과 일치하는가?	"이 URL이 실제 주장을 지원하는가?"
완전성	요청된 모든 측면이 다루어졌는가?	"세 가지 요소를 모두 다루었는가?"
출처 품질	1차 출처를 사용했는가?	"학술 논문 vs 블로그 포스트?"
도구 효율성	적절한 도구를 합리적 횟수 사용했는가?	"50번 검색은 과도한가?"

구현 방법:

- 단일 LLM 호출
- 단일 프롬프트
- 0.0-1.0 점수 출력
- Pass/Fail 등급

효과적인 시나리오:

- 명확한 답이 있는 테스트 케이스
- 예: "R&D 예산 상위 3개 제약회사 정확히 나열했는가?"

3. 인간 평가는 자동화가 놓치는 것을 포착

인간 테스터가 발견하는 것:

- 특이한 쿼리에 대한 환각 답변
- 시스템 실패
- 미묘한 출처 선택 편향

실제 사례:

- **문제 발견**: 초기 에이전트가 SEO 최적화된 콘텐츠 팜을 권위 있는 출처(학술 PDF, 개인 블로그)보다 일관되게 선택
- **해결**: 프롬프트에 출처 품질 휴리스틱 추가

결론:

> "Even in a world of automated evaluations, manual testing remains essential."

4. 창발적 행동 (Emergent Behaviors) 이해

정의:

- 특정 프로그래밍 없이 나타나는 행동
- 리드 에이전트의 작은 변화가 서브 에이전트 행동을 예측 불가능하게 변경

성공 요소:

- 개별 에이전트 행동이 아닌 **상호작용 패턴** 이해
- 엄격한 지시가 아닌 **협업을 위한 프레임워크** 제공
    - 작업 분담
    - 문제 해결 접근법
    - 노력 예산

필요한 것:

- 신중한 프롬프팅
- 도구 설계
- 견고한 휴리스틱
- 관찰 가능성 (Observability)
- 긴밀한 피드백 루프

🔧 Production Reliability & Engineering Challenges

1. 에이전트는 상태를 유지하며 오류가 누적됨

전통 소프트웨어 vs 에이전트 시스템:

측면	전통 소프트웨어	에이전트 시스템
버그 영향	기능 손상, 성능 저하	작은 변화가 대규모 행동 변화로 연쇄
오류 처리	재시작 가능	재시작 비용 높고 사용자에게 불만족
상태 관리	비교적 단순	장시간 다수 도구 호출 중 상태 유지 필요

완화 전략:

(1) 지속 가능한 실행 (Durable Execution)

```
오류 발생 → 처음부터 재시작 ❌
오류 발생 → 중단 지점부터 재개 ✅
```

(2) 모델 지능 활용

- 도구 실패 시 에이전트에게 알림
- 에이전트가 상황에 적응하도록 허용
- **놀랍게도 잘 작동함**

(3) 결정론적 안전장치

- 재시도 로직 (Retry logic)
- 정기적 체크포인트 (Regular checkpoints)

- AI 에이전트의 적응력 + 결정론적 안전장치 결합

2. 디버깅에는 새로운 접근법 필요

에이전트 디버깅의 어려움:

- 동적 의사 결정
- 동일한 프롬프트에도 실행 간 비결정론적
- "명백한 정보를 찾지 못함" 신고 시 원인 불명

질문들:

- 나쁜 검색 쿼리 사용?
- 열악한 출처 선택?
- 도구 실패?

해결책: 전체 프로덕션 추적 (Full Production Tracing)

표준 관찰 가능성 이상

- 에이전트 의사 결정 패턴 모니터링
- 상호작용 구조 모니터링
- **개별 대화 내용은 모니터링하지 않음** (사용자 프라이버시 유지)

고수준 관찰 가능성의 효과:

- 실패 근본 원인 진단
- 예상치 못한 행동 발견
- 일반적 실패 체계적 수정

3. 배포는 신중한 조정 필요

에이전트 시스템의 특징:

- 고도로 상태 유지하는 프롬프트, 도구, 실행 로직의 웹
- 거의 지속적으로 실행
- 업데이트 배포 시 에이전트는 프로세스 어디에나 있을 수 있음

문제:

- 모든 에이전트를 동시에 새 버전으로 업데이트 불가
- 코드 변경이 기존 에이전트를 중단시킬 수 있음

해결책: Rainbow Deployments

- 이전 버전과 새 버전을 동시에 실행
- 트래픽을 점진적으로 이전 → 새 버전으로 전환
- 실행 중인 에이전트를 방해하지 않음

4. 동기 실행이 병목 현상 생성

현재 상태:

- 리드 에이전트가 서브 에이전트를 **동기적으로** 실행
- 각 서브 에이전트 세트가 완료될 때까지 대기

장점:

- 조정 단순화

단점:

- 에이전트 간 정보 흐름에 병목 현상
- 리드 에이전트가 서브 에이전트 조정 불가
- 서브 에이전트 간 조정 불가
- 단일 서브 에이전트 검색 대기로 전체 시스템 차단

비동기 실행의 미래:

장점:

- 추가 병렬화 가능
- 에이전트 동시 작업
- 필요 시 새 서브 에이전트 생성

도전과제:

- 결과 조정 (Result coordination)
- 상태 일관성 (State consistency)
- 서브 에이전트 간 오류 전파

전망:

> "As models can handle longer and more complex research tasks, we expect the performance gains will justify the complexity."

💡 추가 팁 (Appendix)

1. 다중 턴에 걸쳐 상태를 변경하는 에이전트의 최종 상태 평가

도전:

- 다중 턴 대화에서 지속 상태를 수정하는 에이전트 평가
- 각 액션이 후속 단계의 환경을 변경
- 의존성 생성으로 전통적 평가 방법 어려움

해결책: End-State Evaluation

```
❌ 턴별 분석:
Turn 1: 올바른가? → Turn 2: 올바른가? → ...

✅ 최종 상태 평가:
최종 상태가 의도된 결과를 달성했는가?
```

접근법:

- 에이전트가 특정 프로세스를 따랐는지가 아닌
- **올바른 최종 상태를 달성했는지** 판단
- 에이전트가 동일 목표로 가는 대체 경로 인정

복잡한 워크플로우:

- 개별 중간 단계 검증보다
- 특정 상태 변경이 발생해야 하는 **이산 체크포인트**로 나눔

2. 장기 대화 관리 (Long-Horizon Conversation Management)

도전:

- 수백 턴에 걸친 대화
- 표준 컨텍스트 윈도우 불충분
- 지능적 압축 및 메모리 메커니즘 필요

구현 패턴:

(1) 완료된 작업 단계 요약

```
Phase 1 완료 → 요약 생성 → 외부 메모리 저장
Phase 2 시작 → 필수 정보만 유지
```

(2) 컨텍스트 제한 접근 시

```
리서치 계획을 외부 메모리에 저장
↓
새로운 컨텍스트로 새 서브 에이전트 생성
↓
신중한 인계를 통해 연속성 유지
↓
이전 작업 손실 방지
```

(3) 저장된 컨텍스트 검색

- 컨텍스트 제한 도달 시
- 외부 메모리에서 리서치 계획 등 검색
- 이전 작업 손실 방지

효과:

- 컨텍스트 오버플로우 방지
- 확장된 상호작용에서 대화 일관성 유지

3. 서브 에이전트 출력을 파일시스템에 저장하여 '전화 게임' 최소화

전화 게임 문제:

```
서브 에이전트 → 리드 에이전트 → 또 다른 에이전트 → ...
(정보가 전달될수록 왜곡/손실 증가)
```

해결책: Artifact Systems

```
서브 에이전트 출력을 직접 외부 시스템에 저장
↓
리드 에이전트에게는 가벼운 참조만 전달
↓
정보 손실 방지 + 토큰 오버헤드 감소
```

적합한 출력 유형:

- 코드
- 보고서
- 데이터 시각화
- 기타 구조화된 출력

장점:

- 다단계 처리 중 정보 손실 방지
- 대화 기록에 큰 출력을 복사하는 토큰 오버헤드 감소
- 전문 프롬프트를 가진 서브 에이전트가 일반 조정자를 통한 필터링보다 나은 결과 생성

작동 방식:

```
# 서브 에이전트
subagent.write_to_file("report.md", detailed_report)
subagent.return_to_coordinator({"artifact_path": "report.md"})

# 리드 에이전트
# 전체 보고서를 컨텍스트에 포함하지 않고
# 참조만 받음
```

📈 실제 사용 사례 분석 (Clio Embedding)

Anthropic의 Clio 임베딩 플롯 분석 결과, Research 기능의 상위 사용 사례:

순위	사용 사례	비율
1	전문 도메인 전반의 소프트웨어 시스템 개발	10%
2	전문 및 기술 콘텐츠 개발 및 최적화	8%
3	비즈니스 성장 및 수익 창출 전략 개발	8%
4	학술 연구 및 교육 자료 개발 지원	7%
5	사람, 장소, 조직에 대한 정보 조사 및 검증	5%

🎯 핵심 교훈 요약

1. 프로토타입에서 프로덕션까지의 격차

> "The last mile often becomes most of the journey."

도전:

- 개발자 머신에서 작동하는 코드베이스
- 신뢰할 수 있는 프로덕션 시스템이 되려면 상당한 엔지니어링 필요
- 에이전트 시스템에서 오류의 복합적 특성
- 한 단계 실패가 전혀 다른 궤적 탐색 유발

2. Multi-Agent의 가치

사용자 피드백:

- 고려하지 못한 비즈니스 기회 발견
- 복잡한 의료 옵션 탐색
- 까다로운 기술 버그 해결
- 혼자서는 찾지 못했을 리서치 연결 발견
- **최대 수일간의 작업 절약**

3. 성공 요소

필수 요소:

- ✅ 신중한 엔지니어링
- ✅ 포괄적 테스팅
- ✅ 세부 지향적 프롬프트 및 도구 설계
- ✅ 견고한 운영 관행
- ✅ 리서치, 제품, 엔지니어링 팀 간 긴밀한 협업
- ✅ 현재 에이전트 기능에 대한 강력한 이해

4. 미래 전망

> "Multi-agent research systems can operate reliably at scale."

이미 이러한 시스템이 사람들의 복잡한 문제 해결 방식을 변화시키고 있습니다.

📚 참고 자료

오픈소스 프롬프트:

- Anthropic Cookbook - Agent Prompts
- 실제 시스템의 예시 프롬프트 제공

학습 리소스:

- Anthropic Academy
- API 개발, Model Context Protocol, Claude Code 마스터
- 완료 시 인증서 발급

🚀 결론

Multi-agent research systems는 **오픈 엔드 리서치 작업**에서 입증된 가치를 보여주었습니다.

핵심 인사이트:

1. **토큰 사용량이 성능의 핵심**: 문제를 해결하기에 충분한 토큰을 사용하도록 돕는 것이 멀티 에이전트 시스템의 주된 이유
2. **병렬화가 게임 체인저**: 최대 90% 시간 단축
3. **프롬프트 엔지니어링이 중요**: 에이전트의 행동은 프롬프트로 조정
4. **평가는 유연해야 함**: 과정이 아닌 결과 판단
5. **프로덕션은 다름**: 신중한 엔지니어링과 운영 관행 필요

마지막 말:

> "Despite these challenges, multi-agent systems have proven valuable for open-ended research tasks."

프로토타입과 프로덕션 사이의 격차는 예상보다 크지만, 신중한 설계와 구현으로 대규모에서 안정적으로 작동하는 시스템을 구축할 수 있습니다.