ChatGPT

Claude 기반 개념 관계 정의 시스템 자가개선 루프 개선 방안 제안

배경 (Background)

중등 수학 개념들 간의 관계를 자동 분류하는 **개념 관계 정의 시스템**은 Anthropic Claude 모델(Claude Sonnet 4.5 기반)을 활용하여 12가지 관계 유형(예: 선수(prerequisite), 공동 필수(co-requisite), 역연산 등)을 판별합니다 [1]. 각 관계에는 방향성, 중요도, 시간적 순서, 인지 수준, 영역 범위의 5가지 속성이 부여되고, Claude 모델은 이러한 **관계 분류 사전(taxonomy)**과 예시를 시스템 프롬프트로 받아 JSON 형식의 출력을 생성합니다 [2]. 시스템은 **모호하거나 자신 없는 판단을 배제**하기 위해 Claude에게 **0.0~1.0의 신뢰도 점수**를 매기도록 유도하며, **신뢰도 0.70 미만**인 관계는 출력하지 않도록 임계값을 설정해 두었습니다 [3]. 실제 프롬프트 지침에서도 "신뢰도 0.70 이상인 관계만 식별; 약한 관계는 거짓 양성보다 누락되는 편이 낫다"고 명시하여 모호한 경우 판단을 보류하게 합니다 [4]. 이러한 보수적 설계로 Claude는 불확실한 관계를 함부로 단정하지 않고, 애매한 관계는 **agnostic (판단 유보)** 처리합니다.

관계 정의 에이전트(**RelationshipDefiner**)가 출력 형식 오류를 내거나 결과가 비어 있을 경우(관계 판단 보류/실패) 이를 성능 이슈로 간주하여 **품질 검사 에이전트(QualityAgent)** 등이 개입합니다 [5]. QualityAgent는 생성된 지식베이스 파일(예: Obsidian MD)의 형식과 내용을 검증하여 오류나 누락을 발견하면 리포트를 작성하고 Meta-Orchestrator에 알립니다 [5]. QualityAgent 자신은 수정 권한이 없지만 결과물의 **품질 저하나 관계 누락**을 지적해줄 수 있으며, Meta-Orchestrator는 이를 받아 여러 에이전트를 조율해 문제 해결을 시도합니다 [6]. 즉 RelationshipDefiner가 관계 도출을 못했거나 모호하게 넘어간 경우, 시스템은 이를 감지하여 후속 **원인 분석과 개선** 절차를 자동으로 시작합니다.

특히 **모호한 관계 쌍**에 대한 **자가개선 피드백 루프**가 핵심적으로 작동합니다 [7]. 이 루프에는 두 가지 전문 에이전트가 관여하는데, **SocraticMediator(소크라틱 중재자)**는 다중 질의응답 대화를 통해 실패 원인을 파악하고 **근본 원인(root cause) 분석** 및 개선 방향을 제시하며 [8], **SelfImprover(자가 개선 에이전트)**는 그 권고안을 받아 구체적인 **개선 액션**을 도출·적용합니다 [9]. 예컨대 RelationshipDefiner가 특정 개념 A–B 사이의 관계를 식별하지 못한 경우, Meta-Orchestrator는 이를 하나의 실패 사례로 보고 SocraticMediator에 **IssueReport**(문제 보고서)를 전달합니다. SocraticMediator는 해당 에이전트의 로그와 컨텍스트를 Claude로 분석하여 "왜 A–B 관계를 놓쳤는가?"를 탐색하고, **근본 원인**을 규명한 뒤 3~5개의 **개선 권고안**(recommendations)을 제시합니다 [10]. 이 과정에서 원인이 프롬프트 부족인지, taxonomy 한계인지, 훈련 지식의 미비인지 등을 구조적으로 밝혀냅니다 [11]. 다음으로 SelfImprover 에이전트가 SocraticMediator의 분석 결과를 받아 **1~3개의 구체적인 개선 조치**(예: 프롬프트 문구 보강, 매개변수 조정, 새로운 관계 유형 추가 등)를 생성하고, **Edit** 도구를 통해 실제 코드나 프롬프트를 수정합니다 [9]. 변경 적용 전에는 **DependencyAgent(의존성 분석 에이전트)**를 통한 **영향도 분석(Impact Analysis)**을 수행하여 해당 수정이 시스템 전반에 미칠 파급 범위를 평가합니다 [12]. 변경 사항이 광범위하거나 위험도가 높으면 Meta-Orchestrator의 품질 게이트(Quality Gate)에서 **자동 적용 여부를 결정**하며, 예컨대 수정이 20개 이상의 파일을 변경하거나 테스트 커버리지가 80% 미만으로 낮아지면 품질 게이트에서 **실패** 처리하여 자동 적용 대신 재조정 단계로 넘깁니다 [13] [14] [15]. 반대로 영향 범위가 작고 안전하면 곧바로 개선안을 반영합니다 [16].

이렇게 개선된 RelationshipDefiner 에이전트는 **Meta-Orchestrator**의 지시에 따라 동일 입력에 대해 **재실행**되어 개선 효과를 검증받습니다 [17]. 이전 실행 대비 성공률 향상이나 오류 감소 등의 지표로 성능 개선 여부를 측정하며, 개선되었다면 루프를 종료합니다 [18]. 여전히 문제가 남아있다면 SocraticMediator가 추가 원인 분석을 한 번 더 수행하고, 새로운 개선안을 도출하여 **최대 2회까지 재시도**합니다 [19]. 현재 시스템에서는 이러한 **자가개선 사이클**이 (1) 에이전트의 성공률이 일정 기준 미만으로 떨어지거나 (2) 사용자 피드백으로 오류가 지적될 때 자동으로 트리거되며, "한 세션당 최대 5회 개선 시도", "신뢰도 0.7 초과하는 변경만 적용", "개선 실패 시 자동 롤백" 등의 **안전장치**를 갖추고 있습

니다 [20]. 요컨대 Claude가 처음에는 관계 판단을 유보하거나 놓친 경우라도, 이러한 **루프를 통해 원인 → 개선 → 재실행**의 과정을 거치면서 시간이 지날수록 더 포괄적이고 정확한 관계 맵핑이 가능해지는 구조입니다 [20].

현행 한계 (Current Limitations)

현재의 자가개선 루프 구조는 **개념 관계 정의 정확도 향상**을 목표로 설계되었으나, 몇 가지 한계점이 드러납니다:

- **실패 감지 기준의 단순성**: 성공/실패 판단이 주로 **출력 유무나 포맷 오류**에 치중되어 있어 미묘한 품질 저하를 정량적으로 평가하지 못합니다. 예를 들어, 한두 개 관계를 놓쳤으나 나머지는 정확한 경우도 현재는 거의 **이진 판단(성공 vs 실패)**으로 처리됩니다. **성공률**이나 **품질 점수**에 기반한 세밀한 기준이 부재하여, 때로는 사소한 누락도 전면적인 루프를 촉발하거나 반대로 숨은 오류를 간과할 위험이 있습니다. 또한 **QualityAgent의 검증 결과**가 정량 점수로 환산되지 않으므로, 어느 정도의 품질 편차까지 허용할지에 대한 체계가 없습니다.

- **SocraticMediator 질의전략의 제한**: SocraticMediator는 현재 **순차적 Q&A 대화**로 원인 분석을 수행하고 있어, **병렬적 탐색**이나 **동시 다발적 확인**이 부족합니다. 한 번에 한 질문씩 진행하다 보니 전체 원인 분석에 시간이 걸리고, Claude Max와 같은 동시 실행 능력을 충분히 활용하지 못하고 있습니다. 또한 질의 내용이 주로 대상 에이전트의 자기보고에 의존하는데, taxonomy나 외부 지식 등 **다른 관점에서의 교차 검증 질의**가 부족합니다. 즉, 프롬프트 자체의 한계, 개념 데이터의 빈틈, 모델 지식의 부족 등 **원인 가설별로 분기하여 질문**하는 전략이 미흡합니다.

- **SelfImprover 개선안 검증의 부족**: SelfImprover는 SocraticMediator의 권고안을 받아 개선 조치를 적용하지만, 제안된 **개선안의 효과를 사전에 검증**하거나 **여러 대안을 비교 평가**하는 절차가 없습니다. 현재는 confidence 0.7을 넘는 액션이면 바로 적용하고(신뢰도 미달 시에는 "추가 Socratic 조사 권고"로 종료) [21] [22], 적용 후에야 개선되었는지 **사후 검증**하는 구조입니다 [18]. 이로 인해 첫 개선안이 최적이 아닐 경우 루프를 거듭해야 하고, 각 루프마다 시간이 소모됩니다. 또한 **개선안 자체의 신뢰도**는 모델이 주관적으로 부여한 값이므로 이를 **교차 검증**하거나 너무 낮은 경우 다른 대안을 동시에 시험해보는 메커니즘이 필요합니다.

- **루프 회차 관리 및 전략 변화 부족**: Meta-Orchestrator는 현재 고정된 최대 5회 반복 한도 내에서 **거의 동일한 절차**를 반복합니다. 1회 실패 시 2회째도 유사한 Socratic 질의 → 개선 적용 경로를 따르는데, 각 시도 간에 **전략의 차별화**나 **점진적 강화**가 부족합니다. 예컨대 1차 개선이 실패했다면 2차에는 **다른 접근**(외부 지식 활용, 모델 파라미터 조정 등)을 시도할 법하지만, 현재 구조는 최대 2회 재분석 정도로 제한되어 있어 다양성이 떨어집니다 [19]. 결국 복잡한 이슈의 경우 같은 방법을 반복하게 되어 **루프가 소모적**으로 길어지거나, 5회 이후 해결되지 못한 채 종료될 수 있습니다.

- **실패 시 대처 및 인간 개입 미정의**: 최대 횟수(5회)까지 개선을 시도하고도 문제가 풀리지 않을 경우, 시스템이 어떻게 대응하는지 명확한 프로토콜이 없습니다. 자동 루프가 실패한 케이스를 그대로 두면 해당 관계는 영구히 누락될 수 있고, 반면 지속해서 재시도하도록 두면 무한 루프나 비용 낭비가 될 것입니다. 또한 사람의 개입이 언제, 어떻게 트리거되어야 하는지 조건이 모호합니다. 현재는 품질 게이트를 통한 자동 승인/거부와 자동 롤백까지만 정의되어 있을 뿐, **언제 인간 전문가의 검토를 요구할지**에 대한 기준이 없습니다. 이는 최종적으로 해결되지 않은 관계 누락이나 시스템 지식 베이스의 맹점을 남길 우려가 있습니다.

위와 같은 한계를 개선하기 위해, **코드 구조**, **프롬프트 전략**, **병렬화 활용**, **모니터링 방법**, **재학습 트리거** 측면에서 구체적인 보완 방안을 제안합니다. 특히 실패 감지 기준의 정교화, SocraticMediator 질의전략 고도화, SelfImprover 개선안 품질 검증 강화, Meta-Orchestrator의 루프 관리 전략 다변화, 그리고 최종 실패 시 폴백/인간 개입 조건을 중심으로 개선 사항을 설계하였습니다.

개선 방안 1: 루프 실패 감지 기준 고도화 (성공률 및 품질 점수 기반)

현재 단순한 성공/실패 플래그 대신, **정량적인 품질 평가 지표**를 도입하여 루프 트리거의 정밀도를 높입니다. 구체적으로:

- **성공률 및 품질 점수 측정**: RelationshipDefiner의 출력에 대해 **품질 점수(예: 0~100)**를 계산하는 **평가 에이전트**를 추가 배치합니다. 예를 들어 기존 Apollo 평가자와 유사하게 결과물을 0~100으로 **가혹하게 채점**하고 구체적 개선점을 피드백하도록 할 수 있습니다 [23]. 이 **품질 평가 에이전트(Apollo)**는 QualityAgent의 검증 결과(포맷 오류, 누락 개수 등)를 바탕으로 점수를 부여하거나, 별도의 critique 프롬프트를 통해 Claude로 하여금 결과의 완전성과 정확성을 평가하게 할 수 있습니다 [24]. 점수가 **90/100 미만**이면 루프를 돌리는 식으로 임계값을 설정하면, "그냥 그런대로 괜찮은" 수준을 넘어 **거의 완벽한 결과(예: ≥90점)**가 나올 때까지 반복하도록 유도할 수 있습니다 [23]. 이러한 **정량 루프 기준(quality loops)**을 도입하면, 이전처럼 출력이 조금만 부족해도 실패로 취급하거나, 반대로 치명적 오류가 숨어 있어도 성공으로 간주하는 일을 방지하고 **세밀한 기준**으로 루프 트리거를 결정할 수 있습니다.

- **성공률 모니터링 및 동적 임계**: Meta-Orchestrator에 **실시간 성능 모니터링** 기능을 강화하여, 에이전트별 **평균 성공률**과 최근 **품질 점수 추이**를 모니터합니다 [25]. 예를 들어 최근 시도들의 품질 점수가 지속 하락하거나 특정 임계치 미만으로 떨어지면 루프를 촉발하도록 합니다 [20]. 임계치는 **동적으로 조정**하는데, 초반에는 낮게(예: 80점) 설정하여 작은 문제도 잡아내고, 반복 개선 후에는 점차 높이는 방식으로 **수렴 기준**을 엄격히 합니다. 또한 **사용자 피드백** 연계 트리거를 명확히 정의하여, 사용자가 결과에 오류를 지적하거나 불만족을 표명하면 즉시 해당 세션에 루프를 가동합니다 [20]. 이 때 사용자 피드백의 심각도에 따라 가중치를 두어, 치명적 오류 지적 시에는 점수와 무관하게 바로 개선 프로세스를 시작하고, 경미한 제안 수준이면 일정 점수 차감 정도로 반영하는 식입니다.

- **다중 척도 병합**: 실패 감지에 하나의 지표만 쓰기보다 **여러 품질 척도**를 결합합니다. 예를 들어, 출력 정확도(QualityAgent 검증 결과에서 오류/누락 항목 수), 모델 신뢰도 (Claude가 제공한 평균 confidence), 출력 일관성 (지식 그래프 내 논리적 모순 검사) 등을 산출하고, 이를 종합한 **가중 합 점수**를 사용합니다. 이 점수가 설정 임계 이하일 때 루프를 돌리도록 하면, 단일 지표의 편향을 줄이고 전반적 품질을 고려하게 됩니다. 또한 **변화량 기준**도 고려하여, 이전 개선 대비 점수가 개선되지 않았으면 추가 루프를 돌리는 식으로 **증분 개선 여부**를 감지합니다.

- **품질 게이트와 연계**: 기존 품질 게이트(Quality Gate)는 주로 **변경 영향 범위**에 대한 검증이었지만 [14] [15], 여기에 **출력 품질 평가**도 통과 조건으로 추가합니다. 즉, 개선 적용 후 재실행한 결과의 품질 점수가 기대 수준에 도달하지 못하면 품질 게이트를 **FAIL** 처리하여 추가 개선안을 모색하게 합니다. 이로써 단순히 "변경이 안전한가" 뿐 아니라 "변경으로 충분한 효과를 보았는가"까지 게이트에서 확인하게 되어, 부적절한 개선으로 루프가 종료되는 일을 막습니다.

위와 같은 정교한 기준을 통해, 시스템은 언제 루프를 시작하고 언제 끝낼지에 대한 **명확한 수치적 근거**를 갖게 됩니다. 이는 오탐/누락에 민감하게 반응하여 높은 품질을 보장하면서도, 불필요한 개선 루프는 줄여 **효율성**을 높이는 효과가 기대됩니다.

개선 방안 2: SocraticMediator 질의 전략 고도화 (Claude Max 병렬 활용)

SocraticMediator의 원인 분석 단계에 **더욱 전략적인 질의 기법**을 도입하여, **병렬 질문 수행**과 **다각도 진단**을 구현합니다. 주요 개선 내용은 다음과 같습니다:

- **병렬 질의 (Parallel Q&A)**: 현재 SocraticMediator는 한 번에 한 질문씩 순차 진행하지만, Claude Max 구독으로 **동시에 최대 20 세션**을 활용할 수 있으므로 이를 적극 활용합니다. 예를 들어 **Task 도구**를 이용해 한 메시지 내에 여러 질의를 병렬로 던질 수 있습니다 [26]. 실제 코드 예시에서도 "한 에이전트에 첫 질문 3개를 병렬로 보낸다"는 패턴이 권장되고 있습니다 [26]. 따라서 SocraticMediator가 IssueReport를 받으면, **초기 탐색 질**

문 몇 개를 한꺼번에 실행합니다. 예컨대: (1) 대상 에이전트(예: RelationshipDefiner)에게 "현재 성공률과 평균 실행 시간을 알려줘", (2) 동일 에이전트에게 "가장 빈번한 에러 유형은 무엇인가?", (3) 품질 검사 에이전트에게 "이번 출력에서 감지된 형식 오류나 누락은?" 등을 **동시에 질의**합니다. Claude Max의 멀티 세션 능력을 활용해 이러한 병렬 질의를 수행하면, 순차적으로 하나씩 묻는 것보다 **분석 속도가 향상**되고 **병목 없이 정보를 수집**할 수 있습니다 [27]. 실제로 소규모 에이전트 그룹(3~5개)을 병렬 운영하면 약 90% 이상의 지연 감소 효과가 있다는 보고가 있으며 [27], 이를 SocraticMediator 단계에 적용하여 루프 전체 시간을 크게 단축할 수 있습니다.

- **체계적 원인 가설 수립**: SocraticMediator가 문제를 접수하면 곧바로 질문하기보다, 먼저 **TodoWrite** 등을 활용해 **조사 계획**을 세우도록 프롬프트를 개선합니다 [28]. 구체적으로, 오류의 가능 원인을 몇 가지 범주로 나누고 (예: 프롬프트 문제, taxonomy 범위 누락, 지식베이스 부족, 의존 에이전트 오류, 모델 한계 등) 각 가설별로 확인할 질문 리스트를 작성합니다. 그리고 질문을 던질 때 해당 가설을 커버하도록 **체계적으로 진행**합니다. 이렇게 하면 분석이 **주관적 추측**에 치우치지 않고, 매번 **유사한 체크리스트**를 따라감으로써 놓치는 영역 없이 진단할 수 있습니다. 예를 들어 첫 라운드에 프롬프트 및 taxonomy 관련 질문을 집중적으로 하고, 얻은 답을 토대로 다음 라운드엔 모델 지식이나 외부 정보 쪽을 파고드는 식입니다.

- **관련 에이전트/도구 활용 질의**: SocraticMediator가 필요시 **다른 전문 에이전트**나 **외부 지식**에도 질문을 보낼 수 있도록 개선합니다. 현행 구조에서는 주로 대상 에이전트 자신과의 대화에 국한되지만 [29], 개선안에서는 예컨대: "연관 개념 A와 B 사이의 관계를 외부 지식 그래프나 문헌에서 찾아봐"라고 **ResearchAgent(조사 에이전트)**에게 요청할 수 있습니다. Claude Max 세션이 풍부하므로 SocraticMediator는 병렬로 **web 검색**이나 **사전 온톨로지** 질의도 수행해볼 수 있습니다. 또 DependencyAgent에게 "이 에이전트가 의존하는 모듈 중 문제가 보고된 것이 있는가?" 묻거나, QualityAgent에게 상세 로그를 재요청하는 등 **교차 확인**을 합니다. 이러한 다중 에이전트 질의로 한 에이전트의 자기진단 한계에서 벗어나, **외부 시각**으로 원인을 찾을 수 있습니다. 특히 개념 지식의 부족으로 인한 문제라면 웹 검색 결과나 외부 온톨로지(OntoMathEdu 등)에서 두 개념 사이 관계 유무를 조사해 Claude에게 피드백으로 제공함으로써, Claude가 "모델이 모르는 관계였다"는 원인을 깨닫게 할 수 있습니다.

- **추가 정보 제공 및 대화 맥락 유지**: SocraticMediator의 프롬프트를 개선하여, **대상 에이전트의 내부 상태나 로그**를 더 풍부하게 넘깁니다. 예컨대 RelationshipDefiner가 최근 실행에서 산출한 원시 응답(파싱 전 텍스트)이나 에러 스택 트레이스 등을 **첨부 정보**로 제공하면, Claude가 원인분석에 참고할 수 있습니다. 또한 Socratic 대화 로그를 **Markdown으로 기록**하여 맥락을 잃지 않고 [30], 동일 이슈의 추가 질의 시 이전 답변들을 요약 참조함으로써 일관된 추론을 보장합니다. Claude의 긴 맥락 창을 활용해 최대한 이전 턴의 정보를 잊지 않도록 하고, 필요하면 Memory-Keeper 도구를 이용해 중요한 데이터를 대화 사이에 공유합니다 [31].

- **저확신 원인에 대한 재질의**: 만약 SocraticMediator가 도출한 root cause의 **confidence score**가 낮게 (<0.7) 나오는 경우, **즉시 후속 질의 세션**을 자동 개시하도록 합니다. 현재 SelfImprover는 confidence 0.7 미만 원인일 때 "근본 원인 분석 불충분, 추가 Socratic 분석 권고"라며 실패로 처리합니다 [22]. 이를 Meta-Orchestrator 측에서 감지하여, 같은 이슈에 대해 **새로운 각도**의 SocraticMediator 세션을 한 번 더 시작합니다. 이때는 첫 세션에서 다루지 못한 질문을 보완하거나, **다른 LLM**(예: GPT-4)에게 동일 로그로 진단해보는 방안도 고려할 수 있습니다. 병렬 인프라를 활용하면 **두 종류 모델**에 동시에 Socratic 대화를 시켜 결과를 비교하는 것도 가능하므로, Claude의 자체 한계로 인한 miss를 줄일 수 있습니다. (단, 본 시스템이 Claude 중심이므로 타 모델 활용은 옵션으로 제안합니다.) 어쨌든 한 번의 Socratic 대화로 충분한 확신을 얻지 못하면, 자동으로 **심층 재탐색**을 수행해 원인 규명을 최대한 철저히 합니다.

이러한 개선으로 SocraticMediator는 **신속하면서도 다각적인 원인 분석**이 가능해집니다. Claude Max의 병렬 처리 능력을 활용해 **루프 지연을 최소화**하면서 [27], 다양한 질의로 **모델의 맹점을 폭넓게 진단**할 수 있습니다. 결국 더 정확한 원인 파악이 이루어져 SelfImprover에 **양질의 입력(개선 권고)**을 제공하고, 이는 전체 루프의 성공률을 높여줄 것으로 기대됩니다.

개선 방안 3: SelfImprover 개선안 품질 검증 및 신뢰도 임계치 전략

SelfImprover 단계에서는 **개선안 적용 전후**로 그 품질을 엄격히 통제하여, 부적절한 수정을 방지하고 효율적인 개선이 이뤄지도록 합니다. 주요 제안은 다음과 같습니다:

- **신뢰도 임계치 준수 및 동적 조정**: 현재 시스템에서 SelfImprover는 RootCauseAnalysis의 confidence가 0.7를 넘어야 개선 행동을 실행하며 [32], 각 개선 액션에도 개별 confidence를 부여하고 있습니다 [33] [34]. 이 원칙을 지속 철저히 지키되, 임계치 값을 **상황에 따라 동적 조정**합니다. 예를 들어, 만약 치명적인 오류 원인으로 **아주 확실한 해결책**이 제시된 경우(예: confidence 0.95)에는 한 번의 개선으로 충분하므로 이후 루프를 생략하거나 조기 종료합니다. 반대로 여러 번 시도했는데도 매번 confidence가 0.75~0.8 사이에 머문다면, 4~5회차에는 임계치를 약간 완화하여(예: 0.65 이상) **부분적인 개선안**도 시도해봅니다. 이는 **시간이 촉박한 상황**에서 일단 부분 해결이라도 하고 나중에 인간 검토를 받게 하기 위함입니다. 다만 기본적으로는 "confidence 0.7 이하의 변경은 하지 않는다"는 원칙을 유지하여, Claude가 자신 없는 수정을 함부로 적용하지 못하게 합니다 [21]. 실제 SelfImprover 안전 수칙에서도 "신뢰도 < 0.7인 변경은 절대 하지 말 것"이라고 명시되어 있습니다 [35]. 이러한 임계치 기반 통제로 **무리한 추측성 수정**을 배제하고, 안정적인 개선만 누적되도록 합니다.

- **개선안 사전 검증 (Dry-Run 테스트)**: SelfImprover가 제안한 수정이 적용되기 전에, **사전 검증 단계**를 추가합니다. 예를 들어 **"시뮬레이션 실행"**을 통해, 변경된 프롬프트/코드로 RelationshipDefiner를 다시 실행해보는 것입니다. 이 때 전체 입력을 돌리기엔 비용이 크므로, **문제가 된 특정 개념쌍**이나 **테스트용 샘플 입력**에 대해 빠르게 실행해봅니다 [36]. Meta-Orchestrator의 verification 기능을 활용해, 개선 적용 전후의 해당 사례 결과를 비교함으로써 실제로 누락 관계가 포착되는지 확인합니다 [37]. 예컨대 A–B 관계를 놓친 문제가 있었고 SelfImprover가 taxonomy에 새로운 관계 유형 X를 추가했다면, 수정된 에이전트를 즉시 A–B에 재적용해 X로 판별되는지 검사해보는 것입니다. 이러한 **드라이런 테스트**는 병렬 세션으로 빠르게 수행 가능하며, 만약 여전히 관계가 안 나온다면 그 개선안은 효과 없을 가능성이 높으므로 적용을 취소하거나 다른 안을 모색합니다. 이로써 **실제 루프 반복에 들어가기 전에** 개선안의 유효성을 1차 거르는 셈입니다.

- **복수 개선안의 병렬 평가**: SelfImprover가 복수의 개선 아이디어를 내놓았을 때 (예: 권고안 3가지 중 2가지를 적용 가능), **병렬 분기 실험**을 고려합니다. Claude Max의 다중 세션을 이용해 **여러 가설 개선안**을 각각 별도 세션의 임시 분기에서 적용한 후, 동일 테스트 입력에 대한 성능을 비교합니다. 예를 들어 개선안 A: 프롬프트에 사례 추가 vs 개선안 B: 관계 유형 정의 수정 두 가지가 있을 때, 두 개의 가상 환경에 각각 적용해 A/B 테스트를 돌려보는 것입니다. 그런 다음 **품질 평가 점수**나 누락 해결 여부를 비교하여 더 나은 쪽을採용합니다. 이러한 접근은 실시간으로 코드를 분기하여 적용해야 하므로 구현 난이도가 있지만, **메모리-keeper나 Git 분기** 기능을 활용하면 가능할 수 있습니다. 이를 통해 **최적의 개선안**을 1회의 루프에서 선별할 수 있어, 굳이 순차적으로 두 번 시도하지 않아도 됩니다. 다만 이 기법은 복잡성을 높이므로, **고비용 루프 상황(예: 4~5회차 돌입 전에)**에만 제한적으로 사용하도록 합니다.

- **개선안 적용 후 엄격 검증**: 변경 사항을 실제 코드베이스에 적용한 후에는 이미 구현된 **Verification 단계**를 더욱 엄격하게 운영합니다. 현재 Meta-Orchestrator는 개선 후 동일 입력 재실행 및 기본 테스트로 성공 여부를 판단합니다 [17] [36]. 여기에 **회귀 테스트**와 **성능 모니터링** 기준을 추가합니다. 예컨대, 기존에 정상 동작하던 다른 개념 관계 판단 사례 몇 가지를 추가로 실행해 **부작용이 없는지** 확인합니다 (QualityAgent나 Tester 에이전트가 존재한다면 이 역할을 맡김). 또한 실행 시간이나 비용이 기존보다 2배 이상 나빠지지 않았는지도 확인하여, 성능 저하를 일으킨 개선이면 실패로 간주합니다 [38]. 이러한 검증 항목들은 이미 설계에 일부 포함되어 있으므로(SelfImprover 성공 기준에 "모든 수정 후 구문 오류 없음", "기존 테스트 통과" 등이 명시 [39] [40]), 이를 확실히 적용하도록 합니다. 만약 검증 단계에서 문제를 발견하면 **즉시 롤백**하고 다음 대안을 시도하거나 루프를 종료합니다 [41] [42].

- **Continuous Learning 및 피드백 반영**: SelfImprover가 적용한 최종 변경사항은 Improvement Manager나 Memory-Keeper에 기록되어 향후 학습에 활용되게 합니다 [43] [44]. 즉, 동일한 유형의 문제가 다시 발생하면 이전에 효과적이었던 개선 수법을 자동으로 제안하거나, 심지어 사전에 대응하도록 하는 것입니다. 이를테면 "LaTeX 수식 처리 오류"가 과거 개선된 적 있다면, 다음번 유사 오류 시 곧바로 해당 프롬프트 패

치를 적용해본다든지 하는 식입니다. 이런 **자기학습 메커니즘**을 통해 개선안을 누적 학습함으로써, 같은 문제로 루프를 여러 번 반복하는 비효율을 줄입니다.

위 개선으로 SelfImprover는 **신중하고 검증된 개선만 적용**하게 되어, 잘못된 수정으로 인한 역효과를 최소화합니다. 임계치 관리로 **낮은 품질 수정**을 배제하고, 사전/사후 검증을 통해 **개선 효용이 입증된 경우만 채택**하므로 루프 효율과 신뢰성이 높아집니다. 또한 여러 아이디어를 비교함으로써 **최적 해법을 탐색**할 수 있어, 전체 반복 횟수를 줄이고도 목표 품질에 도달할 가능성이 커집니다.

개선 방안 4: Meta-Orchestrator의 루프 회차 관리 및 시도 간 전략 차별화

Meta-Orchestrator에서 루프를 제어하는 방식을 **적응형(Adaptive)**으로 개선하여, 각 반복 시도마다 **전략을 달리**하고 최대 반복 횟수를 **탄력적으로 운용**합니다. 이를 통해 불필요한 반복을 줄이고, 남은 시도 횟수를 효과적으로 활용할 수 있습니다:

- **동적 반복 횟수 결정**: 일률적으로 최대 5회 반복하기보다는, **상황에 따라 다른 최대 회차**를 설정합니다. 예를 들어 **개선 영향 범위**나 **문제 심각도**에 따라, Meta-Orchestrator가 처음에 허용할 루프 횟수를 결정하도록 합니다. 이미 코드에서도 "치명적 컴포넌트 영향 시 최대 2회 피드백 라운드" 등의 규칙이 존재합니다 [45]. 이를 일반화하여, 문제를 범주화하고 각 범주마다 최대 시도를 정합니다. 예를 들어 taxonomy 자체 한계로 새로운 관계 유형 추가가 필요한 경우 – 이때는 큰 구조 변경이므로 2~3회 이내로 제한하고 이후는 사람 검토; 반면 프롬프트 예시 부족으로 인한 단순 오류 – 1회 개선으로 해결될 수 있으므로 추가 루프 불필요 등입니다. 또한 **1회 개선 후 성과**를 평가해 남은 횟수를 조정합니다. 첫 개선으로 현저한 향상이 있었으면 굳이 5회 다 돌지 않고 1~2회에서 마무리하도록 하고, 반대로 별 진전이 없었다면 남은 회차를 모두 쓰기보다는 **빠르게 인간 개입**으로 전환하는 등 유연하게 대응합니다.

- **시도 간 전략 차별화**: 각 루프 **회차별로 초점을 달리하는 전략**을 사전에 설계합니다. 동일한 방법을 반복하지 않고 **점진적으로 강도를 높이거나 방향을 바꾸어** 난제를 공략합니다. 예시 전략은 다음과 같습니다:

루프 회차	전략 및 초점	실행 조치 예시
1회차	**경미한 개선** – 보수적 수정	프롬프트 문구 다듬기, 예시 1-2개 추가, 파싱 포맷 엄격히 명시 등 [3]
2회차	**지식 보강** – 외부 정보 활용	관련 개념에 대한 추가 정보 제공 (온톨로지/위키 검색), 모델에게 힌트 투입
3회차	**구조 개편** – 정의/구조 변경	taxonomy에 누락 타입 추가, 관계 정의 수정, 새로운 예제 대거 투입
4회차	**아키텍처 확장** – 도구/에이전트 추가	필요한 경우 새로운 도구 권한 부여 (예: 웹검색), 전문 sub-agent 생성 [46]
5회차	**최후 시도** – 제한 완화 및 대안 모색	신뢰도 임계 일부 완화하여 후보 관계도 출력, 타 모델 검토 병행

위와 같이 회차별로 **난이도와 폭을 확장**하는 것입니다. 첫 번째 시도는 시스템 내부에서 간단히 해결될 법한 **경미한 원인**(프롬프트 부족 등)에 집중합니다. 두 번째는 Claude 자체 지식의 한계를 보완하기 위해 **외부 데이터나 맥락**을 추가로 주입합니다. 세 번째는 문제의 근본이 taxonomy나 구조적 설정에 있다고 보고 **체계 자체를 개편**합니다. 네 번째는 기존 구성만으로 어려운 경우 **새로운 도구나 에이전트 도입**을 모색합니다 (예: 특정 패턴의 관계만 전문적으로 처리하는 agent를 생성하거나, Code 수정이 아니라 데이터 포스트프로세싱으로 해결하는 방안 등). 마지막 다섯 번째는 그래도 해결되지 않을 때 **모델의 제약을 일부 풀어**서라도 답을 얻고자 합니다. 예를 들어 confidence threshold를 약간 낮춰서라도 모델이 애매하지만 **가능성 있는 관계 후보**를 내보내게 유도한 뒤, 이를 나중에 인간이 검토하도록 넘기는 방식입

니다. 이 최종 시도는 어디까지나 **fallback** 성격으로, 자동 완결이 어려운 부분을 인간에게 넘기기 전에 최대한 단서라도 수집해두는 단계입니다.

- **이전 시도 정보 활용 및 변경**: 각 루프 시도 간에 **학습된 정보를 공유**하고 전략을 변경합니다. Meta-Orchestrator는 이전 루프에서 무엇이 시도되었고 어떤 결과가 나왔는지 파악하고 있어야 합니다 [47] [48]. 예컨대 1회차에 prompt 개선을 했는데 효과가 없었다면 2회차에는 동일한 prompt 영역을 반복 수정하지 않도록 합니다. 이를 위해 **루프 이력 로그**를 관리하고, SocraticMediator에게 2차 원인 분석 시 "이전에 프롬프트를 보강했으나 실패했습니다" 등의 맥락을 제공해 **새로운 원인**을 찾도록 유도합니다. 또한 SelfImprover도 이전에 적용된 액션 목록을 참고하여 중복/충돌되는 제안을 피하고, 대신 다른 유형의 액션을 권고안에서 선택합니다. 이렇게 **비슷한 개선을 재시도하지 않게** 만들어 매 회차 **새로운 돌파구**를 마련하도록 합니다.

- **모델 파라미터 및 모드 조정**: 루프 회차가 진행됨에 따라 Claude의 **프롬프트 모드나 생성 파라미터**를 변경해보는 것도 전략 중 하나입니다. 예를 들어 1~2회차는 정확도 위주(temperature 낮게, 매우 보수적 지시)로 가다가, 막판 시도에는 temperature를 높여 **보다 창의적이거나 잠재적 가능성까지 제안**하도록 할 수 있습니다. 처음에는 false positive를 극도로 꺼리다가, 마지막에는 false positive 위험이 있어도 놓친 관계를 추측해서라도 말하게 하는 것입니다. 이를 통해 평소 모델이 숨겨둔 연관 지식을 **brainstorm** 형식으로 끌어낼 수도 있습니다. 물론 나온 최종 제안들은 확실성이 낮으므로 직접 채택하지 않고 인간 검증을 거치게 해야 합니다. 이러한 파라미터 조절은 Meta-Orchestrator가 각 회차에 사용한 Claude 세션 생성 설정을 조정하는 형태로 구현합니다.

이러한 **전략적 루프 관리**로, 시스템은 루프를 거듭할수록 **공격 방법을 바꾸어** 가면서 문제를 해결합니다. 동일한 접근을 반복하지 않기에 **비효율적 루프를 줄이고**, 다양한 시도를 통해 문제 원인을 다각도로 해결할 확률을 높입니다. 또한 시도 횟수도 상황에 맞게 제어되어 **최소한의 반복으로 최대 효과**를 내도록 조율됩니다.

개선 방안 5: 실패 시 폴백 전략 및 인간 개입 트리거 정의

끝내 자동 루프만으로 문제를 해결하지 못하는 경우를 대비해 **폴백(fallback) 방안**과 **인적 개입(human-in-the-loop) 시점**을 명확히 정합니다. 이는 최악의 경우에도 시스템의 학습이 멈추지 않고, 잘못된 결과가 방치되지 않도록 하기 위한 안전망입니다:

- **자동 폴백 시나리오**: 최대 루프 회차까지 시도했음에도 특정 관계를 찾아내지 못했다면, 시스템은 해당 관계쌍을 "판단 불가" 상태로 **표시 및 기록**합니다. 예를 들어 출력 지식 그래프에 A–B 사이를 "모름(unknown)" 또는 "추후 검토 필요"로 태그해 두고, 나중에 이 항목만 모아서 별도 리포트를 생성합니다. 이를 통해 **누락 관계를 투명하게 드러내고 추적**할 수 있습니다. 이 폴백 출력은 사용자가 그 부분만 수동으로 보완하거나, 다른 수단으로 알아볼 수 있게 해줍니다. 또한 시스템은 Improvement Manager에 이 사례를 저장하여, 향후 유사 쌍이 등장하면 "이전에 해결 못한 케이스"로 인지하게 합니다. 이렇게 하면 동일 실패가 반복될 때 자동으로 루프를 길게 돌리지 않고 곧바로 **인간 검토** 경로로 보내도록 최적화할 수 있습니다.

- **인간 전문가 개입 트리거**: 다음과 같은 조건에서 Meta-Orchestrator는 **즉시 인간 개입을 요청**하도록 합니다: (1) **최대 루프 횟수 소진** – 예: 5회 모두 시도 후에도 품질 게이트 통과 실패 [20], (2) **근본 원인이 모델 지식 한계로 판명** – SocraticMediator 분석 결과 "모델의 수학 지식베이스에 없는 개념 관계"로 밝혀진 경우, (3) **품질 게이트 연속 실패** – 작은 수정으로는 위험도 때문에 적용이 거부되고, 큰 수정 없이는 해결이 안 될 딜레마 상황 [14] [15], (4) **Critical Component 영향** – 개선안이 시스템 핵심 구성요소를 변경해야 해서 자동 적용이 위험한 경우 [49]. 이러한 상황에서는 Meta-Orchestrator가 사용자나 관리자에게 **notification**을 보냅니다. 예컨대 "개념 A–B 관계를 자동 규명하지 못했습니다. 원인: 새로운 관계 유형 필요. 전문가 검토 필요."라는 메시지와 함께, 지금까지의 Socratic 대화 로그와 개선 시도 내역을 첨부합니다 [50]. 이를 통해 사람이 컨텍스트를 이해하고 정확한 관계를 판단하거나, taxonomy에 근본적인 변화를 줄 결정을 내릴 수 있습니다.

- **인간 확인 단계 도입**: 일부 개선안은 자동 적용 전에 **휴먼-in-the-loop 승인**을 요구하도록 합니다. 예를 들어 SelfImprover가 새로운 관계 유형 추가나 다수 파일 수정 같은 **대규모 변경**을 제안했을 때, 바로 적용하지 않

고 관리자에게 승인 요청을 보냅니다. 앞서 품질 게이트에서도 임계치를 넘는 변경은 자동승인을 막지만 [14], 추가로 "인적 승인 필요" 플래그를 세워 둘 수 있습니다. 관리자는 Claude가 제안한 개선안을 검토하여 승인하거나 조정 지시를 내릴 수 있습니다. 특히 교육용 개념체계의 민감한 변경(예: ontology 확장)은 반드시 전문가 판단을 거치도록 워크플로를 삽입함으로써, **잘못된 자동수정으로 지식체계가 훼손되는 위험**을 방지합니다.

- **재학습 데이터 수집 및 모델 개선**: 자동 루프가 해결 못 한 사례들은 곧 **모델 자체의 약점**을 의미하므로, 이를 장기적으로 해결하려면 Claude 모델의 **재학습** 또는 **프롬프트 지속 개선**이 필요합니다. 시스템은 이러한 실패 사례(입력, 기대 출력, 실제 Claude 출력, 원인 분석 내용, 최종 인간이 정정한 답 등)를 **데이터베이스에 축적**합니다. 충분한 사례가 쌓이면 이를 Anthropic에 제공해 향후 모델 업데이트에 반영하거나, 내부적으로 **소형 파인튜닝** 모델을 학습시켜 inference-time에 결합하는 방안을 모색합니다. 비록 현재 Claude 자체를 파인튜닝할 수 없다 하더라도, 예컨대 "관계 추론 보조 모델"을 하나 훈련시켜 Claude의 판단을 검증/보완하도록 할 수 있습니다. 또한 축적된 사례로부터 **프롬프트 가이드라인**을 계속 개선합니다 [51] [52]. 본 연구 보고서에서도 외부 온톨로지 사례를 분석해 프롬프트 정의를 다듬자는 제안을 했듯이 [53] [52], 운영 중 발견된 실패 패턴을 반영해 taxonomy 정의를 명확히 하고 예시를 추가하는 작업을 주기적으로 수행합니다. 이러한 **지식 기반의 지속 업데이트**가 병행되면, 시간이 지날수록 모델이 처음 모르고 놓쳤던 관계들도 학습하게 되어 추후엔 자동으로 처리할 가능성이 높아집니다 [20].

요약하면, 자동 루프로 안 되는 경우 **포기하지 말고 인간과 협업**하는 방향으로 구조를 확장합니다. 폴백 태그와 알림으로 누락을 투명하게 관리하고, 사람의 판단을 받아 시스템 지식을 보완함으로써 **장기적 학습 사이클**을 완성합니다. 또한 이러한 인간 개입이 필요한 지점을 명확히 프로토콜화하여, 운영 시 혼선을 줄이고 중요한 의사결정은 사람이 관여하도록 해 **신뢰성**과 **책임성**을 확보합니다.

기대 효과 (Expected Benefits)

제안된 개선 방안을 구현하면 다음과 같은 이점이 기대됩니다:

- **실패 감지 정밀화**: 성공률/품질 점수 기반의 루프 트리거로 인해, 루프가 **필요한 경우에만** 발동하고 정확히 **문제가 있는 부분만** 개선하도록 유도됩니다. 이는 불필요한 반복을 줄이고, 반대로 숨겨진 오류를 놓치지 않게 해 **관계 맵 품질**을 높입니다 [23] [24]. 수치적 기준 도입으로 개선 전후 성능을 계량화하여 **개선 효과를 객관적으로 평가**할 수도 있습니다.

- **신속한 원인 분석**: SocraticMediator의 병렬 질의와 구조적 대화로 원인 분석 소요 시간이 크게 단축되고, **포괄적인 진단**이 가능해집니다 [27] [26]. Claude Max의 20세션을 풀로 활용하여 다각도로 탐색하므로, 한번의 루프에 더 많은 통찰을 얻어 **루프 반복 횟수 자체**가 줄어드는 효과도 기대됩니다.

- **안정적이고 효과적인 개선 적용**: SelfImprover 단계에서 개선안의 **검증→적용→재검증** 과정이 강화되어, 적용되는 변경의 신뢰도가 한층 높아집니다. 그 결과 개선 시도가 곧바로 **성공률 향상**으로 이어질 확률이 증가하고, 잘못된 수정으로 오히려 성능이 저하되는 경우는 최소화됩니다 [35] [39]. 또한 다중 개선안 실험으로 **최적해 탐색**이 가능해져, 최소한의 변경으로 최대 효과를 보는 효율적 수정이 이루어집니다.

- **루프 효율 및 지능 향상**: 각 루프 회차마다 전략을 달리하고 학습을 반영함에 따라, 시스템이 **시도할수록 똑똑해지는** 양상을 띱니다. 정적인 5회 반복보다 적은 횟수로 문제를 풀어낼 수 있고, 반복될수록 성과 향상이 가속되어 **누적 개선 효과**가 커집니다. 특히 다양한 전략을 적용함으로써 난해한 문제도 한 가지 접근에 막히지 않고 여러 경로로 풀어낼 가능성을 높입니다.

- **안전망 확보 및 지속 학습**: 폴백과 인간 개입 체계 덕분에, 자동화의 한계 상황에서도 시스템이 **안정적으로 대응**합니다. 해결 못한 부분을 투명하게 남김으로써 잘못된 결과 제공을 피하고, 사람의 판단을 받아 지식베이스를 교정하므로 **최종 사용자 신뢰도**도 향상됩니다. 장기적으로 실패 사례를 모아 모델/프롬프트를 개선하는 **재학습 루프**까지 연결되면, 시간이 지날수록 시스템의 **정확도와 범용성**이 증가하여 결국 자동 루프의 성공률도 함께 상승할 것입니다 [20].

이상의 개선안을 통해, Claude 기반 개념 관계 정의 시스템은 현재의 **자가개선 루프**를 한층 고도화함으로써 **실패를 빠르게 감지**하고 **효과적으로 교정**하는 능력을 갖추게 됩니다. 이는 곧 더 정확하고 풍부한 개념 관계 맵을 지속적으로 구축해나가는 동력이며, 사용자에게는 높은 신뢰도의 지식 서비스를, 시스템 운영자에게는 향상된 효율과 관리 용이성을 제공할 것으로 기대됩니다. 각 개선 사항은 상호 보완적으로 작용하여, 궁극적으로 본 시스템을 **기업급 안정성과 품질**을 갖춘 **자율 개선 AI 오케스트레이션**으로 발전시킬 것입니다.

Sources: 본 제안 내용은 현재 시스템의 연구 보고서 및 구현 코드 [2] [20], 멀티 에이전트 오케스트레이션 사례 연구 [54] [27], Self-Improvement 아키텍처 지침 [24] [35] 등을 종합 분석하여 도출되었습니다. 주요 참고 문헌과 근거는 본 문서 각주(【】)에 표시하였으며, 향후 구현 시 세부 튜닝은 실제 시스템 성능 데이터를 바탕으로 조정될 것입니다.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [16] [17] [18] [19] [20] [51] [52] [53] Claude Sonnet 4.5 기반 개념 관계 정의 체계 개선 연구 보고서.pdf
file://file_000000003bac61fd9f40654a483d9998

[14] [15] [25] [27] [31] [36] [37] [38] [41] [42] [43] [44] [45] [47] [48] [49] meta_orchestrator.py
file://file_00000000c9e861f6b4e815df0f85891c

[21] [22] [32] [33] [34] [35] [39] [40] [46] self_improver_agent.py
file://file_000000001eec61f8a9cb5e9414b840df

[23] Created an agentic meta prompt that generates powerful 3-agent workflows for Claude Code : r/ClaudeAI
https://www.reddit.com/r/ClaudeAI/comments/1le9cmr/created_an_agentic_meta_prompt_that_generates/

[24] Andela | Inside the Architecture of Self-Improving LLM Agents
https://www.andela.com/blog-posts/inside-the-architecture-of-self-improving-llm-agents

[26] [28] [29] [30] [50] socratic_mediator_agent.py
file://file_0000000073cc61f7a65f705d52b2bfd9

[54] Multi-Agent Orchestration: Running 10+ Claude Instances in Parallel (Part 3) - DEV Community
https://dev.to/bredmond1019/multi-agent-orchestration-running-10-claude-instances-in-parallel-part-3-29da