# **고급 AI 코딩 어시스턴트 시스템 개발 계획 (PRD)**

## **I. 개요**

본 문서는 AI 코딩 어시스턴트(예: Cursor, Augmentcode)가 직접 활용할 수 있도록 매우 구체적이고 실행 가능한 개발 계획(PRD)을 제시합니다. 이 시스템은 다중 에이전트 아키텍처를 활용하여 복잡한 작업을 분해하고 전문화된 실행을 가능하게 할 것입니다. 핵심 기술 스택은 고성능 백엔드를 위한 FastAPI, 시맨틱 검색을 위한 Qdrant, 전체 텍스트 및 하이브리드 검색을 위한 Meilisearch, 그리고 구조화된 데이터 및 영구 메모리를 위한 PostgreSQL을 포함합니다. LangGraph는 에이전트 워크플로우를 조율하며, 강력한 자체 수정 및 피드백 루프는 지속적인 개선을 보장합니다. Docker Compose는 재현 가능하고 확장 가능한 배포를 용이하게 할 것입니다. 이 PRD는 개발 공수 추정 없이 기술 구현 세부 사항에만 집중합니다.

## **II. 시스템 아키텍처 및 핵심 구성 요소**

### **2.1 비전 및 핵심 기능**

AI 코딩 어시스턴트는 기획 및 연구부터 코드 생성, 개선, 테스트에 이르는 다양한 소프트웨어 개발 단계에서 지능적이고 자율적인 지원을 제공함으로써 소프트웨어 개발 수명 주기를 혁신하는 것을 목표로 합니다. 이 시스템은 개발자의 생산성과 코드 품질을 향상시키고, 반복적이거나 복잡한 코딩 작업을 자동화하여 프로젝트 납기를 단축하는 협업 파트너 역할을 수행할 것입니다. 고수준 요구사항을 이해하고, 이를 세부 작업으로 분해하며, 전문화된 하위 작업을 실행하고, 그 결과를 응집력 있는 솔루션으로 통합할 수 있는 역량을 갖출 것입니다.

### **2.2 다중 에이전트 시스템 설계**

#### **제안된 에이전트 패턴: 감독자-작업자 모델**

이 시스템은 **감독자 에이전트 패턴**을 채택할 것입니다.1 이 모델에서는 중앙 감독자 에이전트가 여러 전문 작업자 에이전트를 조정합니다. 이 패턴은 복잡한 작업을 분해하고, 이를 전문 에이전트에게 위임하며, 그들의 결과물을 통합하여 작업 성공률, 정확성 및 전반적인 생산성을 향상시키는 능력 때문에 선택되었습니다.1

각 전문 에이전트는 자체적인 "스크래치패드" 또는 로컬 컨텍스트를 유지하고, 감독자 에이전트는 전반적인 통신을 관리하며 각 에이전트의 역량에 따라 작업을 위임합니다.1 이러한 분산 접근 방식은 개별 에이전트가 특정 작업에 집중할 수 있도록 하여 효율성을 높이고, 병렬 처리 및 시스템 확장성을 가능하게 합니다.1 이 패턴은 다중 에이전트 시스템에서 설명되는 광범위한 "계층적 에이전트 패턴"의 특정 사례로, 상위 에이전트(감독자)가 하위 에이전트(작업자)를 감독하거나 작업을 위임하여 크고 복잡한 문제를 더 작고 관리하기 쉬운 부분으로 분해하여 효과적으로 관리합니다.2

이러한 설계 선택은 시스템의 견고성과 유지보수성에 직접적인 영향을 미칩니다. 각 작업자 에이전트의 독립적인 운영은 단일 장애 지점이 전체 시스템을 중단시키는 것을 방지하며, 문제 발생 시 감독 에이전트가 작업을 다른 에이전트에게 재할당하거나 자체 수정 메커니즘을 시작할 수 있도록 합니다. 또한, 플래너, 연구원, 코드 생성기 등 에이전트별로 명확한 역할을 정의함으로써 시스템은 '유기적 전문화'를 자연스럽게 달성합니다.3 이는 에이전트가 미리 정의된 기능만 수행하는 것이 아니라 특정 도메인에서 탁월한 성능을 발휘하도록 설계되어, 더욱 견고하고 정확한 문제 해결을 가능하게 합니다. 이러한 구조는 새로운 전문 에이전트를 추가하여 시스템을 쉽게 확장할 수 있도록 하면서도 기존 에이전트에 미치는 영향을 최소화합니다.

#### **에이전트 역할 및 전문화**

* **플래너 에이전트:** 고수준 사용자 쿼리를 수신하고, 이를 순차적 또는 병렬 하위 작업으로 분해하며, 이러한 작업을 실행할 최적의 워크플로우 및 전문 에이전트 순서를 결정하는 역할을 합니다.1 이 에이전트는 반복적인 성찰과 개선을 통해 "느린 사고" 패러다임을 구현합니다.4
* **연구원 에이전트:** 문서, API, 기존 코드베이스를 검색하여 정보를 수집하는 데 특화되어 있습니다. 시맨틱 및 전체 텍스트 검색 기능(Qdrant, Meilisearch)을 활용하여 관련 컨텍스트를 검색합니다.
* **코드 생성기 에이전트:** 제공된 계획 및 연구를 기반으로 코드 스니펫, 함수 또는 전체 모듈을 생성하는 데 중점을 둡니다. 코드 합성을 위해 LLM과 상호 작용합니다.
* **개선 에이전트:** 생성된 코드의 품질을 향상시키는 데 전념하며, 리팩토링, 성능 최적화, 코딩 표준 준수 보장 등을 포함합니다. 이 에이전트는 자체 수정 메커니즘을 통합합니다.
* **테스터 에이전트:** 테스트 케이스를 생성하고, 실행하며, 생성된 코드의 버그 또는 불일치를 식별하는 역할을 합니다. 개선 에이전트 또는 코드 생성기 에이전트에 피드백을 제공합니다.
* **문서화 에이전트:** 생성된 코드 및 식별된 기능을 기반으로 문서를 생성하거나 업데이트합니다.

#### **LangGraph를 사용한 워크플로우 오케스트레이션**

LangChain의 일부인 LangGraph는 그래프 기반 아키텍처로 복잡한 프로세스를 효과적으로 처리하고 에이전트 상호 작용 전반에 걸쳐 컨텍스트를 유지하므로, 에이전트 워크플로우를 오케스트레이션하기 위한 프레임워크로 선택될 것입니다.1 이는 조정에 감독 제어 패턴과 강력한 메모리 시스템을 사용합니다.1

주요 그래프 구조는 다양한 에이전트 노드 간의 흐름을 정의합니다. 각 에이전트는 노드로 표현되어 단계를 실행하고 실행을 완료할지 또는 다른 에이전트로 라우팅할지 결정합니다.6 에이전트들은 상태 기반 그래프 시스템을 통해 통신하며, 감독자 에이전트가 정보 및 작업 흐름을 지시합니다.1 통신은 핸드오프 또는 도구 호출을 통해 이루어질 수 있습니다.6 시스템은 에이전트 간에 관련 상태 부분을 전달하기 위한 명확한 프로토콜을 정의할 것입니다.6

각 에이전트의 워크플로우 및 의사 결정 로직은 LangGraph 상태 머신을 구현하는 graph.py 스크립트에 정의됩니다. 에이전트 기능(비즈니스 로직)의 구체적인 구현은 tools.py에 있습니다.1 워크플로우와 구현의 이러한 명확한 분리는 깔끔한 아키텍처를 촉진합니다.1

#### **표: 다중 에이전트 프레임워크 비교**

이 표는 주요 다중 에이전트 프레임워크에 대한 명확하고 비교적인 개요를 제공하여 LangGraph 선택을 정당화합니다.

| **프레임워크 이름** | **핵심 패러다임** | **강점** | **약점/과제** | **최적 사용 사례** | **메모리 관리** | **추상화 수준** |
| --- | --- | --- | --- | --- | --- | --- |
| LangChain/LangGraph | 그래프 기반 워크플로우 | 유연성, 사용자 정의 가능성, 영구 메모리 | 가파른 학습 곡선, 사용자 정의 메모리의 복잡성 | 복잡한 다중 에이전트 워크플로우, 장기 실행/미션 크리티컬 애플리케이션 | 사용자 정의 가능한 단기/장기(외부 DB), 엔티티 메모리, 영구 메모리 5 | 저수준, 세밀한 제어 7 |
| CrewAI | 역할 기반 협업 | 역할 전문화, 쉬운 구성, 내장 메모리 | LangGraph보다 유연성 부족, SQLite3 확장성 제한 | 구조화된 팀 기반 작업, 콘텐츠 생성 | 내장(엔티티, 컨텍스트, 사용자), RAG, SQLite3 5 | 고수준 "Crew" 추상화 8 |
| AutoGen | 비동기 대화 | 다중 턴 대화, 실시간 도구 호출, 연구 기반 | 복잡한 작업에 대한 유연성/확장성 부족 | 동적 대화, 빈번한 역할 전환 | 메시지 목록, 외부 통합 5 | 비동기 대화 8 |

이 비교는 기술 선택을 정당화하는 데 필수적입니다. 다중 에이전트 프레임워크 선택은 시스템의 기반이 되는 결정입니다. 이 표는 대안을 제시하고 LangGraph가 복잡하고 사용자 정의 가능한 워크플로우에 최적의 선택인 이유를 강조하여, AI/ML 엔지니어의 제어 및 유연성에 대한 요구 사항과 일치합니다.7 또한, 구조화된 비교 데이터를 통해 AI 코딩 어시스턴트가 선택된 기술의 장단점과 근거를 효율적으로 처리할 수 있도록 하여 PRD의 "AI 준비성"을 강화합니다.

### **2.3 백엔드 및 데이터 인프라**

#### **백엔드 API 프레임워크: FastAPI**

**선택 근거:** FastAPI는 고성능, 네이티브 비동기 지원 및 자동 OpenAPI 문서 생성 기능 때문에 주요 백엔드 API 프레임워크로 선택되었습니다.9 이는 Starlette(비동기 서버 구성 요소용) 및 Pydantic(데이터 유효성 검사용)을 기반으로 구축되어 Python 3.6+에서 빠르고 효율적인 API를 생성하는 데 매우 적합합니다.12

**성능 및 확장성:** FastAPI는 속도를 위해 설계되었으며 비동기 처리 및 양방향 웹 소켓을 지원하여 고트래픽 및 I/O 바운드 애플리케이션에 이상적입니다.9 모듈식 설계는 수평 확장을 용이하게 합니다.9

**Python 생태계 호환성:** FastAPI는 Python의 풍부한 생태계로부터 이점을 얻어, AI 기반 애플리케이션에 필수적인 TensorFlow 및 PyTorch와 같은 AI/ML 라이브러리와 원활하게 작동합니다.10

FastAPI에서 async/await에 대한 강조 9는 단순한 성능 최적화를 넘어 백엔드가 동시 작업을 처리하는 방식의 근본적인 변화를 나타냅니다. 다중 에이전트 시스템에서 에이전트는 데이터베이스 쿼리, LLM에 대한 외부 API 호출, 검색 엔진과 같은 I/O 바운드 작업을 자주 수행할 것입니다. 동기식 작업은 주 스레드를 차단하여 여러 동시 에이전트 작업 또는 사용자 요청을 처리하는 시스템의 능력을 심각하게 제한합니다. 비동기 프로그래밍은 API가 데이터베이스 및 외부 서비스와 상호 작용할 때 전체 프로세스의 속도를 늦추지 않도록 하여 반응형 다중 에이전트 시스템에 필요한 높은 동시성을 가능하게 합니다.14 이러한 선택은 모든 데이터 계층 통합(PostgreSQL ORM, Qdrant 클라이언트, Meilisearch 클라이언트) 및 외부 API 호출도 FastAPI의 기능을 최대한 활용하고 병목 현상을 피하기 위해 비동기 작업을 지원해야 함을 의미합니다.

#### **벡터 데이터베이스: 시맨틱 검색 및 임베딩을 위한 Qdrant**

**역할:** Qdrant는 연구원 에이전트 및 기타 시맨틱 이해 작업에 필수적인 효율적인 유사성 검색을 위한 고성능 벡터 데이터베이스 역할을 할 것입니다.11 대규모 벡터 검색 쿼리를 낮은 지연 시간으로 처리하는 데 최적화되어 있습니다.11

**비동기 특성:** Qdrant의 비동기 특성은 메모리 사용량을 줄이고 여러 동시 사용자를 지원하여 처리할 수 있는 요청 수를 개선합니다.11 AsyncQdrantClient는 초기화에 사용될 것입니다.11

**컬렉션 관리:** 정확한 시맨틱 검색을 위해 vector\_size(예: all-MiniLM-L6-v2와 같은 Sentence-Transformer 모델의 경우 384차원) 및 distance metric(코사인 유사성)이 지정된 컬렉션이 생성될 것입니다.11

**데이터 저장:** Sentence-Transformers와 같은 모델을 사용하여 텍스트 데이터(예: 코드 스니펫, 문서, 연구 논문)에서 생성된 벡터 임베딩을 저장합니다.11

#### **전체 텍스트 검색 엔진: 키워드 및 하이브리드 검색을 위한 Meilisearch**

**역할:** Meilisearch는 Qdrant를 보완하여 정확한 키워드 일치 또는 하이브리드 검색이 필요한 시나리오를 위해 번개처럼 빠르고 오타 허용 전체 텍스트 검색 기능을 제공합니다.15 최종 사용자가 프론트엔드에서 직접 사용할 수 있도록 설계되었으며 최소한의 지연 시간을 가집니다.17

**하이브리드 검색:** Meilisearch는 키워드 및 시맨틱 검색을 결합한 하이브리드 검색을 지원하여 컨텍스트 깊이 수준을 제어할 수 있습니다.15 이는 모호한 쿼리에도 불구하고 정확한 일치와 시맨틱 이해를 혼합하여 고품질 결과를 제공합니다.15

**데이터 지속성:** Meilisearch는 콘텐츠를 디스크에 기록하여 데이터 지속성을 보장하므로 재부팅이 즉시 이루어지고 재인덱싱 비용이 발생하지 않습니다.18 비용을 피하기 위해 임베딩을 디스크에 유지합니다.18

Qdrant(벡터 DB)와 Meilisearch(전체 텍스트/하이브리드 검색)를 모두 선택하는 것은 중복이 아니라 시너지 효과를 냅니다. Qdrant는 쿼리의 *의미*를 이해하는 시맨틱 유사성에서 탁월하며, 이는 개념적으로 유사한 코드나 관련 연구 논문을 찾는 작업에 매우 중요합니다. 반면 Meilisearch는 대규모 코드베이스나 문서 내에서 특정 함수 이름, 오류 메시지 또는 정확한 구문을 찾는 데 필수적인 빠르고 정확한 키워드 일치 및 하이브리드 기능을 제공합니다.15 둘 중 하나에만 의존하면 연구원 에이전트의 검색 기능이 제한될 것입니다. 예를 들어, "Python에서 팩토리 패턴을 구현하는 방법"을 묻는 사용자는 Qdrant의 시맨틱 검색의 이점을 얻을 수 있지만, "AuthService.java에서 SQLException의 모든 발생을 찾기"와 같은 쿼리는 Meilisearch의 키워드 검색이 필요합니다. 이러한 이중 데이터베이스 전략은 AI 에이전트의 포괄적이고 효율적인 정보 검색을 보장하여 개념적 및 어휘적 검색 요구 사항을 모두 충족합니다. 또한, 기본 데이터 소스(PostgreSQL)와 두 검색 인덱스를 최신 상태로 유지하기 위한 강력한 데이터 동기화 메커니즘의 필요성을 내포합니다.

#### **관계형 데이터베이스: 구조화된 데이터 및 영구 메모리를 위한 PostgreSQL**

**역할:** PostgreSQL은 에이전트 메모리(장기 에피소드 저장소), 작업 상태, 사용자 프로필, 구성 및 코드 아티팩트에 대한 메타데이터를 포함한 구조화된 데이터 저장소를 위한 기본 관계형 데이터베이스 역할을 할 것입니다.3 이는 ACID 준수, 고도로 안전하며 오류 허용 RDBMS입니다.20

**비동기 ORM 지원:** PostgreSQL은 SQLAlchemy 및 Tortoise ORM과 같은 Python의 비동기 ORM과 잘 통합됩니다.21 SQLAlchemy는 유연성, 강력한 아키텍처, 강력한 비동기 지원 및 성능으로 유명하며, 특히 FastAPI와 함께 사용할 때 더욱 그렇습니다.21 Tortoise ORM은 비동기 기능 때문에 실시간 애플리케이션에 적합한 선택입니다.21

**메모리 관리:** 이는 에이전트의 장기 메모리에 매우 중요하며, 에이전트가 과거 상호 작용을 기반으로 응답을 조정하고 상황이 풍부한 시나리오에서 의사 결정 능력을 향상시킬 수 있도록 합니다.3

#### **비동기 메시징 시스템**

**필요성:** 비동기 메시징 시스템(예: RabbitMQ, Apache Kafka)은 마이크로서비스 및 에이전트 간의 효율적이고 비차단 통신을 가능하게 하는 데 필수적이며, 특히 장기 실행 또는 백그라운드 작업에 그렇습니다.23 이는 신뢰할 수 있는 작업 위임 및 중앙 집중식 조정을 보장합니다.2

**이벤트 기반 아키텍처:** 이 시스템은 오케스트레이터(감독자 에이전트)가 명령 메시지를 파티션에 분배하고 작업자 에이전트가 이러한 이벤트를 소비하는 이벤트 기반 오케스트레이터-작업자 패턴을 지원할 것입니다.2 이는 에이전트를 분리하고 오류 복구 메커니즘을 제공하여 작업을 단순화합니다.2

**사용 사례:** 무거운 계산 오프로드, 백그라운드 코드 분석 처리 또는 주 API 스레드를 차단하지 않고 다른 에이전트 간의 순차적 단계 조정과 같은 작업에 중요합니다.13

비동기 메시징 시스템(Kafka 또는 RabbitMQ와 같은) 2의 통합은 단순한 통신을 넘어선 중요한 아키텍처 결정입니다. 이는 서비스와 에이전트의 진정한 분리를 가능하게 합니다. 즉, 코드 생성기 에이전트가 바쁘더라도 플래너 에이전트는 동기적으로 기다릴 필요 없이 단순히 작업을 대기열에 추가할 수 있습니다. 이러한 분리는 개별 에이전트가 워크로드에 따라 독립적으로 확장될 수 있으므로 시스템의 확장성을 크게 향상시킵니다. 또한, 에이전트가 실패하더라도 메시지를 재생하거나 재라우팅할 수 있어 작업 손실이 발생하지 않으므로 복원력이 향상됩니다.2 이는 "에이전트 중복을 통한 내재적 오류 허용"을 제공합니다.3 이러한 아키텍처 선택은 성능 저하 또는 데이터 손실 없이 복잡하고 동시적인 대량의 요청을 처리할 수 있는 강력하고 처리량이 높은 AI 코딩 어시스턴트를 구축하는 데 필수적입니다. 또한 메시지 직렬화/역직렬화 및 메시징 계층 내에서 강력한 오류 처리의 필요성을 도입합니다.

#### **표: 백엔드 프레임워크 비교**

이 표는 AI API 통합을 위한 Python 백엔드 프레임워크를 비교하여 FastAPI 선택을 정당화합니다.

| **프레임워크** | **핵심 특성** | **장점** | **단점** | **AI에 가장 적합** |
| --- | --- | --- | --- | --- |
| FastAPI | 고성능, 비동기, OpenAPI | 속도, 확장성, 비동기 지원, 자동 문서화, 타입 힌트 9 | 최신, 덜 성숙, 전체 앱에 추가 기능 필요 9 | 고트래픽 AI API, 실시간 통합 |
| Django | "배터리 포함", MVT, ORM | 광범위한 기능, 쉬운 설정, 대규모 커뮤니티 9 | 무거움, 모놀리식 아키텍처 10 | 내장 도구가 있는 AI 기반 웹 애플리케이션 |
| Flask | 최소, 경량, 유연 | 유연성, 모듈식, 학습 곡선이 얕음 9 | 기본적으로 확장성 없음, 구조 부족 10 | 빠른 AI API 개발, 연구 프로젝트 |

이 표는 사용자 쿼리가 "코딩 어시스턴트" 개발 계획을 요구하므로 백엔드 선택이 매우 중요합니다. 이 표는 가장 인기 있는 Python 백엔드 프레임워크를 직접 비교하여 9 FastAPI 선택에 대한 데이터 기반 정당화를 가능하게 합니다. 이는 정보에 입각한 기술 결정을 내리는 AI/ML 엔지니어의 역할과 일치합니다. 또한, 구조화된 표 형식으로 제시함으로써 AI 코딩 어시스턴트가 선택된 기술의 컨텍스트와 장단점을 효율적으로 이해할 수 있도록 하여 PRD의 "AI 준비성"을 강화합니다.

#### **표: 핵심 데이터 구성 요소 개요**

이 표는 기본 데이터 저장 및 검색 시스템의 역할, 주요 기능 및 통합 지점을 요약합니다.

| **구성 요소** | **주요 역할** | **주요 기능** | **통합 지점** | **고려 사항** |
| --- | --- | --- | --- | --- |
| PostgreSQL | 구조화된 데이터, 영구 메모리 | ACID, SQL/JSON 지원, 비동기 ORM 옵션 20 | LangGraph (에이전트 메모리용), FastAPI (API 엔드포인트) | 동기화를 위한 CDC |
| Qdrant | 벡터 임베딩, 시맨틱 검색 | 고성능 벡터 검색, 비동기 클라이언트, 코사인 유사성 11 | LangGraph (연구원 에이전트용), FastAPI (API 엔드포인트) | 벡터 크기/거리 측정 |
| Meilisearch | 전체 텍스트 검색, 하이브리드 검색 | 오타 허용, 하이브리드 검색, 디스크 우선 지속성 15 | LangGraph (연구원 에이전트용), FastAPI (API 엔드포인트) | 인덱싱 전략, 하이브리드 검색을 위한 시맨틱 비율 |

이 아키텍처는 여러 전문 데이터 저장소를 사용합니다. 통합된 표는 이러한 복잡성을 관리하는 데 도움이 됩니다. 이 표는 각 데이터베이스의 고유한 목적과 시스템의 데이터 요구 사항을 공동으로 지원하는 방법을 명확히 합니다(예: Qdrant와 Meilisearch가 모두 필요한 이유 15). AI 코딩 어시스턴트의 경우 이 표는 각 데이터 구성 요소의 "이유"와 "무엇"을 이해하기 위한 빠른 참조 역할을 하며, 이는 올바른 통합 코드를 생성하는 데 중요합니다.

## **III. 상세 할 일 목록: 구현 계획**

### **3.1 기본 설정**

#### **3.1.1 프로젝트 초기화 및 환경 구성**

* **할 일 3.1.1.1: Python 프로젝트 구조 초기화:**
  + 프로젝트의 루트 디렉토리를 생성합니다.
  + backend/(FastAPI 애플리케이션), agents/(LangGraph 정의), data/(데이터베이스 스키마, 초기 데이터), config/, tools/에 대한 하위 디렉토리를 설정합니다.
  + backend/ 디렉토리 내에 Python 가상 환경을 초기화합니다 (python -m venv venv).
  + 가상 환경을 활성화합니다 (source venv/bin/activate for Unix/macOS, .\venv\Scripts\activate for Windows).24
* **할 일 3.1.1.2: 핵심 종속성 설치:**
  + FastAPI 및 Uvicorn을 설치합니다 (pip install fastapi uvicorn).
  + Qdrant 클라이언트를 설치합니다 (pip install qdrant-client).
  + Meilisearch 클라이언트를 설치합니다 (pip install meilisearch).
  + PostgreSQL 비동기 ORM을 설치합니다 (예: pip install sqlalchemy[asyncio] psycopg2-binary 또는 pip install tortoise-orm).
  + OpenAI Python 라이브러리를 설치합니다 (pip install openai).25
  + LangChain/LangGraph를 설치합니다 (pip install langchain langgraph).
  + 환경 변수 관리를 위한 dotenv를 설치합니다 (pip install python-dotenv).25
  + requirements.txt 파일을 생성합니다 (pip freeze > requirements.txt).
* **할 일 3.1.1.3: 컨테이너화된 서비스를 위한 Docker Compose 구성:**
  + 프로젝트 루트에 docker-compose.yml 파일을 생성합니다.
  + 다음을 위한 서비스를 정의합니다:
    - **FastAPI 백엔드:** backend/Dockerfile에서 빌드하고 관련 포트를 노출합니다.
    - **Qdrant:** qdrant/qdrant 이미지를 사용하고, 포트 6333을 매핑하며, 데이터를 위한 영구 볼륨을 정의합니다 (-v $(pwd)/qdrant\_data:/qdrant\_data).11
    - **Meilisearch:** getmeili/meilisearch 이미지를 사용하고, 포트 7700을 매핑하며, 데이터를 위한 영구 볼륨을 정의합니다 (-v $(pwd)/meili\_data:/meili\_data), MEILI\_MASTER\_KEY 환경 변수를 설정합니다.19
    - **PostgreSQL:** postgres 이미지를 사용하고, 포트 5432를 매핑하며, 데이터를 위한 영구 볼륨을 정의하고, 데이터베이스 이름, 사용자 및 암호에 대한 환경 변수를 설정합니다.
    - **RabbitMQ/Kafka (메시징 시스템):** 적절한 Docker 이미지(예: rabbitmq:management 또는 confluentinc/cp-kafka)를 사용합니다.
  + 모든 서비스가 Docker의 내부 네트워크를 통해 통신하도록 구성되었는지 확인합니다.
  + FastAPI 백엔드를 위한 Dockerfile을 추가합니다.

모든 핵심 서비스(FastAPI, Qdrant, Meilisearch, PostgreSQL, 메시징)에 Docker Compose 19를 사용하도록 명시적으로 지시하는 것은 재현성을 위한 중요한 단계입니다. 복잡한 다중 구성 요소 AI 시스템에서 올바른 버전, 종속성 및 구성으로 각 서비스를 수동으로 설정하는 것은 "지루하고 오류가 발생하기 쉽습니다".27 Docker 컨테이너는 애플리케이션이 다른 환경에서 동일하게 실행되도록 보장합니다.19 이는 일관되고 격리된 개발 환경을 제공하여 "내 컴퓨터에서는 작동하는데"와 같은 문제를 줄이고 자동화된 테스트 및 배포를 단순화함으로써 AI 코딩 어시스턴트에 직접적인 이점을 제공합니다. 이러한 기본 설정은 AI 어시스턴트와 인간 개발자 모두에게 시스템 설정, 테스트 및 배포에 대한 마찰을 크게 줄입니다. 또한 docker-compose.yml이 코드와 함께 버전 관리되어야 함을 암시합니다.

#### **3.1.2 로깅 및 모니터링 구현**

* **할 일 3.1.2.1: Loguru를 사용한 구조화된 로깅 구현:**
  + FastAPI 애플리케이션에 Loguru(pip install loguru)를 통합합니다.28
  + 쉽게 구문 분석하고 분석할 수 있도록 구조화된 로깅을 위해 Loguru를 구성합니다.28
  + TRACE, DEBUG, INFO, SUCCESS, WARNING, ERROR, CRITICAL과 같은 로그 수준을 정의하고 애플리케이션 전체에서 적절하게 사용합니다.28
  + 콘솔 및 영구 파일로 출력하기 위한 로그 싱크를 설정합니다.28
  + 디버깅 및 사후 분석을 위해 에이전트 상호 작용 및 결정이 충분한 세부 정보로 기록되는지 확인합니다.25
* **할 일 3.1.2.2: 메트릭을 위한 Prometheus 및 Grafana 통합:**
  + prometheus-fastapi-instrumentator를 설치합니다 (pip install prometheus-fastapi-instrumentator).29
  + 기본 메트릭(예: http\_requests\_total, http\_request\_duration\_seconds)으로 FastAPI 애플리케이션을 계측하고 메트릭 엔드포인트(metrics)를 노출합니다.29
  + Instrumentator().add\_metric()을 사용하여 에이전트별 작업(예: 에이전트 작업 완료율, 도구 호출 수, LLM 토큰 사용량)에 대한 사용자 정의 메트릭을 정의합니다.30
  + FastAPI 애플리케이션에서 메트릭을 스크랩하기 위해 Prometheus를 docker-compose.yml에 추가합니다.
  + Grafana를 docker-compose.yml에 추가하고, Prometheus에 대한 데이터 소스를 구성하며, 핵심 API 및 에이전트 성능 메트릭을 시각화하기 위한 초기 대시보드를 생성합니다.29
* **할 일 3.1.2.3: (선택 사항) 오류 추적을 위한 Sentry 통합:**
  + sentry-sdk를 설치합니다 (pip install sentry-sdk).29
  + 처리되지 않은 예외 및 오류를 캡처하고 보고하도록 FastAPI 애플리케이션에서 Sentry SDK를 초기화합니다.
  + Sentry DSN 및 환경을 구성합니다.

다중 에이전트 시스템과 같은 AI 시스템은 본질적으로 복잡하고 종종 비결정적입니다.31 기존 디버깅 방법으로는 불충분합니다. 강력한 로깅 및 모니터링 28은 "관찰 가능성" 12을 제공하여 개발자가 에이전트 동작을 이해하고, 병목 현상을 식별하며, 성능을 추적하고, 프로덕션에서 문제를 진단할 수 있도록 합니다. 구조화된 로깅은 에이전트 의사 결정 경로의 자동화된 분석에 도움이 되며, 메트릭(예: 에이전트 처리 시간, 도구 호출 성공률)은 시스템 상태 및 효율성에 대한 정량적 정보를 제공합니다. 이는 LLM 기반 구성 요소를 반복하고 개선하는 데 중요합니다.31 이러한 설정은 운영 위생을 위한 것뿐만 아니라 AI 에이전트의 효과적인 반복 개발 및 개선을 위한 전제 조건입니다. 이는 데이터 기반 최적화 및 예기치 않은 동작 또는 회귀의 신속한 감지를 가능하게 하며, 이는 신뢰할 수 있는 코딩 어시스턴트를 유지하는 데 필수적입니다.

### **3.2 데이터 계층 개발**

#### **3.2.1 PostgreSQL 데이터베이스 설정**

* **할 일 3.2.1.1: 에이전트 메모리 및 작업 상태를 위한 PostgreSQL 스키마 정의:**
  + SQL 스크립트를 생성하거나 ORM의 마이그레이션 시스템을 사용하여 다음 테이블을 정의합니다:
    - agents: 에이전트 구성, 역할 및 고유 ID를 저장합니다.
    - tasks: 작업의 세부 정보(ID, 상태, 입력, 출력, 할당된 에이전트 ID, 타임스탬프)를 저장합니다.
    - agent\_memory\_long\_term: 유연한 스키마를 위해 JSONB를 잠재적으로 사용하여 에이전트의 장기 에피소드 메모리를 저장합니다.3
    - code\_artifacts: 생성된 코드, 개선된 코드, 테스트 결과 및 관련 메타데이터를 저장합니다.
    - user\_profiles: 개인화를 위한 사용자별 정보를 저장합니다.5
    - feedback\_data: 지속적인 개선을 위한 명시적 및 암시적 사용자 피드백을 저장합니다.31
* **할 일 3.2.1.2: 비동기 ORM 통합 구현:**
  + 비동기 ORM(예: asyncio 지원이 있는 SQLAlchemy 또는 Tortoise ORM)을 선택하고 통합합니다.21
  + Docker Compose에 정의된 PostgreSQL 서비스에 연결하도록 ORM을 구성합니다.
  + 비차단 I/O를 보장하기 위해 정의된 모든 스키마에 대한 비동기 데이터베이스 작업(CRUD)을 구현합니다.14
  + 효율적인 리소스 관리를 위해 연결 풀링을 활용합니다.13

#### **3.2.2 Qdrant 벡터 데이터베이스 통합**

* **할 일 3.2.2.1: Qdrant 클라이언트 초기화 및 컬렉션 생성:**
  + FastAPI 애플리케이션에서 AsyncQdrantClient를 초기화하고 Docker 네트워크 주소(예: http://qdrant:6333)를 통해 Qdrant 서비스에 연결합니다.11
  + 애플리케이션 시작 시 Qdrant 컬렉션이 생성되거나 확인되도록 FastAPI 수명 주기 이벤트(시작 후크)를 구현합니다.11
  + 다양한 유형의 임베딩(예: 코드 스니펫용 code\_embeddings, 문서용 doc\_embeddings, 연구 자료용 research\_paper\_embeddings)을 위한 컬렉션을 정의합니다.
  + 각 컬렉션에 대한 VectorParams를 지정합니다. 여기에는 size(예: all-MiniLM-L6-v2 또는 OpenAI의 text-embedding-3-large와 같은 일반적인 임베딩 모델의 경우 384 또는 1536) 및 distance 메트릭(예: Cosine)이 포함됩니다.11
* **할 일 3.2.2.2: 코드 임베딩을 위한 비동기 데이터 수집 구현:**
  + 사전 훈련된 Sentence-Transformer 모델(예: all-MiniLM-L6-v2 또는 다국어 지원이 필요한 경우 한국어 특정 모델인 Jina Embeddings v3)을 사용하여 텍스트 데이터(코드, 문서)에서 벡터 임베딩을 생성하는 비동기 함수(upsert\_data\_to\_qdrant)를 개발합니다.11
  + 고유한 id 및 원본 콘텐츠에 대한 payload를 포함하여 데이터 포인트의 효율적인 비동기 삽입 및 업데이트를 위해 qdrant.upsert()를 구현합니다.11
* **할 일 3.2.2.3: 시맨틱 검색 기능 개발:**
  + 쿼리 문자열을 가져와 동일한 Sentence-Transformer 모델을 사용하여 벡터 임베딩으로 변환하고 qdrant.search()를 사용하여 시맨틱 검색을 수행하는 비동기 함수(qdrant\_search)를 생성합니다.11
  + 결과가 관련성(점수)에 따라 순위가 매겨지고 원본 페이로드 데이터와 함께 반환되는지 확인합니다.11

#### **3.2.3 Meilisearch 전체 텍스트 검색 통합**

* **할 일 3.2.3.1: Meilisearch 클라이언트 초기화 및 데이터 인덱싱:**
  + Meilisearch 클라이언트를 초기화하고 Docker 네트워크 주소(예: http://meilisearch:7700)를 통해 Meilisearch 서비스에 연결합니다.15
  + 다양한 유형의 텍스트 데이터에 대한 Meilisearch 인덱스(예: code\_index, documentation\_index, api\_index)를 생성합니다.
  + Meilisearch 인덱스에 문서를 추가하거나 업데이트하는 비동기 함수를 구현합니다.33
  + 정보 손실을 피하기 위해 인덱싱 전에 데이터 전처리(예: 구두점, 이모티콘, 이메일 제거, 텍스트 정규화)가 적용되는지 확인합니다.33
* **할 일 3.2.3.2: 하이브리드 검색 기능 구현:**
  + Meilisearch의 하이브리드 검색 기능을 활용하는 함수를 개발합니다.15
  + 쿼리 컨텍스트에 따라 시맨틱 대 키워드 일치의 중요도를 조정하도록 semanticRatio 매개변수를 구성합니다.15
  + 하이브리드 검색의 시맨틱 구성 요소를 위해 임베딩 모델(예: OpenAI의 임베딩 API 또는 로컬 SentenceTransformer 모델)과 통합합니다.15

#### **3.2.4 데이터 동기화 (PostgreSQL에서 벡터/검색)**

* **할 일 3.2.4.1: 변경 데이터 캡처(CDC) 전략 구현:**
  + **전략 선택:** PostgreSQL의 논리적 복제를 사용하는 **로그 기반 CDC** 접근 방식을 구현합니다.20 이 방법은 기본 데이터베이스에 부담을 덜 주고 이벤트 기반 메커니즘을 사용하여 실시간 데이터 변경 캡처를 허용하므로 트리거 기반 CDC보다 선호됩니다.20 쿼리 기반 CDC는 실시간 업데이트에 덜 효율적입니다.20
  + **메커니즘:** 논리적 복제를 위해 PostgreSQL을 구성합니다. 별도의 서비스(예: Python 마이크로서비스 또는 Kafka Connect 커넥터)는 복제 스트림(WAL 로그)을 소비하여 삽입, 업데이트 및 삭제를 식별합니다.20
  + **데이터 변환 및 전달:** PostgreSQL에서 캡처된 각 변경 이벤트에 대해:
    - Qdrant(벡터 임베딩) 및 Meilisearch(검색 가능한 문서)에 적합한 형식으로 데이터를 변환합니다.
    - Qdrant의 경우 Sentence-Transformer 모델을 사용하여 업데이트/삽입된 텍스트 콘텐츠에 대한 새 임베딩을 생성합니다.11
    - Qdrant 컬렉션에서 데이터 포인트를 비동기적으로 upsert 또는 delete합니다.11
    - Meilisearch 인덱스에서 문서를 비동기적으로 add 또는 delete합니다.
  + **메시징 큐 통합:** 비동기 메시징 시스템(예: RabbitMQ/Kafka)을 중간자로 사용하여 캡처된 변경 사항을 Qdrant 및 Meilisearch로 스트리밍합니다.34 이는 데이터 파이프라인의 확장성 및 신뢰성을 보장합니다.34

이 시스템은 폴리글랏 지속성 접근 방식(구조화된 데이터용 PostgreSQL, 벡터용 Qdrant, 검색용 Meilisearch)을 사용합니다. 이러한 아키텍처의 중요한 과제는 이러한 이기종 저장소 전반에 걸쳐 데이터 일관성 및 최신 상태를 유지하는 것입니다. 강력한 동기화 메커니즘이 없으면 에이전트가 오래되거나 일관성 없는 정보를 검색하여 잘못된 코드 생성 또는 권장 사항으로 이어질 수 있습니다. CDC 20는 소스(PostgreSQL)에서 변경 사항을 캡처하고 거의 실시간으로 전문 데이터 저장소로 전파함으로써 이를 직접 해결합니다. 이는 AI 에이전트가 항상 최신 정보를 기반으로 작동하도록 보장하며, 이는 진화하는 코드베이스 및 문서를 처리하는 코딩 어시스턴트에게 매우 중요합니다. 특히 로그 기반 CDC를 선택하면 기본 데이터베이스에 미치는 영향이 최소화됩니다.20 이러한 동기화 계층은 AI 코딩 어시스턴트의 신뢰성과 정확성을 위한 복잡하지만 필수적인 구성 요소입니다. 이는 AI가 환각을 일으키거나 잘못된 행동을 하는 원인이 될 수 있는 데이터 불일치를 방지하여 "두뇌"(에이전트)와 "메모리"(데이터베이스)가 항상 일치하도록 보장합니다.

### **3.3 에이전트 프레임워크 및 핵심 로직 개발**

#### **3.3.1 LangGraph 워크플로우 정의**

* **할 일 3.3.1.1: 감독자 에이전트를 위한 주요 그래프 구조 정의:**
  + 전체 다중 에이전트 워크플로우를 위한 StateGraph 인스턴스를 생성합니다.6
  + 각 전문 에이전트(플래너, 연구원, 코드 생성기, 개선 에이전트, 테스터, 문서화)에 대한 노드를 정의합니다.
  + 감독자 에이전트의 로직을 (전용 노드 내 또는 그래프의 진입점으로) 구현하여 다음을 수행합니다:
    - 초기 사용자 쿼리를 수신합니다.
    - 쿼리를 하위 작업으로 분해합니다(예: 1의 "목적지 추천", "항공편 검색", "호텔 예약").
    - 에이전트 역량 및 작업 요구 사항에 따라 조건부 라우팅을 사용하여 적절한 전문 에이전트 노드에 작업을 위임합니다.1
    - 에이전트 작업의 순차적 또는 병렬 실행을 조율합니다.
    - 작업자 에이전트의 결과물을 포괄적인 응답으로 통합합니다.1
* **할 일 3.3.1.2: 에이전트 노드 및 에이전트 간 통신 구현:**
  + 각 전문 에이전트에 대해 내부 워크플로우가 복잡한 경우 자체 LangGraph(하위 그래프)를 정의합니다(예: 1의 항공편 에이전트의 그래프는 채팅 및 도구 작업 간의 흐름을 관리합니다).
  + 한 에이전트가 다른 에이전트로 제어를 넘기는 "핸드오프"를 구현합니다.6
  + 명확한 컨텍스트 공유를 보장하기 위해 에이전트 간에 전달되는 메시지 구조를 정의합니다.6
  + LangGraph의 상태 관리를 활용하여 에이전트 상태 및 지식이 세션 및 상호 작용 전반에 걸쳐 유지되도록 합니다.5

#### **3.3.2 에이전트 툴링 및 외부 API 통합**

* **할 일 3.3.2.1: 각 에이전트를 위한 전문 도구 개발:**
  + **연구원 에이전트용:**
    - search\_qdrant(query: str, limit: int): Qdrant에서 시맨틱 검색을 수행하는 도구.11
    - search\_meilisearch(query: str, semantic\_ratio: float = 0.5): Meilisearch에서 하이브리드 검색을 수행하는 도구.15
    - fetch\_documentation(query: str): 내부/외부 소스에서 특정 문서를 검색하는 도구.
  + **코드 생성기 에이전트용:**
    - generate\_code(prompt: str, context: str, language: str): 코드 생성을 위해 LLM을 호출하는 도구.
    - refactor\_code(code: str, instructions: str): 코드 리팩토링을 위한 도구.
  + **테스터 에이전트용:**
    - generate\_tests(code: str, requirements: str): 단위/통합 테스트를 생성하는 도구.
    - execute\_tests(test\_code: str, target\_code: str): 테스트를 실행하고 결과를 캡처하는 도구.
  + **개선 에이전트용:**
    - analyze\_code\_quality(code: str): 코드 품질을 평가하는 도구(예: 정적 분석).
    - fix\_code\_issues(code: str, issues: List[str]): 식별된 문제를 기반으로 수정 사항을 적용하는 도구.
  + **문서화 에이전트용:**
    - generate\_documentation(code: str, format: str): 코드에서 문서를 생성하는 도구.
* **할 일 3.3.2.2: LLM 상호 작용을 위한 OpenAI API 통합:**
  + OpenAI 모델과 상호 작용하기 위해 openai Python 라이브러리를 활용합니다.26
  + 환경 변수(OPENAI\_API\_KEY)에서 OpenAI API 키를 안전하게 로드합니다.25
  + 텍스트 생성(예: 코드 생성, 추론, 대화형 응답)을 위해 /v1/chat/completions 엔드포인트에 대한 호출을 구현합니다.35
  + 텍스트 임베딩 생성을 위해 /v1/embeddings 엔드포인트에 대한 호출을 구현합니다.38
  + model(예: gpt-4o, gpt-4o-mini), temperature, max\_output\_tokens, 함수 호출을 위한 tools와 같은 모델 매개변수를 구성합니다.35
  + 속도 제한 및 잘못된 요청을 포함하여 API 호출에 대한 오류 처리를 구현합니다.35
* **할 일 3.3.2.3: 표준화된 API 호출을 위한 모델 컨텍스트 프로토콜(MCP) 구현:**
  + **MCP 서버:** FastAPI 백엔드 내에 MCP 서버 구성 요소를 구현합니다.25
    - FastMCP를 사용하여 서버 및 해당 도구 인터페이스를 정의합니다.25
    - 내부 기능(예: 데이터베이스 스키마 검사, SQL 쿼리 실행)을 MCP 리소스(예: database://schema) 및 도구(예: execute\_query)로 노출합니다.25
    - 이를 통해 AI 모델이 내부 데이터 소스와 상호 작용하고 표준화된 방식으로 특정 기능을 수행할 수 있습니다.40
  + **MCP 클라이언트:** 에이전트 프레임워크 내에 MCP 클라이언트를 구현합니다(예: 에이전트용 사용자 정의 도구로).25
    - 클라이언트는 MCP 서버에 연결합니다(예: 개발 중 stdio 전송을 통해).25
    - 에이전트는 이 클라이언트를 사용하여 MCP 서버에서 사용 가능한 도구 및 리소스를 검색하고 호출합니다.25
  + **보안:** MCP 엔드포인트에 대한 인증 메커니즘, 속도 제한 및 접근 제어를 구현합니다.25 외부 API에 대한 OAuth 2.0 통합을 고려할 수 있습니다.40

모델 컨텍스트 프로토콜(MCP) 25의 명시적인 통합은 미래 지향적인 아키텍처 결정입니다. 전통적으로 각 AI-외부 시스템 통합은 맞춤형으로 구축되어 파편화 및 유지보수 어려움으로 이어졌습니다.25 MCP는 AI 모델이 외부 데이터 소스 및 API와 상호 작용하기 위한 "범용 커넥터" 및 "표준화된 방식" 40을 제공합니다. 이는 일관성과 이식성을 보장합니다.25 코딩 어시스턴트의 경우, 에이전트가 데이터베이스를 쿼리하거나, 셸 명령을 실행하거나, 버전 제어 시스템과 상호 작용해야 하는 경우에도 MCP를 통해 상호 작용 패턴이 일관되게 유지됨을 의미합니다. 이는 LLM 도구 호출과 관련된 "취약성"을 줄입니다.7 MCP는 AI 코딩 어시스턴트의 확장성 및 유지보수성을 크게 향상시킵니다. 새로운 도구 또는 외부 서비스를 에이전트의 핵심 로직을 광범위하게 재설계할 필요 없이 더 쉽게 통합할 수 있습니다. 또한 AI가 사용 가능한 기능을 이해하고 활용하기 위한 더 명확한 인터페이스를 제공하여 도구 사용 시 잠재적인 "환각"을 줄입니다.

#### **표: 주요 OpenAI API 엔드포인트 및 사용법**

이 표는 코딩 어시스턴트와 관련된 특정 OpenAI API 엔드포인트와 주요 매개변수를 자세히 설명합니다.

| **엔드포인트** | **주요 사용 사례** | **주요 매개변수** | **인증** | **호환성 참고 사항** |
| --- | --- | --- | --- | --- |
| /v1/chat/completions | 텍스트/코드 생성, 추론, 대화 | model (예: gpt-4o, gpt-4o-mini), messages (role, content), temperature (0-2), max\_output\_tokens, tools (함수 호출용), response\_format (JSON 모드) 35 | OPENAI\_API\_KEY 환경 변수를 통한 Bearer 토큰 25 | /v1/completions는 채팅 모델용 레거시; GPT-3.5/4 모델에는 v1/chat/completions 사용 37 |
| /v1/embeddings | 텍스트의 벡터 표현 생성 | model (예: text-embedding-3-small, text-embedding-3-large), input (임베딩할 텍스트) 38 | OPENAI\_API\_KEY 환경 변수를 통한 Bearer 토큰 25 |  |

OpenAI의 API는 LLM 기반 에이전트의 기본입니다. 이 표는 상호 작용의 핵심 방법을 직접 다룹니다. AI 코딩 어시스턴트의 경우 정확한 엔드포인트 세부 정보, 필수 매개변수 및 호환성 참고 사항은 올바르고 기능적인 API 호출을 생성하는 데 중요합니다. 이는 더 이상 사용되지 않는 엔드포인트를 사용하는 것과 같은 일반적인 함정을 피합니다.37 또한, 안전한 API 키 관리(환경 변수) 25를 강조하는 것은 중요한 보안 지침입니다.

#### **3.3.3 에이전트 메모리 관리**

* **할 일 3.3.3.1: 단기 컨텍스트 메모리 구현:**
  + 에이전트 실행 흐름 내에서 즉각적인 컨텍스트 및 최근 상호 작용을 유지하는 LangGraph의 네이티브 기능을 활용하여 단기 작업 메모리를 관리합니다.3
  + 관련 메시지 기록을 턴 사이에 전달하여 대화 일관성을 보장합니다.6
* **할 일 3.3.3.2: 장기 에피소드 메모리 통합:**
  + LangGraph의 메모리 시스템을 PostgreSQL에 연결하여 중요한 경험, 학습된 패턴 및 상세한 상호 작용 기록을 영구적으로 저장합니다.3
  + 장기 실행 애플리케이션에서 신뢰성 및 일관성을 향상시키기 위해 세션 전반에 걸쳐 에이전트 상태 및 지식이 유지되도록 저장합니다.5
  + 엔티티별 메모리의 경우, 구조화된 엔티티 데이터를 위해 PostgreSQL 또는 전용 키-값 저장소를 활용합니다.5
  + RAG 기반 엔티티 메모리의 경우, Qdrant와 통합하여 특정 엔티티와 관련된 임베딩을 저장하고 검색합니다.5

#### **3.3.4 자체 수정 및 피드백 루프**

* **할 일 3.3.4.1: LLM 자체 수정 메커니즘 통합:**
  + \*\*자체 개선(Self-Refine)\*\*을 구현합니다 41: 에이전트는 초기 결과물(예: 코드)을 생성한 다음, 검토 및 개선을 통해 단계별로 반복적으로 개선합니다.41 이는 인간의 반복적인 개발을 모방합니다.
  + \*\*검증 체인(CoVe)\*\*을 구현합니다 41: 초기 응답을 생성한 후, 에이전트는 자체 결과물을 평가하기 위한 검증 질문을 생성하고, 이에 답한 다음, 검증 결과를 기반으로 최종 결과물을 개선합니다.41 이는 코드 정확성, 논리적 일관성 또는 요구 사항 준수를 검증하는 데 적용될 수 있습니다.
  + \*\*자체 검증(Self-Verification)\*\*을 고려합니다 41: 여러 후보 솔루션(예: 대체 코드 구현)을 생성하고, 원본 문제의 일부를 마스킹하여 각 솔루션을 평가한 다음, 누락된 정보를 예측합니다.41
  + 이러한 기술을 개선 에이전트의 워크플로우 내에 또는 감독자 에이전트의 품질 보증 단계의 일부로 통합합니다.
* **할 일 3.3.4.2: 명시적 및 암시적 피드백 루프 설계:**
  + **명시적 피드백:** 사용자가 에이전트 결과물에 대한 직접적인 피드백(예: "좋아요/싫어요", 별점, 피드백 양식)을 제공할 수 있는 메커니즘을 구현합니다.31 이 피드백을 PostgreSQL의 feedback\_data 테이블에 저장합니다.
  + **암시적 피드백:** 응답 시간, 후속 질문 수 또는 생성된 코드의 수락/거부와 같은 사용자 행동에서 추론된 피드백을 수집합니다.31 이러한 상호 작용을 기록하고 feedback\_data 테이블에 저장합니다.
  + **피드백 분석:** 수집된 피드백(명시적 및 암시적)을 분석하는 구성 요소(잠재적으로 다른 전문 에이전트)를 개발합니다.31
  + **모델 개선 통합:** 분석된 피드백을 사용하여 LLM 기반 에이전트의 반복적인 개선에 정보를 제공합니다.31 여기에는 다음이 포함될 수 있습니다:
    - 시스템 프롬프트 수정.31
    - RAG 전략 개선.31
    - 에이전트 미세 조정 또는 재훈련을 위한 평가 데이터셋 생성.31
    - SiriuS가 제안한 대로 최적화를 위해 자체 생성된 합성 데이터 활용.42
  + **에이전트 피드백 루프(AFL):** 에이전트가 다른 에이전트로부터의 피드백(예: 테스터 에이전트 피드백을 코드 생성기 에이전트에)을 분석하여 이해도를 개선하는 AFL을 구현합니다.43 에이전트 동작을 개선하기 위해 이러한 상호 작용 기록을 메모리에 저장합니다.43

자체 수정 41 및 피드백 루프 31에 대한 강조는 "지속적인 학습"을 AI 코딩 어시스턴트의 핵심 기능으로, 단순한 배포 후 최적화가 아닌, 기본으로 자리매김합니다. LLM은 비결정적이며 그 결과물은 신뢰할 수 없을 수 있습니다.31 에이전트 워크플로우에 자체 수정 메커니즘을 직접 통합하면 시스템이 자체 오류를 식별하고 수정하여 정확성과 신뢰성을 *자율적으로* 향상시킬 수 있습니다. 또한, 명시적 및 암시적 사용자 피드백은 에이전트 피드백 루프와 결합하여 모델이 실제 상호 작용 및 내부 비판으로부터 학습하는 폐쇄 루프 시스템을 생성합니다.31 이는 코딩 관행, 프레임워크 및 사용자 기대치가 끊임없이 진화함에 따라 AI 코딩 어시스턴트의 장기적인 생존 가능성과 효과에 매우 중요합니다. 이러한 설계는 AI 코딩 어시스턴트가 정적인 존재가 아니라 끊임없이 진화하는 존재임을 보장합니다. 이를 위해서는 피드백 수집, 분석 및 에이전트 훈련/개선 파이프라인에 통합하기 위한 전용 인프라가 필요하며, 잠재적으로 A/B 테스트 변경 사항도 포함됩니다.31

#### **표: LLM 자체 수정 기술**

이 표는 에이전트 시스템에 통합될 다양한 LLM 자체 수정 기술을 설명하고 비교합니다.

| **기술** | **목적** | **구현 원리** | **코딩 어시스턴트 적용** |
| --- | --- | --- | --- |
| 자체 보정 (Self-Calibration) | 신뢰도 평가 | 모델이 자체 결과물을 평가 | 코드 품질 평가, 리팩토링 |
| 자체 개선 (Self-Refine) | 반복적 개선 | 초기 결과물의 반복적 개선 | 반복적인 코드 생성/버그 수정 |
| 역방향 사고 체인 (RCoT) | 환각 감지 | 솔루션에서 새 문제 생성하여 불일치 감지 | 생성된 코드의 논리적 결함 감지 |
| 자체 검증 (Self-Verification) | 오류 수정 | 여러 솔루션 생성, 마스킹하여 평가 | 요구 사항에 대한 테스트 결과 검증 |
| 검증 체인 (CoVe) | 응답 개선 | 검증 질문 생성 및 답변 | 생성된 코드/계획 비판 |
| 누적 추론 (CR) | 단계별 문제 해결 | 각 단계를 평가, 수락/거부 결정 | 복잡한 코딩 작업을 세분화 |

LLM은 강력하지만 부정확하거나 신뢰할 수 없는 결과물을 생성할 수 있습니다.41 이 표는 이러한 한계를 완화하는 방법을 직접적으로 다룹니다. 다양한 자체 수정 기술 41을 자세히 설명함으로써, 이 표는 AI 코딩 어시스턴트가 구현할 수 있는 옵션 메뉴를 제공하여 품질 보증에 대한 다각적인 접근 방식을 허용합니다. AI 어시스턴트가 이를 구현해야 하는 경우, 이 표는 각 기술에 대한 명확한 "방법" 원칙을 제공하여 통합 프로세스를 명시적으로 만듭니다.

### **3.4 API 계층 개발 (FastAPI)**

#### **3.4.1 API 엔드포인트 정의**

* **할 일 3.4.1.1: 에이전트 상호 작용을 위한 비동기 엔드포인트 정의:**
  + 다중 에이전트 시스템과의 사용자 대면 상호 작용을 위한 FastAPI 경로(async def 사용)를 생성합니다.14
  + 예시:
    - POST /tasks/submit: 사용자 쿼리/작업 설명을 수락하고 플래너 에이전트에 위임합니다.
    - GET /tasks/{task\_id}/status: 작업의 현재 상태 및 진행 상황을 검색합니다.
    - GET /tasks/{task\_id}/result: 최종 결과물(예: 생성된 코드, 문서)을 검색합니다.
    - POST /feedback: 주어진 작업/결과물에 대한 명시적 사용자 피드백을 수락합니다.
  + 엔드포인트가 I/O 바운드 작업(예: LLM 호출, 데이터베이스 쿼리, 검색 쿼리)을 비동기적으로 처리하도록 보장합니다.13
  + 재사용 가능한 구성 요소(예: 데이터베이스 세션, Qdrant 클라이언트) 관리를 위해 FastAPI의 의존성 주입을 활용합니다.13
* **할 일 3.4.1.2: 장기 실행 작업을 위한 백그라운드 작업 구현:**
  + 즉각적인 클라이언트 응답이 필요 없는 작업(예: 광범위한 코드 분석, 대규모 문서 생성)의 경우 FastAPI의 BackgroundTasks를 활용합니다.13

#### **3.4.2 인증 및 권한 부여**

* **할 일 3.4.2.1: API 보안을 위한 OAuth2/JWT 구현:**
  + FastAPI의 내장 보안 유틸리티, 특히 JWT(JSON Web Tokens)와 함께 OAuth2를 활용합니다.45
  + Authorization 헤더에서 토큰 추출을 위한 OAuth2PasswordBearer 스키마를 구현합니다.45
  + 사용자 데이터, 토큰 요청 및 토큰 응답을 위한 Pydantic 모델을 정의합니다.45
  + 다음을 위한 함수를 구현합니다:
    - 사용자 등록(예: signup).
    - JWT 토큰 생성을 위한 사용자 인증(signin).45
    - JWT 토큰 생성, 인코딩 및 디코딩.45
    - 토큰 유효성 검사 및 사용자 정보 검색을 위한 get\_current\_user 의존성.45
  + 보호된 API 경로에 Depends(get\_current\_user)를 적용하여 인증을 강제합니다.45
  + JWT가 단기적이고 취소될 수 있는지 확인합니다.45
  + 남용을 방지하기 위해 속도 제한을 구현합니다.13

API를 통해 노출되는 AI 코딩 어시스턴트는 여러 사용자 또는 팀에게 서비스를 제공할 가능성이 높습니다. 지적 재산(코드), 사용자 데이터를 보호하고 무단 접근 또는 남용을 방지하기 위해 보안 45이 가장 중요합니다. OAuth2/JWT는 인증 및 권한 부여를 위한 표준적이고 강력한 메커니즘을 제공합니다. 이는 인증된 사용자만 시스템에 접근할 수 있도록 보장하고, 그들의 행동이 역할 또는 권한에 따라 승인되도록 합니다. 적절한 보안 없이는 시스템이 데이터 도난 및 클라우드 침입에 취약합니다.47 이 보안 계층은 프로덕션 준비가 된 AI 코딩 어시스턴트에게 필수적입니다. 또한 API 키, 비밀 및 잠재적으로 엔터프라이즈 배포를 위한 ID 공급자(예: 45에 언급된 AWS Cognito)와의 통합에 대한 안전한 처리가 필요함을 의미합니다.

#### **3.4.3 OpenAPI 문서 사용자 정의**

* **할 일 3.4.3.1: 자동 생성된 OpenAPI 문서 개선:**
  + FastAPI의 자동 OpenAPI 스키마 생성을 활용합니다.48
  + 다음과 같이 OpenAPI 문서를 사용자 정의합니다:
    - API에 대한 사용자 정의 제목, 버전, 요약 및 설명을 추가합니다.49
    - 다양한 환경에 대한 서버 URL을 지정합니다.48
    - 문서에서 더 나은 그룹화를 위해 모든 API 작업에 tags를 추가합니다.48
    - 설명 및 외부 문서 링크가 포함된 openapi\_tags 메타데이터를 태그에 포함합니다.48
    - 생성된 SDK에서 사용성을 향상시키기 위해 각 엔드포인트에 대한 operation\_id를 사용자 정의합니다.48
    - OpenAPI 문서에 보안 스키마(예: OAuth2)를 추가합니다.46
  + 요청할 때마다 재생성되는 것을 피하기 위해 생성된 OpenAPI 스키마에 대한 캐싱 메커니즘을 구현합니다.49

OpenAPI 문서 48를 사용자 정의하는 데 명시적으로 초점을 맞추는 것은 "AI 코딩 어시스턴트"를 대상으로 하는 PRD에 매우 중요합니다. OpenAPI는 API에 대한 기계 판독 가능한 계약 역할을 합니다. AI의 경우, 잘 문서화되고 구조화된 OpenAPI 사양은 상세한 사용 설명서와 같습니다. 이를 통해 AI는 인간의 개입 없이 사용 가능한 엔드포인트, 매개변수, 예상 응답 및 보안 요구 사항을 이해하여 자동화된 API 상호 작용 및 클라이언트용 코드 생성을 용이하게 합니다. 이는 단순한 인간의 가독성을 넘어 API를 "AI 네이티브"로 만드는 것입니다. 고품질의 기계 판독 가능한 API 문서는 AI 코딩 어시스턴트가 시스템의 자체 API와 효과적으로 상호 작용하고, 잠재적으로 다른 애플리케이션이 통합될 수 있도록 클라이언트 측 코드를 생성하는 직접적인 이점을 제공합니다.

## **IV. 향후 개선 사항 및 고려 사항**

* **확장성 및 고가용성 전략:**
  + FastAPI 인스턴스, 에이전트 서비스 및 데이터 저장소에 대한 수평 확장을 구현합니다.
  + 로드 밸런싱 솔루션(예: Nginx, Kubernetes Ingress)을 탐색합니다.23
  + 고가용성을 위해 데이터베이스 복제(PostgreSQL 스트리밍 복제)를 고려합니다.
  + 자주 접근하는 데이터에 대한 캐싱 전략(예: Redis)을 구현합니다.13
* **고급 자체 개선 메커니즘:**
  + 사용자 상호 작용을 기반으로 에이전트를 미세 조정하기 위한 RLHF(인간 피드백을 통한 강화 학습) 또는 기타 RL 기술을 탐색합니다.44
  + 프로덕션에서 에이전트 개선 사항을 평가하기 위한 A/B 테스트 프레임워크를 구현합니다.31
  + 에이전트를 지속적으로 늘리는 "협업 확장 법칙"을 조사합니다.4
* **IDE 및 버전 제어 시스템과의 통합:**
  + IDE 플러그인(예: VS Code, IntelliJ)을 개발하여 개발 환경 내에서 실시간 코딩 지원, 코드 완성 및 리팩토링 제안을 직접 제공합니다.
  + 자동화된 코드 커밋, 풀 리퀘스트 생성 및 브랜치 관리를 위해 Git과 통합합니다.
* **휴먼-인-더-루프 메커니즘:**
  + 인간의 감독 및 개입을 위한 명확한 인터페이스를 설계하여 개발자가 에이전트 생성 결과물을 검토, 승인 또는 수정할 수 있도록 합니다.
  + 에이전트가 불확실할 때 인간의 설명 또는 지원을 요청하는 메커니즘을 구현합니다.
  + 인간 검토 및 규정 준수를 용이하게 하기 위해 모든 AI 기반 작업 및 결정에 대한 강력한 감사 추적을 보장합니다.50

#### Works cited

1. Build multi-agent systems with LangGraph and Amazon Bedrock - AWS, accessed May 30, 2025, <https://aws.amazon.com/blogs/machine-learning/build-multi-agent-systems-with-langgraph-and-amazon-bedrock/>
2. Four Design Patterns for Event-Driven, Multi-Agent Systems - Confluent, accessed May 30, 2025, <https://www.confluent.io/blog/event-driven-multi-agent-systems/>
3. LLM-based Multi-Agent Systems: Techniques and Business Perspectives - arXiv, accessed May 30, 2025, <https://arxiv.org/html/2411.14033v2>
4. Scaling Large Language Model-based Multi-Agent Collaboration - arXiv, accessed May 30, 2025, <https://arxiv.org/html/2406.07155v3>
5. AI Agent Memory: A Comparative Analysis of LangGraph, CrewAI ..., accessed May 30, 2025, <https://dev.to/foxgem/ai-agent-memory-a-comparative-analysis-of-langgraph-crewai-and-autogen-31dp>
6. LangGraph's Multi-Agent Systems - Overview, accessed May 30, 2025, <https://langchain-ai.github.io/langgraph/concepts/multi_agent/>
7. Langgraph vs CrewAI vs AutoGen vs PydanticAI vs Agno vs OpenAI Swarm - Reddit, accessed May 30, 2025, <https://www.reddit.com/r/LangChain/comments/1jpk1vn/langgraph_vs_crewai_vs_autogen_vs_pydanticai_vs/>
8. Comparing Open-Source AI Agent Frameworks - Langfuse Blog, accessed May 30, 2025, <https://langfuse.com/blog/2025-03-19-ai-agent-comparison>
9. Which Is the Best Python Web Framework: Django, Flask, or FastAPI ..., accessed May 30, 2025, <https://blog.jetbrains.com/pycharm/2025/02/django-flask-fastapi/>
10. Top Backend Frameworks for AI API Integration: Pros, Cons ..., accessed May 30, 2025, <https://scalablehuman.com/2025/03/23/top-backend-frameworks-for-ai-api-integration-pros-cons-comparison/>
11. Async Similarity Search with FastAPI & Qdrant - FutureSmart AI Blog, accessed May 30, 2025, <https://blog.futuresmart.ai/building-an-async-similarity-search-system-from-scratch-with-fastapi-and-qdrant-vectordb>
12. Top Microservices Frameworks: From Python & Go - vFunction, accessed May 30, 2025, <https://vfunction.com/blog/best-microservices-frameworks/>
13. FastAPI Performance Tuning: Tricks to Enhance Speed and Scalability - LoadForge Guides, accessed May 30, 2025, <https://loadforge.com/guides/fastapi-performance-tuning-tricks-to-enhance-speed-and-scalability>
14. Asynchronous Programming with FastAPI: Building Efficient APIs - DEV Community, accessed May 30, 2025, <https://dev.to/dhrumitdk/asynchronous-programming-with-fastapi-building-efficient-apis-nj1>
15. Hybrid search: Definition, how it works, benefits and more - Meilisearch, accessed May 30, 2025, <https://www.meilisearch.com/blog/hybrid-search>
16. The Backend Shift: Leveraging Open Source Powerhouses for Faster, Leaner Apps, accessed May 30, 2025, <https://dev.to/glennlaysonjr/the-backend-shift-leveraging-open-source-powerhouses-for-faster-leaner-apps-11ha>
17. Meilisearch vs Qdrant: Tradeoffs, Strengths and Weaknesses - Clément Renault, accessed May 30, 2025, <https://blog.kerollmops.com/meilisearch-vs-qdrant-tradeoffs-strengths-and-weaknesses>
18. Meilisearch – search engine API bringing AI-powered hybrid search ..., accessed May 30, 2025, <https://news.ycombinator.com/item?id=43680699>
19. Using Meilisearch with Docker, accessed May 30, 2025, <https://www.meilisearch.com/docs/guides/docker>
20. The Change Data Capture (CDC) Guide for PostgreSQL - Astera Software, accessed May 30, 2025, <https://www.astera.com/type/blog/change-data-capture-postgresql/>
21. Modern ORM Frameworks in 2025: Django ORM, SQLAlchemy, and Beyond, accessed May 30, 2025, <https://www.nucamp.co/blog/coding-bootcamp-backend-with-python-2025-modern-orm-frameworks-in-2025-django-orm-sqlalchemy-and-beyond>
22. Async ORM Application - PyNest, accessed May 30, 2025, <https://pythonnest.github.io/PyNest/async_orm/>
23. How to Use Python to Create and Implement Microservices - SayOne Technologies, accessed May 30, 2025, <https://www.sayonetech.com/blog/python-microservices/>
24. OpenAPI Python Generator - IBM, accessed May 30, 2025, <https://www.ibm.com/docs/en/api-connect/saas?topic=tools-openapi-python-generator>
25. MCP (Model Context Protocol): Standardizing How LLMs Connect to ..., accessed May 30, 2025, <https://www.getambassador.io/blog/model-context-protocol-mcp-connecting-llms-to-apis>
26. Open AI API Integration with Python: the Complete Guide - PLANEKS, accessed May 30, 2025, <https://www.planeks.net/open-ai-api-integration-guide/>
27. Dockerizing a RAG Application with FastAPI, LlamaIndex, Qdrant and Ollama, accessed May 30, 2025, <https://otmaneboughaba.com/posts/dockerize-rag-application/>
28. The 5 Best Logging Libraries for Python - Highlight.io, accessed May 30, 2025, <https://www.highlight.io/blog/5-best-python-logging-libraries>
29. FastAPI Monitoring Setup - AI Prompt - DocsBot AI, accessed May 30, 2025, <https://docsbot.ai/prompts/technical/fastapi-monitoring-setup>
30. prometheus-fastapi-instrumentator - PyPI, accessed May 30, 2025, <https://pypi.org/project/prometheus-fastapi-instrumentator/>
31. LLM Feedback Loop - Nebuly, accessed May 30, 2025, <https://www.nebuly.com/blog/llm-feedback-loop>
32. Top 12 Open Source Models on HuggingFace in 2025 - Analytics Vidhya, accessed May 30, 2025, <https://www.analyticsvidhya.com/blog/2024/12/top-open-source-models-on-hugging-face/>
33. How to Make a Search Engine in Python: Step-by-Step Tutorial - Meilisearch, accessed May 30, 2025, <https://www.meilisearch.com/blog/python-search-engine>
34. Change Data Capture: Keeping Your Systems Synchronized in Real-Time - Zilliz, accessed May 30, 2025, <https://zilliz.com/glossary/change-data-capture-(cdc)>
35. API Reference - OpenAI API, accessed May 30, 2025, <https://platform.openai.com/docs/api-reference>
36. OpenAI Chat completion - Documentation - Products - Langdock, accessed May 30, 2025, <https://docs.langdock.com/api-endpoints/completion/openai>
37. OpenAI API: Is the /v1/chat/completions endpoint a legacy endpoint? - Stack Overflow, accessed May 30, 2025, <https://stackoverflow.com/questions/78565484/openai-api-is-the-v1-chat-completions-endpoint-a-legacy-endpoint>
38. Add `/api/v1/embeddings` endpoint for 100% OpenAI compatibility · Issue #8719 - GitHub, accessed May 30, 2025, <https://github.com/open-webui/open-webui/issues/8719>
39. Clarification on Specifying Embedding Models in OpenAI Vector Stores - API, accessed May 30, 2025, <https://community.openai.com/t/clarification-on-specifying-embedding-models-in-openai-vector-stores/1241740>
40. Exploring the Integration of MCP Servers with External APIs and Services - Arsturn, accessed May 30, 2025, <https://www.arsturn.com/blog/exploring-the-integration-of-mcp-servers-with-external-apis-and-services>
41. Introduction to Self-Criticism Prompting Techniques for LLMs, accessed May 30, 2025, <https://learnprompting.org/docs/advanced/self_criticism/introduction>
42. SiriuS: Self-improving Multi-agent Systems via Bootstrapped Reasoning - arXiv, accessed May 30, 2025, <https://arxiv.org/html/2502.04780v1>
43. Agentic Feedback Loop Modeling Improves Recommendation and User Simulation - arXiv, accessed May 30, 2025, <https://arxiv.org/html/2410.20027v2>
44. Google Publishes LLM Self-Correction Algorithm SCoRe - InfoQ, accessed May 30, 2025, <https://www.infoq.com/news/2024/10/google-deepmind-score/>
45. How to Protect Your API with OAuth2, JWT, and AWS Cognito | CamelEdge, accessed May 30, 2025, <https://cameledge.com/post/web-development/secure-api-oauth2-jwt-aws-cognito-part-1>
46. Security - FastAPI, accessed May 30, 2025, <https://fastapi.tiangolo.com/tutorial/security/>
47. 12 AI Agent Frameworks for Enterprises in 2025 - AI21 Labs, accessed May 30, 2025, <https://www.ai21.com/knowledge/ai-agent-frameworks/>
48. How to generate an OpenAPI document with FastAPI - Speakeasy, accessed May 30, 2025, <https://www.speakeasy.com/openapi/frameworks/fastapi>
49. Extending OpenAPI - FastAPI, accessed May 30, 2025, <https://fastapi.tiangolo.com/how-to/extending-openapi/>
50. 8 Best Multi-Agent AI Frameworks for 2025 - Multimodal, accessed May 30, 2025, <https://www.multimodal.dev/post/best-multi-agent-ai-frameworks>