SPAK: 사양 기반 프로그래머블 에이전트 커널을 통한 AI 엔지니어링 교육 프레임워크

초록

대규모 언어 모델(LLM)의 발전으로 인해 에이전트 기반 AI 시스템이 빠르게 확산되고 있으나, 현재의 에이전트 개발 방식은 여전히 프롬프트 조정과 경험적 반복에 크게 의존하고 있다. 이는 에이전트 시스템의 구조적 복잡성을 이해하고 교육하는 데 한계를 초래한다. 본 논문에서는 사양(Specification) 기반 접근법을 통해 에이전트를 체계적으로 구성, 검증, 확장할 수 있는 프로그래머블 에이전트 커널인 SPAK(Spec-Driven Programmable Agent Kernel)를 제안한다. SPAK는 AgentSpec이라는 의미적 도메인 특화 언어(DSL)를 통해 에이전트의 구조와 동작을 명시적으로 정의하며, 에이전트를 의미 범주 위의 자기 함수(endofunctor)로 정식화한다. 또한 Level 0부터 Level 5까지의 단계적 에이전트 성숙도 프레임워크를 제시하고, 각 단계에서 실제로 에이전트 시스템을 생성하고 실행한 사례를 통해 교육적·공학적 유효성을 검증한다. 본 연구는 프롬프트 중심의 AI 활용을 넘어 사양 기반 AI 엔지니어링 교육의 새로운 방향을 제시한다.

---

1. 서론

1.1 연구 배경

최근 LLM을 활용한 AI 에이전트는 소프트웨어 개발, 교육, 코칭, 운영 자동화 등 다양한 영역에서 활용되고 있다. 그러나 대부분의 에이전트 시스템은 프롬프트 설계(prompt engineering)와 시행착오에 의존하여 개발되며, 명확한 구조적 모델이나 검증 가능한 명세를 갖지 않는다. 이러한 방식은 복잡한 에이전트 시스템의 유지보수와 확장, 그리고 교육적 전달에 큰 어려움을 야기한다.

1.2 문제 정의

기존 접근 방식의 핵심 문제는 다음과 같다. - 에이전트 동작의 의미적 구조가 암묵적으로만 존재 - 상태, 도구 사용, 계획, 협업 등의 개념이 비정형적으로 결합 - 교육 과정에서 학습자가 에이전트의 내부 작동 원리를 체계적으로 이해하기 어려움

1.3 연구 목표

본 논문의 목표는 다음과 같다. 1. 에이전트를 사양 기반으로 정의하고 실행하는 커널 구조 제안 2. 에이전트의 의미적 동작을 수학적으로 정식화 3. 단계적 에이전트 성숙도 모델을 통한 교육 프레임워크 제시 4. 실제 Level 0~5 에이전트 생성 사례를 통한 실증

---

2. SPAK 개요

2.1 SPAK의 핵심 개념

SPAK는 에이전트를 단순한 응답 생성기가 아닌, 명시적 의미 상태를 변환하는 계산 주체로 본다. 이를 위해 다음의 요소로 구성된다. - **AgentSpec DSL**: 에이전트의 상태, 함수, 효과, 워크플로우를 선언적으로 기술 - **Semantic IR**: 에이전트 동작을 중간 표현으로 변환하여 검증 및 실행 - **Agent Kernel**: 사양을 해석하고 효과를 제어하는 런타임

2.2 사양 기반 접근의 장점

- 구조적 명확성
- 정적 검증 가능성
- 교육적 단계화 용이성
- 재현성과 확장성 확보

---

3. 형식적 의미론: 에이전트를 자기 함수로 보기

3.1 의미 범주 정의

에이전트가 동작하는 공간을 의미 범주 $\mathcal{C}$로 정의한다. - 객체(Object): 시스템의 의미적 상태 - 사상(Morphism): 상태 간의 순수 의미 변환

3.2 에이전트의 정식화

에이전트는 $F : \mathcal{C} \to \mathcal{C}$ 형태의 자기 함수(endofunctor)로 정의된다. 이는 동일한 의미 공간 내에서 상태를 반복적으로 변환함을 의미한다.

3.3 효과와 커널의 역할

외부 세계와의 상호작용은 대수적 효과(algebraic effects)로 모델링되며, 실제 실행은 커널이 제어한다. 이를 통해 정책(LLM 판단)과 효과 실행을 분리한다.

---

4. 에이전트 성숙도 프레임워크

본 연구에서는 에이전트의 복잡도를 단계적으로 확장하는 교육·평가 프레임워크를 제안한다.

Level	명칭	의미적 구성 요소	구현 결과
0	Static Responder	function (입력→출력)	성공
1	Context-Aware Bot	state (기억/이력)	성공
2	Tool-Use Agent	effect (외부 API)	성공
3	Planning Agent	workflow (조건/반복)	성공
4	Multi-Agent System	composition (공유 의미 공간)	성공
5	Self-Improving Agent	meta-build (자기 재귀)	성공

---

5. Level별 구현 사례

5.1 Level 0: Static Responder

단일 function 사양을 기반으로 입력을 출력으로 매핑하는 에이전트를 생성하였다. Builder는 Python 함수 코드를 생성하고, Verifier는 타입 및 입출력 일관성을 검증하였다.

5.2 Level 1: Context-Aware Bot

state 블록을 추가하여 대화 이력을 유지하는 에이전트를 구현하였다. 의미 상태의 누적이 응답에 반영됨을 확인하였다.

5.3 Level 2: Tool-Use Agent

effect 선언을 통해 외부 계산기 및 파일 시스템 접근을 허용하였다. 커널은 효과 호출을 가로채 안전하게 실행하였다.

5.4 Level 3: Planning Agent

workflow를 통해 조건 분기와 반복 구조를 정의하였다. 에이전트는 목표 달성을 위한 다단계 계획을 생성하고 실행하였다.

5.5 Level 4: Multi-Agent System

여러 에이전트가 공유 의미 공간을 통해 협력하도록 구성하였다. 역할 분담과 결과 통합이 사양 수준에서 정의되었다.

5.6 Level 5: Self-Improving Agent

에이전트가 자신의 AgentSpec을 수정하고 재빌드하는 meta-build 구조를 구현하였다. 실패 사례 분석 후 사양을 개선하는 자기 개선 루프가 관찰되었다.

---

6. 관련 연구 비교

기존 MetaGPT, ChatDev 등은 인간 협업 모델을 모방하는 데 집중한다. 반면 SPAK는 커널과 사양을 중심으로 기계 검증 가능한 제약을 우선한다. Reflexion 계열 접근법과 달리, SPAK는 테스트 이전 단계에서 의미적 불변조건을 검증한다.

---

7. 교육적 의의

SPAK는 AI 엔지니어링을 다음과 같이 재정의한다. - 프롬프트 작성 → 사양 설계 - 경험적 조정 → 형식적 검증 - 단일 모델 활용 → 시스템 구성 능력

Level 기반 접근은 학습자가 점진적으로 에이전트 복잡도를 이해하도록 돕는다.

---

8. 결론

본 논문에서는 사양 기반 프로그래머블 에이전트 커널 SPAK를 제안하고, Level 0~5까지의 에이전트 생성 사례를 통해 그 유효성을 입증하였다. 본 연구는 AI 에이전트 개발을 소프트웨어 공학적 대상으로 전환하며, 차세대 AI 엔지니어링 교육을 위한 기반을 제공한다.

부록 A. AgentSpec 문법 요약

```
AgentSpec ::= MetaDef SystemDef
SystemDef ::= "system" Name "{" ComponentDef* "}"
ComponentDef ::= "component" Name "{" MemberDef* "}"
MemberDef ::= "state" Name "{" Field* "}"
            | "function" Name "(" Params ")" "->" Type "{" Body "}"
            | "effect" Name
            | "workflow" Name
            | "invariant" StringLiteral
```