SPAK: A Spec-Driven Programmable Agent Kernel for AI Engineering Education

Authors: [Author Name]
Affiliation: [Institution Name]

Repository: https://github.com/taewony/SPAK

Abstract

The rapid advancement of Large Language Models (LLMs) has accelerated the development of AI agents, yet current prompt-driven methodologies suffer from non-deterministic outputs, lack of interpretability, and difficulty in verification. This paper introduces the **Spec-driven Programmable Agent Kernel (SPAK)**, a novel architecture that treats AI software synthesis as a formal compilation process rather than probabilistic text generation. SPAK introduces three key innovations: (1) **AgentSpec**, a semantic Domain-Specific Language (DSL) that formalizes agent structure, state, effects, and workflows as mathematical objects; (2) an **Algebraic Effect-based Runtime Kernel**, which isolates decision-making policy from execution to ensure safety and observability; and (3) an **Agent Maturity Framework**, a six-level curriculum (Level 0–5) that systematically guides learners from building simple functions to complex, self-improving recursive agents. We demonstrate SPAK's efficacy by synthesizing and verifying agents across all maturity levels on a local PC environment using the **Ollama** inference tool, showing that formal specifications can reliably bridge the gap between natural language intent and robust software execution. This work proposes a shift from prompting agents to programming agent kernels, offering a foundational platform for both AI engineering education and dependable agentic systems.

Keywords: AI Engineering, Agent Kernels, Formal Specification, Large Language Models, Algebraic Effects, Recursive Agents, Software Engineering Education.

---

1. Introduction

1.1 Background: From Prompt Engineering to Agent Engineering

The paradigm of AI engineering is shifting from single-turn chat interfaces toward autonomous agents capable of long-horizon planning, tool usage, and self-correction. However, most existing development practices remain rooted in **prompt engineering**, which relies on empirical tuning of natural language instructions. From a software engineering perspective, this approach lacks structural guarantees and fails to ensure **intent preservation**, **reproducibility**, and **verifiability**.

1.2 Problem Statement

Current agent frameworks emphasize behavioral performance while neglecting formal structure. This results in:

- **Implicit Semantics:** Agent logic is embedded in unstructured prompts, invisible to static analysis.

- **Tight Coupling:** State, tools, and control flow are entangled, hindering reasoning and maintenance.
- **Pedagogical Fragility:** Learners acquire model-specific tricks rather than transferable engineering principles.

These limitations make it difficult to scale agent systems reliably or to teach agent construction systematically.

1.3 Contribution

This paper proposes **SPAK (Spec-driven Programmable Agent Kernel)** as a response to these challenges. SPAK reframes agent development as a **specification-driven compilation process**, grounded in formal semantics and category-theoretic structure.

Our contributions are:

1. **AgentSpec DSL:** A semantic specification language for agents, modeling state, transitions, effects, and invariants explicitly.
2. **Algebraic Effect Kernel:** A runtime that strictly separates decision logic from side effects, enabling safety, observability, and replay.
3. **Agent Maturity Framework:** A six-level curriculum (Level 0–5) that incrementally introduces agentic capabilities in a pedagogically sound manner.
4. **End-to-End Implementation:** A working kernel that synthesizes, verifies, and executes agents locally using Ollama-based LLM inference.

---

2. Formal Semantics: Agents as Endofunctors

SPAK models agents as mathematical objects operating over semantic state spaces.

2.1 Semantic Category $\mathcal{C}$

We define a **Semantic Category** $\mathcal{C}$ where:

- **Objects:** Immutable snapshots of an agent's semantic state (e.g., memory, goals, context).
- **Morphisms:** Pure functions $f : S \to S'$ representing valid semantic transitions.

2.2 Agent as Endofunctor

An agent is modeled as an **Endofunctor** $F : \mathcal{C} \to \mathcal{C}$ that:

- Iteratively applies semantic transformations ($S_{t+1} = F(S_t)$).
- Preserves structural invariants defined in the specification.

This formulation provides a precise notion of intent preservation as structure preservation under F.

2.3 Algebraic Effects

Interactions with the external world (LLM calls, file I/O, tools) are represented as **Algebraic Effects**. Rather than executing effects directly, agents emit effect requests that are interpreted by the Kernel. This ensures:

- Clear separation between **policy** and **mechanism**.
- Deterministic replay and controlled execution.
- Compatibility with formal verification techniques.

---

3. Agent Maturity Framework

To evaluate SPAK as an educational and engineering framework, we introduce a graded **Agent Maturity Framework**.

Level	Name	Capability	Formal Concept	Status
0	Static Responder	Input → Output	Morphism	✓
1	Context-Aware Bot	Persistent State	Objects	✓
2	Tool-Use Agent	Side Effects	Algebraic Effects	✓
3	Planning Agent	Control Flow	Endofunctor	✓
4	Multi-Agent System	Coordination	Category Composition	✓
5	Self-Improving Agent	Recursive Build	Meta-Endofunctor	✓

Each level builds on the previous one, enabling learners to progress from simple functional mappings to self-recursive agent systems.

---

4. Implementation and Evaluation

The SPAK system was implemented in Python and evaluated entirely on a **local PC environment using Ollama** for LLM inference.

4.1 Architecture

- **Compiler:** Parses AgentSpec Markdown into a Semantic IR using Lark.
- **Builder:** Uses LLMs to synthesize code and tests from the IR.
- **Verifier:** Enforces invariants via static checks and dynamic test execution.
- **Runtime:** Executes agents while mediating all declared effects.

4.2 Case Studies

We successfully synthesized and verified agents at all maturity levels, including:

- **Level 2:** Tool-using agents with complete side-effect isolation.
- **Level 3:** Planning agents with self-repair loops triggered by test failures.

- **Level 5:** Recursive agents that spawn sub-kernels to overcome context window limitations.

These results demonstrate that formal specifications can reliably constrain probabilistic LLM behavior.

---

5. Related Work

SPAK differs from existing agent frameworks (e.g., LangChain, MetaGPT, Reflexion) by prioritizing **formal structure and verifiability** over heuristic orchestration. Unlike traditional formal methods, SPAK maintains practical usability through a Python-like DSL and LLM-assisted synthesis.

---

6. Conclusion and Future Work

This paper presented SPAK, a spec-driven programmable agent kernel that elevates AI agent development from prompt-centric experimentation to a rigorous engineering discipline. By combining formal semantics, algebraic effects, and a curriculum-oriented maturity framework, SPAK enables both reliable system construction and effective AI engineering education.

Future work will focus on the following directions:

1. **Enhanced Meta-Build Capabilities:** Extending Level 5 agents to autonomously refine and optimize their own AgentSpec definitions through recursive verification and synthesis loops.
2. **Web-Based AI Engineering Playbooks:** Developing agents that generate interactive web pages documenting design decisions, specifications, and build trajectories as educational artifacts.
3. **Project-Oriented Learning Agents:** Applying SPAK to end-to-end educational build projects, such as implementing an LLM inference engine from scratch, allowing students to experience the full stack from kernel optimization to agent orchestration.

We believe these directions will further strengthen SPAK's role as a foundational platform for scalable, trustworthy, and teachable AI engineering.

References

[1] OpenAI. GPT-4 Technical Report, 2023.
[2] Chase, H. LangChain, 2022.
[3] Weng, L. LLM-powered Autonomous Agents, 2023.
[4] Plotkin, G. D., Power, A. J. Algebraic Effects, 2003.
[5] ML105: Agentic AI Curriculum.

Appendix A. AgentSpec Grammar (Summary)

```
AgentSpec ::= MetaDef SystemDef
SystemDef ::= "system" Name "{" ComponentDef* "}"
ComponentDef ::= "component" Name "{" MemberDef* "}"
MemberDef ::= "state" Name "{" Field* "}"
```

```
        | "function" Name "(" Params ")" "->" Type
        | "effect" Name
        | "workflow" Name
        | "invariant" StringLiteral
```