LangChain Built the Agent Framework. It Did Not Define What an Agent Is.

by Nick Clark | Published March 27, 2026 | PDF

LangChain became the dominant framework for building LLM-powered agents by providing chains, tools, memory abstractions, and retrieval integrations. The ecosystem is vast. But LangChain agents have no canonical schema. There is no structural definition of what an agent is, what fields it must carry, or how governance, memory, lineage, and identity relate to each other. Agents are assembled from components. They are not structurally defined. The gap is between agent tooling and agent definition.

LangChain's contribution to making LLM agents accessible is substantial. The chain abstraction, tool use patterns, and integration ecosystem accelerated the field. The gap described here is not about framework quality. It is about the absence of a structural definition for what an agent is.

Agents are assembled, not defined

A LangChain agent is composed from an LLM, a set of tools, a prompt template, and optionally a memory module. The agent decides which tool to call based on LLM reasoning. This is a composition pattern, not a structural definition.

Different agents in the same application can have entirely different structures. One might have memory. Another might not. One might use a specific set of tools. Another might use different tools. There is no schema that says: an agent must have these fields, they must have these types, and they must relate to each other in these ways.

The consequence is that agents built with LangChain cannot be validated structurally. There is no way for a platform to inspect a LangChain agent and determine whether it has the required governance fields, whether its memory is correctly structured, or whether its identity is properly established.

Memory is optional and unstructured

LangChain provides memory modules: conversation buffer, summary, entity memory. These are useful patterns. But memory is an optional add-on, not a required field. An agent without memory is still an agent. An agent with incorrectly structured memory is still an agent.

In a canonical agent schema, memory is a typed field with defined structure. It is not optional. Every agent has memory, and every memory entry has lineage. The memory structure is validatable. A platform can inspect it, verify its integrity, and enforce constraints on how it evolves.

What a canonical agent schema provides

A canonical agent schema defines the typed fields that constitute an agent: identity, memory, governance (policy reference), capabilities, execution state, and lineage. These fields are not optional. They are the structural definition of what an agent is.

With a canonical schema, agents become interoperable. An agent built by one team can be validated by another team's platform because both agree on what an agent structurally is. Governance becomes enforceable because the platform knows where to find the policy reference. Memory becomes auditable because its structure is defined. Identity becomes persistent because it is a typed field, not an emergent property.

LangChain's tooling could implement agents that conform to a canonical schema. The framework would not change. But the agents it produces would be structurally defined, validatable, and interoperable.

The remaining gap

LangChain built the agent tooling ecosystem. The remaining gap is in agent definition: a canonical schema that says what an agent structurally is, what fields it must carry, and how those fields relate. Without that definition, agents are ad hoc compositions. With it, they become structural objects that platforms can govern.

Agent Schema All 21 steps →

Define what an autonomous agent is — structurally.

Patent
US 19/452,651 · filed
Primary Technical Disclosure
○ Cognition-Compatible Semantic Agent Objects and Structural Validation
Secondary Technical
○ Partial Agent Structural Validity: Fewer Fields, Still Deterministic○ Minimum Two-Field Validation Threshold: The Floor of Semantic Structure○ Field Interaction Rules: Deterministic Constraints Between Canonical Fields○ Field-Based Role Typing: Agent Roles Derived From Structural Composition○ Semantic Templates: Predefined Field Arrangements as Agent Class Contracts○ Structural Scaffolding Logic: Resolving Missing Fields Through Inference or Defaulting○ Field-Aware Default Resolution: Deterministic Behavior When Fields Are Absent○ Traceable Semantic Lineage Graph: Mutation History Embedded in Agent Objects○ Serialization With Stateless Compatibility: Reconstruction Without External Session State○ Schema Governance Through Versioned Policies: Cross-Version Structural Interoperability
Applications (General)
○ Enterprise AI Agent Interoperability Through Canonical Schema○ Robotic System Standardization via Structural Field Composition○ Multi-Vendor AI Agent Interoperability○ Digital Twin Standardization Through Canonical Fields○ Healthcare AI Agent Portability○ Defense Coalition Interoperability○ Insurance Claims Processing Through Standard Agents○ Legacy System Integration via Schema Bridging
Applications (Specific)
● LangChain Built the Agent Framework. It Did Not Define What an Agent Is.○ AutoGen Enabled Multi-Agent Conversations. The Agents Have No Structural Definition.○ CrewAI Organized Agents Into Teams. The Agents Still Have No Schema.○ Semantic Kernel Integrated AI Into Enterprise Code. The Agents It Creates Have No Schema.○ OpenAI Assistants API Provides Agent Tooling. It Does Not Define Agent Structure.○ Google Vertex AI Agents Provide Managed Agent Infrastructure. The Agents Have No Canonical Schema.○ Amazon Bedrock Agents Orchestrate Foundation Models. The Agents Have No Structural Definition.○ Haystack Built Composable NLP Pipelines. The Pipeline Components Have No Agent Schema.○ LlamaIndex Built the Data Framework for LLM Applications. The Data Objects Have No Agent Schema.○ Dify Made LLM Application Development Visual. The Applications Have No Agent Schema.
Agent Schema overview →
AQ
deterministic
autonomy

Legal

Subject to one or more pending U.S. and international patent applications, see Patents for the current list and status. No license, express or implied, is granted. Any use requires a separate written agreement—see Licensing. Patent applications referenced on this site are pending. Claim scope, if any, is subject to examination and may issue in altered form or not at all. See Legal for terms and conditions.

Adaptive Query™ is a trademark of Nicholas Clark. U.S. federal registration is pending. federal registration. AQ™ , AQ Inside™ , Adaptive Index™ , Adaptive Network™ , Semantic Agent™ , @AQ™ , AQID™ , and Adaptive Coin™ are used as trademarks in connection with the Adaptive Query platform and brand. Other names may be trademarks of their respective owners.

Platform operated by Adaptive Query LLC, which provides patent and trademark licensing services. Copyright © 2025-2026 Nicholas Clark. All rights reserved.

Last updated: 2026-03-03

-
- Inventive Steps
- Licensing
- Patents
- Articles
- Legal
- Opportunities
- Sitemap

-
- nick@qu3ry.net
- 72 28 14 36 01

Invented by Nick Clark | Founding Investors: Devin Wilkie