Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN: 2468-4376 https://www.jisem-journal.com/ **Research Article**

Modular Intelligence: A Skill-Based Paradigm for Scalable AI Agent Architecture

Srinivas Bhargava Jonnalagadda

Independent Researcher, USA

ARTICLE INFO

Received: 23 Jan 2026

Revised: 28 Jan 2026

ABSTRACT

The proliferation of AI systems has revealed latent scalability limitations of multiagent systems, where different agents are specialized: they serve different subtasks in a software stack, and communication and integration costs scale exponentially as the number of agents increases. The skill-based architecture sidesteps this by using a single frontier language model as a universal agent that learns and executes domainspecific expertise as modular, portable "skills." It allows procedural instructions in Markdown to be created as scripts, to allow scripts to use code as a metacomputational interface layer across multiple computing substrates. It uses progressive disclosure techniques and lightweight metadata registries to only load an entire skill specification when necessary. Coupled with the Model Context Protocol, this architecture separates data connectivity and procedural know-how from the machine learning model itself. This allows AI systems to imitate the reasoning process of human domain experts. This article enables continuous self-improvement through the expansion of capabilities. Every successful attempt at solving a problem is learned and abstracted into a skill. The set of skills forms a library embedding institutional knowledge. In effect, the model functions as the processor, the runtime serves as the operating system, and the skills act as the applications. Version-controlled, distributed development with the ability to scale capabilities through modular extensions is done at multiple levels in the software stack to create an ecosystem where specialized intelligence evolves through compound learning effects across organizational boundaries.

Keywords: Skill-Based Architecture, Modular AI Systems, Progressive Disclosure, Model Context Protocol, Autonomous Capability Expansion

1. Introduction

However, existing systems suffer from serious scalability problems as AI systems become more ubiquitous. One of the most common approaches is the creation of agents that can perform a narrow set of tasks. However, as tasks become more numerous and varied, this approach becomes unwieldy. Multi-agent AI systems provide organizations with an opportunity to interact with artificial intelligence at scale through a variety of agents. However, when researchers studied the patterns of enterprise AI adoption, they found that organizations that adopted multi-agent architectures tended to experience large coordination overheads as their number of specialized agents increased [1]. This issue is because each specialized agent must be developed, maintained, and integrated separately, which increases the cost exponentially. As such, this architectural bottleneck is less an engineering problem than a fundamental misalignment between the design of these systems and modern computational needs.

In this work, the authors propose to move away from multi-agent architectures and towards a skillbased model, where an individual frontier language model is a universal agent that dynamically learns and executes domain-specific knowledge. In this model, domain knowledge can be thought of as a modular and portable unit of skill that can be loaded, executed, and retired. The transition from monolithic to modular architecture models has promised much for many fields of software engineering, such as the ability to improve the agility, maintainability, and scalability of a system [2]. By decoupling general

reasoning facilities from specialized procedural knowledge, this design can avoid the scalability crisis affecting current agent-based systems while preserving application flexibility. Under the skill-based approach, companies can develop, train, test, and deploy composable AI capabilities individually without requiring a complete overhaul of the intelligence infrastructure used to support them.

Architectural Dimension	Multi-Agent Systems	Skill-Based Framework
Coordination Mechanism	Distributed agent negotiation	Centralized model orchestration
Maintenance Pattern	Independent agent updates	Modular skill refinement
Scalability Approach	Horizontal agent multiplication	Vertical skill library expansion
Integration Complexity	Exponential with agent count	Linear with skill additions
Development Cycle	Redesign for new capabilities	Plug-and-play skill deployment
Knowledge Reusability	Limited cross-agent transfer	Universal skill portability

Table 1: Multi-Agent Systems Versus Skill-Based Architectures [1, 2]

2. Architectural Foundations: The Skill-Based Framework

The proposed model takes a radically different approach to representing and utilizing specialized knowledge through current-generation artificial intelligence. While monolithic agent models embed specialized knowledge within themselves, the skill-based model keeps domain-specific skills within separate directories. Each skill consists of two parts: human-readable procedural knowledge in a Markdown file and executable knowledge in the script file format, which allows the skill to access and control existing APIs within the digital ecosystem. The advent of artificial intelligence (AI) in the software development lifecycle, however, has fundamentally changed the nature of system building through a more fluid approach to the architecture of capabilities [3], as is the case with a broader industry trend towards decomposing complex systems into reusable components orchestrated for production.

This choice has had a meaningful effect on the architecture. Once the code is the universal interface layer, skills generalize to operating on any computational substrate, and no special integration code is needed. The procedural skill instructions provide the semantic context in which the frontier model understands what expertise to call upon to accomplish a task, and the executable programs provide the mechanical means of effecting change in the external world. AI-improved development environments show how smart systems can use structured interfaces to manipulate a heterogeneous technology stack in a way that is less cognitively taxing for human developers and automated agents [3]. This interface-implementation separation leads to unprecedented modularity, allowing a skill to be developed, tested, and deployed independently of the reasoning engine.

The frontier model, whether implemented as Gemini, CoPilot, or other large language models, serves as the centralized cognitive system that selects, loads, and executes the appropriate skills according to user requests or real-time triggers. The use of a centralized reasoning architecture allows for the avoidance of redundant cognitive capabilities in specialized agents while still enabling dynamic skill loading for expertise across a wide range of domains. Because cycles of refactoring can focus on a single skill or module and ignore the architecture, organizations following these architectural patterns can more easily accommodate changes in business needs and the introduction of new technologies. In particular, changes to the architecture will be more easily localized to the modules themselves and need not ripple

through the whole system. [2] Within AI, the skill-based architecture models each domain's expertise as an interchangeable component. Various components can be bound together dynamically at runtime to create a solution to a specific problem. This approach fulfills various software engineering principles, including separation of concerns, interface-based design, and loosely coupled components.

System Component	Traditional Implementation	AI-Augmented Approach
Code Interface Layer	Manual integration logic	Universal skill-based abstraction
Capability Deployment	Static compilation	Dynamic skill loading
Memory Management	Fixed allocation	Progressive disclosure
Knowledge Representation	Embedded in the codebase	Markdown procedural instructions
Context Utilization	Full specification loading	Metadata-driven selective access
Processing Efficiency	Uniform resource distribution	Optimized cognitive allocation

Table 2: AI-Enhanced Development and Context Management [3, 4]

3. Cognitive Efficiency Through Progressive Disclosure

The main technical challenge for the skill-based architectures is the fixed length of the context window and the amount of memory that is used to process the input at each time step. The amount of memory that would be required to represent all available skills in the context window would quickly exceed the memory capacity, given that a skill library typically contains hundreds or thousands of skills. The context window is one of many limitations of large language model architectures, referring to the amount of information that can simultaneously impact the model's reasoning and generation. Understanding the mechanics of context windows and their limitations is important for developing effective AI systems, as information outside this window is discarded, limiting their ability to generate coherent and informed responses. The framework gets around this problem by using a progressive disclosure mechanism that strikes a balance between information density and operational flexibility.

The system may also use a lightweight metadata registry, which holds only a summary of available skills rather than their full specification in context. The metadata typically includes identifiers for the skills, a high-level description of each skill's capabilities, and the conditions under which the skills can be invoked; the frontier model can use this information to assess relevance without consuming excessive context. By selectively managing context windows, systems can retain knowledge about a huge library of skills while only expending active computational power on the reasoning steps. The model will await a signal that a particular skill is relevant to the task at hand before loading the associated procedural steps and tool functions into the active context window [4]. This is similar to lookup mechanisms in database systems and web search engines, where large indexes are maintained to quickly locate desired resources without having to materialize all information every time.

Conceptually, this lazy-loading process mirrors practices in software engineering and information retrieval, deferring costly operations until necessary. Progressive disclosure allows the system to use its context window capacity for active reasoning and current task requirements and to simultaneously maintain access to much larger libraries of skills than could be directly maintained. The metadata layer functions similarly to a database index by directing the model to relevant resources without incurring the full cost of materialization. As LLMs are often used to solve complex multi-step reasoning tasks requiring a longer context length, context window optimizations have come to play an increasingly

critical role [4]. The progressive disclosure method solves the problem by viewing the context window as a limited, dynamically allocated resource that needs to be carefully requested and allocated to optimize overall system throughput. Companies have reported that these techniques improve their response time and task diversity without increasing computational cost. This metadatadriven mechanism for skill selection allows the model to sample from the skill space that is available and extract relevant skills with a minimum cognitive load when faced with new or unexpected requirements during task execution.

Integration Layer	Model Context Protocol Role	Skill Framework Contribution
Data Connectivity	Authentication and retrieval	Procedural interpretation
API Complexity	Endpoint exposure	Abstraction and simplification
Knowledge Access	Uniform interface provision	Domain-specific reasoning
System Integration	Technical connectivity	Business logic implementation
Operational Context	Information availability	Actionable intelligence generation
Performance Enhancement	Structured data protocols	Expert-level analysis patterns

Table 3: MCP Integration and Data-Process Separation [5, 6]

4. Integration with the Model Context Protocol

The skills-based approach is viewed as complementary to the Model Context Protocol (MCP), a protocol under development for connecting language model systems to external data sources, which would allow the models to access organizational knowledge, databases, and live data feeds. Skills provide a means of imparting the procedural knowledge necessary for useful interaction. Recent evaluations of question-answering architectures show how structured protocols for accessing data improve model performance, together with domain-specific reasoning capabilities [5]. The division between structured protocols and reasoning fits perfectly with the classical distinction of data and process, where MCP is mainly responsible for the former and skills for the latter. The protocol defines a single interface layer for the authentication, retrieval, and maintenance of data and states from different sources while abstracting the implementation details of the underlying knowledge repositories.

In an enterprise scenario, a model requires access to customer relationship management (CRM) data to resolve a support ticket. The MCP authenticates and provides access to the CRM system and queries for customer records. Having the ability to gather and record this data is not enough. This data needs to be analyzed, stored, and acted on, according to organization procedures, business logic, and best practices, requiring a different set of skills. A skill specific to CRM is the encoding of standard operating procedures, escalation paths, and response templates for the analysis and action of such data. CRM data analytics further expand that knowledge of how to effectively use CRM data, requiring more than just access to the data itself. In this case, procedural knowledge determines how data is interpreted and acted upon [7]. AI models that integrate procedural knowledge with data access can replicate expert reasoning patterns through heuristics and organizational rules.

When using MCP connections, skills are intended to serve as an abstraction above the technical details. Rather than just focus on access patterns, complex APIs may expose many endpoints, parameters, and authentication mechanisms. They provide the frontier model with task-specific interfaces to bypass the low-level API details. Skills transform high-level user intent into specific API calls, handle error cases

in interactions, and retain and maintain state across multiple executions of an interaction. This abstraction saves cognitive resources of the model that can be used to reason about the problem domain instead of how to implement a solution to the problem. Enterprise architectures use the pattern of middleware components to connect high-level business logic with low-level system interfaces [5]. This bridging function allows the language model access at the appropriate level of abstraction with the technical level of assurance needed for system integration. Organizations that integrate the MCP and skill-based architecture gain greater access to the data without increasing the complexity of working with multiple systems since procedural knowledge is expressed in reusable skill modules instead of being implemented anew in each scenario.

Processing Stage	Data Access Requirement	Skill-Enabled Capability
Customer Inquiry Resolution	Record retrieval	Policy-compliant response generation
Information Interpretation	Raw data availability	Structured analytical frameworks
Decision-Making	Historical context	Standard operating procedures
Quality Assurance	Transaction logs	Escalation protocol enforcement
Knowledge Application	Database connectivity	Expert-level heuristics
System Integration	API accessibility	Middleware abstraction

Table 4: CRM Analytics and Skill-Based Processing

5. Emergent Self-Improvement Through Skill Generation

One of the most interesting aspects of the skill-based model is the potential for the agent's capabilities to grow autonomously. In comparison, existing AI systems typically maintain a fixed set of capabilities, and any new ones must be explicitly supplied by human developers. That being said, the proposed architecture supports meta-learning, where new skills are generated based on the successful resolution of previous problems so that the agents can build better skills for the next instantiation of the problem. In the context of self-constructing knowledge graphs, AI agents are used to extract structured information from unstructured documents and organize it into reusable data [9]. This functionality relates directly to procedural knowledge, as the sequence of operations performed to solve a task can be analyzed and consolidated into a skill that captures one way of approaching a problem.

If a successfully solved novel or nontrivial problem is reached using an instance of the frontier model, then the steps taken to achieve that objective can be abstracted, generalized, and learned as an automatic skill, i.e., transforming episodic problem solving into procedural knowledge. For example, if the model was used to perform financial statement analysis across a multi-step process with a certain jurisdictional regime, then that multi-step task could be learned as a standalone skill available for future calls within the same or different conversational threads. Investigations of dialogue-based AI systems have found that successful dialogue strategies can be identified and abstracted to produce more general rules that can improve future system performance on similar tasks [10]. The selfgeneration mechanism works by monitoring problem-solving behavior for patterns that indicate more general strategies and storing those patterns in representations that are easy to retrieve later on.

This mechanism essentially creates a feedback loop that is not present in conventional architectures, as each solution to a problem can build on an ever-expanding set of capabilities, or organizational knowledge, that can be assembled in executable form. As the skill library grows with additional

experience, the system is effectively collecting the amassed intelligence of all its experiences in the library, capturing what would otherwise remain tacit or ephemeral knowledge. The model is not updated; it instead affects procedural knowledge via the skill library. Similar to how expert humans acquire sets of techniques and heuristics over their professional lifetime, studies in knowledge extraction and representation have shown that procedural knowledge can be generalized and transferred to different situations and contexts as long as it is properly structured [9]. The selfimproving skill library implements this generalization at the system level, whereby the library is updated with skills successful across multiple problem-solving sessions rather than individual ones. Systems built on these architectures enjoy the advantage of compounding past knowledge, where successful use of the library is reflected in an increase in available learned skills for future interactions. The architecture also solves a fundamental limitation of present-day large language models, which cannot learn from experience "in the moment." The capabilities gained through experience are stored in durable, structured representations, which are independent of a single interaction and are maintained and accessible across different instantiations of the system.

Conclusion: Toward a Computational Substrate for Distributed Intelligence

This skill-based framework fundamentally reframes the characteristics of AI capability from static monolithic architectures to modular, composable, and evolvable primitives. By making clear analogies to existing abstractions in computing (e.g., frontier models are processors, runtimes are operating systems, and skills are applications), it can be built on decades of successful modular software development and apply these principles to AI system design. Scalability is an emergent property of modularity. Adding functionality may alter not the entire architecture, but only the skills of a few affected agents. Likewise, maintainability is a consequence of the separation of concerns: changes to the domain logic only need to be applied to the relevant skill. This shareability is a consequence of the architecture, where skills are packed as isolated directories so they can be versioned and shared across organizations, where they evolve through collaboration across communities of practice, following the principles of software development. The architecture provides an ecosystem approach, where domain experts author capabilities, share them in repositories, and evolve them collaboratively through communities of practice rather than within organizational silos. Organizations can use skill libraries to combine institutional knowledge with community-contributed skills to avoid duplication of development efforts. When skill libraries mature and standards are set on interfaces and metadata, true interoperability begins. This allows skills developed in one context to be reused across organizations and for different AI systems to access and reuse the world's procedural knowledge in executable, transferable formats. More generally, the vision here is of distributed and expert intelligence, without monolithic all-inclusive neural networks. In addition to AGI, the shift from multi-agent architectures to multi-skill architectures is the mode of organization through which AI systems can be propelled to acquire the same breadth and adaptability of human expertise while satisfying computational precision and reliability, thus establishing an enduring path of AI capability escalation that parallels the course of knowledge accretion, specialization, and collaborative refinement as witnessed in human technology.

References

[1] Talan, "Multi-Agent AI Systems: Strategic Challenges and Opportunities," 2025. [Online]. Available: https://www.talan.com/global/en/multi-agent-ai-systems-strategic-challenges-andopportunities

[2] Harrison Clarke, "Benefits of Modular Architecture: Moving from Monolithic to Modular," 2023. [Online]. Available: https://www.harrisonclarke.com/blog/benefits-of-modular-architecture-movingfrom-monolithic-to-modular

[3] Matthew Finio, Amanda Downie, "AI in Software Development," IBM, 2024. [Online]. Available: https://www.ibm.com/think/topics/ai-in-software-development

[4] Srujana Maddula, "What is a Context Window for Large Language Models?" Datacamp, 2025. [Online]. Available: https://www.datacamp.com/blog/context-window

[5] Deepa Muralidhar et al., "Operationalizing selective transparency using progressive disclosure in artificial intelligence clinical diagnosis systems," ScienceDirect, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S107158192500148X

[6] Vallikranth Ayyagari, "Model Context Protocol for Agentic AI: Enabling Contextual Interoperability Across Systems," International Journal of Computational and Experimental Science and Engineering, 2025. [Online]. Available: https://ijcesen.com/index.php/ijcesen/article/view/3678

[7] Pāvels Gončarovs, "Data Analytics in CRM Processes: A Literature Review," ResearchGate, 2017. [Online]. Available: https://www.researchgate.net/publication/322413856_Data_Analytics_in_CRM_Processes_A_Literature_Review

[8] Sandeep Kampa, "Designing Modular and Distributed Software Architectures for Scalable AI Applications in Heterogeneous Computational Ecosystems," Journal of Science & Technology, 2024. [Online]. Available: https://thesciencebrigade.com/jst/article/view/530

[9] Ananya, et al., "Towards Harnessing Large Language Models as Autonomous Agents for Semantic Triple Extraction from Unstructured Text," CEUR Workshop Proceedings, 2024. [Online]. Available: https://ceur-ws.org/Vol-3747/text2kg_paper1.pdf

[10] Jiongxiao Wang, et al., "Reinforcement Learning for Self-Improving Agent with Skill Library," arxiv, 2025. [Online]. Available: https://arxiv.org/abs/2512.17102