Towards Autonomous Agents and Recursive Intelligence

Product • April 1, 2025


Aditya Vempaty
Research Scientist


Ashish Jagmohan
Research Scientist


Prasenjit Dey
SVP Innovation


Ravi Kokku
Co-Founder & CTO


Satya Nitta
Co-Founder & CEO


Sharad Sundararajan
Co-Founder & CIO

Introduction

Emergence's Agents platform is evolving into a system that can automatically create agents and assemble multi-agent systems with minimal human intervention. At the heart of this platform is an orchestrator with built-in coding and planning abilities, designed to spawn tools and agents that handle complex tasks.

Our vision is ambitious: agents should build agents and dynamically self-assemble to make multi-agent systems, so humans no longer have to. In this model, agents create goals, simulate tasks, evaluate themselves and others, learn from failure and evolve into more capable versions. Through recursive self-improvement, they develop a deeper

alignment with their objectives—continuously refining the tools, strategies, and collaborators needed to a

We are actively building this vision into the Emergence platform. Recent work such as the STOP paper (Self-Taught Optimizing Programs) demonstrates that recursively self-improving code generation is no longer theoretical—it is becoming practical, with systems learning to evolve and optimize themselves [1]. The Emergence Orchestrator drives dynamic collaboration among agents, enabling adaptive self-assembling systems. Unlike traditional RPA solutions or Agent frameworks where humans program automation pipelines, this system is designed to become fully autonomous, yet guided by human-defined goals, compliance guardrails, and expert oversight. The result is a new model of automation—where creativity and intelligence scale fluidly, without human bottlenecks, but always within human-defined boundaries.

Self-Exploration, Simulation and Self-evaluation: How the Orchestrator Builds Intelligence

Starting from a single task, the orchestrator expands contextually—cascading into new tasks, creating agents, and designing multi-agent systems tailored to solve them. It plans the needed capabilities for each task, reuses existing agents when applicable, and creates new ones if the existing agents are not sufficient to solve the task. These agents are then assembled by the orchestrator into multi-agent systems customized to address the tasks.

The orchestrator tests each multi-agent configuration on data, systems or test cases, critiques performance using multiple evaluation strategies, and validates outputs through ongoing feedback. Humans remain in the loop—they can inspect, correct, and guide agent behavior at any time. Over multiple iterations, the orchestrator not only adapts the system to current needs but anticipates future ones by learning what works.

Example from an Enterprise context

Let us consider a deployment scenario involving one of our clients in the Semiconductor industry, where the platform had access to relevant data sources, business context, and domain knowledge. In large enterprises, data transformation and modeling are inherently complex— requiring the integration, cleaning, standardization, and structuring of data from multiple, often inconsistent, sources. Manually performing these tasks is time-consuming and frequently leads to inefficiencies in downstream operations.

In this case, the Orchestrator begins with a single task: **Identify the chips with the lowest yield in the lot**. It automatically decomposes this task into a discrete set of steps — from data ingestion to analytics and insight generation—verifies that none of the existing agents could address the task, and generates new specialized agents (as code) to handle each step. Verifying that these new agents perform effectively poses a non-trivial challenge, and involves a multi-pronged verification approach. While the system reliably produces useful agents for simpler tasks, such as ingestion and analytics, generating agents for more complex steps remains our active area of research.

To reduce time-to-insight for future use cases, the Orchestrator then anticipates upcoming tasks, and creates agents using the same self-assembly process. Starting with a seed task, the orchestrator expands its capabilities to solve a variety of related tasks (what does yield variation look like across the wafer, which circuits are contributing to low yield, what has lot to lot variation looked like historically and what does chip size have to do with yield, etc.) This forward-looking behavior enables rapid reuse and adaptation in subsequent workflows.

Figure 1 illustrates how the Orchestrator expands a single objective into a series of related tasks, and automa
to solve them. Each column in the figure represents a distinct task variant, while each row corresponds to a unique agent. The vertical stack of agents in a column forms the multi-agent system assembled to complete that specific task.

Some of these multi-agent systems are reused across tasks, while others are built anew to address evolving requirements. Through **iterative self-exploration** and **reflection**, the Orchestrator continuously **creates new agents to tackle an expanding set of problems**. The figure is based on representative enterprise data and real-world tasks from the field. Each agent and multi-agent configuration is tested for successful execution on actual data to ensure reliability and effectiveness.

Figure 2(a) shows the growth of tasks, agents and multi-agent systems with self-exploration across iterations. As new tasks get generated, the previously created agents get reused and their capabilities are expanded as required, or new agents are generated. While many tasks are generated in each iteration, the percentage of tasks

that succeed in the initial iterations is lower since the system is still figuring out the context. **Figure 2(b)** show

iterations. **Figure 2(c)** shows that the average sophistication of tasks increases with iterations, as is evident from relatively longer plans.

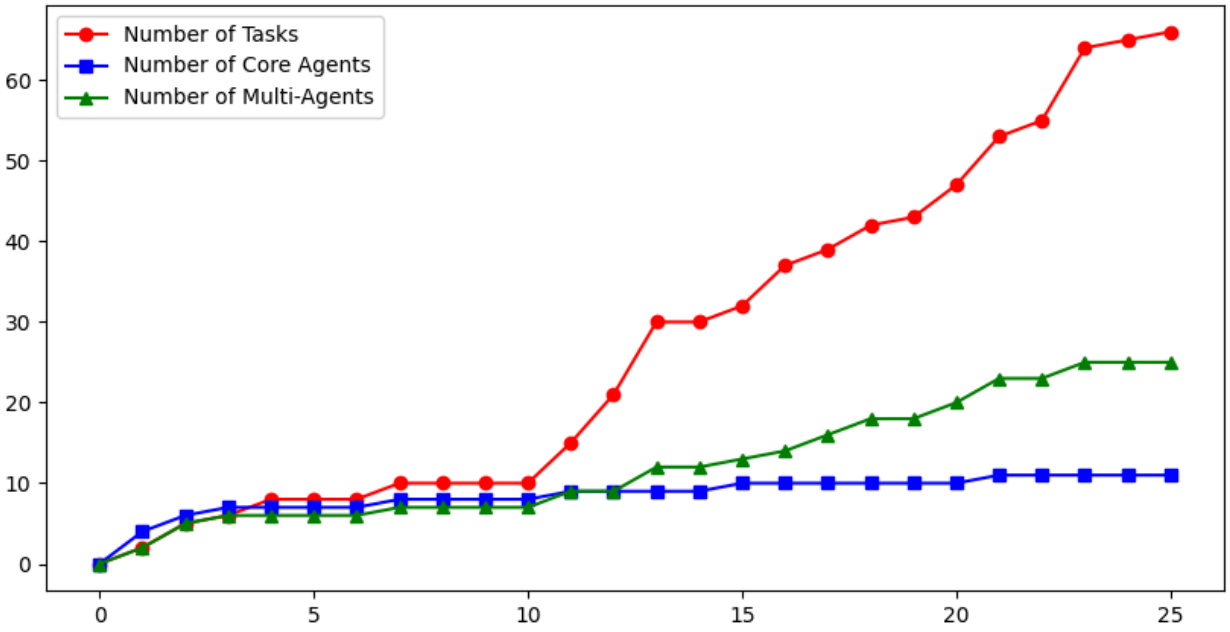


Figure 2(a): Growth of tasks, agents and multi-agent systems with self-exploration across iterations.

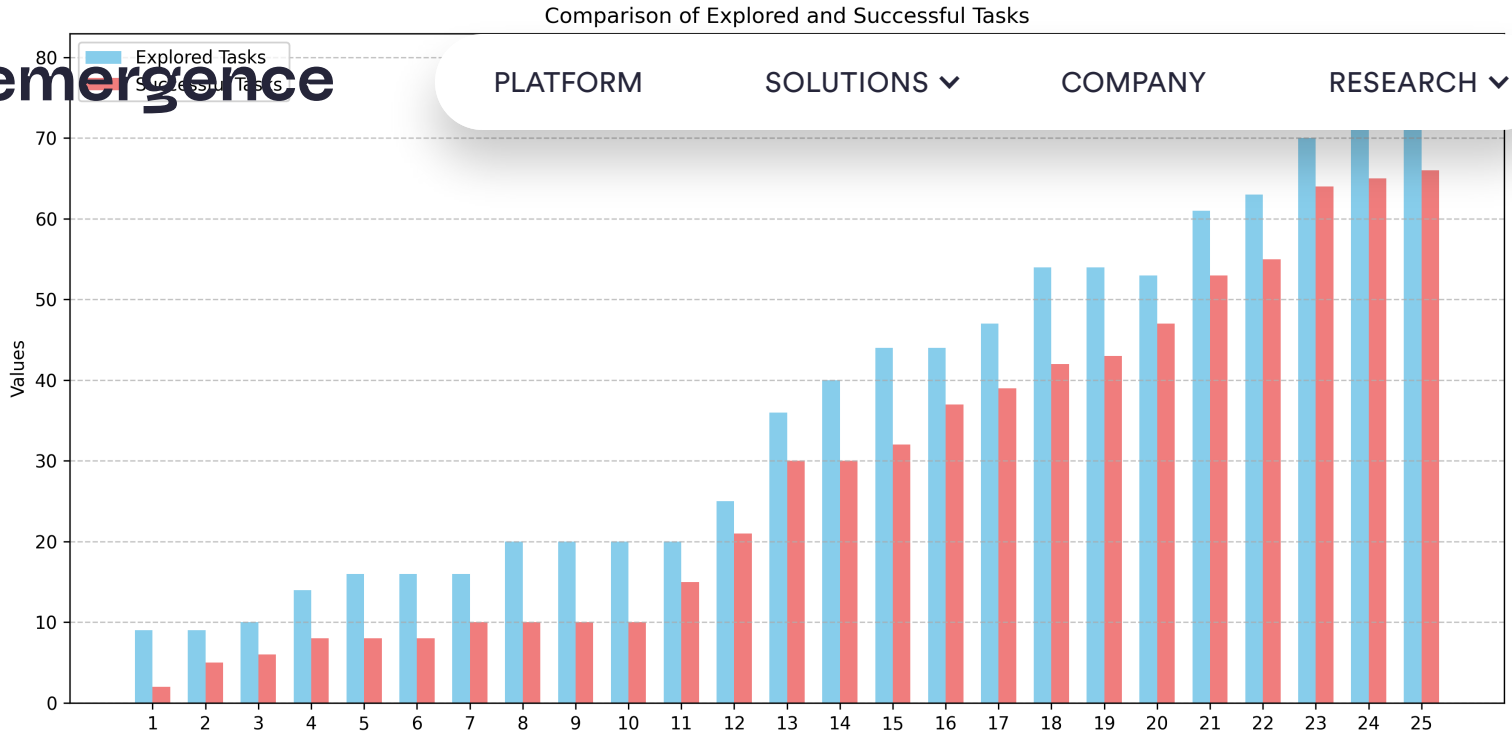


Figure 2(b): Number of tasks generated, and number of tasks that succeed in verification. The percentage of success increases with iterations.

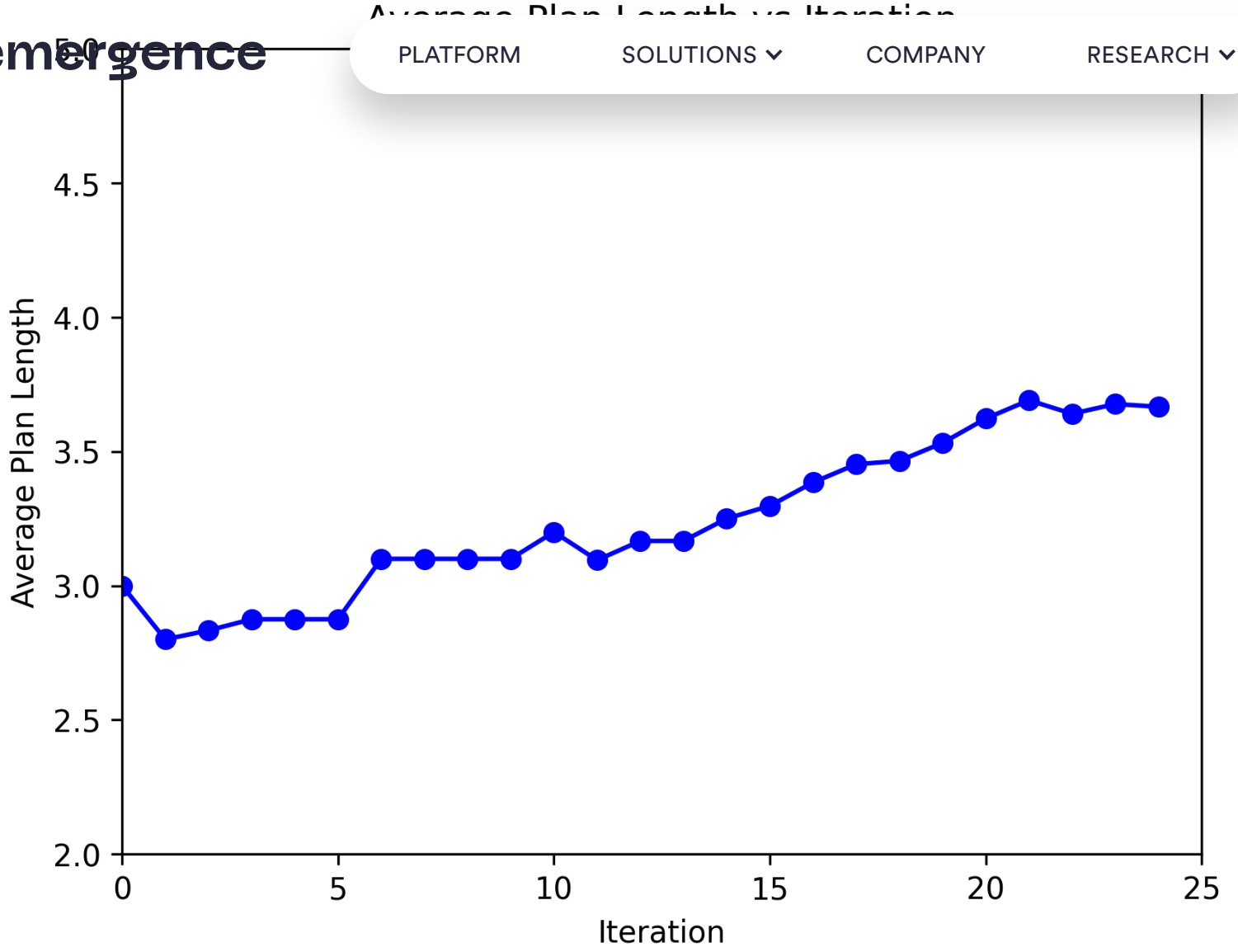


Figure 2(c): Tasks sophistication increases with iterations as the Orchestrator builds on previous success and understanding of data

Through self-exploration, simulation and verification, the orchestrator brings in several advantages:

- **Accelerated Learning**: By simulating complex situations, agents can learn faster and more comprehensively in their contexts.

- **Broad Scenario Exploration**: Edge cases and rare scenarios are tested offline without risking real-world

- **Optimization Across Versions**: Agents compare performance across iterations to refine logic, prompts, or architectures.

While this capability is still in its early days, we're building on the momentum of the broader AI community—particularly the rapid improvements in code generation models [4, 5, 6]. This foundation allows us to iterate quickly, and we'll be sharing new demos in the coming weeks that reflect this accelerating progress. The result is an intelligence infrastructure that is inherently flexible, composable, and capable of evolving alongside the business it serves.

Challenges and What Could Go Wrong

Despite its promise, recursive intelligence comes with challenges:

- **Bias in Simulation**: Agents may learn from skewed data or environments, leading to poor generalization.

- **Opaque Decision-Making**: As agents evolve, understanding why they make certain decisions becomes harder.

- **Goal Misalignment**: Without strict boundaries, agents might optimize for the wrong objectives.

- **System Sprawl**: Agents that create agents can quickly overwhelm infrastructure if not properly controlled.

In general, enterprise settings are well-suited to mitigate these risks due to the following:

- **Guardrails and Role Limits**: Strong boundaries and access controls prevent agents from going rogue.

- **Verification Rubrics**: Agents are tested against checklists for performance, safety, and compliance before deployment.

- **Human-in-the-Loop**: Humans remain involved at key checkpoints to validate decisions and maintain oversight.

- **Workflow Simplicity**: Most enterprise processes are structured and predictable—making them ideal for responsible agent automation in the initial years.

By designing systems that are both autonomous and auditable, enterprises can take full advantage of agentic creation while minimizing the downside. We are still in the early days of enterprise-wise autonomy, and we are addressing it with one problem at a time. Nevertheless, the vision of autonomous and agile Enterprises makes the pursuit worthy of exploration.

Agentic Self Assembly

In addition to creating agents, the Orchestrator also self assembles multi-agent systems in response to user tasks. We first previewed the capability in our December 2024 release, where the Orchestrator successfully self-assembled systems using two foundational system agents: the API agent and the Web agent.

Today, we're excited to announce an expansion of this capability. The Orchestrator can now build more complex [illegible] including connector agents, as well as data and text intelligence agents. The Orchestration Platform also introduces an intuitive agent SDK for seamless integration of 2nd and 3rd party agents, supported by a comprehensive agent registry. It can also be deployable across any cloud environment. The Orchestration Platform offers broad functionality, showcased through a range of demos below. One multi-agent solution that we have developed on top of the platform automates the testing of web software —already being rolled out in enterprise environments.

Demos

Multi-Agent Orchestration Demo - Example 1

Multi-Agent Orchestration Demo - Example 2

Multi-Agent Orchestration Demo - Example 3

Multi-Agent Orchestration Demo - Example 4

Data Intelligence Use Case

Autonomous Tool Generation

Whether an enterprise is interested in building full-scale solutions such as test automation or exploring other innovative applications, our system agents along with 2nd and 3rd party agents, can be assembled into multi-agent systems without any manual code writing and are designed to meet enterprise needs.

Over the past year, agentic systems have made dramatic progress in solving complex tasks, leveraging the advances in reasoning capabilities of foundation models [5, 6]. Agentic code generation exemplifies these rapid strides; on SWE-bench Verified [2,20] code generation agents have improved resolution rates from a baseline of 7% in April 2024 [3] to over 60% attained by Claude Sonnet 3.7 less than a year later [4]. Similarly impressive advances have been made in other domains. Web agents have improved task completion rates on the WebVoyager benchmark [7], from 30% for GPT-4 to over 80% demonstrated by Emergence's Agent-E, Google's Project Mariner, Claude Computer Use and others [8, 9, 10].

Accompanying these strides, there has been a concomitant dramatic reduction in the barriers to building multi-agent AI systems for real-world use-cases. Popular open-source and proprietary frameworks like AutoGen [12, 13], LangGraph [14], CrewAI [15], and the Emergence Orchestrator [16] have made it significantly easier for practitioners to create and deploy both individual AI agents as well as multi-agent systems.

We believe that the natural next chapter in the evolution of real-world agentic systems is the emergence of agents that can design and spawn other agents in practice. In the field of computer science, there has been a long history of related ideas, dating back to Turing [17], von Neumann [18] and others (see [19] and [20] for more comprehensive overviews of the literature, and [21] and related work on the theory of optimal recursively self-improving agentic systems). In software systems, autonomic systems that self-configure, self-optimize, and self-heal have been hypothesized [22] as an antidote to the problem of managing system complexity. More recently, with the advent of deep neural nets, and large language models, there has been renewed interest in self-correction and self improvement (e.g. [23-25] and many other works) and systems that automate tool building and skill acquisition (e.g. [26-28] among many others). Some of the most interesting recent applications deal with recursively self-improving code

generation loops [20] and multi-agent systems that create new scientific discoveries [29].

The emergence of agentic systems that spawn agents mirrors how complex systems evolve in nature and physics. In biology, cells self- organize into tissues, organs, and ultimately, life itself. The notion of self-assembly creates hierarchies in nature that allow for robustness, scale, and constant evolution.

AI agents engaging in self-exploration, simulation, and self-evaluation for recursive creation represent a digital parallel to these natural phenomena. Just as natural systems evolve under the influence of physical laws and survival imperatives, agentic systems self-assemble under the influence of governing constraints and objective functions defined by humans. These objectives act as guiding forces that shape the design, coordination, and optimization of agents over time. The orchestrator guides the formation of intelligent architectures without direct human coding, yet ensures that these architectures are aligned to the overall human goals.

Conclusion

The shift from "What can we build with AI?" to "What can AI build for us?" is subtle but profound. Philosophically, this shift may force us to rethink our role: not as builders of every piece, but as curators of the conditions that allow intelligence to assemble and scale itself. Practically, we believe this shift will result in increasing both the ease-of-use of multi-agent AI and its capabilities.

References

[1] E. Zelikman et al., “Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation”, cursively Self-Improving Code Generation”

[2] Introducing SWE-bench

[3] SWE-bench leaderboard

[4] Claude 3.7 Sonnet and Claude Code

[5] Deepseek-AI, “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”

[6] Introducing OpenAI o1

[7] He et al., “WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models”

[8] AbuelSaad et al., “Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems”

[9] Project Mariner

[10] Developing Computer Use.

[11] Yao et al., “τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains”

[12] Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation”

[13] Autogen, AG2

[14] LangGraph

[15] CrewAI

[16] Emergence Orchestrator

[17] A.M. Turing, “Computing Machinery and Intelligence”

[18] J.V. Neumann, “Theory of Self Reproducing Automata”

[19] R.V. Yampolskiy, “From Seed AI to Technological Singularity via Recursively Self-Improving Software”

[20] Jimenez et al., “SWE-bench: Can Language Models Resolve Real-world Github Issues?”

[21] Schmidhuber, “Goedel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements”

[22] J. O. Kephart and D.M. Chess, “The Vision of Autonomic Computing”

[23] E. Zelikman et al., “STaR: Bootstrapping Reasoning With Reasoning”

[24] A. Kumar et al., “Training Language Models to Self-Correct via Reinforcement Learning”

[25] A. Hosseini, “V-STaR: Training Verifiers for Self-Taught Reasoners”

[26] Wang et al., “JARVIS-1: Open-World Multi-task Agents with Memory-Augmented Multimodal Language Models”

[27] Cai et al. “Large Language Models as Tool Makers”

[28] Wang et al., “Voyager: An Open-Ended Embodied Agent with Large Language Models”

[29] Gottweis et al., “Towards an AI co-scientist”

[30] Zhou ety al., “Large Language Models Are Human-Level Prompt Engineers”

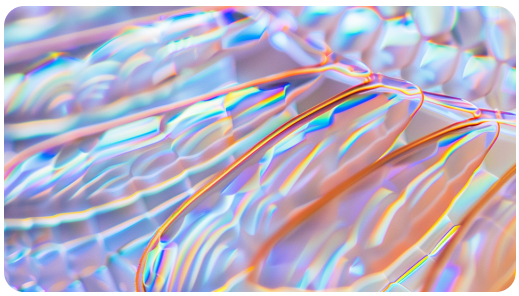
[31] T. Wu et al., “Meta-Rewarding Language Models: Self-Improving Alignment with LLM-as-a-Meta-Judge”

[32] T. Hospedales et al., “Meta-Learning in Neural Networks: A Survey”

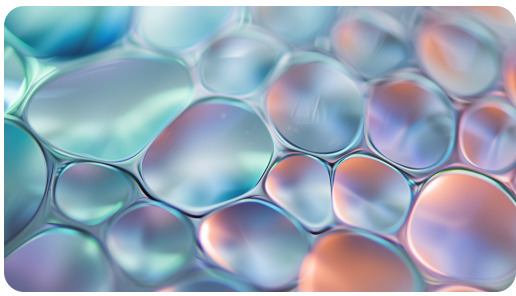
[33] C. Fernando et al., “Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution”

More from the Journal

April 22, 2025



April 15, 2025



April 15, 2025

Building Agentic Systems from First Principles Inspired by Unix and Kubernetes

A first-principles architecture for agentic systems, inspired by Unix and Kubernetes. It introduces nine core abstractions—such as Execution Contexts, Skills (as high-level system calls), and Dynamic Agent Instantiation—to enable runtime agent creation, recursive delegation, and asynchronous execution.

Layer by Layer: A Framework to Benchmarking AI Agents in the Enterprise

A structured five-layer framework provides standardized benchmarking for AI agent capabilities across the full spectrum of enterprise task complexity, from UI to infrastructure.

Beyond the Browser: The Next Generation of Enterprise AI Agents

Discover why standard AI benchmarks fall short for enterprise needs and how agent performance is truly measured on realistic, multi-application workflows using both UI and APIs.

See what Emergence can do for you.

Let's set up a demo

emergence

PLATFORM SOLUTIONS COMPANY RESEARCH

Last Name

E-mail*

Industry

What's on your mind?

I agree to receive marketing communications from Emergence AI.

protected by reCAPTCHA
Privacy - Terms

SUBMIT

OUR OFFICES

New York (HQ)

8 W 40th St
Floor 20
New York NY 10018

Irvine (West Coast Office)

16511 Scientific Way
Suite 100
Irvine CA 92618

India

2nd Floor, 25, Phoenix Citadel, Castle St, Ashok Nagar
Bengaluru, Karnataka 560025

Privacy Policy

Terms of Use

Consent Preferences