Multi-Agent Network and Event-Sourced Memory

A Decentralized Architecture for Production-Scale Agent Coordination

Adya Research Team *Hyderabad · Bengaluru · Goa*

Breaking the scalability–reliability–coordination triad through event-driven messaging, gossip-based peer discovery, and immutable event logs.

Abstract

Multi-agent systems built on current LLM-based frameworks — LangGraph, AutoGen, CrewAI — have democratized agentic AI development but inherit a structural limitation: centralized coordination. Whether the central authority is an orchestrator graph, a manager agent, or a static workflow definition, all three paradigms route inter-agent traffic through a single point that becomes the bottleneck before reasoning ever begins. We argue that no current framework simultaneously satisfies the three pressures of production deployment — scalability to thousands of agents, reliability against cascading hallucinations and node failure, and sub-second coordination — and that this is an architectural constraint rather than an implementation one. We present **MAN+ESM** (Multi-Agent Network with Event-Sourced Memory), a decentralized substrate built on three primitives: an event-driven Kafka backbone, gossip-based capability discovery propagating in O(log n) hops, and an append-only event log from which all state is derived by replay. We describe the three-layer coordination model (Decision, Execution, Interface), the Flow ID propagation system that yields complete distributed observability, the dynamic agent registry that enables runtime capability evolution, and six well-defined micro-protocols that compose into arbitrary workflows. We provide a mathematical reliability proof showing distributed redundancy converts 90%-reliable components into 99.9%-reliable systems, a task-level evaluation against the three dominant frameworks on SWE-bench and T-bench, and a comparative analysis across six architectural dimensions. The architecture is operational and forms the substrate of Adya's Vanij Agent Studio platform; the substrate, SDK, evaluation harness, and benchmark suite are scheduled for public release in 2026.

Keywords: multi-agent systems, decentralized coordination, event-driven architecture, gossip protocol, event sourcing, distributed observability, agentic AI

1. Introduction

Multi-agent systems have evolved over three decades from academic distributed-computing research into production infrastructure managing material economic value across manufacturing, logistics, financial services, and smart-city deployments. The contemporary renaissance — driven by the integration of Large Language Models with agent coordination frameworks — has democratized access to multi-agent development. Teams that previously could not justify the distributed-systems investment now deploy sophisticated agentic workflows in days rather than months.

This success has revealed a ceiling. Frameworks that handle ten agents elegantly struggle with one hundred, and degrade catastrophically by one thousand. Production teams attempting enterprise-scale deployments encounter a consistent failure mode: an architecture that worked at pilot scale collapses under load, not because of bugs, but because the coordination model itself is structurally incompatible with the requirements. This is not a tooling problem; it is a paradigmatic one.

The paradigm in question is centralization. LangGraph's stateful workflow graphs, AutoGen's manager-coordinator pattern, and CrewAI's role-hierarchy crew model each route the substantive coordination through a single point — an orchestrator, a manager agent, or a graph executor. That single point becomes the bottleneck against which every other improvement must contend. Throughput, latency, fault tolerance, and adaptability all depend on a component that cannot be parallelized away.

This paper argues that **the next generation of multi-agent infrastructure must be decentralized by construction**, not as an optimization but as a primary architectural commitment. We present MAN+ESM, an operational decentralized substrate that achieves linear horizontal scaling, deterministic event replay, sub-second coordination at production scale, and graceful degradation under partial failure — properties that compose rather than compete.

The contribution of this paper is fourfold:

1. We articulate the **coordination triad** — *scalability*, *reliability*, and *coordination latency* — as three irreducibly

competing pressures, and demonstrate that no current framework satisfies all three simultaneously.

2. We present the **MAN+ESM architecture** — its three-layer coordination model, event-driven communication backbone, dynamic agent registry, gossip-based discovery protocol, Flow ID observability primitive, event-sourced memory layer, and six composable micro-protocols.
3. We provide a mathematical reliability proof showing how distributed redundancy converts moderately-reliable components into highly-reliable systems, and contrast the resulting memory and update-latency profiles against centralized alternatives.
4. We describe the production translation through Vanij Agent Studio, demonstrating that decentralized coordination need not impose a complexity tax on end users — the architecture can be hidden behind interfaces that domain experts can use without distributed-systems training.

The remainder of the paper is organized as follows. Section 2 surveys the historical evolution from centralized standardization to modern LLM-driven frameworks. Section 3 formalizes the coordination triad. Section 4 analyzes the limitations of LangGraph, AutoGen, and CrewAI through an architectural lens. Section 5 presents the MAN+ESM design across six subsections corresponding to the major architectural innovations. Section 6 derives the reliability and memory mathematics. Section 7 reports comparative measurements across two evaluation dimensions: task-level resolution rate on SWE-bench and T-bench, and internal infrastructure measurements. Section 8 describes the Vanij implementation. Section 9 discusses open research frontiers.

2. Background: From Standardization to the Modern Ceiling

Multi-agent systems research dates to the 1980s, with the Contract Net Protocol (Smith & Davis, 1980) establishing the foundational pattern of decentralized task allocation through bidding. The 1990s saw the emergence of standardization efforts — the FIPA agent communication language and supporting platforms such as JADE — which transitioned the field from research projects into production deployments in manufacturing process control, logistics route optimization, and traffic management.

These early production deployments validated the core thesis: multi-agent coordination could solve real-world problems at meaningful scale. They also embedded a constraint that would shape the next two decades. The successful systems of that era achieved coordination through centralized control: an agent management system, a custom orchestrator, or a hand-built coordinator that owned the global state and routed all inter-agent traffic. This worked at the scale at which it was deployed — typically tens of agents, occasionally low hundreds — and it became the unconsidered default.

The integration of Large Language Models in the early 2020s triggered a renaissance. Frameworks such as LangGraph (Chase et al., 2024), AutoGen (Wang et al., 2023), and CrewAI democratized multi-agent development by providing high-level abstractions: stateful workflow graphs, conversational manager-worker patterns, role-based crew hierarchies. Organizations that had previously been unable to justify the distributed-systems investment could now deploy production agentic workflows with a few hundred lines of Python.

The renaissance frameworks inherited the centralized assumption. LangGraph executes stateful graphs through a single runtime that holds the complete graph state and routes execution along edge transitions. AutoGen orchestrates through a manager agent that mediates conversational turn-taking. CrewAI organizes specialists into a crew with a coordinator that distributes tasks. In each case, the substantive coordination — routing decisions, state transitions, message ordering — passes through one component.

At the scale of pilot deployments, this is fine. At the scale of enterprise production — financial services running concurrent risk analyses, healthcare workflow optimization handling load spikes, manufacturing supply-chain orchestration requiring continuous operation — it ceases to be fine. The pattern is consistent across domains and consistent across frameworks: initial success at small scale, followed by a predictable scaling failure that cannot be resolved within the existing paradigm.

The software industry has faced this transition before. Early web applications succeeded with monolithic centralized architectures until scale requirements forced a transition to event-driven and microservices architectures. The constraints that drove that transition — single points of failure, coordinator bottlenecks, exponential memory growth in centralized state, system-wide downtime for updates — are structurally identical to the constraints multi-agent systems face today. The resolution is also structurally identical: decentralize coordination.

3. The Coordination Triad

We frame the architectural problem as three competing pressures that any production multi-agent system must satisfy simultaneously.

Scalability demands linear horizontal growth. Adding the 101st agent should not slow down the first 100. A system that handles ten agents well but degrades catastrophically at a hundred has not solved scalability — it has merely deferred the failure.

Reliability demands graceful degradation. The failure of any single agent, link, or coordination event must not cascade. In particular, hallucination cascades — where one agent's incorrect output becomes another agent's input premise, compounding error through a multi-step workflow — are the dominant production failure mode in current LLM-based systems. Without explicit validation gates between stages, errors propagate unchecked until downstream effects become visible, often too late for recovery.

Coordination latency demands sub-second responsiveness. Agents that must agree on shared facts, hand off tasks, or compose results cannot afford the round-trip costs of synchronous consensus. Distributed agreement among even a small number of nodes takes meaningful time when implemented naively; in real-time applications the latency budget for the entire pipeline is often well under a second.

These three pressures are not independent. Optimizing for one tends to penalize the others under standard architectural assumptions. Aggressive parallelization (scalability) without validation gates degrades reliability. Strong consensus protocols (reliability) introduce serialization and latency. Fast routing (latency) without coordination overhead reintroduces single points of failure.

The architectural claim of this paper is that these pressures appear irreconcilable only under centralized assumptions. Once coordination is decentralized — once routing decisions, validation gates, and state transitions are distributed across the agent network rather than concentrated in a single coordinator — the three pressures stop competing. They can be satisfied simultaneously by an architecture that treats them as design requirements rather than as trade-off axes.

4. Limitations of Current Frameworks

We analyze the three dominant frameworks across six architectural dimensions: agent management, task execution, state management, adaptability, risk management, and observability. The analysis is qualitative; quantitative comparison appears in Section 7.

4.1 LangGraph: Centralized Graph Orchestration

LangGraph models multi-agent workflows as stateful graphs where nodes are agents and edges define execution transitions. The model is intuitive, debuggable on small graphs, and offers predictable execution flow. These are real strengths.

The limitations are structural. The graph executor holds the complete workflow state at runtime and is responsible for every transition; it cannot scale horizontally because it *is* the runtime. Agent capabilities are bound to graph nodes at design time, so adding a new capability requires rewriting and redeploying the graph. Failure of the executor terminates every workflow currently in flight. Past approximately fifty concurrent agents, latency grows roughly linearly with the agent count as the executor becomes the bottleneck against which every transition contends.

In short: LangGraph optimizes for predictability at the cost of scalability and adaptability.

4.2 AutoGen: Isolated Conversational Agents

AutoGen provides true horizontal scaling at the agent level — agents are independent runtime entities, and parallel deployment is straightforward. Individual agent fault tolerance is high. The model is well-suited to embarrassingly parallel tasks where agents operate on independent inputs.

The limitation appears the moment workflows require shared context or coordinated reasoning. AutoGen has no canonical state, no consensus mechanism, and no shared coordination surface. Each agent maintains its own conversation history, leading to inconsistent context across instances and duplicated work. There is no native validation gating, so when one agent's output feeds another's input, errors propagate without checks. The framework is excellent for parallel execution but cannot natively support workflows that require multi-step reasoning under shared constraints.

In short: AutoGen optimizes for parallelism at the cost of coordination and consistency.

4.3 CrewAI: Role-Based Hierarchical Crews

CrewAI organizes agents into role-defined crews coordinated by a manager. Workflows are clearly structured, roles are explicit, and small-team coordination is straightforward. For workflows that match a fixed crew structure, this works well.

The limitations emerge as workflow requirements evolve. The crew hierarchy is statically defined at initialization. Adding new roles or modifying the workflow requires reconfiguration. The manager agent is a centralization point — when the manager fails, the crew is paralyzed. Memory overhead scales poorly because the manager retains complete conversational context across all agent interactions. There is no mechanism for real-time consensus when agents must agree on shared facts, and the architecture breaks ungracefully on individual agent failure.

In short: CrewAI optimizes for structural clarity at the cost of adaptability and resilience.

4.4 The Pattern

Each framework excels along the dimensions it is built for and fails on the dimensions it cannot accommodate. None satisfies all three pressures of the coordination triad. The reason is structural: each places a single locus of control — graph executor, manager agent, crew coordinator — on the critical path of every coordination decision, and that locus becomes the constraint against which all other properties must contend.

5. The MAN+ESM Architecture

MAN+ESM is built on a different starting premise: *there is no central locus of control*. Coordination intelligence is distributed across the agent network. Agents discover peers through gossip, route requests through a shared event bus, validate through critic agents, persist through an append-only event log, and degrade gracefully through redundancy.

We describe the architecture across six subsections corresponding to the major innovations: the three-layer coordination model, the event-driven communication backbone, the dynamic agent registry with gossip-based discovery, the Flow ID propagation system, the event-sourced memory layer, and the micro-protocol catalogue.

5.1 The Three-Layer Coordination Model

Agents in MAN+ESM are organized across three specialized layers. The layering is not a hierarchy — there is no upward escalation of authority — but a separation of concern that allows specialization without rigidity.

The **Decision Layer** (Primary Agents) handles task decomposition and high-level coordination. When a request enters the system, a Primary Agent inspects it, decomposes it into sub-tasks, identifies dependencies, and emits routing events for downstream execution. Primary Agents are stateless — each invocation operates on the current event context — and any number can run concurrently.

The **Execution Layer** (Specialist Agents) performs domain-specific work. Specialists subscribe to capability topics relevant to their function — code generation, data analysis, document parsing, validation, and so forth — and respond to routed events. Specialists may emit further events of their own, including escalation events back to the Decision Layer if a sub-task requires further decomposition. Parallel execution is the default; serialization is a special case enforced explicitly through dependency events.

The **Interface Layer** (Gateway Agents) mediates between MAN+ESM and the world outside it. Gateway Agents handle authentication boundaries, format transformation between internal event schemas and external API contracts, and security enforcement at the perimeter. Legacy systems integrate as Gateway Agents wrapping their interfaces; modernization can proceed gradually without breaking existing functionality.

The crucial property of this organization is that agents choose their behaviour at runtime. Each agent in MAN+ESM exposes three primitive capabilities: decompose a task into sub-tasks, execute a task directly, or forward a task to a more capable peer. The agent inspects the incoming event, consults its capability registry and the available peer mesh, and selects which capability to invoke. This dynamic decision-making is what distinguishes MAN+ESM from frameworks where roles are fixed at initialization (CrewAI) or bound to graph nodes (LangGraph).

5.2 Event-Driven Communication

The communication substrate is an event bus implemented over Apache Kafka. Every inter-agent message is an event published to a capability-named topic. Agents subscribe to topics matching their capabilities; routing happens through subscription rather than through point-to-point addressing.

This matters for three reasons. First, it eliminates the orchestrator. There is no single component holding routing decisions — routing is an emergent property of who subscribes to what. Second, it enables asynchronous coordination. Agents do not block on each other; they publish events and continue. Backpressure is handled by Kafka, not by application logic. Third, it provides natural fault isolation. A crashed agent does not propagate failure to its peers; the event remains in the topic, and any other subscriber capable of handling it can take over.

The intelligence loop — the canonical request lifecycle — operates as follows. A producer publishes a request to the `requests` topic. A Policy LLM subscribes, reads the request, and emits a routing event identifying the appropriate

downstream capability. A Critic LLM evaluates outputs at each stage and emits either a validation event (allowing progression) or a needs-fix event (looping back for refinement). Once execution completes, an Executor Agent emits a signed receipt to the `receipts` topic. A consumer subscribed to receipts reads the final result. The loop is bounded by configurable iteration limits and confidence thresholds.

Schema validation is enforced at every event boundary using an Avro schema registry. Type-safe inter-agent communication is guaranteed by construction — an agent publishing a malformed event is rejected at the boundary before any subscriber sees it.

5.3 Dynamic Agent Registry and Gossip Discovery

Traditional frameworks rely on static agent catalogues defined at deployment time. MAN+ESM replaces the catalogue with a self-maintaining registry where agents announce their capabilities at startup, refresh those announcements through TTL-bounded heartbeats, and propagate capability information through gossip rather than through a central registry node.

The discovery protocol operates as follows. When an agent boots, it publishes a `capabilities.announce` event listing the capability topics it can serve, along with a TTL indicating how long the announcement is valid. The announcement propagates through the gossip network — each receiving agent re-broadcasts to a small fanout of peers, and the announcement reaches the entire cluster in roughly $log_k(n)$ hops, where k is the gossip fanout and n is the cluster size. At a fanout of four, this means roughly 3.3 hops at 100 agents, 5 hops at 1,000 agents, and 6.6 hops at 10,000 agents — the discovery latency grows logarithmically rather than linearly with cluster size.

When an agent needs to find a peer with a specific capability, it queries its local registry — which is a projection over the gossip-propagated announcements it has received — rather than querying a central directory. This is the structural difference from the registry-based alternative: the registry-based approach puts every discovery query on the critical path of a single machine, where latency grows linearly with both query volume and cluster size; the gossip approach puts discovery on the network, where latency is bounded by round-trip time and grows logarithmically.

The registry is also tamper-resistant in a useful sense: there is no single registry to compromise. Agents announcing false capabilities are detected by the Critic agents that validate their outputs; consistent failure leads to capability removal through TTL expiry without explicit revocation.

5.4 Flow ID and Distributed Observability

Distributed systems are notoriously hard to debug. When a workflow involves dozens of agents communicating asynchronously across a cluster, reconstructing what happened — and why — has historically required heroic effort. MAN+ESM addresses this through the **Flow ID** primitive.

Every task initiation generates a Flow ID with the format `{timestamp}_{session}_{version}_{routing_hint}`. The Flow ID propagates through every event derived from the original request — every routing decision, every Critic evaluation, every tool invocation, every signed receipt. The result is a complete causally-ordered trace of the entire workflow, available for inspection at any granularity.

This enables four operational capabilities that are difficult or impossible in centralized frameworks. **Complete task traceability:** any workflow can be reconstructed from its Flow ID by querying the event log for matching events. **Performance bottleneck identification:** timing data attached to each event allows precise analysis of where latency accumulates. **Error root cause analysis:** when a downstream failure occurs, the Flow ID identifies which upstream decisions led to it. **Optimization opportunity discovery:** aggregate analysis of Flow IDs across many runs reveals common inefficient patterns that can be addressed through routing improvements or new specialist agents.

Flow IDs combine with Task IDs (sub-task-level identifiers) to provide hierarchical traceability — a single Flow ID may contain dozens of Task IDs corresponding to decomposed sub-tasks executed in parallel. The combined granular tracking is what enables MAN+ESM to provide what we believe is the strongest distributed observability surface of any current multi-agent framework.

5.5 Event-Sourced Memory

State in MAN+ESM is not a mutable store. State is the projection of an append-only event log — the sequence of all events ever emitted in the system, ordered, immutable, signed.

This is event sourcing in the sense established by Fowler (2005) and Kreps (2013), applied to the multi-agent setting. Every observable property of the system — the current state of any agent, the contents of any task's working context, the audit trail of any decision — is derivable by replaying events from the log up to a chosen point in time. Two memory tiers cooperate as projections over the same log: an episodic projection retaining long-term

experience and learned preferences across sessions, and a scratchpad projection holding task-local working context that evicts on completion.

Four properties follow from event sourcing by construction:

Deterministic replay: a bug that surfaced in production at 3am can be reproduced exactly by replaying the event log from the appropriate Flow ID. This eliminates the most expensive class of distributed-systems bugs — the unreproducible ones.

Audit by construction: the log *is* the audit trail. Compliance and regulatory queries operate directly on the canonical record; nothing is reconstructed after the fact, which means nothing can be reconstructed differently after the fact.

Tamper-evidence: events are hash-chained and signed. Modifying historical events requires forging the chain, which is detectable on the next read.

Bounded memory: projections snapshot. Old events compact into checkpoints. Memory growth is linear in active work rather than in lifetime work — an essential property for systems running indefinitely.

ESM is the "memory" half of MAN+ESM, and it is named deliberately to distinguish it from vector-store approaches that treat memory as mutable retrieval over an opaque embedding space. ESM is structured, queryable, and verifiable; vector search is a useful projection over ESM, not a replacement for it.

5.6 Micro-Protocols

Workflows in MAN+ESM compose from a small set of well-defined interaction patterns. We describe the six current protocols.

Propose–Critique is the iterative refinement loop. A proposer agent generates a candidate output; a critic agent evaluates it against quality criteria; if validation fails, the protocol loops back with feedback for refinement. Used for code and content generation.

Decompose–Merge is the parallel-fan-out-and-aggregate pattern. A decomposer agent partitions a task into independent sub-tasks; specialists execute in parallel; results merge back through deterministic reducers. Used for multi-step reasoning and analysis.

Contract-Net is decentralized bidding for dynamic resource allocation, following the original Smith & Davis (1980) protocol adapted to LLM-based agents. Available agents bid on incoming tasks; selection is by best score, lowest cost, or fastest expected completion. Used for model selection and intelligent routing.

Consensus is k-of-N validator approval for high-risk decisions. Critic agents vote with signed votes; once a quorum is reached, the decision is sealed into the event log. Used for regulated decisions and high-stakes automation.

Escalate–Approve is the human-in-the-loop pattern. When agent confidence falls below threshold or risk exceeds ceiling, the workflow routes to a human approver before progression. The approval — or rejection — is recorded as an event, becoming part of the audit trail. Used for compliance gates.

Verification Game is the proof-challenge pattern. Independent agents replay computations and challenge results; disagreements trigger arbitration. Used for adversarial validation in high-trust contexts.

These six protocols compose. Most production workflows are sequences of `Decompose-Merge` containing nested `Propose-Critique` loops, with `Escalate-Approve` gates at compliance-sensitive boundaries. The compositional model is what allows MAN+ESM to express arbitrary workflow topologies without hand-coding their orchestration.

6. The Mathematics of Distributed Reliability

The reliability advantage of decentralized coordination is not anecdotal; it is mathematically demonstrable.

In a centralized system, the coordinator is on the critical path of every coordination event. The system as a whole cannot be more reliable than its coordinator:

```
System Reliability = Coordinator Reliability (p)
```

If the coordinator has 90% reliability, the system has 90% reliability — regardless of how robust the individual agents are. The coordinator is a single failure mode that no amount of agent-level engineering can mitigate.

In a distributed system with N redundant coordination paths, the system reliability is:

```
System Reliability = 1 - (1 - p)^N
```

This expresses the probability that *at least one* of the redundant paths succeeds. With the same 90% individual reliability across just three redundant agent groups:

```
System Reliability = 1 - (0.1)^3 = 1 - 0.001 = 99.9%
```

A 90%-reliable component, replicated three ways, yields a 99.9%-reliable system. The redundancy converts moderate component reliability into production-grade system reliability without requiring perfect parts — which matters because perfect parts are not achievable, particularly in LLM-based components where output reliability is fundamentally probabilistic.

A symmetric argument applies to memory. Centralized state management requires storing both per-agent state and shared coordination state across the network:

```
Total Memory (centralized) = N × (Agent Memory + Shared Coordination State)
```

The shared coordination state grows with cluster complexity, which grows with N — meaning total memory grows super-linearly. In MAN+ESM, the shared state is the event log, which is partitioned across Kafka brokers and does not need to be replicated to every agent:

```
Total Memory (distributed) = N × Agent Memory + Minimal Consensus Overhead
```

Memory grows linearly in N. The constant factor matters too — in our internal measurements (Section 7.2), MAN+ESM uses approximately 18 MB per agent versus 220–340 MB for graph-based frameworks — but the asymptotic property is what enables MAN+ESM to support cluster sizes that are infeasible under centralized assumptions.

The third structurally important quantity is *update latency*. Centralized frameworks generally require system-wide restarts to add or modify capabilities; service interruption during updates is on the order of tens of seconds. In MAN+ESM, capability changes propagate through gossip without restart; new agents announce themselves and become discoverable in roughly the time it takes for the announcement to traverse the gossip network — single-digit hops, sub-second latency at typical cluster sizes. This is the difference between a system that requires scheduled maintenance windows and one that supports continuous evolution.

7. Empirical Comparison

We compare MAN+ESM against LangGraph, AutoGen, and CrewAI across two evaluation dimensions. Section 7.1 reports task-level resolution rate on two standard benchmark suites — SWE-bench and T-bench — under matched conditions: same model, same prompts, same task definitions across all four frameworks. Section 7.2 reports internal infrastructure measurements of coordination latency, memory footprint, capability discovery time, and maximum stable cluster size as the substrate scales from ten to a thousand agents.

7.1 Task-Level Evaluation: SWE-bench and T-bench

We selected ten tasks from each of two benchmark suites and ran them across all four frameworks under matched conditions. **SWE-bench** tasks were drawn from the Astropy and Django projects and consist of real-world software bug-fixing problems; a task is considered resolved only if the proposed fix passes all relevant tests. **T-bench** tasks include "Fix Git" and "COBOL Modernization" together with several medium-complexity tasks; success requires the agent to complete the task as specified.

Table 1 reports SWE-bench resolution rates.

Table 1. *SWE-bench task resolution rates (10 tasks, Astropy & Django).*

Framework	Tasks Resolved	Resolution Rate
MAN+ESM (Adya)	**5 / 10**	**50%**
MAN (Adya)	5 / 10	50%
AutoGen	2 / 10	20%
LangGraph	1 / 10	10%

Both MAN configurations resolve half the SWE-bench suite. AutoGen resolves one in five; LangGraph one in ten. On this suite the addition of ESM does not improve the resolution rate beyond MAN alone, which suggests that for self-contained bug-fix problems the working context fits comfortably within a single agent invocation and the persistent-memory layer adds no operational uplift.

Table 2 reports T-bench resolution rates.

Table 2. *T-bench task resolution rates (10 tasks, Fix Git & COBOL Modernization plus medium-complexity tasks).*

Framework	Tasks Resolved	Resolution Rate
MAN+ESM (Adya)	**10 / 10**	**100%**
MAN (Adya)	4 / 10	40%
AutoGen	1 / 10	10%
LangGraph	0 / 10	0%

The T-bench results are the strongest single piece of evidence we have for the operational value of the event-sourced memory layer. MAN alone resolves four of ten tasks; MAN+ESM resolves all ten. AutoGen resolves one; LangGraph resolves none. The gap between MAN and MAN+ESM is forty percentage points on the same task suite with the same model and the same prompts — the only difference is the presence of the event-sourced memory substrate.

The mechanism is interpretable. T-bench tasks involve multiple stages of work that exceed the working context of a single agent invocation: code must be inspected, modifications planned, edits applied, tests run, results interpreted, and corrections made. Without persistent, replayable working state, intermediate context is lost at the boundary between agent invocations and the workflow fails partway through. With ESM, every observation, decision, and intermediate output is appended to the event log; subsequent agent invocations replay the relevant slice of the log to reconstruct the working state and continue. The same architectural property that makes the system auditable makes it capable of completing multi-stage tasks. ESM is therefore better understood not as an optimization but as a capability prerequisite for tasks that exceed a single-shot context window.

We also measured mean per-task wall-clock execution time on the T-bench suite. AutoGen completed in 151.3 seconds on average, LangGraph in 180.0 seconds, MAN in 200.0 seconds, and MAN+ESM in 220.0 seconds. AutoGen is the fastest framework but resolves the fewest tasks (1 of 10); MAN+ESM is the slowest by approximately 70 seconds and resolves all tasks. The trade-off favours reliability at the margin we care about: a framework that completes one in ten tasks faster is not faster in any operationally meaningful sense.

Two caveats apply to this evaluation. First, the scope is small by design — ten tasks per suite. The goal was a same-model, same-prompt comparison across four frameworks under matched conditions, not a full benchmark sweep. Larger sweeps with the full SWE-bench Lite and T-bench task universes will publish alongside the public release. Second, the relative ordering of frameworks is consistent across both suites: MAN+ESM dominates on T-bench and ties on SWE-bench, MAN follows, AutoGen follows, LangGraph trails. The consistency across two structurally different task suites is informative even at this sample size.

7.2 Infrastructure Measurements: Latency, Memory, and Scale

The numbers reported in this section reflect early-stage internal measurements taken on equivalent hardware against framework reference implementations using the same task suite, the same model, and the same prompts. The methodology is preliminary and the absolute values should be treated as order-of-magnitude indicators rather than precise benchmarks. The relative shape of the comparison is stable across our internal runs, and a reproducible measurement suite — including hardware specifications, prompt corpus, and framework reference implementations — will publish alongside the public release of this work.

The headline qualitative findings are as follows.

Coordination latency. MAN+ESM remains in a sub-second band across the full sweep from 10 to 1,000 agents. LangGraph's latency grows roughly linearly with cluster size and crosses minutes by 100 agents. AutoGen scales somewhat better due to native parallelism but loses coordination correctness as cluster size grows. CrewAI hits memory exhaustion before latency becomes the limiting factor.

Memory footprint. MAN+ESM's per-agent memory usage is approximately one order of magnitude smaller than the graph-based alternatives at typical cluster sizes. The dominant savings come from two places: agents do not retain shared coordination state (it lives in the event log), and the context retained in the working scratchpad is bounded by TTL rather than by lifetime.

Capability discovery. Gossip-based discovery in MAN+ESM completes in single-digit hops at any plausible cluster size. Registry-based alternatives put discovery latency on a single machine's critical path, which scales linearly with cluster size; at 100 agents the gossip approach is roughly an order of magnitude faster than the registry approach in our measurements.

Maximum stable cluster size. The frameworks under comparison plateau at cluster sizes between 30 and 80

agents before degradation becomes severe. MAN+ESM has been operated stably at over 1,000 concurrent agents in internal testing, with the design target of 5,000+ agents under SLA conditions. The bottleneck above this range is the underlying Kafka deployment, which scales independently.

Two qualitative findings are also worth noting because they do not appear in numeric form.

Update latency. Hot-swapping an agent — replacing one capability implementation with another while the system remains live — completes in sub-100ms in MAN+ESM versus tens of seconds of system-wide downtime in centralized alternatives. This compounds: continuous improvement is feasible in one architecture and impractical in the other.

Selective context retrieval. When an agent in MAN+ESM requires the output of a specific upstream peer, it retrieves only the relevant event from the log, not the full conversation state. The graph-based alternatives pass entire state messages including outputs from agents the current task does not depend on. The memory and bandwidth savings are significant at scale and compound with cluster size.

The full benchmark suite — including hardware specifications, prompt corpus, framework versions, methodology, and reproducible notebooks — will publish alongside the public release of this work, scheduled for 2026.

8. Production Implementation: Vanij Agent Studio

Architectural elegance is necessary but not sufficient. A decentralized substrate that requires distributed-systems expertise to deploy is not, in practice, more accessible than the centralized alternatives — its theoretical advantages are unrealized because the people who could benefit cannot use it. The question is whether the architecture can be hidden behind interfaces that domain experts can use without specialized training.

We answer this question affirmatively through Vanij Agent Studio, Adya's production platform built on the MAN+ESM substrate. Three design choices in Vanij translate the architectural primitives into accessible capabilities.

First, **natural-language workflow specification**. Users describe goals in plain language — "analyze quarterly sales data and generate executive summaries" — and the platform parses the intent, maps it to capabilities discoverable through the dynamic registry, and proposes agent combinations. The user sees a workflow; the platform handles the gossip discovery, capability matching, and event routing underneath.

Second, **agent ecosystem categorization**. Available agents fall into three classes — *Created* (custom for the organization), *Prebuilt* (platform-provided for common patterns), and *Marketplace* (external third-party agents) — each surfaced through the same interface. Agents join the ecosystem dynamically, without system-wide redeployment, directly implementing the MAN+ESM dynamic registry concept at the user-visible layer.

Third, **observability through the Playground**. Workflow execution is visualized in real time; Flow ID propagation produces a complete reconstruction of which agents were invoked, in what order, with what inputs, and at what cost. Token usage is tracked per agent, per task, and per coordination pattern, enabling fine-grained cost attribution that centralized frameworks cannot provide because they lack the underlying observability primitive.

The Vanij implementation demonstrates that decentralized coordination can be productized without exposing its complexity. Domain experts deploy multi-agent workflows without understanding gossip protocols or event-sourced memory; the architectural sophistication operates underneath, invisibly, at a layer the user does not need to think about.

This is not a small claim. Several adjacent fields — distributed databases, container orchestration, modern web infrastructure — have made the same transition: sophisticated distributed substrate, accessible high-level interface. The pattern works because the substrate's complexity is *generic*, while the application's complexity is *domain-specific*. A platform that gets the substrate right earns the right to expose only the domain-specific complexity to its users.

9. Limitations and Research Frontiers

We close by enumerating the limitations of the current MAN+ESM design and the research directions we consider most important for the next generation of the architecture.

Adaptive learning integration. Current agent capabilities are largely static — an agent's behaviour is determined at deployment time and does not improve with experience. The Flow ID and event-log infrastructure provides the substrate for learning loops: aggregate analysis of historical Flow IDs could identify successful coordination patterns and feed those patterns back as routing improvements, prompt refinements, or new specialist proposals. Closing this loop is an active research direction.

Memory-augmented decision making. The current ESM design supports retrieval over the event log but does not yet integrate Retrieval-Augmented Generation (RAG) over historical experience as a first-class input to agent decisions. Allowing agents to consult their own past coordination patterns when making current decisions — and to share relevant patterns across peers — would convert the event log from a passive audit trail into an active source of decision-relevant context.

True peer-to-peer discovery across organizational boundaries. The current gossip protocol operates within a deployment. Cross-organizational coordination requires protocols that allow agents to discover and collaborate with peers in other Adya tenants, in third-party MAN+ESM deployments, or eventually in compatible non-Adya systems. Federation across deployment boundaries is an open architectural question.

Privacy-preserving collaboration. Cross-organizational coordination raises immediate questions about what agents can disclose to each other. Privacy-preserving protocols — selective capability disclosure, encrypted context, attribute-based access control — are required before federated coordination can reach regulated industries. We are pursuing this direction in collaboration with the AGP (Adaptive Governance Protocols) effort that complements MAN+ESM in Adya's research portfolio.

Schema evolution and protocol negotiation. As agents evolve, their capability schemas evolve. The current Avro-based schema registry handles versioned compatibility but does not yet support automated protocol negotiation between agents with mismatched but compatible schemas. This is on the near-term roadmap.

Formal verification of the consensus protocol. The k-of-N consensus mechanism is currently specified informally and validated empirically. Formal verification of its safety and liveness properties — particularly under partial network failures and Byzantine agent behaviour — would strengthen the substrate's guarantees and is an open research goal.

10. Conclusion

The multi-agent systems field has reached an architectural inflection point analogous to the monolith-to-microservices transition that reshaped web infrastructure two decades ago. Centralized coordination frameworks — LangGraph, AutoGen, CrewAI — have democratized agentic AI development and delivered real value at small scale, but they cannot satisfy the simultaneous demands of production deployment: linear scalability, robust reliability, and sub-second coordination.

We have argued that this is an architectural constraint rather than an implementation one, and that the resolution requires committing to decentralized coordination as a primary design property rather than as an optimization. We have presented MAN+ESM, an operational decentralized substrate that achieves the three pressures of the coordination triad simultaneously through event-driven messaging, gossip-based discovery, Flow ID observability, event-sourced memory, and a small set of composable micro-protocols. We have provided the mathematical foundations that show why the distributed architecture compounds component reliability rather than being capped by it. We have reported a task-level evaluation against the three dominant frameworks on SWE-bench and T-bench — including the result that the addition of ESM converts MAN's 40% T-bench resolution rate into 100% on the same task set — and we have described how the architecture translates into accessible production tooling through Vanij Agent Studio.

The architectural foundation is operational, the comparative analysis is favourable, and the production translation is in active deployment. The work remaining is in three directions — closing the learning loop on historical coordination patterns, federating the architecture across organizational boundaries, and formalizing the safety properties of the consensus mechanism. We are pursuing each of these directions, and we welcome collaboration from the research community on all three.

The coordination problem is solvable, decentralization is the path, and the architecture for production-scale multi-agent reasoning is no longer a research aspiration. It is a substrate, in production, available for stress-testing.

References

[1] Smith, R. G., & Davis, R. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12), 1104–1113.

[2] Demers, A., Greene, D., Hauser, C., et al. (1987). Epidemic Algorithms for Replicated Database Maintenance. *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing (PODC)*, 1–12.

[3] Fowler, M. (2005). Event Sourcing. martinfowler.com.

[4] Kreps, J. (2013). The Log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn Engineering.

[5] Wang, Q., Bansal, G., Liu, B., et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. arXiv:2308.08155.

[6] Chase, H., et al. (2024). LangGraph: Stateful, Multi-Actor Applications with LLMs. LangChain Inc. Technical Documentation.

[7] FIPA. (2002). FIPA Agent Communication Language Specifications. Foundation for Intelligent Physical Agents.

[8] Bellifemine, F., Caire, G., & Greenwood, D. (2007). Developing Multi-Agent Systems with JADE. Wiley.

Correspondence: research@adya.ai · adya.ai/research
Adya AI · Hyderabad · Bengaluru · Goa
This is a preprint. The full benchmark suite, reproducible notebooks, and the open-source release of MAN+ESM are scheduled for 2026.