International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Comparative Analysis of Agentic Orchestration Layers: Personalization, Planning, Search, MCP Integration, and Reasoning Across Twelve Open-Source Frameworks

Ankur Aggarwal

Abstract

INTRODUCTION: The rapid growth of LLM-powered agentic systems has produced a fragmented ecosystem of open-source orchestration frameworks addressing agent memory, planning, retrieval, and reasoning.

OBJECTIVES: To comparatively evaluate twelve leading open-source orchestration frameworks across dimensions critical to production deployment.

METHODS: Twelve frameworks were evaluated across five dimensions: personalization, memory architecture, planning paradigms, retrieval integration, and Model Context Protocol (MCP) adoption depth.

RESULTS: Findings include a personalization spectrum from stateless to memory-first designs, a planning-control trade-off as the primary selection axis, and near-universal MCP adoption where integration depth is the differentiating factor.

CONCLUSION: Framework selection affects reliability, scalability, and production viability; the five-dimensional evaluation provides actionable heuristics for matching framework architecture to deployment context.

Keywords: *Agentic AI, LLM Orchestration, Retrieval-Augmented Generation, Model Context Protocol, Multi-Agent Systems, Open-Source Frameworks, Reasoning Patterns*

1. Introduction

The transition from single-turn prompt-response language model interactions to goal-directed autonomous agents represents one of the most consequential shifts in applied artificial intelligence since the transformer architecture became the dominant foundation for natural language processing. Where a prompt-response model produces an output and terminates, an agentic system perceives its environment, decomposes complex goals into executable steps, invokes external tools, observes results, and iterates until a termination condition is reached. Wang et al. [1] survey the landscape of LLM-based autonomous agents comprehensively, identifying memory, planning, action, and environment interaction as the four canonical agent subsystems. What their survey and related works do not resolve, because the ecosystem was still forming, is which open-source orchestration frameworks best implement each subsystem in 2025, and how those implementation choices interact across the full five-dimensional profile that production deployments require.

The orchestration layer, the software framework coordinating how agents perceive, plan, act, and learn, is not a commodity choice. The same agent behavior implemented across different frameworks may differ substantially in debugging access, failure mode characteristics, horizontal scalability, and integration costs with existing infrastructure. A framework that excels for a RAG-first knowledge agent (LlamaIndex) may be poorly suited for a software engineering pipeline requiring SOP-structured multi-agent coordination (MetaGPT), which in turn differs from a general-purpose production workflow requiring checkpointed replay (LangGraph). The selection decision is consequential and underspecified in the current literature.

Several developments since the most recent comprehensive surveys make a current analysis necessary. Microsoft consolidated AutoGen and Semantic Kernel into the Microsoft Agent Framework (MAF), a production-ready successor with stable APIs and full MCP, Agent-to-Agent (A2A), and AG-UI protocol support, signaling the

Meta, USA

end of the framework proliferation phase and the beginning of convergence. Google introduced the Agent Development Kit (ADK) [10] with native A2A support. The Model Context Protocol (MCP) [4], introduced by Anthropic in November 2024, achieved near-universal adoption across all major frameworks within twelve months. Meta launched Muse Spark, a model with built-in multi-agent orchestration, raising structural questions about the long-term role of external orchestration layers. Each development reshapes the framework selection landscape in ways prior surveys do not capture.

This paper evaluates twelve leading open-source frameworks across five dimensions chosen for direct practical relevance to production agent deployments. The analysis is grounded in primary documentation, published research, and empirical benchmark data. The paper additionally examines the model layer and concludes with practical selection heuristics, emerging trend analysis, and predictions for ecosystem consolidation. The analysis draws on the taxonomy of agentic architectures developed by Abou Ali et al. [21] as a structural reference frame.

2. Survey Scope and Evaluation Methodology

2.1 The Twelve Frameworks

Table 1 presents the twelve frameworks included in this analysis. The selection covers the dominant open-source frameworks as of 2025, spanning stateful graph execution (LangGraph), unified enterprise multi-agent (MAF), role-based crews (CrewAI), lightweight primitives (OpenAI Agents SDK, Agno), hierarchical multi-agent (Google ADK), RAG-first workflows (LlamaIndex), production pipeline DAG (Haystack), SOP-encoded assembly line (MetaGPT), declarative prompt optimization (DSPy), and MCP-native composition (mcp-agent). The Microsoft consolidation of AutoGen and Semantic Kernel into MAF is treated as a resolved development: AutoGen is in maintenance mode, and MAF is the current enterprise-recommended successor offering cross-runtime Python and .NET interoperability alongside the full MCP/A2A/AG-UI protocol suite.

Table 1: The Twelve Frameworks, Maintainer, Architecture Style, and License

Framework	Maintainer	Architecture Style	Language(s)	License
LangGraph	LangChain	Stateful directed graph	Python/JS	MIT
MAF	Microsoft	Unified multi-agent (AutoGen + Semantic Kernel)	Python/.NET	MIT
CrewAI	CrewAI Inc.	Role-based crews	Python	MIT
OpenAI Agents SDK	OpenAI	Lightweight agent primitives	Python	MIT
Google ADK	Google	Hierarchical multi-agent	Python	Apache 2.0
Agno	Agno AGI	Minimal multi-modal agents	Python	MIT
LlamaIndex	LlamaIndex	RAG-first agentic workflows	Python/TS	MIT
Haystack	deepset	Pipeline-based DAG	Python	Apache 2.0
MetaGPT	DeepWisdom	SOP-encoded assembly line	Python	MIT
DSPy	Stanford NLP	Declarative prompt optimization	Python	MIT
mcp-agent	LastMile AI	MCP-native composable patterns	Python	MIT

Survey scope: twelve leading open-source agentic orchestration frameworks as of 2025. MAF (Microsoft Agent Framework) unifies AutoGen (now maintenance mode) and Semantic Kernel under a single enterprise-ready framework with stable APIs and full protocol interoperability.

Four primary architectural styles are represented. Stateful graph execution frameworks (LangGraph, MAF, and Haystack) model workflows as directed graphs where nodes represent computational steps and conditional edges govern transitions. Role-based frameworks (CrewAI, MetaGPT) organise

agents around defined roles with structured inter-agent communication, fitting naturally for multi-agent pipelines that mirror human organizational structures. Lightweight primitive frameworks (OpenAI Agents SDK, Agno, Google ADK) minimize framework abstraction, relying on the underlying model's native function-calling and reasoning capabilities for task decomposition. Specialized frameworks (LlamaIndex, DSPy, and mcp-agent) optimize deeply for a single architectural concern, retrieval, prompt optimization, and protocol-native composition, respectively.

2.2 Five Evaluation Dimensions

The five evaluation dimensions span the complete agent operation cycle. Personalization and memory architecture determine whether agents can build and maintain user-specific context across sessions. Planning paradigms govern goal decomposition, whether implicit in the model's function-calling behavior or explicit in framework-level structures such as graphs, role assignments, or standard operating procedures (SOPs). Search and retrieval integration covers Retrieval-Augmented Generation (RAG) capabilities, vector database ecosystem breadth, and position on the on-device versus on-server retrieval spectrum. MCP adoption depth measures how architecturally central MCP is to the framework's tool model. Reasoning pattern support characterizes which of the five major reasoning patterns the framework natively enables and whether reasoning is primarily model-dependent or framework-controlled. A methodological note on scoring: all dimension scores in Table 6 are assigned based on documented framework capabilities as described in primary framework documentation, published research papers, and empirical benchmark data available as of mid-2025; they do not reflect independent benchmark testing conducted by the authors. Practitioners should validate scores against their specific deployment requirements and model configurations.

3. Dimension 1 - Personalization and Memory Architecture

3.1 Memory Architecture Across Frameworks

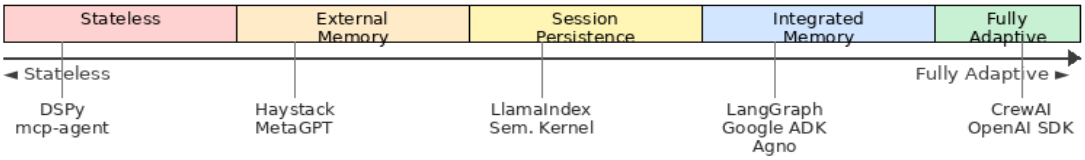
Effective personalization in agentic systems depends on memory architectures operating across four functional layers. Buffer memory (short-term) retains recent conversation turns within a session and resets at session end. Long-term memory persists across sessions with facts about the user, preferences, past decisions, and domain expertise, enabling agents to adapt based on accumulated individual knowledge. Entity memory stores structured knowledge about specific people, places, and concepts encountered in interactions. Episodic memory captures specific past interaction outcomes, preserving records of which approaches succeeded or failed under which conditions. The surveyed frameworks diverge substantially on which layers they implement natively, how they manage inter-layer coordination, and how memory content is integrated into the agent's active prompting pipeline. CrewAI's unified memory system is the most architecturally sophisticated in the survey. Its single Memory class uses an LLM to analyze content at save time, automatically inferring scope, category, and importance without developer annotation. Memory is organized in hierarchical scopes analogous to a file system, and recall uses composite scoring blending semantic similarity, recency decay, and importance weighting. Before each agent task, the framework automatically recalls relevant context and injects it into the prompt; after each task, discrete facts are extracted and stored. OpenAI's Agents SDK takes a philosophically distinct approach centered on context engineering, deliberately shaping what the model knows at any moment. The RunContextWrapper provides a typed shared-state container; Sessions maintain a within-run working context and can be persisted across runs using a structured profile-and-notes accumulation pattern, with distillation during runs and deduplication at session end. LangGraph integrates memory with its graph execution model through persistent checkpointing: every graph node can read from and write to shared state, and the checkpoint mechanism ensures state survives across sessions, failures, and human-in-the-loop interruptions.

3.2 Key Finding: The Personalization Spectrum

Figure 1 positions the twelve frameworks on a personalization spectrum from stateless to fully adaptive. At the stateless end, DSPy and mcp-agent treat memory as a developer-managed external concern; the frameworks maintain no cross-request state. Haystack and MetaGPT provide vector store integrations but do not embed memory management as a first-class framework concern. LlamaIndex and Semantic Kernel support within-session context natively and cross-session retrieval through vector store integrations. LangGraph, Google ADK, and Agno embed memory as a core primitive, graph state, or session state, persisting automatically.

CrewAI and the OpenAI Agents SDK anchor the fully adaptive end, embedding intelligent memory analysis and automatic recall into the execution loop through architecturally distinct mechanisms: CrewAI through LLM-analyzed hierarchical scoping and OpenAI SDK through developer-specified context engineering with framework-managed lifecycle.

Figure 1: Framework Personalization Spectrum, Stateless to Fully Adaptive



Frameworks positioned on the personalization spectrum. Stateless frameworks treat memory as an external concern; fully adaptive frameworks embed intelligent memory management as a first-class execution primitive, automating recall, storage, and cross-session context injection.

The practical selection implication is a direct trade-off between personalization richness and operational complexity. Stateless frameworks scale horizontally without shared-state synchronization overhead. Fully adaptive frameworks deliver contextually rich agent behavior but introduce state management, conflict resolution, and storage infrastructure requirements. For consumer-facing products where personalization is the primary value proposition, integrated memory architecture is typically justified; for batch processing or stateless API-serving workloads, automated memory management provides no return.

4. Dimension 2- Planning Paradigms

4.1 Planning Taxonomy and Framework Analysis

Task decomposition, converting a complex goal into an ordered sequence of executable steps, is the dimension on which frameworks diverge most sharply in both design philosophy and practical capability. Implicit function-calling planning (OpenAI Agents SDK, Google ADK, and Agno) delegates decomposition entirely to the underlying LLM. The framework presents available tools and awaits the model's tool-selection decisions, making planning quality a function of the deployed model's capability. This approach requires minimal framework configuration and performs well with frontier models on bounded tool sets; it sacrifices developer control over execution order and makes debugging dependent on the model's internal reasoning. Graph-based planning (LangGraph, MAF, Haystack) makes execution topology explicit. LangGraph's checkpointing is uniquely valuable; it enables the agent to rewind to any prior state and explore an alternative branch, a capability absent from every other framework in the survey.

Role-based delegation (CrewAI, MetaGPT) assigns tasks to specialized agents based on defined roles, with inter-agent communication structured around role boundaries. Hong et al. [2] demonstrated that MetaGPT's SOP-encoded planning approach, embedding Standard Operating Procedures into each agent role's execution model with verified intermediate outputs before downstream handoff, substantially reduces hallucination propagation in software generation tasks relative to unconstrained multi-agent generation. DSPy [3] represents the most conceptually distinct approach: developers specify input-output signatures and a quality metric, and DSPy's optimizer searches the prompt space to discover the strategy that best satisfies the metric, treating planning as an engineering optimization problem rather than an architectural design decision.

4.2 Key Finding: The Planning-Control Trade-off

Table 2 presents the planning paradigm trade-off matrix. Planning control and deployment simplicity trade inversely. Implicit planning minimises framework configuration but maximises dependence on model behavior, which is rational when the deployed model is frontier-class, and the task involves a bounded, well-defined tool set. Graph-based planning maximizes developer control and auditability at the cost of upfront design complexity, which is non-negotiable for production workflows where reproducibility and debugging access are hard requirements. Role-based and SOP-encoded paradigms suit domain-structured multi-agent pipelines. Programmatic optimization is most valuable when a quality metric can be defined precisely, and the task executes repeatedly at scale.

Table 2: Planning Paradigm Trade-off Analysis

Paradigm	Advantages	Limitations	Best Suited For
Implicit (function calling)	Simple; leverages native model capability; minimal framework overhead	Limited control; execution order depends on model behavior	Well-defined tool sets with capable frontier models
Graph-based	Full execution control; debuggable; replayable via checkpointing	Complex to design; topology must be explicitly specified	Complex production workflows requiring auditability
Role-based delegation	Intuitive; mirrors human team structures; natural task handoff	Overhead for simple single-agent tasks	Multi-agent collaboration with specialized domain roles
SOP-encoded	Reduces hallucination cascades; verifies intermediate results	Domain-specific; requires SOP design upfront	Software engineering and structured multi-step pipelines
Programmatic optimization	Automatically discovers optimal prompts/strategies; measurable	Requires evaluation metrics and a labeled validation set	Repeatable tasks with quantitatively measurable quality

Planning paradigm comparison. Framework examples, Implicit: OpenAI SDK, Google ADK, Agno; Graph-based: LangGraph, MAF, Haystack; Role-based: CrewAI, MetaGPT; SOP-encoded: MetaGPT; Programmatic: DSPy.

5. Dimension 3 - Search, Retrieval, and Vector Database Architecture

5.1 The On-Device vs. On-Server Dimension

Retrieval-Augmented Generation (RAG) addresses the fundamental limitation of parametric LLM knowledge, fixed at training, by retrieving current, domain-specific information at inference time. The architectural decision that most consequentially shapes a RAG system's privacy, latency, and scalability profile is vector index placement. Cloud-hosted vector databases dominate production deployments: Pinecone provides fully managed auto-scaling. Weaviate offers hybrid dense vector and keyword retrieval; Qdrant delivers high-performance filtering; Azure AI Search integrates with Microsoft's enterprise stack; and Elasticsearch provides the mature full-text-plus-vector hybrid that many organizations already operate. Their shared limitation is the privacy boundary; queries traverse an external network connection, making cloud RAG architecturally incompatible with strict data locality requirements.

On-device vector search solutions directly address this limitation. SQLite-vec extends SQLite with vector search capability and WebAssembly (WASM) support, viable for browser-based and mobile agents. LanceDB provides a serverless columnar vector store for embedded deployments without requiring a running server process. Xu et al. [16] surveyed on-device language model deployment comprehensively, documenting that device memory typically constrains personal indexes to fewer than 100,000 vectors. Table 3 presents the decision matrix for choosing between deployment modes. The hybrid architecture, a small personal index on-device for low-latency private retrieval combined with a large organizational knowledge base on-serve,r is increasingly the production default for mobile and edge agent deployments.

Table 3: Vector Database Deployment Decision Matrix, On-Device vs. On-Server

Factor	On-Device / Edge	On-Server / Cloud
Query latency	<10ms	50–200 ms, including network round-trip
Data privacy	Data stays local; no external trust boundary	Data transmitted to the server requires contractual or technical trust
Index scale	<100K vectors (constrained by device memory)	Millions to billions of vectors
Cost model	Device compute only; no per-query pricing	Per-query or per-storage cloud pricing
Index freshness	Sync delay relative to server state	Real-time updates possible

Offline capability	Full functionality available offline	Degraded or unavailable without connectivity
Best for	Personal context, edge AI, mobile agents, privacy-sensitive workloads	Organizational knowledge bases, shared data, high-scale retrieval

Decision matrix for vector index placement. The hybrid architecture (personal index on-device + organizational knowledge base on-server + selective sync) is the emerging production default for privacy-sensitive mobile and edge agent deployments.

5.2 Framework RAG Capabilities

LlamaIndex [13] was purpose-built for retrieval and remains the strongest framework in the survey for RAG depth, natively supporting over 40 vector stores and a comprehensive document loader ecosystem. Its distinctive architectural contribution is Agentic RAG: each document collection receives a dedicated sub-agent with search and summarization tools, and a top-level orchestrating agent coordinates across all document sub-agents, enabling multi-hop reasoning chains that traverse multiple knowledge sources. Haystack [12] provides production-grade retrieval pipelines through a composable directed acyclic graph (DAG) architecture with native support for pipeline evaluation and A/B testing of retrieval strategies, a quality assurance capability absent from most other frameworks in the survey. MAF provides first-class enterprise vector integration with Azure AI Search alongside content filtering through Microsoft's Responsible AI toolkit. CrewAI takes a distinctive approach by using vector stores primarily for agents. memory rather than document retrieval, short-term, entity, and episodic memories are stored as embeddings and retrieved semantically, unifying memory management and knowledge retrieval under a single infrastructure layer.

6. Dimension 4, MCP Integration and Protocol Ecosystem

6.1 MCP Architecture and the Layered Protocol Stack

The Model Context Protocol (MCP) [4], introduced by Anthropic in November 2024, standardizes tool and data source integration for LLM applications through a JSON-RPC 2.0 host-client-server architecture. Hosts (LLM applications) initiate connections through clients (protocol connectors embedded within the host), which maintain dedicated connections to servers exposing three capability types: tools (callable functions), resources (queryable data sources), and prompts (templated interaction patterns). MCP's core contribution is standardized tool discovery: any MCP-compatible agent can connect to any MCP server without custom integration code, enabling composable tool ecosystems. By 2025, MCP achieved near-universal adoption across all survey frameworks, a consolidation speed reflecting both the protocol's technical quality and the industry's appetite for a universal integration standard.

Two complementary protocols have emerged alongside MCP. Agent-to-Agent (A2A), introduced by Google and adopted by MAF, provides JSON-RPC over HTTP for structured inter-agent communication, enabling multi-agent systems where specialised agents expose capabilities to orchestrating agents without direct code coupling. AG-UI standardizes HTTP and server-sent events (SSE) streaming for user-facing agent interactions. MCP (tool access), A2A (agent-to-agent), and AG-UI (user-to-agent) together constitute a three-layer interoperability stack with direct structural analogies to the OSI network model, suggesting agent protocol interoperability is converging toward a solved-infrastructure state.

6.2 MCP Integration Depth Across Frameworks

Table 4 presents MCP integration depth ratings. The critical observation is not that all frameworks support MCP; by 2025, they do, but that architectural centrality varies from foundational to incidental. mcp-agent treats MCP as the complete and sufficient architecture for agentic systems: every tool, resource, and capability is exposed through MCP servers, and the framework's composition primitives are built entirely around MCP's capability model. The OpenAI Agents SDK integrates MCP at the same depth as native function tools; MCP server tools are indistinguishable from locally defined Python functions within the agent loop. Google ADK and MAF provide first-class MCP support alongside their own protocol extensions (A2A and AG-UI, respectively). Frameworks in the Supported tier offer fully functional MCP integrations but treat MCP as one of multiple integration patterns rather than the primary architectural primitive.

Table 4: MCP Integration Depth Across Twelve Frameworks

Framework	Integration Level	Key Details
mcp-agent	Native (★★★★★)	Built entirely around MCP; treats MCP as sufficient architecture for agentic systems
OpenAI Agents SDK	Native (★★★★★)	Built-in MCP server tool integration; MCP tools behave identically to function tools
Google ADK	First-class (★★★★)	MCP tools in rich tool ecosystem; A2A protocol for agent-to-agent communication
MAF	First-class (★★★★)	Full protocol suite: MCP server + client, A2A, AG-UI
LangGraph	First-class (★★★★)	MCP integration via langchain-mcp package; deep ecosystem support
CrewAI / Agno / Haystack / LlamaIndex	Supported (★★★)	MCP tool integration available; Haystack pipelines can be exposed as MCP servers
DSPy / MetaGPT	Basic (★★)	MCP integration available but not central to the framework design philosophy

MCP adoption across the twelve surveyed frameworks. Integration levels, Native: MCP is the primary tool architecture; First-class: MCP is a primary integration with full ecosystem support; Supported: MCP available as a fully functional option; Basic: MCP available but not central to framework design.

7. Dimension 5, Reasoning Patterns and Cognitive Architecture

7.1 Reasoning Pattern Taxonomy

Table 5 presents the five major reasoning patterns across the survey. ReAct [5], the interleaved Thought-Action-Observation loop, is the near-universal baseline: every framework supports ReAct-style reasoning through model function-calling, and it is the appropriate pattern for the majority of tool-using agent tasks. Its limitation is convergence: poorly specified ReAct agents can cycle through reasoning loops without approaching a termination condition. Reflexion [6] addresses this by adding verbal reinforcement learning: after completing or failing a task, the agent generates a self-critique stored in episodic memory, enabling subsequent attempts to incorporate explicit awareness of prior failure modes. Shinn et al. demonstrated that Reflexion substantially improves performance across decision-making, coding, and language-reasoning benchmarks relative to non-reflective baselines. Plan-and-Execute separates planning from execution, producing structured and predictable execution at the cost of brittleness when the upfront plan is incorrect. Tree of Thoughts explores multiple parallel reasoning branches before committing to an execution path, maximizing coverage of ambiguous problems at substantially higher LLM compute cost.

Table 5: Reasoning Pattern Taxonomy, Mechanism, Strengths, and Weaknesses

Pattern	Mechanism	Strengths	Weaknesses
ReAct	Interleaved Thought → Action → Observation loops	Fast; grounded in tool output	Can enter non-converging reasoning loops
Reflexion	Self-critique after task completion; episodic memory of failures	Improves over successive trials	Requires external evaluation signal

Plan-and-Execute	Complete plan generated upfront, then executed sequentially	Structured and predictable	Brittle when the initial plan is incorrect
Tree of Thoughts	Explores multiple parallel reasoning branches; evaluates each path	Handles ambiguous multi-path problems	Computationally expensive; requires many LLM calls
Chain-of-Thought	Step-by-step reasoning within a single model completion	Simple; broadly effective	No tool use; limited to a single forward pass

Five primary reasoning patterns across the agentic framework ecosystem. ReAct is the near-universal baseline. Reflexion adds self-improvement through episodic critique. Plan-and-Execute trades flexibility for predictability. Tree of Thoughts maximizes reasoning breadth at high compute cost. Chain-of-Thought provides single-pass structured reasoning without tool invocation.

7.2 Framework Reasoning Capabilities

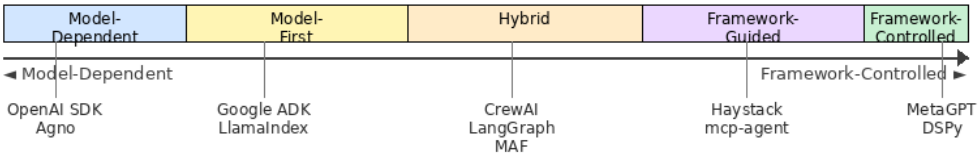
LangGraph [7] provides greater reasoning flexibility than any other framework in the survey. Its graph execution model can express any reasoning pattern through the composition of graph primitives: ReAct loops are cycles; Plan-and-Execute maps to a sequential planning node followed by execution nodes. Reflection is a dedicated evaluation node writing self-critique to a persistent checkpoint state. LangGraph's unique capability is path rewind; checkpointing allows the agent to return to any prior reasoning state and explore an alternative branch, enabling Tree-of-Thoughts-style exploration without a dedicated ToT implementation. MetaGPT and DSPy represent the framework-controlled reasoning pole. MetaGPT embeds reasoning into organizational structure; each role follows structured thinking templates that decompose reasoning into verifiable intermediate steps, preventing hallucination propagation across agent boundaries. DSPy [3] treats reasoning as an optimization problem: the developer specifies input-output signatures and a quality metric, and DSPy's compiler discovers the reasoning chain that best satisfies the metric.

Google ADK [10] implements hierarchical reasoning through agent delegation: the orchestrating agent decomposes complex tasks by routing sub-problems to specialised sub-agents, producing more predictable behavior than flat ReAct loops for multi-step problems. The OpenAI Agents SDK exemplifies the model-dependent pole: the framework provides minimal reasoning scaffolding, trusting the underlying model, particularly the o1 and o3 models with extended thinking capabilities, to reason effectively. LlamaIndex [13] specializes in retrieval-grounded multi-hop reasoning: its agentic RAG architecture enables iterative query refinement where each retrieval step informs the next query, synthesizing evidence from multiple knowledge sources before reaching a conclusion.

7.3 Key Finding: Reasoning Depth vs. Control Trade-off

Figure 2 positions the frameworks on the reasoning control spectrum from model-dependent to framework-controlled. Frameworks at the model-dependent end trust the LLM to reason effectively with minimal scaffolding, appropriate when deploying frontier-class models with strong native reasoning capability and when reasoning determinism is not a hard requirement. Frameworks at the framework-controlled end provide structures that guide or optimize reasoning independently of model capability, appropriate for smaller or specialised models, auditable reasoning requirements, or domain-specific reasoning discipline. LangGraph, MAF, and CrewAI occupy a productive middle ground: framework structures govern high-level reasoning flow while detailed step-level reasoning remains model-dependent.

Figure 2: Reasoning Spectrum, Model-Dependent to Framework-Controlled



Frameworks positioned on the reasoning control spectrum. Model-dependent frameworks rely on the underlying LLM's native reasoning capability. Framework-controlled frameworks provide explicit structures, graphs, SOPs, and optimization loops that guide or constrain reasoning independently of model behavior.

8. Unified Comparison and Selection Heuristics

8.1 Unified Scoring Matrix

Table 6 presents the unified scoring matrix across eleven frameworks and five dimensions using a five-point scale. LangGraph leads overall with best-in-class scores on planning and reasoning, reflecting its graph-based execution model's unique combination of execution control, checkpointed replay, and pattern flexibility. CrewAI and MAF share second position at 3.8, each dominating different dimensions. CrewAI on personalization through its intelligent unified memory system, MAF on MCP, and enterprise protocol interoperability. LlamaIndex scores highest on search and RAG. mcp-agent achieves its peak score on MCP integration, tied with OpenAI SDK for the top rating, while scoring lowest overall due to deliberate minimalism on all other dimensions. DSPy and MetaGPT score identically on reasoning (best-in-class) while diverging sharply on personalization and search, reflecting their respective specializations in prompt optimization and software engineering pipelines.

Table 6: Unified Framework Scoring Matrix, Five Dimensions, Eleven Frameworks

Framework	Personalization	Planning	Search/RAG	MCP	Reasoning	Avg.
LangGraph	●●●●○	●●●●●	●●●●○	●●●●○	●●●●●	4.4
CrewAI	●●●●●	●●●●○	●●●○○	●●●○○	●●●●○	3.8
MAF	●●●○○	●●●●○	●●●●○	●●●●○	●●●●○	3.8
OpenAI SDK	●●●●○	●●●○○	●●○○○	●●●●●	●●●○○	3.4
Google ADK	●●●●○	●●●○○	●●●○○	●●●●○	●●●○○	3.4
LlamaIndex	●●●○○	●●●○○	●●●●●	●●●○○	●●●●○	3.6
Haystack	●●○○○	●●●○○	●●●●●	●●●○○	●●●○○	3.2
MetaGPT	●●○○○	●●●●●	●●○○○	●●○○○	●●●●●	3.2
DSPy	●○○○○	●●●●○	●●○○○	●●○○○	●●●●●	2.8
Agno	●●●●○	●●○○○	●●●○○	●●●○○	●●○○○	2.8
mcp-agent	●○○○○	●●○○○	●○○○○	●●●●●	●●○○○	2.4

Five-point scoring: ●●●●● = best-in-class; ●○○○○ = minimal or no native support. Average is the unweighted mean across five dimensions. Production teams should weigh dimensions by deployment requirement; the unweighted average is not a direct selection criterion.

8.2 Selection Heuristics

The matrix supports practical selection heuristics across eight deployment scenarios. For complex production workflows requiring auditability, replayability, and post-hoc debugging access, LangGraph's graph-based execution with checkpointing is the clear choice. For deployments where personalization and role-based multi-agent collaboration are the primary product requirements, CrewAI's unified intelligent memory and delegation model provides the strongest native support. For rapid development within the OpenAI ecosystem, the OpenAI Agents SDK minimizes framework overhead while providing native MCP integration. For Google Cloud enterprise deployments requiring Gemini-native hierarchical agents with A2A protocol support, Google ADK is the structurally natural choice. For organizations migrating from AutoGen or Semantic Kernel, or requiring multi-language enterprise deployment with the full MCP/A2A/AG-UI protocol suite, MAF provides continuity and stability.

Knowledge-intensive applications where the primary challenge is grounding agent responses in large document corpora should select LlamaIndex; its RAG architecture depth has no peer in the survey. When retrieval pipeline evaluation and A/B testing are production requirements, Haystack's evaluation-first pipeline architecture provides capabilities absent from other frameworks. When software engineering agents with SOP-structured hallucination-resistant workflows are the target, MetaGPT's assembly-line paradigm is appropriate. When programmatic prompt optimization with measurable quality metrics matters more than framework flexibility, DSPy is the only framework

in the survey with automated optimization capability. When an MCP-native composable architecture is the requirement, systems where MCP is the complete tool integration model, the MCP-agent provides the most architecturally consistent implementation.

9. The Model Layer: Llama Family and Muse Spark

9.1 Llama Model Progression

Every orchestration framework in this survey supports Meta's Llama model family, and Llama's architectural evolution directly shapes what orchestration frameworks need to provide. Groeneveld et al. [18] document the Llama 3 family's technical architecture, the foundation on which Llama 4's Mixture-of-Experts (MoE) extensions build. Llama 3.2 1B and 3B models enable on-device inference via ExecuTorch on mobile CPUs with INT4 quantization, supporting basic tool calling for local device actions and combining naturally with on-device vector indexes to create fully local agent architectures where user data never leaves the device, directly addressing the privacy concerns that Xu et al. [16] identify as the primary driver of on-device LLM adoption. Llama 4 introduced MoE architecture for the first time in the Llama family: Scout (109B total parameters, 17B active via 16 experts) and Maverick (400B total, 17B active via 128 experts) [15] deliver frontier-class reasoning quality while activating only 17B parameters per token, making multi-agent deployments economically viable. Maverick's 1M-token context window and Scout's 10M-token window reduce dependence on external vector databases for many use cases; entire interaction histories or document sets can fit within context, enabling in-context personalization without retrieval infrastructure.

9.2 Muse Spark and the Evolving Role of Orchestration Frameworks

Meta Superintelligence Labs launched Muse Spark on April 8, 2026, a proprietary closed-weight model that integrates multi-agent orchestration capabilities directly into the model architecture, departing structurally from the Llama series. Where Llama models rely on external orchestration frameworks to coordinate multiple agents, Muse Spark's "contemplating" mode spawns multiple parallel reasoning agents internally, enabling concurrent multi-agent reasoning without external framework coordination. The model achieves an Intelligence Index of 52 on the Artificial Analysis [17] benchmark, in the top five globally, while requiring substantially less compute than comparable quality-level models. Tool use is part of the native architecture rather than a fine-tuned behavior. Visual chain-of-thought enables step-by-step reasoning over images without requiring multimodal framework plugins.

Muse Spark's built-in orchestration reshapes the framework selection calculus. In traditional deployments, Llama 4 plus a full orchestration framework, the framework provides cognitive coordination (planning, routing, state management, and tool dispatch) while the model provides per-step reasoning. In hybrid deployments, Muse Spark, plus a lightweight framework, the model handles multi-agent reasoning and tool orchestration internally; the framework provides infrastructure services: state persistence, MCP server connections, human-in-the-loop interrupts, monitoring, and guardrails. The direction of travel is clear: as models absorb orchestration responsibilities, frameworks will evolve from cognitive coordinators toward infrastructure providers. The frameworks best positioned for that future excel at persistence, protocol interoperability, observability, and guardrails, rather than compensating for model limitations through complex orchestration logic. A recency limitation applies to this analysis: Muse Spark launched in April 2026, and independent technical evaluations, third-party benchmarks, and published architectural documentation beyond Meta's own announcements were limited at the time of writing. The analysis of Muse Spark's implications for orchestration frameworks should therefore be treated as directional rather than definitive, and readers are encouraged to consult emerging literature as the model's capabilities and ecosystem integrations become better characterized.

10. Emerging Trends and Future Directions

Four trends are actively reshaping the orchestration framework landscape. Protocol convergence is the most immediate: MCP, A2A, and AG-UI are consolidating into a layered interoperability stack that mirrors the historical consolidation of network protocols around the TCP/IP model. Agent protocol interoperability is approaching a solved-infrastructure horizon, shifting competitive differentiation from protocol support to developer experience and operational depth.

Memory as a first-class primitive is the second trend. The contrast between frameworks that treat memory as external infrastructure (Haystack, DSPy) and

those embedding intelligent memory management as a first-class execution primitive (CrewAI, OpenAI Agents SDK) represents an architectural frontier that all frameworks are moving to close. The combination of massive context windows (Llama 4 Scout's 10M token capacity) and LLM-analyzed hierarchical memory is driving convergence in which the distinction between in-context personalization and cross-session persistent memory becomes increasingly architectural rather than fundamental.

The on-device plus on-server hybrid RAG architecture is emerging as the production default for mobile and edge agent deployments. As on-device models mature through the Llama 3.2 family [16, 18] and on-device vector search becomes viable through SQLite-vec and LanceDB, the hybrid pattern is displacing both purely cloud-based and purely local architectures. Frameworks natively supporting this hybrid topology will hold a structural advantage in mobile and edge deployment contexts over the next two to three years. Framework ecosystem consolidation is the fourth trend: Microsoft's merger of AutoGen and Semantic Kernel into MAF signals the end of the proliferation phase. Production deployments are clustering around a small number of cloud-provider-backed frameworks, and the commercially supported production ecosystem will likely converge on three to four dominant frameworks within two to three years.

11. Conclusion

This comparative analysis of twelve open-source agentic orchestration frameworks across five dimensions yields three principal findings. First, no single framework dominates all five evaluation axes: LangGraph leads on planning and reasoning flexibility, CrewAI on personalization depth, LlamaIndex and Haystack on retrieval, and mcp-agent and OpenAI Agents SDK on MCP integration depth. The selection decision is necessarily context-dependent, and the selection heuristics in Section 8.2 provide a practical guide organized around eight deployment-context archetypes. Second, the planning-control trade-off is the primary selection axis for production deployments: implicit function-calling planning suits capable frontier models on bounded tool sets; graph-based planning suits workflows requiring auditability and replayability; role-based and SOP-encoded planning suit domain-structured multi-agent pipelines. Third, MCP adoption has crossed the universality threshold; the differentiating factor is now integration depth, not presence.

The boundary-blurring finding, that the distinctions between memory and retrieval, planning and reasoning, and model capability and framework capability are actively dissolving, carries the most significant implications for framework selection strategy. CrewAI's memory system is architecturally a specialised retrieval system. LangGraph's graph-based planning is a form of structured reasoning. Muse Spark's built-in multi-agent orchestration absorbs capabilities previously requiring external frameworks. Each development point toward a future in which orchestration frameworks provide infrastructure services, persistence, protocol integration, monitoring, guardrails, and human-in-the-loop control, while increasingly capable models provide cognitive coordination. Abou Ali et al. anticipate this shift, framing it as the transition from orchestration-heavy to model-heavy agent architectures.

The practical recommendation that follows is framework selection by elimination rather than optimization. Identify the non-negotiable dimension requirements for the deployment context; eliminate frameworks failing any non-negotiable threshold; select the simplest remaining option. Simplicity in framework architecture correlates with lower debugging complexity, faster team onboarding, and better long-term maintainability, qualities that consistently matter more in production than marginal capability advantages on noncritical dimensions. The frameworks best suited to the emerging model-heavy landscape are those investing in infrastructure quality rather than cognitive scaffolding, a criterion that will prove increasingly decisive as the model layer continues to absorb orchestration responsibilities.

References

1. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci. 2024;18:186345. doi:10.1007/s11704-024-40231-1

2. Hong S, Zhuge M, Chen J, Zheng X, Cheng Y, Zhang C, et al. MetaGPT: meta programming for a multi-agent collaborative framework. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024); 2024 May 7–11; Vienna, Austria. OpenReview.net; 2024. Available from: https://openreview.net/forum?id=VtmBAGCN7o

3. Khattab O, Singhvi A, Maheshwari P, Zhang Z, Santhanam K, Vardhamanan S, et al. DSPy: compiling declarative language model calls into self-improving pipelines. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024); 2024 May 7–11; Vienna, Austria. OpenReview.net; 2024. Available from: https://openreview.net/forum?id=sY5N0zY5Od
4. Anthropic. Introducing the Model Context Protocol [Internet]. Anthropic; 2024 Nov [cited 2025 Jun 3]. Available from: https://www.anthropic.com/news/model-context-protocol
5. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al. ReAct: synergizing reasoning and acting in language models. In: Proceedings of the 11th International Conference on Learning Representations (ICLR 2023); 2023 May 1–5; Kigali, Rwanda. OpenReview.net; 2023. Available from: https://openreview.net/forum?id=WE_vluYUL-X
6. Shinn N, Cassano F, Berman E, Gopinath A, Narasimhan K, Yao S. A. Reflexion: an autonomous agent with dynamic memory and self-reflection. arXiv:230311366 [cs] [Internet]. 2023 Mar 20; Available from: https://arxiv.org/abs/2303.11366
7. LangChain. LangGraph [Internet]. www.langchain.com. Available from: https://www.langchain.com/langgraph
8. Mintlify. CrewAI [Internet]. Crewai.com. CrewAI; 2024 [cited 2026 Jun 3]. Available from: https://docs.crewai.com/concepts/memory
9. OpenAI Agents SDK [Internet]. Github.io. 2025. Available from: https://openai.github.io/openai-agents-python
10. Agent Development Kit [Internet]. Google Cloud Documentation. 2026. Available from: https://docs.cloud.google.com/gemini-enterprise-agent-platform/build/adk
11. LastMile AI. mcp-agent: build powerful agents using Model Context Protocol [Internet]. GitHub; 2025 [cited 2025 Jun 3]. Available from: https://github.com/lastmile-ai/mcp-agent
12. deepset. Haystack: the open-source framework for building production-ready LLM applications [Internet]. GitHub; 2025 [cited 2026 Jun 3]. Available from: https://github.com/deepset-ai/haystack
13. LlamaIndex. Agentic RAG with LlamaIndex [Internet]. LlamaIndex Blog; 2024. Available from: https://www.llamaindex.ai/blog/agentic-rag-with-llamaindex-2721b8a49ff6
14. Microsoft. Microsoft Agent Framework overview [Internet]. Microsoft Learn; 2026 [cited 2026 Jun 3]. Available from: https://learn.microsoft.com/en-us/agent-framework/overview
15. Meta AI. Model cards and prompt formats: Llama 4 [Internet]. Meta AI; [cited 2026 Jun 3]. Available from: https://www.llama.com/docs/model-cards-and-prompt-formats/llama4/
16. Xu J, Li Z, Chen W, Wang Q, Gao X, Cai Q, et al. On-device language models: a comprehensive review. arXiv [Internet]. 2024 Aug. doi:10.48550/arXiv.2409.00088
17. Artificial Analysis. Independent analysis of AI models and API providers [Internet]. Artificial Analysis; 2026 [cited 2026 Jun 3]. Available from: https://artificialanalysis.ai
18. Grattafiori A, Dubey A, Jauhri A, Pandey A, Kadian A, Al-Dahle A, et al. The Llama 3 herd of models. arXiv [Internet]. 2024 Jul. Report No.: arXiv:2407.21783. doi:10.48550/arXiv.2407.21783
19. Model Context Protocol. MCP specification transports [Internet]. Model Context Protocol; 2025 Mar [cited 2025 Jun 3]. Available from: https://modelcontextprotocol.io/specification/2025-11-25/basic/transports
20. Patil SG, Mao H, Yan F, Ji CC, Suresh V, Stoica I, et al. The Berkeley Function Calling Leaderboard (BFCL): from tool use to agentic evaluation of large language models. In: Proceedings of the 42nd International Conference on Machine Learning (ICML 2025); 2025; Vancouver, Canada. PMLR; 2025. vol. 267. p. 48371–48392. Available from: https://proceedings.mlr.press/v267/patil25a.html
21. Abou Ali M, Dornaika F. Agentic AI: a comprehensive survey of architectures, applications, and future directions. arXiv [Internet]. 2025 Oct [cited 2025 Jun 3]. Report No.: arXiv:2510.25445. doi:10.48550/arXiv.2510.25445