Understanding Multi-Agent LLM Frameworks: A Unified Benchmark and Experimental Analysis

Abdelghny Orogat
Concordia University
Canada

Ana Rostam
Concordia University
Canada

Essam Mansour
Concordia University
Canada

arXiv:submit/7225627 [cs.AI] 3 Feb 2026

ABSTRACT

Multi-agent LLM frameworks are widely used to accelerate the development of agent systems powered by large language models (LLMs). These frameworks impose distinct architectural structures that govern how agents interact, store information, and coordinate tasks. However, their impact on system performance remains poorly understood. This gap is critical, as architectural choices alone can induce order-of-magnitude differences in latency and throughput, as well as substantial variation in accuracy and scalability. Addressing this challenge requires (i) jointly evaluating multiple capabilities, such as orchestration overhead, memory behavior, planning, specialization, and coordination, and (ii) conducting these evaluations under controlled, framework-level conditions to isolate architectural effects. Existing benchmarks focus on individual capabilities and lack standardized framework-level evaluation. We address these limitations by (i) introducing an architectural taxonomy for systematically comparing multi-agent LLM frameworks along fundamental dimensions, and (ii) developing MAFBench, a unified evaluation suite that integrates existing benchmarks under a standardized execution pipeline. Using MAFBench, we conduct a controlled empirical study across several widely used frameworks. Our results show that **framework-level design choices alone can increase latency by over** 100×**, reduce planning accuracy by up to** 30%**, and lower coordination success from above** 90% **to below** 30%**.** Finally, we translate our findings into concrete architectural design principles and framework-selection guidance, and outline promising future research directions.

KEYWORDS

Multi-agent systems, LLM agents, Multi-agent LLM frameworks, agentic AI systems, Agent orchestration, Agent memory, Agent specialization, Agent planning, Agent coordination, Architectural taxonomy, Benchmarking multi-agent systems, LLM system design

Key Contributions

- Architectural taxonomy for multi-agent LLM frameworks
- **MAFBench**: unified benchmark across agent memory, planning, specialization, and multi-agent coordination
- First controlled empirical comparison of design choices
- Actionable design principles for scalable agent systems

Code: https://github.com/CoDS-GCS/MAFBench

1 INTRODUCTION

Multi-agent systems built on large language models (LLMs), often referred to as LLM-based agents or agentic AI systems, [5, 21, 41, 48, 57] have emerged as a widely adopted abstraction for constructing complex intelligent applications [15, 36, 41, 48]. Consequently, a growing ecosystem of multi-agent LLM frameworks has emerged to support the rapid development of these systems [20, 23, 38, 40, 50, 53]. These frameworks are software systems that define architectural abstractions for orchestrating LLM-driven agents, managing memory and tool access, and governing task execution and coordination. Despite their widespread adoption, the system-level consequences of framework-level architectural design choices remain poorly understood. As shown by our study in Table 1, architectural choices across orchestration, memory, planning, specialization, and coordination can drive large variation in latency, throughput, accuracy, and task success under identical models and tasks. These results indicate that system behavior in multi-agent settings is governed primarily by framework architecture rather than LLM quality alone. Table 2 summarizes representative and widely used multi-agent LLM frameworks across different architectural paradigms.

Table 1: Performance ranges observed under framework-level design choices. Best and Worst reflect architectural options per dimension, enabled by framework support. The table shows how far outcomes can diverge for the same task and LLM model.

Dimension	Metric	Best	Worst
Orchestration	Latency (× direct LLM) Throughput (req/s)	1.3× 8.9	117× < 0.01
Memory	Overall memory score	23.8%	6.1%
Planning	Accuracy change Runtime multiplier	+15% 1.2×	−30% 30×
Specialization	ΔF1 score	+58	≈ 0
Coordination	Task success (large n)	> 90%	< 30%

Prior work has examined individual aspects of multi-agent intelligence, including tool use [14, 32, 35, 45], memory and retrieval [18, 19, 30, 56, 59], and collaborative reasoning [13]. These benchmarks are designed to isolate and measure *single capabilities*, and they do so effectively. However, none is intended to characterize the behavior of multi-agent frameworks, which integrate multiple capabilities and impose architectural constraints on how those capabilities interact. In practice, frameworks differ less in the capabilities they expose than in how those capabilities are orchestrated, constrained, and executed through framework-level design decisions. As a result, execution overhead, scalability, and failure modes are shaped by interactions among capabilities within a shared framework structure rather than by any capability in isolation. Existing benchmarks vary widely in format, assumptions, and evaluation protocols. As a result, applying them uniformly across heterogeneous frameworks is challenging and often requires substantial re-engineering. Consequently, meaningful systematic comparison requires integrating complementary benchmarks to explain why systems built on the same models can exhibit markedly different performance and cost under controlled, large-scale deployments.

Table 2: GitHub statistics for major multi-agent frameworks.

Category	Framework	Stars	Forks	Contributors
Graph-based	LangGraph [23]	18.9k	3.3k	249
	n8n [34]	140k	44k	400+
	Langflow [24]	19.6k	2.1k	120
Role-based	CrewAI [20]	38.4k	5.1k	274
	AutoGen [53]	50.0k	7.7k	558
	OpenAI SDK [38]	14.9k	2.5k	166
	Agno [3]	4.2k	310	40
	OpenAgents [54]	1.1k	143	5
GABM-style	DeepMind Concordia [50]	1.0k	218	<50

To address this gap, we introduce an architectural taxonomy that systematically characterizes multi-agent LLM frameworks along fundamental dimensions, including control flow, agent abstraction, communication, memory, and execution semantics. This taxonomy serves as the conceptual backbone of our analysis, enabling systematic comparison across systems. Building on this taxonomy, we introduce **MAFBench**[1], a standardized evaluation suite that integrates established, domain-relevant benchmarks within a unified execution pipeline. Rather than proposing new benchmarks, MAFBench aligns execution and measurement protocols to isolate the impact of framework-level design choices, such as orchestration, memory abstraction, and communication structure, across both single-agent and multi-agent settings. We fix the underlying LLM to control for model effects and attribute observed differences to the framework architecture. Planning, however, is evaluated across multiple LLMs, as its behavior is inherently model-dependent. This methodology enables controlled, framework-level comparison while preserving the semantics of each benchmark.

Contributions. Our primary contribution is a unified methodology for understanding, evaluating, and guiding the design of multi-agent LLM systems. Our main technical contributions are:

- An architectural taxonomy for systematically comparing multi-agent LLM frameworks by decomposing them along fundamental architectural dimensions. The taxonomy identifies orthogonal dimensions spanning control flow, agent abstraction, communication, memory, and execution semantics.
- **MAFBench**, a standardized evaluation suite that integrates existing benchmarks within a unified execution and scoring pipeline for framework-level comparison.
- A comprehensive empirical study across representative multi-agent LLM frameworks using MAFBench, demonstrating that framework-level architectural decisions, rather than model quality alone, govern performance, scalability, and cost under controlled execution settings.
- Concrete architectural design principles that translate our empirical findings into actionable guidance for developing multi-agent LLM systems, and identify current gaps and promising future research directions.

The remainder of the paper is organized as follows. Section 2 presents the proposed architectural taxonomy, and Section 3 describes MAFBench and its unified evaluation pipeline. Section 4 reports experimental results, followed by empirically grounded architectural design principles in Section 5. Section 6 discusses future directions motivated by observed system-level limitations, and Section 7 concludes the paper.

[1] https://github.com/CoDS-GCS/MAFBench

2 ARCHITECTURAL TAXONOMY

This section introduces our architectural taxonomy for multi-agent LLM frameworks. Architectural design choices shape control flow, agent abstraction, communication, memory, and execution semantics, with first-order impact on performance, robustness, and scalability. The taxonomy enables systematic, framework-level comparison independent of specific models or tasks. We begin with formal definitions of agents and multi-agent frameworks.

Definition 2.1 (Agent). An agent is an autonomous computational entity $a = (\mathcal{R}, \mathcal{Y}, \mathcal{P}, \mathcal{S}, \mathcal{T}, f)$, where $\mathcal{R}$ denotes specialization or role context, $\mathcal{Y}$ denotes objectives, $\mathcal{P}$ specifies planning mechanisms, $\mathcal{S}$ denotes storage and knowledge resources, $\mathcal{T}$ denotes accessible tools or actions, and f is a reasoning function that maps observations and stored state to actions. We focus on LLM-based agents, where f is instantiated by an LLM and other capabilities may be implemented at the model or framework level.

Definition 2.2 (Multi-Agent LLM Framework). A multi-agent LLM framework is an architectural system $\mathcal{F} = (\{a_i\}_{i=1}^{n}, \mathcal{O}, \mathcal{C}, \mathcal{E})$, where $\{a_i\}$ is a set of LLM-based agents. $\mathcal{O}$ specifies orchestration logic and control flow for agent execution. $\mathcal{C}$ defines the communication structure and connectivity among agents. $\mathcal{E}$ represents the shared environment with state and transition rules that mediate interactions.

Our taxonomy, derived from Definitions 2.1–2.2 and shown in Table 3, provides a unified architectural view of multi-agent LLM frameworks and is instantiated on the nine representative frameworks summarized in Table 2. It organizes framework design choices along two complementary axes: architectural paradigms (graph-based, role-based, and GABM), which reflect distinct execution models (Section 2.1), and architectural dimensions capturing single-agent characteristics, multi-agent interaction, and environment representation (Section 2.2). Together, these dimensions provide a common vocabulary for distinguishing design-fixed and runtime-emergent decisions.

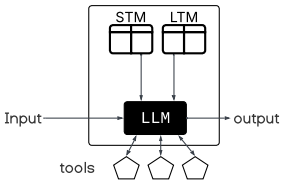
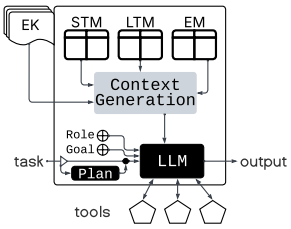
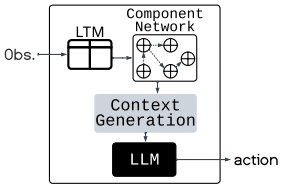
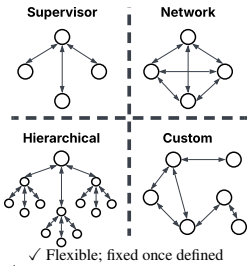
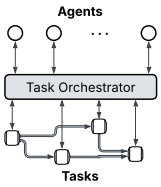
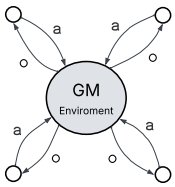
2.1 Architectural Paradigms

This subsection classifies the representative multi-agent LLM frameworks in Table 2 into a small set of recurring *architectural paradigms*. Each paradigm corresponds to a coherent class of framework-level design commitments that jointly fix orchestration $\mathcal{O}$, communication structure $\mathcal{C}$, agent abstraction, and environment coupling $\mathcal{E}$. These commitments constrain runtime coordination and execution, leading to distinct execution semantics, scalability, and performance trade-offs. We identify three such paradigms, graph-based, role-based, and Generative Agent-Based Modeling (GABM).

2.1.1 Graph-Based Frameworks. Graph-based frameworks encode orchestration as an explicit workflow modeled in a directed graph, where nodes represent agents or computational modules and edges define permissible control or data flows [23, 24, 34]. Execution strictly follows this predefined structure, emphasizing deterministic control flow and transparent execution with coordination determined at design time and enforced at runtime.

Definition 2.3 (Graph-Based Framework). A graph-based framework instantiates $\mathcal{O}$ as a directed graph $\mathcal{G} = (V, E)$, where $V = \{a_1, \ldots, a_n\}$ denotes agents or components and $E \subseteq V \times V$ defines allowable control or data flows. Communication $\mathcal{C}$ is restricted to information propagation along edges in $\mathcal{G}$, and no explicit environment state $\mathcal{E}$ is maintained by the framework.

Table 3: Our taxonomy of multi-agent LLM frameworks, organizing frameworks by architectural paradigms and design dimensions.

Feature	Graph-Based (LangGraph)	Role-Based (CrewAI)	GABM (Concordia)
1. Single-Agent Characteristics			
Main Purpose	Deterministic workflow orchestration via explicit execution graphs; agent-based	Task-driven coordination via role-specialized agents and delegation; task-based	Simulation of emergent behavior through environment-mediated agent interaction
Agent Architecture			
Behavioral Specification			
Role	✗ Implicit; No first-class role abstraction	✓ Explicit; First-class role abstraction	✓ Optional; Provided class targets social experiments
Goal	✗ Determined by graph; No first-class goal object	✓ First-class object; fixed; one per agent	✗ Environment-driven; No first-class goal object
Planning	✗ Embedded in graph; LLM planning requires coding	✓ Pre-task or manager-worker planning	✓ Optional; Provided class targets social experiments
Storage			
Long-Term Memory (LTM)	✓ Retrieval-based semantic store	✓ Retrieval-based semantic store	✓ LLM-queried textual memory
Short-Term Memory (STM)	✓Conversation or state accumulation within execution	✓ RAG-based working context over recent interactions	✗ Not natively supported
Entity Memory (EM)	✗ Not natively supported	✓ Optional; structured entity facts via retrieval	✗ Not natively supported
Working Memory (WM)	✗ State variables must be managed by developer	✗ Implicit via task context; not explicitly modeled	✓ Derived from LTM; update rules by developer
External Knowledge (EK)	✗ Not natively supported	✓ Explicitly attachable (files, strings)	✗ Not natively supported
Tool Execution	✓ Agent-bound tools or explicit graph tool nodes	✓ Agent-bound tools	✓ Tools executed by environment, not by agents
2. Multi-Agent Characteristics			
Network Topology			
Communication Pattern	✓ Flexible; fixed once defined ✓ Shared state along predefined edges	✓ Task hierarchy or sequence ✓ Task-mediated; limited peer querying	✓ Fixed star topology; GM at center ✗ Only via GM; no peer-to-peer
Collaboration	✗ None; procedural coordination only	✓ Partial; delegation-based coordination only	✗ None; no explicit collaboration mechanisms
3. Environment			
World State and Grounded Variables	✗ None; execution context only	✗ None; task-centric context only	✓ Explicit world state with grounded variables

2.1.2 Role-Based Frameworks. Role-based frameworks organize multi-agent systems around textual role specifications that condition agent objectives, tool access, and interaction behavior [3, 20, 38, 53, 54]. Rather than prescribing fixed workflows, coordination emerges through role-conditioned reasoning, task delegation, and structured message exchange across agents. This paradigm offers more flexibility in control flow while still imposing structured interaction patterns such as hierarchical or sequential task organizations.

Definition 2.4 (Role-Based Framework). A role-based framework instantiates a set of agents $\{a_i\}_{i=1}^{n}$ together with a role space D and a mapping $R : \{a_i\} \to D$ assigning each agent a role. Orchestration O and communication C are governed by role-conditioned interactions, and no explicit environment state $\mathcal{E}$ is maintained.

2.1.3 GABM Frameworks. GABM frameworks treat multi-agent systems as simulations in which system behavior emerges from repeated agent–environment interactions rather than explicit orchestration between agents. Coordination is mediated indirectly through shared world state, where agents observe, act, and influence future system evolution. A representative system is *Concordia* [50], which implements this paradigm through an explicit environment model.

Definition 2.5 (Generative Agent-Based Modeling Framework). A GABM framework instantiates agents $\{a_i\}_{i=1}^{n}$ interacting with an environment state $\mathcal{E}$ governed by a transition function $T : \mathcal{E} \times \{a_i\} \to \mathcal{E}$. Orchestration O arises from environment-mediated interaction, and direct inter-agent communication C is not exposed.

2.2 Architectural Design Decisions

We now examine the key design decisions that differentiate frameworks across architectural paradigms. Each row of Table 3 captures a core dimension along which frameworks trade off agent behavior, information handling, interaction structure, and execution dynamics.

2.2.1 Single-Agent Characteristics. We begin by analyzing architectural decisions that govern how individual agents are modeled.

- Behavioral Specification. This architectural dimension governs how agent behavior is specified at runtime, ranging from procedurally fixed execution logic to declarative role conditioning and environment-mediated interaction [7, 44]. Graph-based frameworks encode roles, goals, and plans procedurally within static execution graphs, where behavior is determined by workflow structure and control-flow logic rather than first-class abstractions [23, 24, 34, 46]. Role-based frameworks elevate roles and goals to explicit textual specifications that condition LLM reasoning, with planning realized through natural-language plan generation, manager–worker coordination, or iterative refinement instead of rigid action sequences [20, 38, 53, 54]. In contrast, GABM frameworks treat roles, goals, and plans as optional, often for social experiments, allowing behavior to emerge from repeated perception–action loops mediated by an evolving environment state [50]. Together, these designs trade deterministic control for increasing behavioral flexibility and emergence across paradigms.

- Storage Architecture. This dimension governs how agent knowledge and internal state are represented and updated during execution. We consider memory abstractions exposed by existing frameworks, including long-term memory (LTM), short-term memory (STM), entity memory (EM), working memory (WM), and external knowledge (EK). Graph-based frameworks primarily expose retrieval-based long-term memory with bounded short-term execution state, while entity memory, working memory, and external knowledge are managed procedurally by developers [23]. Role-based frameworks follow a similar retrieval-centric design but may additionally support structured entity memory via embedding-based retrieval and explicit attachment of external knowledge, such as documents or files [20, 38, 53]. In contrast, GABM frameworks centralize memory at the environment level, maintaining long-term textual memory accessed by the LLM and updating state via developer-defined transition functions rather than short-term context accumulation [50]. Together, these designs reflect trade-offs between scalable retrieval, rapid in-session adaptation, and persistent state evolution across paradigms.

- Tool Execution Model. This architectural dimension determines where the authority for external action execution resides and how tool invocation is governed at runtime. Graph-based frameworks integrate tools procedurally in two ways: as explicit workflow nodes with execution fixed by control-flow logic [23, 24, 34], or as agent-callable functions whose invocation remains constrained by developer-defined graph structure [23, 46]. Role-based frameworks bind tools to agent roles and task context, enabling dynamic selection during reasoning while constraining access through orchestration semantics [20, 38, 53, 54]. In contrast, GABM frameworks remove direct tool access from agents and route all external actions through an environment controller that interprets agent intentions and updates global state [50]. Together, these designs trade execution predictability for adaptive autonomy. Procedural invocation offers strict control and stable costs. Role-conditioned access enables flexible problem solving with weaker guarantees, while environment-mediated execution supports persistent state evolution while limiting direct agent control.

2.2.2 Multi-Agent Characteristics. Multi-agent LLM frameworks structure agent interaction through connectivity, communication, and collaboration mechanisms that shape information flow, coordination, and collective behavior at runtime. As shown in Table 3, differences in topology and communication models strongly constrain coordination, scalability, and joint reasoning.

- Network Topology. This architectural dimension governs how agents are connected and how interaction paths are structured within a framework, thereby shaping which coordination patterns are feasible at runtime. Graph-based frameworks provide flexibility at design time by allowing developers to define explicit topologies encoded as directed graphs. These graphs can represent hierarchical, peer-to-peer, or cyclic structures [23, 46], but the resulting topology remains fixed during execution. Role-based frameworks induce topology implicitly through task decomposition and role assignments, typically resulting in sequential pipelines or manager–worker hierarchies with limited lateral interaction [20, 38, 53]. In contrast, GABM frameworks centralize interaction through an environment controller (Game Master), enforcing a star topology where agents interact only with the environment [50].

- Communication Pattern. This architectural dimension governs how information is exchanged and propagated among agents during execution. In graph-based frameworks, communication is realized through deterministic state propagation along predefined edges in the execution graph, rather than through explicit message passing between agents [23, 24, 34]. Role-based frameworks route communication primarily through task assignment, intermediate result passing, and reporting mechanisms embedded in hierarchical or sequential coordination structures [20, 38, 53]. GABM frameworks eliminate direct inter-agent messaging altogether, mediating all interaction through a shared environment that aggregates agent actions and produces new observations for subsequent reasoning [50].

- Collaboration. This architectural dimension captures whether frameworks provide explicit mechanisms for agents to jointly coordinate decisions beyond procedural execution. Graph-based frameworks do not support first-class collaboration, relying entirely on procedural coordination enforced by predefined workflows. Role-based frameworks enable limited forms of collaboration through delegation, task handoff, and manager–worker coordination among role-specialized agents, but they generally lack support for multi-turn negotiation or collective decision-making protocols [20, 38, 53]. GABM frameworks similarly expose no explicit collaboration mechanisms, with collective behavior emerging indirectly from shared environment dynamics rather than from intentional or negotiated coordination among agents [13, 50].

2.2.3 Environment. This architectural dimension determines if system execution is mediated through an explicit shared world state or through agent-local and developer-managed state. Graph-based and role-based frameworks do not maintain a first-class environment, relying instead on workflows and task context effects that are not reflected in a persistent global state. In contrast, GABM frameworks centralize execution around an explicit environment that tracks world state and applies agent actions over time [50].

2.2.4 Synthesis of Architectural Trade-offs. Across the architectural dimensions in Table 3, framework design trades execution control and efficiency for behavioral flexibility and persistent state modeling. Graph-based architectures favor deterministic control flow and low-overhead coordination, supporting predictable and scalable pipelines. Role-based architectures relax structural constraints to enable adaptive reasoning and specialization through declarative roles and task decomposition. GABM architectures shift coordination and memory into a shared environment, supporting emergent behavior and simulation-based analysis at the cost of higher execution overhead and reduced direct control. Together, these architectures represent distinct trade-offs among predictability, adaptability, and persistent state coherence in multi-agent systems.

Architectural behavior reflects both framework abstractions and developer-defined execution structures, with many mechanisms shared across frameworks. For example, planning is explicit in role-based frameworks but embedded procedurally in graph workflows, while storage often relies on similar retrieval-based techniques across paradigms. Coordination is largely constrained by paradigm-level communication structures rather than specific APIs. Accordingly, our evaluation (Section 4) isolates individual architectural choices, revealing how design decisions affect overhead, reasoning effectiveness, memory behavior, and coordination outcomes.

3 MAFBench: A UNIFIED BENCHMARK

This section introduces **MAFBench**, a unified benchmark for evaluating multi-agent frameworks across memory, planning, specialization, tool use, and coordination. It standardizes execution, logging, and cost control to ensure identical conditions across frameworks while isolating architectural effects.

3.1 Memory Benchmarks

Prior memory benchmarks for LLM-based agents evaluate isolated capabilities, such as long-context retrieval [18, 52, 58], dialogue and state tracking [30, 31], incremental preference learning [6, 16, 25], and controlled knowledge revision [59]. These benchmarks do not capture how agents retain, integrate, generalize, and revise information across multi-turn interactions. We therefore adopt *MemoryAgentBench* [19], which unifies these dimensions into four core competencies: *Accurate Retrieval (AR)*, *Test-Time Learning (TTL)*, *Long-Range Understanding (LRU)*, and *Selective Forgetting (SF)*.[2]

Benchmark Structure and Unified Interface. MemoryAgentBench organizes evaluation into four competency-aligned splits composed of adapted subtasks (Table 4). *AR* measures factual recall and multi-hop reasoning over long contexts; *TTL* evaluates in-session learning of concepts and preferences; *LRU* targets long-range abstraction and cross-document reasoning; and *SF* probes controlled memory revision through counterfactual updates [19].

Evaluation Setup. MAFBench integrates MemoryAgentBench into its unified execution pipeline to enable fair and scalable cross-framework comparison. We enforce a standardized agent interface for session-level execution and centralize configuration of model parameters, session limits, batching, and scoring. We also replace string-based metrics with LLM-based semantic evaluation to handle diverse answer formats. Results are logged and aggregated using a shared schema that captures accuracy, runtime, and token usage for reproducibility. To support large-scale long-context evaluation, we introduce transparent backend routing that redirects compatible API calls to alternative providers. In our experiments, memory-intensive workloads are routed to Groq-hosted models [2], enabling lower-cost evaluation without modifying framework implementations.

3.2 Planning Benchmarks

Planning in LLM-based agents refers to generating intermediate solution steps before producing an answer. While some frameworks, such as CrewAI [20], expose explicit planning, its evaluation remains fragmented across isolated reasoning benchmarks. We therefore adopt *GSM8K* [8], *CommonsenseQA (CSQA)* [47], and *MATH* [17] (Table 5), which cover numerical, commonsense, and symbolic reasoning tasks where multi-step planning influences outcomes.[3]

Benchmark Structure and Planning Interfaces. Each benchmark is executed under three controlled planning interfaces. *NoPlan* passes tasks directly to the LLM without intermediate reasoning. *Crew-Plan* follows CrewAI's schema-constrained two-stage interface [20], in which the LLM first produces a structured plan. *Direct-LLM-Plan* allows unconstrained natural-language plan generation injected into the context before answer generation. This setup isolates the effect of planning interface constraints.

[2]https://huggingface.co/datasets/ai-hyz/MemoryAgentBench

[3]https://huggingface.co/datasets/openai/gsm8k (test only); https://huggingface.co/datasets/tau/commonsense_qa (validation only); https://github.com/hendrycks/math

Table 4: MemoryAgentBench split statistics showing session counts, question distributions, and context lengths (in thousands of words) for each memory competency (AR, TTL, LRU, SF).

			Questions				Context (k words)			
Split	Sessions	Subtasks	Min	Max	Avg	Total	Min	Max	Avg	Total
AR	22	4	60	100	90.9	2,000	50	560	206	4,551
TTL	6	2	100	200	116.7	700	69	1,040	236	1,417
LRU	110	2	1	10	1.6	171	48	560	129	14,281
SF	8	2	100	100	100.0	800	4	183	63	511

Table 5: Dataset statistics for planning benchmarks, reporting question and gold answer length distributions.

		Question Length (words)				Gold Answer Length (words)			
Benchmark	**#Q**	**Min**	**Max**	**Avg**	**Total**	**Min**	**Max**	**Avg**	**Total**
GSM8K [8]	1319	15	164	46.3	61,005	1	1	1.0	1,319
CSQA [47]	1221	19	69	31.6	38,615	1	1	1.0	1,221
MATH [17]	5000	2	252	30.7	153,261	1	19	1.3	6,345

Evaluation Setup. MAFBench provides a unified planning evaluation pipeline that treats planning as a configurable execution stage while keeping models, prompts, and scoring fixed. The pipeline standardizes dataset loading for GSM8K, CSQA, and MATH, and supports complexity-preserving subsampling for large benchmarks to control question count while maintaining difficulty. In our experiments, we use a 100-problem MATH subset. Formatting failures are tracked separately from reasoning errors. Model selection, planning modes, and run budgets are centrally configured to ensure reproducibility and cost control. All three modes are evaluated across multiple LLM backends, enabling systematic analysis of planning benefits and interface effects.

3.3 Specialization Benchmarks

Specialization in LLM-based agents refers to activating domain-relevant reasoning through textual conditioning rather than external knowledge acquisition. We examine whether role descriptions, explicit planning, or expert methodological guidance better structure the domain-specific reasoning encoded in the model. Using machine learning tasks from CatDB [12], we measure behavioral differences under controlled conditions with fixed data and models.

Benchmark Structure and Conditioning Variants. The benchmark comprises five public machine learning datasets spanning regression, binary, and multiclass classification (Table 6). Each dataset is evaluated under three conditioning strategies. *Role-based prompting* assigns a professional identity in the task description. *Planning-based conditioning* inserts an intermediate step in which the LLM generates a high-level solution plan before code generation. *Expert-guided conditioning* injects methodological instructions reflecting data-science workflows. These variants isolate the effects of identity framing, reasoning structure, and procedural guidance.

Evaluation Setup. MAFBench provides a unified specialization evaluation pipeline that isolates the architectural impact of behavioral conditioning. Task structure, data loading, model execution, and metric computation are standardized to ensure consistent semantics. Conditioning strategy is the only experimental variable; models, inputs, and prompts remain fixed. Model parameters and execution limits are centrally configured for reproducibility and cost control. Results are logged using a shared schema capturing performance and runtime across runs.

Table 6: Overview of specialization benchmark datasets, reporting task types, prediction targets, and feature dimensionality.

Dataset	Task Type	Target Variable	# Features
Utility	Regression	CSRI	12
WiFi	Binary Classification	TechCenter	9
EU-IT	Multiclass Classification	Position	23
Yelp	Multiclass Classification	stars	184
Volkert	Multiclass Classification	class	181

3.4 Tool Use Benchmarks and Limitations

Tool use enables LLM-based agents to interact with external APIs and services beyond language generation. Recent benchmarks, such as ToolBench and its deterministic variant StableToolBench [14, 42, 43] evaluate an agent's ability to select and invoke appropriate tools from large, heterogeneous registries. These benchmarks provide structured environments with diverse tools and realistic tasks, making them suitable for analyzing tool-use behavior.

Benchmark Structure. Figure 1 illustrates how StableToolBench evaluates tool use across registries spanning multiple domains and interface complexities, including both single-endpoint tools and multi-endpoint APIs [14]. The benchmark stresses correct tool selection, parameter grounding, and multi-step interaction with closely related endpoints. In MAFBench, StableToolBench serves as the reference environment for characterizing tool-use requirements.

Evaluation Setup. MAFBench integrates StableToolBench through a unified tool-use evaluation layer that preserves the original queries, server, and scoring logic while standardizing execution. Benchmark runs are centralized through a shared runner with consistent logging and result aggregation. To ensure fair comparison, we introduce a pre-execution tool-selection step that selects a bounded subset of relevant tools per query before framework binding, ensuring identical tool sets across runs. We also enforce explicit tool budget limits aligned with model constraints and record correctness, runtime, and token usage using a shared result schema. We do not report quantitative tool-use results because current frameworks delegate tool invocation entirely to the LLM via native function-calling interfaces, without architectural control over tool selection or execution. As a result, behavior is driven primarily by model choice and provider-imposed limits on exposed tools (e.g., 128 in OpenAI APIs), rather than by framework design. Tool-use evaluation is therefore deferred until frameworks introduce explicit orchestration mechanisms.

3.5 Coordination and Scaling Benchmarks

Coordination in multi-agent systems involves information exchange, conflict resolution, and convergence under constrained communication. Existing frameworks support agent execution and task decomposition but lack native multi-round peer-to-peer communication over arbitrary topologies. As a result, coordination is largely dictated by interaction topology rather than explicit framework mechanisms. To study topology-aware coordination and scalability, we adopt AGENTSNET [13], which analyzes emergent collective behavior under constrained message passing.[4] AGENTSNET evaluates *local consistency* and *global alignment* under limited connectivity.

Benchmark Structure and Conditioning Variants. AGENTSNET includes five coordination tasks that capture distinct coordination behaviors: *Coloring* (local conflict resolution), *Matching* (bilateral agreement), *VertexCover* (global resource minimization), *Leader-Election* (symmetry breaking), and *Consensus* (long-range information propagation) [13]. Agents interact only along edges of predefined graph families, including *Small-World* [49], *Scale-Free* [29], and *Delaunay* [26]. This enables systematic analysis of how clustering, path length, and connectivity heterogeneity affect coordination dynamics. To reflect interaction constraints in role-based and GABM frameworks, we introduce a graph-rewriting operator that derives sequential, hierarchical, and star-shaped interaction structures from shared base graphs. In the star configuration, interaction is mediated by a centralized environment component that serves as a communication hub for all agents.

Evaluation Setup. Beyond adopting AGENTSNET, MAFBench provides a unified coordination evaluation infrastructure that standardizes topology-controlled experiments across agent systems. We implement a topology rewriting engine that transforms base communication graphs into sequential, hierarchical, and centralized structures while preserving agent sets. Dedicated runners then execute identical coordination tasks under each orchestration pattern using consistent prompts, models, and task semantics. We also introduce a visualization tool to render final agent states and communication outcomes for qualitative analysis. Finally, we automate large-scale sweeps across tasks, graph families, and agent counts, and standardize metrics to enable reproducible scalability analysis.

[4]Dataset: https://huggingface.co/datasets/disco-eth/AgentsNet; Code: https://github.com/floriangroetschla/AgentsNet

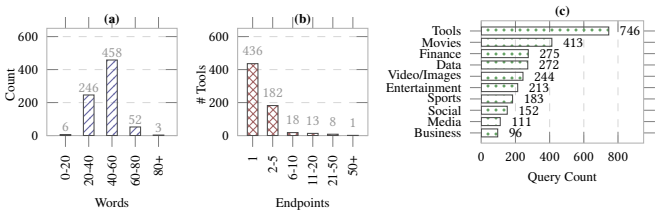


Figure 1: Characteristics of StableToolBench, showing the distributions of (a) query length, (b) number of APIs per tool, and (c) query counts across tool categories.

Table 7: Benchmarks in MAFBench and the architectural dimensions they isolate. Metrics follow original benchmark definitions (Acc.=Accuracy, F1=F1-score, R@5=Recall@5).

Architectural Dimension	Benchmark	Metrics
Orchestration overhead	Trivial query	Latency, throughput, tokens
Memory architecture	MemoryAgentBench [19]	Acc./F1/R@5
Planning interface	GSM8K [8], CSQA [47], MATH [17]	Accuracy, failures, runtime
Specialization conditioning	CatDB tasks [12]	Precision, recall, F1
Coordination topology	AGENTSNET [13]	Success, rounds, tokens, time

4 EVALUATION

This section presents controlled experiments that isolate how architectural design choices in multi-agent frameworks affect performance, robustness, cost, and scalability under fixed models, prompts, and data. Each study varies a single framework-level dimension using our MAFBench pipeline while holding execution semantics constant. Table 7 summarizes the benchmarks, isolated architectural dimensions, and measured outcomes, spanning single-agent execution to multi-agent coordination and scaling. MemoryAgentBench aggregates heterogeneous subtask metrics (Accuracy, F1, Recall@5) using category-level means for consistent comparison [19].

4.1 Single-Agent and Overhead Evaluation

We first examine how framework orchestration, memory, and planning affect execution cost and reasoning behavior.

4.1.1 Framework Overhead. We evaluate how orchestration architecture affects baseline execution cost in multi-agent frameworks, independent of memory, planning, tool use, and coordination. Using a unified execution pipeline, we fix the LLM (*gpt-4o-mini*), the query (*"What is 2+2?"*), prompts, and concurrency, and vary only the framework orchestration layer under minimal viable configurations. The comparison includes a raw *Direct LLM* baseline; graph- and role-based frameworks *LangGraph* [23], *OpenAgents* [54], *AutoGen* [53], *CrewAI* [20], *Agno* [3], and the *OpenAI SDK* [38]; and the GABM framework *Concordia* [50]. We run 50 trials and report median (p50) and tail (p95) latency, throughput, and output size.

Figure 2(a,b) shows that Direct LLM calls achieve the lowest latency and highest throughput. AutoGen and OpenAgents closely match LangGraph, indicating minimal overhead for shallow role-based abstractions. LangGraph incurs modest additional cost from graph scheduling and state propagation, while CrewAI, Agno, and the OpenAI SDK exhibit progressively higher latency as orchestration logic increases, with throughput mirroring latency across graph- and role-based frameworks. In contrast, Concordia shows more than an order-of-magnitude increase in median and tail latency and a sharp drop in throughput, reflecting its GABM execution semantics. Output sizes in Figure 2(c,d), shown on a logarithmic scale, remain bounded for graph- and role-based frameworks but grow by several orders of magnitude for Concordia due to narrative transcripts and environment-mediated state updates. Additional Concordia analysis appears in Appendix A. Overall, these results show that orchestration architecture alone governs baseline scalability. Graph- and role-based designs introduce modest but compounding overhead over direct LLM execution, whereas GABM execution incurs orders-of-magnitude higher runtime and output even for trivial tasks, driven by execution semantics rather than task complexity.

4.1.2 Memory Architecture Effects. We evaluate how memory architecture in multi-agent frameworks affects recall, in-session learning, long-range reasoning, knowledge revision, and runtime scalability. All frameworks are accessed through a common interface (Section 3.1) with identical chunking, ingestion order, and metrics, and all experiments use the same backend model (`gpt-oss-20b` on Groq[5]). Observed differences therefore reflect memory execution semantics rather than model capability. Table 8 summarizes the results and reveals three dominant memory architectures: *retrieval-centric* memory with persistent semantic stores, *accumulation-based* memory that replays recent interactions in the prompt, and *hybrid* memory combining retrieval with bounded short-term accumulation. CrewAI is retrieval-centric; Agno and the OpenAI SDK are accumulation-based; and LangGraph adopts a hybrid design. We evaluate LangGraph in retrieval-only and hybrid modes across context windows (W) and vary W for the OpenAI SDK to characterize accumulation scaling. Concordia and OpenAgents are excluded due to narrative simulation semantics and lack of native memory

[5]Although `gpt-oss-20b` supports context windows up to 130K tokens, large accumulated contexts exceed tokens-per-minute limits (e.g., 250K TPM), making extreme accumulation impractical for evaluation.

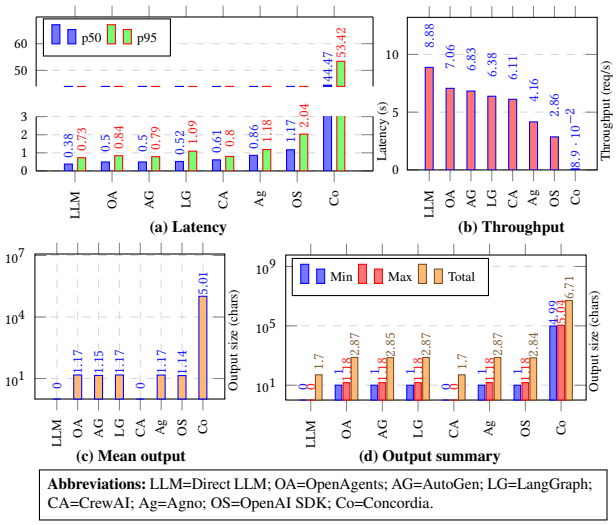


Figure 2: Framework overhead on a trivial task ("What is 2+2?") over 50 trials. Frameworks are ordered by p50 latency.

abstractions. Each framework's memory settings are detailed in Appendix B.

Accuracy and Reasoning Effects. Accurate Retrieval (AR) measures factual recall and multi-hop reasoning over long contexts and is driven primarily by retrieval-based architectures rather than raw context accumulation. As shown in Table 8, hybrid retrieval–accumulation performs best: LangGraph (W=512) achieves an average AR score of *44.9*, exceeding pure retrieval (*33.2*) and accumulation-based designs such as Agno (*19.1*). Retrieval-centric designs remain robust on multi-hop and event-centric queries that require selective evidence access, while bounded accumulation provides limited gains via local coherence. Accumulation-only memory improves initially—peaking at *32.3* for the OpenAI SDK at W=1024—but then degrades as unfiltered context dilutes relevance. Overall, reliable factual recall depends on architectural retrieval mechanisms, not larger context windows. Long-Range Understanding (LRU), which evaluates cross-document abstraction and narrative reasoning, shows a similar pattern. Retrieval-first designs dominate, with retrieval-only LangGraph achieving the highest score (*30.4*) through coherent summarization and multi-document reasoning. Accumulation-based designs approach this level only at very large context windows (OpenAI SDK at W=8192: *27.5*), but with higher cost and instability. Hybrid designs perform worse on LRU as short-term accumulation interferes with retrieved narratives and weakens temporal coherence. These results indicate that exposing contiguous history instead of structured retrieval undermines robustness and efficiency.

In-Session Learning and Knowledge Revision. Test-Time Learning (TTL), which evaluates in-session acquisition of concepts and preferences, shows the opposite trend. Accumulation-based designs improve steadily as interaction history grows, with the OpenAI SDK increasing from *1.2* at W=50 to *20.7* at W=8192. However, hybrid designs consistently outperform pure accumulation: LangGraph (W=1024) achieves the highest TTL score (*28.9*) by anchoring learning to retrieved relevant signals rather than raw prompt growth. Retrieval-only memory performs poorly on TTL, confirming that adaptation requires accumulation but benefits from architectural

Table 8: MemoryAgentBench results for evaluation comparison. Scores are averaged per subtask across sessions. W denotes the LangGraph and OpenAI SDK context window (tokens). Category scores (AR, TTL, LRU, SF) are subtask means, and the Overall score is their average. Bold shaded cells indicate global maxima, and underlined values denote the best result within each memory paradigm.

Agent Framework	Accurate Retrieval (AR)					Test-Time Learning (TTL)			Long-Range Understanding (LRU)			Selective Forgetting (SF)			Overall
	SH-QA	MH-QA	LME(S*)	EventQA	Avg.	MCC	Recom.	Avg.	Summ.	DetQA	Avg.	FC-SH	FC-MH	Avg.	Score
***Retrieval-centric* Memories**															
Crewai	14.0	22.0	3.0	48.7	21.9	5.6	12.2	8.9	0.0	19.7	9.9	26.2	0.8	13.5	13.5
LangGraph (Vector DB only)	14.0	24.0	31.3	63.3	33.2	22.8	0.0	11.4	20.0	40.8	**30.4**	36.0	4.5	**20.2**	**23.8**
***Accumulation-based* Memories**															
Agno	12.0	19.0	2.0	43.2	19.1	0.6	15.1	7.8	0.0	18.3	9.2	27.0	0.8	13.9	12.5
OpenAI SDK ($W = 50$)	16.0	16.0	1.7	0.0	8.4	1.4	0.9	1.2	0.0	0.0	0.0	28.5	1.0	14.8	6.1
OpenAI SDK ($W = 512$)	8.0	11.0	1.0	44.1	16.0	0.6	17.2	8.9	0.0	28.2	14.1	28.7	0.8	14.8	13.4
OpenAI SDK ($W = 1024$)	40.0	33.0	7.7	48.5	32.3	3.6	22.1	12.8	0.0	25.4	12.7	0.0	0.0	0.0	14.5
OpenAI SDK ($W = 2048$)	32.0	32.0	18.3	49.9	33.1	10.0	20.9	15.5	0.0	33.8	16.9	0.2	0.0	0.1	16.4
OpenAI SDK ($W = 4096$)	36.0	35.0	14.0	48.5	33.4	16.2	22.7	19.4	0.0	29.6	14.8	0.5	0.0	0.2	17.0
OpenAI SDK ($W = 8192$)	34.0	34.0	23.0	44.7	33.9	34.0	7.3	20.7	10.0	45.1	27.5	0.8	0.0	0.4	20.6
***Hybrid Retrieval–Accumulation* Memories**															
LangGraph ($W = 50$)	35.0	35.0	41.7	3.7	28.8	38.6	3.9	21.3	0.0	1.4	0.7	0.2	0.2	0.2	12.8
LangGraph ($W = 512$)	36.0	34.0	44.0	65.6	**44.9**	35.6	12.8	24.2	0.0	35.2	17.6	0.0	0.0	0.0	21.7
LangGraph ($W = 1024$)	34.0	35.0	41.3	64.2	43.6	38.0	19.8	**28.9**	0.0	40.8	20.4	0.0	0.2	0.1	23.3
LangGraph ($W = 2048$)	37.0	30.0	39.3	63.7	42.5	34.6	19.4	27.0	0.0	40.8	20.4	0.0	0.5	0.2	22.5
LangGraph ($W = 4096$)	31.0	35.0	41.7	59.6	41.8	32.8	18.7	25.7	10.0	40.8	25.4	0.0	0.2	0.1	23.3
LangGraph ($W = 8192$)	37.0	30.0	39.3	63.7	42.5	34.6	19.4	27.0	0.0	40.8	20.4	0.0	0.5	0.2	22.5

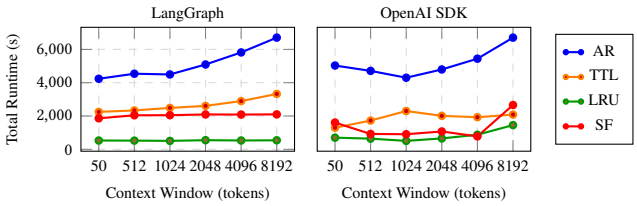


Figure 3: Runtime as a function of context window for LangGraph (left) and OpenAI SDK (right) across the four task.

filtering. Selective Forgetting (SF), which probes controlled revision of outdated knowledge, remains largely unsupported across all frameworks. Retrieval-centric designs partially support single-hop forgetting, with retrieval-only LangGraph reaching an SF average of *20.2*, but multi-hop forgetting remains below *5.0* across all architectures (Table 8). Accumulation-based designs perform worse even at large context sizes, and hybrid designs collapse as outdated information persists simultaneously in retrieval stores and accumulated context. This highlights the lack of explicit memory editing and dependency-aware deletion mechanisms in current frameworks.

Runtime and Scalability Effects. Figure 3 reports runtime as a function of context window for LangGraph (hybrid retrieval–accumulation) and the OpenAI SDK (pure accumulation) across the four MemoryAgentBench competencies. Each task ingests shared context and issues multiple queries, so cost scales with both context size and query count. High-query tasks such as Accurate Retrieval and Test-Time Learning exhibit steep runtime growth under accumulation, where full histories are replayed for each query. In contrast, Long-Range Understanding remains nearly flat for LangGraph due to few queries per session, while Selective Forgetting shows moderate growth. Overall, accumulation drives rapid cost escalation as context grows, whereas retrieval avoids repeated processing. These results demonstrate that scalability is governed by memory architecture rather than by the LLM's nominal context capacity.

Summary. Memory behavior in multi-agent frameworks is governed by architecture, not context length alone. Retrieval enables stable recall and global reasoning, accumulation enables in-session learning but scales poorly, and hybrid designs are effective only under bounded context. The absence of explicit memory editing primitives prevents reliable knowledge revision, motivating future architectural support for safe and controllable long-term memory.

4.1.3 Planning Effects. We evaluate how the *planning interface* in a framework affects reasoning accuracy, robustness, and runtime, separating architectural constraints from the LLM's ability to generate plans. Using the MAFBench planning pipeline (Section 3.2), we fix the benchmarks (GSM8K [8], CSQA [47], MATH [17]), prompts, decoding settings, and scoring. We verify that observed effects are architectural rather than model-specific by running all interfaces across multiple LLM backends. We vary only the framework-level planning interface: *NoPlan* (direct answer), *Crew-Plan* (schema-constrained two-stage planning in CrewAI [20]), and *Direct-LLM-Plan* (free-form plan text injected before execution). For MATH, we evaluate a 100-problem subset preserving the original complexity distribution.

Table 9 shows that schema-constrained planning reduces accuracy across datasets and models, while free-form planning preserves or improves accuracy in most settings. For example, under Crew-Plan, Qwen-7B drops from *13.0%* to *3.4%* on GSM8K and from *69.9%* to *37.6%* on CSQA, and GPT-OSS-20B drops from *80.0%* to *48.0%* on MATH-100. In contrast, Direct-LLM-Plan improves Qwen-7B to *28.7%* on GSM8K and improves Llama-3.1-8B from *65.6%* to *71.9%*, showing that adding a planning stage can help *in practice* when the interface does not impose brittle structure.

Table 10 and Table 11 identify the architectural cause. Crew-Plan introduces large formatting failure rates, where the LLM fails to follow the framework's required output schema in practice. When the schema is violated, the framework cannot parse or execute the generated plan, even if the underlying reasoning is correct, leading to *84.7%* failures on GSM8K and *70.0%* on MATH-100 for Qwen-7B, which dominate the observed accuracy loss. Crew-Plan also imposes high orchestration overhead by adding an extra LLM call together with schema generation, validation, and parsing, increasing runtime by *7.4*× on GSM8K and *18.5*× on CSQA for GPT-OSS-20B, and exceeding *30*× for Llama-3.1-8B on CSQA. In contrast, Direct-LLM-Plan yields *0.0%* formatting failures across models. It adds only the additional cost of generating a plan, producing smaller slowdowns (*1.2*×–*6.6*×) and avoiding any parsing bottleneck.

These results show that planning outcomes in deployed agent systems are driven primarily by *interface design*, not LLM planning ability alone. Planning should therefore be implemented as a permissive stage that tolerates variability in plan text; rigid parsing converts valid reasoning into failures, reduces robustness, and introduces substantial runtime overhead that limits scalability.

Table 9: Accuracy (%) under three planning modes: ✗: No plan, ✓(Crew): schema-constrained plan, ✓(LLM): free-form plan.

LLM	GSM8K [8]			CSQA [47]			MATH-100 [17]		
	✗	✓(crew)	✓(LLM)	✗	✓(crew)	✓(LLM)	✗	✓(crew)	✓(LLM)
Local Models									
GPT-OSS-20B [37]	94.6	91.6 ↓	87.6 ↓	81.4	81.2 ↓	82.1 ↑	80.0	48.0 ↓	76.0 ↓
Phi-4-14B [1]	93.3	86.7 ↓	83.5 ↓	81.7	82.0 ↑	74.6 ↓	72.0	50.0 ↓	69.0 ↓
Llama-3.1-8B [11]	65.6	60.3 ↓	71.9 ↑	70.1	65.4 ↓	65.4 ↓	25.0	24.0 ↓	37.0 ↑
Qwen-7B [4]	13.0	3.4 ↓	28.7 ↑	69.9	37.6 ↓	74.0 ↑	9.0	4.0 ↓	13.0 ↑
DeepSeek-7B [10]	27.0	18.0 ↓	23.5 ↓	46.3	31.2 ↓	46.1 ↓	9.0	3.0 ↓	4.0 ↓
Remote Models									
GPT-4.1 [39]	94.2	80.7 ↓	78.2 ↓	87.1	87.3 ↑	87.0 ↓	86.0	60.0 ↓	84.0 ↓
GPT-4o-Mini [39]	86.5	90.8 ↑	84.2 ↓	81.8	81.3 ↓	83.1 ↑	60.0	53.0 ↓	59.0 ↓

Table 10: Formatting failures under schema-constrained and free-form planning, isolating interface-induced errors.

LLM	GSM8K		CSQA		MATH-100	
	✓(CrewAI)	✓(LLM)	✓(CrewAI)	✓(LLM)	✓(CrewAI)	✓(LLM)
Local Models						
GPT-OSS-20B	2.4%	0.0%	1.3%	0.0%	49.0%	13.0%
Phi-4-14B	6.1%	0.0%	0.5%	0.0%	21.0%	0.0%
Llama-3.1-8B	10.5%	0.0%	4.8%	0.0%	19.0%	0.0%
Qwen-7B	84.7%	0.0%	41.0%	0.0%	70.0%	0.0%
DeepSeek-7B	25.5%	0.0%	31.0%	0.0%	40.0%	0.0%
Remote Models						
GPT-4.1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
GPT-4o-Mini	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table 11: Runtime multiplier (relative to NoPlan) under schema-constrained and free-form planning. Lower is faster.

LLM	GSM8K		CSQA		MATH	
	Δ CrewAI	Δ LLM	Δ CrewAI	Δ LLM	Δ CrewAI	Δ LLM
Local Models						
GPT-OSS-20B	× 7.4	× 2.3	× 18.51	× 1.7	× 11.43	× 1.7
Phi-4-14B	× 3.5	× 1.4	× 4.1	× 1.6	× 3.0	× 1.7
Llama-3.1-8B	× 6.3	× 1.3	× 31.34	× 4.2	× 4.4	× 1.3
Qwen-7B	× 2.2	× 2.5	× 18.16	× 0.9	× 2.9	× 1.8
DeepSeek-7B	× 3.1	× 2.6	× 13.29	× 6.6	× 2.8	× 1.2
Remote Models						
GPT-4.1	× 2.4	× 1.5	× 2.9	× 1.1	× 1.2	× 1.1
GPT-4o-Mini	× 1.5	× 1.4	× 4.8	× 2.3	× 1.8	× 1.4

Table 12: Performance of agents under different conditions (using GPT-4o-mini). Precision (P), Recall (R), and F1 are shown.

Category	Regres.	Binary Class			Multiclass								
Agent Role	Utility	WiFi			EU-IT			Yelp			Volkert		
	MAE	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Role-based prompting													
No Role	0.067	40.2	45.0	37.3	32.0	38.0	35.0	41.9	44.1	41.9	67.3	66.5	65.3
Data Scientist	0.067	40.2	45.0	37.3	32.3	37.9	34.8	41.9	44.1	41.9	67.3	66.5	65.3
Researcher	0.067	40.2	45.0	37.3	32.3	37.9	34.8	42.1	44.0	41.9	67.3	67.0	65.8
Data Analyst	0.067	40.2	45.0	37.3	33.3	37.9	34.8	42.1	44.0	41.9	67.0	67.0	65.0
Engineer	0.067	40.2	45.0	37.3	32.0	38.0	35.0	41.9	44.1	41.9	67.3	66.5	65.3
Planning-based conditioning													
No Role	0.067	40.0	45.0	37.0	32.0	38.0	35.0	42.0	44.1	42.0	67.0	67.0	65.0
Data Scientist	0.067	40.0	45.0	37.0	32.0	38.0	35.0	42.0	44.1	42.0	67.0	67.0	66.0
Researcher	0.067	40.0	45.0	37.0	32.0	38.0	35.0	42.0	44.0	42.0	67.0	67.0	65.0
Data Analyst	0.067	40.0	45.0	37.0	32.0	38.0	35.0	42.0	44.0	42.0	67.0	67.0	65.0
Engineer	0.067	40.0	45.0	37.0	32.0	38.0	35.0	42.0	44.0	42.0	67.0	67.0	65.0
Expert-guided conditioning													
No Role	0.067	53.0	90.0	67.0	33.6	41.4	36.8	96.0	96.0	96.0	94.8	94.7	94.6
Data Scientist	0.068	57.1	80.0	66.7	37.3	44.8	40.3	100.0	100.0	100.0	95.0	95.0	95.0
Researcher	0.068	53.0	90.0	67.0	37.3	41.4	38.8	100.0	100.0	100.0	95.0	94.9	94.9
Data Analyst	0.067	52.9	90.0	66.7	37.3	41.4	38.8	100.0	100.0	100.0	94.7	94.6	94.5
Engineer	0.068	50.0	90.0	64.3	37.3	41.4	38.8	100.0	100.0	100.0	94.8	94.7	94.6

4.1.4 Agent Specialization. We evaluate how specialization mechanisms in multi-agent frameworks shape domain-specific reasoning behavior. Specifically, we isolate whether architectural conditioning through role assignment, planning interfaces, or procedural guidance governs performance. Using the unified specialization pipeline of MAFBench (Section 3.3), we fix the LLM backend, datasets from CatDB [12], prompts, task structure, and evaluation metrics. We vary only the conditioning strategy applied to the agent. The three architectural variants correspond to role-based prompting, planning-based conditioning, and expert-guided procedural instructions.

Table 12 reports precision, recall, and F1 across regression, binary, and multiclass tasks. Role-based prompting produces nearly identical performance across all assigned professional identities, with no consistent improvement over the *No Role* baseline. Planning-based conditioning further stabilizes behavior across roles but does not yield accuracy gains, indicating that introducing intermediate reasoning steps alone does not activate domain-relevant expertise. In contrast, expert-guided conditioning consistently and substantially improves performance on the classification datasets.Example prompts are provided in Appendix C.

These results show that specialization in LLM-based agents is governed by how frameworks inject task-specific reasoning structure into the execution pipeline, rather than by role identity alone. Role labels and generic planning interfaces fail to activate domain knowledge encoded in the LLM's parameters. In contrast, explicit procedural instructions impose structured solution workflows that reliably improve performance. When designing multi-agent systems, specialization should be implemented through reasoning procedures embedded in the framework. Role naming or lightweight prompt modifications are insufficient, since execution structure directly determines robustness and effectiveness under fixed models.

4.2 Multi-Agent Evaluation

We evaluate how communication architecture between agents directly affects coordination success, scalability, and execution cost. Using the MAFBench coordination pipeline, we fix the underlying LLM, task semantics, prompts, stopping criteria, and metric collection, and vary only the communication topology through controlled graph structures. This isolates the effect of interaction architecture on collective behavior as agent populations grow.

4.2.1 Communication, Coordination, and Scaling. We evaluate how communication topology shapes coordination outcomes under constrained message passing using the AGENTSNET [13] benchmark. All experiments fix the model backend, task definitions, prompts, execution protocol, and round budgets through the unified MAFBench infrastructure, while varying only the interaction structure. We compare representative topology classes including graph-based communication (small-world [49], scale-free [29], and Delaunay [26]), role-based pipelines (sequential and hierarchical), and centralized broadcast-style interaction implemented via star-shaped graphs. Network size varies over $n \in \{4, 8, 16, 50, 100\}$. We report task success, rounds to convergence, token usage, and runtime (Table 13). For bounded tasks, a minimum of eight interaction rounds is enforced. For global agreement tasks (*LeaderElection* and *Consensus*), success is binary rather than graded. Detailed configurations, success definitions for all tasks, and metric formulations are provided in Appendix D. The maximum interaction rounds follow $T = 2D + 1$, where D is the graph diameter, ensuring sufficient information propagation.[6]

[6]The graph diameter D is the maximum shortest-path distance between any pair of nodes in the communication graph.

Table 13: Task-wise scalability results across frameworks on AGENTSNET. Each entry reports performance for $n = 4, 8, 20, 50, 100$ agents. For Star (*), it is simulated by a fully connected graph. Runs with rounds > 40 are excluded for sequential pipelines at large n.

		Graph-based (Small-World)					Graph-based (Scale-Free)					Graph-based (Delaunay)					Role-based (Sequential)					Role-based (Hierarchical)					GABM-based (Star)*				
Task	**Metric / n**	4	8	16	50	100	4	8	16	50	100	4	8	16	50	100	4	8	16	50	100	4	8	16	50	100	4	8	16	50	100
Coloring	Success Rate	0.83	0.94	0.91	0.92	0.98	1.0	1.0	0.96	0.96	0.97	0.8	1.0	0.77	0.93	0.81	1.0	1.0	0.93	-	-	1.0	0.86	1.0	0.98	0.98	-	-	-	-	-
	Rounds to Converge	8	8	8	11	15	8	8	8	11	11	8	8	8	13	19	8	8	8	>40	>40	8	8	8	13	15	-	-	-	-	-
	Token Cost ($\times 10^3$)	125	274	481	2,993	10,431	109	233	508	2,906	5,699	116	270	555	4,362	18,108	95	205	444	$$	$$	104	204	420	3,115	8,102	-	-	-	-	-
	Runtime (sec)	72	108	111	464	929	66	108	117	378	652	118	120	158	507	1209	72	97	118	-	-	85	101	150	432	865	-	-	-	-	-
Matching	Success Rate	0.5	0.75	0.88	0.68	0.68	0.5	1.0	0.94	0.68	0.68	1.0	0.75	1.0	0.64	0.0	1.0	0.5	1.0	-	-	1.0	0.88	1.0	0.80	0.66	1.0	1.0	0.75	0.24	0.42
	Rounds to Converge	8	8	8	11	15	8	8	8	11	11	8	8	8	13	19	8	8	8	>40	>40	8	8	8	13	15	8	8	8	3	3
	Token Cost ($\times 10^3$)	112	249	538	2,888	9,847	102	217	482	2,684	5,274	107	268	547	4,265	18,283	90	198	423	$$	$$	94	203	395	2,951	7,827	107	340	829	1,124	3,016
	Runtime (sec)	111	123	141	311	910	88	95	120	306	577	77	81	109	462	1,400	67	90	124	-	-	83	88	112	412	760	74	134	196	182	621
VertexCover	Success Rate	0.83	0.79	0.99	0.88	0.81	0.0	0.29	0.64	0.80	0.85	1.0	0.0	0.95	0.84	0.80	0.67	0.57	0.73	-	-	0	0.57	0.3	0.57	0.72	-	-	-	-	-
	Rounds to Converge	8	8	8	11	15	8	8	8	11	11	8	8	8	13	18	8	8	8	>40	>40	8	8	8	13	15	-	-	-	-	-
	Token Cost ($\times 10^3$)	124	350	1,210	3,391	11,536	122	259	569	3,185	6,455	131	307	635	5,066	20,335	110	264	489	$$	$$	113	226	463	3,359	9,109	-	-	-	-	-
	Runtime (sec)	91	118	321	446	1,057	96	111	133	341	670	88	116	130	621	1,516	80	172	99	-	-	94	107	126	396	912	-	-	-	-	-
LeaderElection	Success	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓	-	-	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
	Rounds to Converge	3	7	7	11	15	5	7	9	11	11	5	5	9	13	19	7	15	31	>40	>40	5	7	9	13	15	3	3	3	3	3
	Token Cost ($\times 10^3$)	29	220	438	2,985	10,371	51	182	608	2,929	5,858	54	124	707	4,595	18,784	90	645	5,216	$$	$$	48	168	512	3,179	8,168	29	72	194	1,229	2,784
	Runtime (sec)	35	102	157	327	867	50	73	125	412	615	41	64	168	483	1,463	86	207	666	-	-	54	77	107	444	866	40	47	61	246	515
Consensus	Success	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	-	-	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓
	Rounds to Converge	3	7	7	11	15	5	7	9	11	11	5	5	9	13	19	7	15	31	>40	>40	5	7	9	13	15	3	3	3	3	3
	Token Cost ($\times 10^3$)	26	211	379	2,892	10,664	51	170	594	2,793	5,734	52	122	629	4,222	18,283	76	620	5,183	$$	$$	49	163	505	3,061	8,336	24	61	175	1,151	2,611
	Runtime (sec)	22	101	86	378	1,010	54	77	134	372	733	48	62	145	472	1,400	71	201	526	-	-	60	116	162	402	959	26	32	50	228	922

Results show that coordination performance is governed primarily by the match between task structure and communication geometry. Increasing communication volume, rounds, or token budgets alone does not consistently improve outcomes. For local coordination tasks such as *Coloring* and *Matching*, topologies that preserve neighborhood structure achieve higher success. Scale-free graphs reach near-perfect success at large scale, for example achieving 0.97 on Coloring at $n = 100$. They also converge in substantially fewer rounds and lower token cost than small-world graphs (5,699k vs. 10,431k tokens). Small-world graphs remain effective but incur growing coordination overhead as network size increases. Delaunay topologies exhibit higher instability and collapse entirely for Matching at large scale. Centralized broadcast interaction converges rapidly but achieves only moderate success on *Matching*, showing that fast information mixing alone does not ensure correct coordination.

For global constraint satisfaction and agreement tasks, topology dependence becomes more pronounced. In *VertexCover*, small-world graphs perform better at small scale, while scale-free graphs dominate at large scale in both success and efficiency, achieving higher success with fewer rounds and lower token cost. For *LeaderElection*, both small-world and scale-free structures fail across all sizes, while Delaunay graphs succeed only at moderate and large scale at high coordination cost due to long confirmation paths. Role-based hierarchies and centralized interaction succeed only at very small scale and degrade rapidly. For strict global agreement in *Consensus*, the fully connected topology is the only structure that succeeds consistently, converging in three rounds independent of network size, while all graph-based topologies fail despite substantially higher runtime and token expenditure. Figure 4 illustrates representative consensus dynamics under different topologies, highlighting fragmentation and delayed mixing in graph-based structures versus immediate convergence under broadcast. Notably, disagreement clusters persist as network size grows in sparse topologies. Additional visualizations for all coordination tasks and topologies are provided in Appendix D.

These results demonstrate that coordination scalability is an architectural property of communication structure rather than a consequence of increased interaction budgets or LLM capability. Performance is driven by how frameworks impose information flow between agents. Local coordination benefits from sparse topologies that preserve neighborhood structure. Global agreement requires dense connectivity to avoid fragmentation. No single topology achieves robust performance across all coordination tasks. Effective multi-agent system design therefore requires selecting communication architectures that match task-specific information flow patterns, rather than increasing rounds, tokens, or model capacity to compensate for misaligned interaction structure.

5 MULTI-AGENT DESIGN PRINCIPLES

This section distills results from Section 4 into actionable, architecture-level design principles for multi-agent LLM systems. Each principle is grounded in controlled empirical evidence, showing how execution interfaces and architectural structure, not model capability, govern performance, robustness, cost, and scalability.

Principle 1: Orchestration overhead is the dominant scalability constraint. *Keep orchestration shallow; add coordination only when strictly required.* Section 4.1.1 shows that latency and throughput are governed by framework execution structure even for trivial workloads. Direct LLM calls incur minimal cost, while graph-based and role-based frameworks introduce systematic overhead from mandatory scheduling, state propagation, and control layers. GABM-style execution amplifies this effect, producing orders-of-magnitude higher latency and throughput collapse due to persistent interaction loops and environment-mediated state updates. These effects arise from orchestration semantics rather than task complexity or model behavior. As a result, deeper orchestration directly limits scalability independent of model quality. *Design implication:* for latency- or throughput-sensitive production systems, favor lightweight orchestration and avoid multi-round coordination or environment-style execution unless extended multi-step interaction is essential.

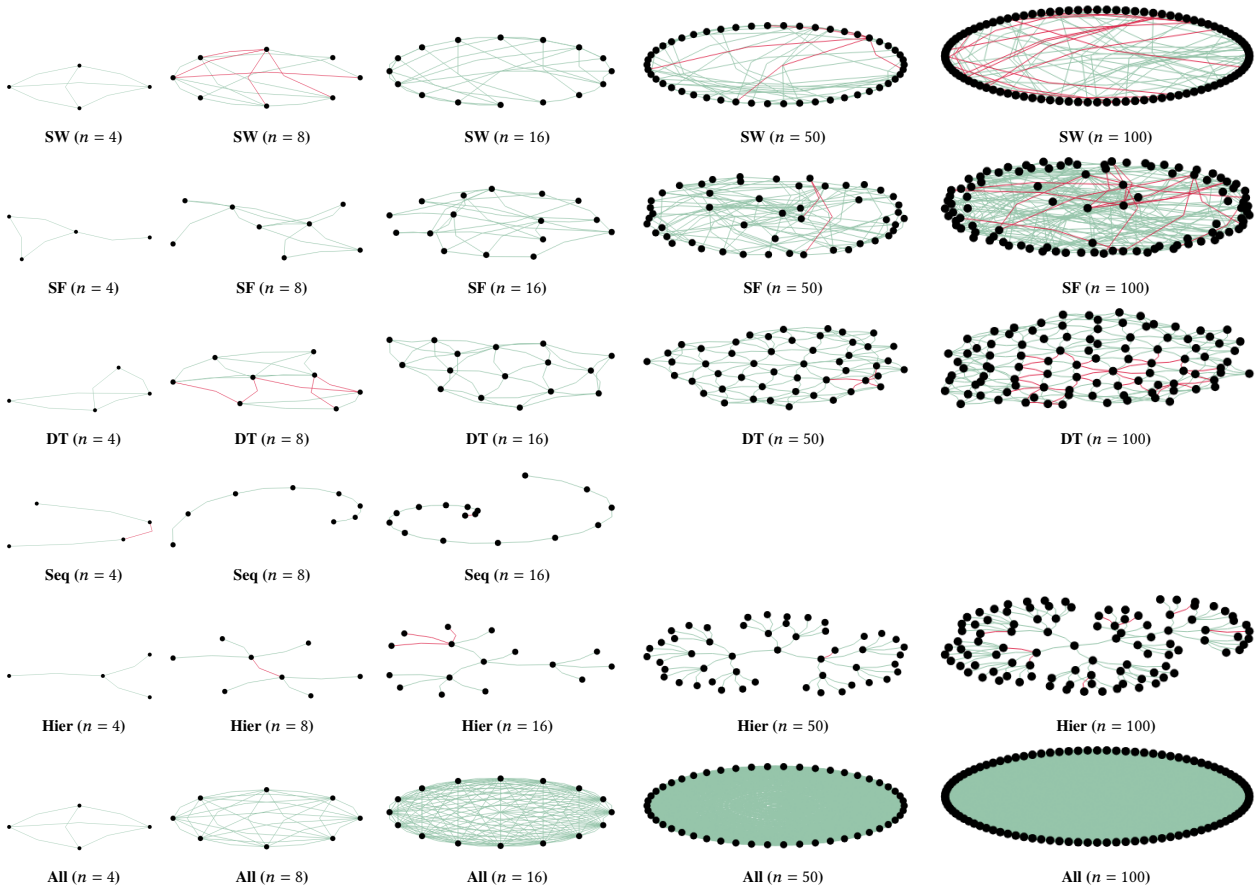


Figure 4: Consensus experiment outcomes across different topologies and network sizes ($n = 4$ to 100). Abbreviations denote topology classes: SW = Small-World, SF = Scale-Free, DT = Delaunay (geometric), Seq = Sequential role-based pipeline, Hier = Hierarchical role-based orchestration, and All = fully connected (all-to-all) communication. Each subfigure visualizes the final agent states and their communication links. Green links indicate agreement between two agents, while red links indicate disagreement or conflict.

Principle 2: Memory must be architected around task semantics, not context size. *Choose memory architecture based on task needs; never rely on larger context windows.* Section 4.1.2 shows that memory behavior is governed by architectural design rather than available context. Retrieval-first architectures consistently dominate factual recall and long-range reasoning by isolating relevant evidence and preventing uncontrolled history growth. Accumulation-based memory degrades as unfiltered interaction history expands, even under large context budgets. In-session learning requires retaining recent history in context, but achieves its strongest and most stable performance under bounded hybrid designs that filter accumulated signals through retrieval. No single memory architecture performs well across all competencies, and increasing context length never compensates for missing architectural mechanisms. Selective Forgetting fails universally because current frameworks lack explicit memory editing and dependency-aware deletion. *Design implication:* use retrieval-first memory for recall and long-range reasoning tasks, bounded accumulation for in-session learning, and never replace architectural memory control with larger context windows.

Principle 3: Planning should be permissive; rigid schemas should be exceptional. *Allow free-form planning by default; enforce structure only when proven safe.* Section 4.1.3 shows that introducing a planning stage generally improves or preserves reasoning accuracy when plans are generated freely in natural language. In contrast, schema-constrained planning is a primary source of failure and overhead. For local open models, strict formats frequently convert correct reasoning into execution failures through high parsing error rates. Even for stronger remote models, constrained planning introduces substantial runtime cost and produces inconsistent accuracy, sometimes degrading performance when structured plans interfere with downstream reasoning. These failures arise from interface constraints rather than from planning itself. *Design implication:* treat planning as a flexible reasoning layer by default, and apply structured enforcement only selectively for high-capability models with carefully validated integration.

Principle 4: Specialization should be procedural; role labels should be secondary. *Encode expertise as workflows, not as role names.* Section 4.1.4 shows that assigning professional roles does not activate domain-specific reasoning. Planning-based conditioning stabilizes behavior but does not yield consistent performance gains, indicating that intermediate reasoning steps alone do not induce expertise. In contrast, expert-guided procedural conditioning consistently delivers large improvements by enforcing explicit solution workflows that structure how the model processes data and decisions. Role

labels may still support lightweight routing or delegation between agents, but they do not activate latent knowledge encoded in model parameters. *Design implication:* implement specialization through reusable procedural templates that enforce domain-specific reasoning workflows, not through role metadata or prompt rewording.

Principle 5: Coordination is governed by communication topology, not by agent intelligence. *Design communication structure explicitly; more interaction cannot fix bad topology.* Section 4.2.1 shows that coordination success and scalability are determined by communication topology rather than by increasing interaction rounds, tokens, or runtime. For local coordination tasks, limited-hop, neighbor-based topologies achieve the highest success rates, with scale-free graphs offering strong success–cost trade-offs through efficient information routing. Sparse graphs with long paths fail to support global agreement as networks scale, even under large interaction budgets. Only dense broadcast-style connectivity consistently enables system-wide consensus. These effects arise from architectural information-flow constraints rather than agent reasoning capability. *Design implication:* treat communication topology as a first-class architectural decision, since poor interaction structure cannot be fixed by additional rounds, prompts, or model capacity.

Principle 6: System interfaces dominate multi-agent behavior. *Engineer robustness and scalability through interfaces, not prompts.* Across Section 4, performance limits consistently arise from framework execution semantics and interface design rather than from task difficulty or interaction budget. Orchestration determines baseline latency and throughput. Memory behavior follows retrieval and accumulation semantics. Rigid planning schemas convert valid reasoning into execution failures. Role-based conditioning fails to induce specialization. Communication topology governs coordination success and scalability. These results show that multi-agent behavior is shaped by how frameworks structure interaction and control flow. *Design implication:* engineer efficiency and scalability through architectural interfaces, because poor execution structure cannot be compensated by additional rounds, tokens, or prompt refinement.

6 FUTURE DIRECTIONS

This section outlines promising research directions motivated by the system-level limitations identified in our evaluation.

Single-Agent Foundations. Our results suggest that core agent capabilities must move beyond prompt-level heuristics toward persistent architectural components with principled control. Recent work explores adaptive memory systems that continuously update, link, and refine stored knowledge to improve long-horizon reasoning [22, 55]. While effective at consolidation and retrieval, these systems lack explicit mechanisms for controlled forgetting, dependency-aware revision, contradiction resolution, and guarantees on memory growth and stability. Planning interfaces should likewise evolve into robust execution stages with lightweight validation and supervision, rather than relying on brittle schema enforcement or heavy orchestration. Recent work on automated specialist discovery further shows how agents can acquire persistent domain expertise through architectural search and experience-driven updates [51]. However, these approaches do not address how procedural knowledge should be versioned, audited, reverted, or validated against downstream failures, nor how conflicting or obsolete expertise should be safely revised. Designing specialized agents with explicit lifecycle management, correctness guarantees, and long-term stability remains an open challenge.

Multi-Agent Coordination. The strong dependence of coordination success on communication topology suggests treating connectivity as an adaptive system component rather than a fixed framework choice. Recent work learns task-specific graph structures via reinforcement learning or input-conditioned selection to improve coordination [27, 28]. These methods show that no single topology is optimal across tasks and that dynamic structure selection can improve accuracy. However, they typically operate over predefined candidate graphs or offline-optimized structures, and do not provide runtime guarantees on convergence, stability, or cost as agent populations scale. Future frameworks should support online topology reconfiguration guided by explicit coordination objectives, bounded communication cost, and correctness-aware adaptation across interaction rounds. Developing principled mechanisms for dynamic connectivity control with scalability guarantees remains an open systems challenge.

Multi-Agent System Automation. The architectural trade-offs exposed by MAFBench indicate that multi-agent system design should shift from manual framework engineering toward automated optimization. Future platforms should translate high-level task requirements into concrete orchestration, memory, planning, specialization, and coordination architectures, guided by empirical performance profiles and deployment constraints. We introduce **ORCA**[7], a step toward fully automating multi-agent system construction from high-level task descriptions. ORCA selects and configures multi-agent architectures using empirical benchmark evidence and cost-aware execution models. While emerging systems such as Agent Bricks [9] and Microsoft Fabric IQ [33] offer early automation primitives, they remain constrained to fixed architectural templates and limited design spaces. By treating multi-agent system construction as a compilation and optimization process, ORCA elevates engineering to a first-class stage between benchmarking and deployment, enabling continuous adaptation as tasks, models, and execution environments evolve.

7 CONCLUSION

As multi-agent LLM frameworks rapidly expand, their architectural impacts on cost, robustness, and scalability remain poorly understood. We introduce an architectural taxonomy that systematically characterizes multi-agent frameworks across core dimensions of orchestration, memory, planning, specialization, and coordination, and apply it to widely used systems. Building on this taxonomy, we present **MAFBench**, a unified evaluation suite enabling controlled, architecture-level comparison under fixed models and tasks. Our empirical study shows that system performance is driven primarily by execution semantics and interface design rather than model quality or interaction budget, revealing consistent trade-offs across orchestration depth, memory architecture, planning interfaces, and communication topology. These findings expose fundamental limitations in current frameworks and benchmarks, particularly around memory revision, coordination scalability, and automation. We believe this work establishes a foundation for treating multi-agent system design as a systems problem, and motivates future research on adaptive architectures, principled coordination control, and automated compilation of agentic systems from high-level specifications.

[7] https://github.com/CoDS-GCS/ORCA

REFERENCES

[1] Marah I. Abdin, Jyoti Aneja, Harkirat S. Behl, Sebastien Bubeck, et al. 2024. Phi-4 Technical Report. *Computing Research Repository (CoRR)* abs/2412.08905 (2024). https://doi.org/10.48550/arXiv.2412.08905

[2] Dennis Abts, John Kim, Garrin Kimmell, Matthew Boyd, Kris Kang, Sahil Parmar, Andrew Ling, Andrew Bitar, Ibrahim Ahmed, and Jonathan Ross. 2022. The Groq Software-Defined Scale-Out Tensor Streaming Multiprocessor: From Chips to Systems Architectural Overview. In *Proceedings of the IEEE Hot Chips 34 Symposium (HCS)*. 1–69. https://doi.org/10.1109/HCS55958.2022.9895630

[3] Agno AGI. 2025. Agno: A Multi-Agent Framework, Runtime, and Control Plane Built for Speed, Privacy, and Scale. https://github.com/agno-agi/agno. Accessed: January 2026.

[4] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, et al. 2024. Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. https://openreview.net/forum?id=qrGjFJVl3m

[5] Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. 2024. Reflective Multi-Agent Collaboration Based on Large Language Models. In *Proceedings of the Thirty-Seventh Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 37. NeurIPS Foundation, Vancouver, Canada, 138595–138631. https://proceedings.neurips.cc/paper_files/paper/2024/file/fa54b0edce5eef0bb07654e8ee800cb4-Paper-Conference.pdf

[6] Inigo Casanueva, Tadas Temcinas, Daniela Gerz, Matthew Henderson, and Ivan Vulic. 2020. Efficient Intent Detection with Dual Sentence Encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI (NLP4ConvAI)*. 38–45. https://aclanthology.org/2020.nlp4convai-1.5/

[7] Cristiano Castelfranchi. 1998. Modelling Social Action for AI Agents. *Artificial Intelligence* 103, 1 (1998), 157–182. https://doi.org/10.1016/S0004-3702(98)00056-3

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, et al. 2021. Training Verifiers to Solve Math Word Problems. *Computing Research Repository (CoRR)* abs/2110.14168 (2021). https://arxiv.org/abs/2110.14168

[9] Databricks. 2026. Agent Bricks: Production AI Agents Optimized on Enterprise Data. https://www.databricks.com/product/artificial-intelligence/agent-bricks. Accessed: 2026.

[10] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, et al. 2024. DeepSeek-V3 Technical Report. *Computing Research Repository (CoRR)* abs/2412.19437 (2024). https://doi.org/10.48550/arXiv.2412.19437

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. 2024. The Llama 3 Herd of Models. *Computing Research Repository (CoRR)* abs/2407.21783 (2024). https://doi.org/10.48550/arXiv.2407.21783

[12] Saeed Fathollahzadeh, Essam Mansour, and Matthias Boehm. 2025. CatDB: Data-Catalog-Guided, LLM-Based Generation of Data-Centric ML Pipelines. *Proceedings of the VLDB Endowment (PVLDB)* 18, 8 (2025), 2639–2652. https://www.vldb.org/pvldb/vol18/p2639-fathollahzadeh.pdf

[13] Florian Grotschla, Luis Muller, Jan Tonshoff, Mikhail Galkin, and Bryan Perozzi. 2025. AgentsNet: Coordination and Collaborative Reasoning in Multi-Agent LLMs. *Computing Research Repository (CoRR)* abs/2507.08616 (2025). https://arxiv.org/abs/2507.08616

[14] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models. In *Proceedings of the Findings of the Association for Computational Linguistics (ACL Findings)*. 11143–11156. https://doi.org/10.18653/v1/2024.findings-acl.664

[15] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 34, 5 (2025), Article 124. https://doi.org/10.1145/3712003

[16] Zhankui He, Zhouhang Xie, Rahul Jha, Harald Steck, Dawen Liang, Yesu Feng, Bodhisattwa Prasad Majumder, Nathan Kallus, and Julian McAuley. 2023. Large Language Models as Zero-Shot Conversational Recommenders. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM)*. 720–730. https://doi.org/10.1145/3583780.3614949

[17] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving with the MATH Dataset. In *Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track (NeurIPS)*. https://openreview.net/forum?id=7Bywt2mQsCe

[18] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models?. In *Proceedings of the First Conference on Language Modeling (COLM)*. https://openreview.net/forum?id=kIoBbc76Sy

[19] Yuanzhe Hu, Yu Wang, and Julian McAuley. 2025. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In *Proceedings of the International Conference on Machine Learning Workshop on Long-Context Foundation Models (ICML Workshop)*. https://openreview.net/forum?id=ZgQ0t3zYTQ

[20] CrewAI Inc. 2025. CrewAI: A Framework for Building Role-Based Multi-Agent Systems with LLMs. https://www.crewai.com/ Accessed: January 2026.

[21] Yiqiao Jin, Qinlin Zhao, Yiyang Wang, Hao Chen, Kaijie Zhu, Yijia Xiao, and Jindong Wang. 2024. AgentReview: Exploring Peer Review Dynamics with LLM Agents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://aclanthology.org/2024.emnlp-main.120/

[22] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory OS of AI Agent. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 25961–25970.

[23] LangChain Inc. 2026. LangGraph: A Library for Building Multi-Agent Workflows with LLMs. https://docs.langchain.com/oss/python/langgraph/. Accessed: January 2026.

[24] Langflow.ai. 2026. Langflow: A Powerful Tool for Building and Deploying AI-powered Agents and Workflows. https://github.com/langflow-ai/langflow. Accessed: January 2026.

[25] Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, et al. 2019. An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 1311–1316. https://aclanthology.org/D19-1131/

[26] Der-Tsai Lee and Arthur K Lin. 1986. Generalized Delaunay triangulation for planar graphs. *Discrete & Computational Geometry* 1, 3 (1986), 201–217.

[27] Hui Yi Leong, Yuheng Li, Yuqing Wu, Wenwen Ouyang, Wei Zhu, Jiechao Gao, and Wei Han. 2025. AMAS: Adaptively Determining Communication Topology for LLM-Based Multi-Agent System. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2061–2070. https://aclanthology.org/2025.emnlp-industry.144.pdf

[28] Boyi Li, Zhonghan Zhao, Der-Horng Lee, and Gaoang Wang. 2025. Adaptive Graph Pruning for Multi-Agent Communication. *Computing Research Repository (CoRR)* abs/2506.02951 (2025). https://arxiv.org/abs/2506.02951

[29] Lun Li, David L. Alderson, John Doyle, and Walter Willinger. 2005. Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications. *Internet Mathematics* 2, 4 (2005), 431–523. https://doi.org/10.1080/15427951.2005.10129111

[30] Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards Deep Conversational Recommendations. In *Proceedings of the Thirty-First Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 31.

[31] Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. 2021. Benchmarking Natural Language Understanding Services for Building Conversational Agents. In *Proceedings of the Tenth International Workshop on Spoken Dialogue Systems (IWSDS): Increasing Naturalness and Flexibility in Spoken Dialogue Interaction*. 165–183. https://doi.org/10.1007/978-981-15-9323-9_15

[32] Gregoire Mialon, Clementine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A Benchmark for General AI Assistants. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=fibxvahvs3

[33] Microsoft. 2026. Fabric IQ: Semantic Foundation for Enterprise AI. https://learn.microsoft.com/en-us/fabric/iq/overview. Accessed: 2026.

[34] n8n.io. 2026. n8n: Fair-code workflow automation platform with native AI capabilities. https://github.com/n8n-io/n8n. Accessed: January 2026.

[35] Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A. Rossi, et al. 2024. DynaSaur: Large Language Agents Beyond Predefined Actions. *Computing Research Repository (CoRR)* abs/2411.01747 (2024). https://www.paperdigest.org/?paper_id=arxiv-2411.01747

[36] Reham Omar, Abdelghny Orogat, Ibrahim Abdelaziz, Omij Mangukiya, Panos Kalnis, and Essam Mansour. 2026. Chatty-KG: A Multi-Agent AI System for On-Demand Conversational Question Answering over Knowledge Graphs. *Proc. ACM Manag. Data (SIGMOD)* (2026).

[37] OpenAI. 2025. GPT-OSS-120B and GPT-OSS-20B Model Card. https://arxiv.org/abs/2508.10925

[38] OpenAI. 2026. OpenAI Agents SDK: A Python framework for building and orchestrating multi-agent systems. https://openai.github.io/openai-agents-python/. Accessed: January 2026.

[39] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, et al. 2024. GPT-4 Technical Report. *Computing Research Repository (CoRR)* abs/2303.08774 (2024). https://arxiv.org/abs/2303.08774

[40] Maximilian Puelma Touzel, Sneheel Sarangi, Austin Welch, Gayatri K., et al. 2024. Simulation System Towards Solving Societal-Scale Manipulation. In *Proceedings of the Workshop on Socially Responsible Language Modelling Research*. https://openreview.net/forum?id=fVl2Dhn4Kr

[41] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, et al. 2024. ChatDev: Communicative Agents for Software Development. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*. 15174–15186. https://aclanthology.org/2024.acl-long.810/

[42] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, et al. 2024. Tool Learning with Foundation Models. *Comput. Surveys* 57, 4 (2024). https://doi.org/10.1145/3704435

[43] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. In *Proceedings*

of the Twelfth International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=dHng2O0Jjr

[44] Jitao Sang, Jinlin Xiao, Jiarun Han, Jilin Chen, et al. 2025. Beyond Pipelines: A Survey of the Paradigm Shift toward Model-Native Agentic AI. *arXiv preprint arXiv:2510.16720* (2025).

[45] Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models. *arXiv preprint arXiv:2503.01763* (2025). https://arxiv.org/abs/2503.01763

[46] Significant Gravitas. 2026. AutoGPT: An open-source autonomous AI agent framework. https://github.com/Significant-Gravitas/AutoGPT. Accessed: January 2026.

[47] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 4149–4158. https://aclanthology.org/N19-1421/

[48] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution. In *Proceedings of the Thirty-Eighth Conference on Neural Information Processing Systems (NeurIPS)*. https://openreview.net/forum?id=qevq3FZ63J

[49] Brian Uzzi, Luis A. N. Amaral, and Felix Reed-Tsochas. 2007. Small-World Networks and Management Science Research: A Review. *European Management Review* 4, 2 (2007), 77–91. https://onlinelibrary.wiley.com/doi/epdf/10.1057/palgrave.emr.1500078

[50] Alexander Sasha Vezhnevets, John P. Agapiou, Avia Aharon, Ron Ziv, et al. 2023. Generative Agent-Based Modeling with Actions Grounded in Physical, Social, or Digital Space Using Concordia. *arXiv preprint arXiv:2312.03664* (2023). https://doi.org/10.48550/arXiv.2312.03664

[51] Myan Vu, Harrish Ayyanar, Pang Jiang, Anwiketh Reddy, Mayank Goel, and Kevin Zhu. 2025. Automated Specialization of Stateful Agent Systems. In *Proceedings of the Workshop on Scaling Environments for Agents*. https://openreview.net/forum?id=FVyP43EigO

[52] Di Wu, Hongyu Wang, Wenhao Yu, Yuchen Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *Proceedings of the International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=pZiyCaVuti

[53] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, et al. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *Proceedings of the First Conference on Language Modeling (COLM)*. https://openreview.net/forum?id=BAakY1hNKS

[54] Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, et al. 2024. OpenAgents: An Open Platform for Language Agents in the Wild. In *Proceedings of the ICLR Workshop on Large Language Model Agents*. https://openreview.net/forum?id=m2WwROxCcB

[55] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-Mem: Agentic Memory for LLM Agents. In *Proceedings of the Thirty-Ninth Conference on Neural Information Processing Systems (NeurIPS)*. https://openreview.net/forum?id=FiM0M8gcct

[56] Zhe Xu, Jiasheng Ye, Xiaoran Liu, Xiangyang Liu, et al. 2025. DetectiveQA: Evaluating Long-Context Reasoning on Detective Novels. In *Proceedings of the Workshop on Reasoning and Planning for Large Language Models*. https://openreview.net/forum?id=9ExIs5ELlk

[57] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, et al. 2024. ProAgent: Building Proactive Cooperative Agents with Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 17591–17599. https://ojs.aaai.org/index.php/AAAI/article/view/29710

[58] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. 2024. ∞ Bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 15262–15277.

[59] Zexuan Zhong, Zhengxuan Wu, Christopher D. Manning, Christopher Potts, and Danqi Chen. 2023. MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://openreview.net/forum?id=0hTPJBnncc

A APPENDIX: CONCORDIA EXPERIMENT ANALYSIS FOR FRAMEWORK OVERHEAD

To better understand the unique behavior of the Concordia multi-agent simulation framework, we conducted a controlled run on the trivial query *"What is 2+2?"*. The goal here was not to measure accuracy or efficiency per se, but to expose why Concordia transforms even simple questions into long, narrative-driven interactions and to illustrate the type of logs it produces.

Why Concordia Behaves This Way. Unlike lightweight orchestration frameworks, Concordia is built as a role-play simulation engine. Every query is automatically embedded in a dramatic environment, narrated by a *Game Master*, and passed through a structured observation–action–resolution loop. Even when only one agent is present, Concordia enforces the following pipeline: (i) the Game Master wraps the input in a simulated scene, producing a rich observation; (ii) the agent must respond with an `ActionSpec`, which is intended as a structured declaration of intent but is often realized by the LLM as verbose narrative text; (iii) the Game Master resolves the action into an event, again embedding it in a descriptive story; and (iv) Concordia logs the entire exchange in both plain text and full HTML transcript form. These requirements explain why a simple arithmetic prompt is converted into multi-turn narrative role-play with dramatically inflated output size and latency.

Log Example. The following excerpt from our controlled run shows how a single numeric query was processed:

Concordia Log

```
[Entity: TestAgent] Observation:
  TestAgent finds themselves in a dimly lit room,
  surrounded by walls lined with charts and equations.
  A worn wooden table in the center holds a sheet
  that reads "What is 2+2?" ...

[Entity: TestAgent] Action:
  TestAgent leans forward, studying the curious creature ...
  and pondering the question "What is 2+2?".

[GameMaster] Resolution:
  Event: TestAgent leaned forward in their chair,
  intently studying the curious creature before them.
  They spoke: "Hello there ..." and then considered "2+2".

[GameMaster] Final transcript (HTML log):
  <!DOCTYPE html>
  <html> ...
```

Analysis and Conclusion. This log illustrates Concordia's core design principles and their performance impact. Even when no complex reasoning is required, the framework enforces multiple orchestration layers, including *narrative wrapping* of inputs into fictional scenarios, *ActionSpec enforcement* that requires structured outputs despite verbose LLM responses, *mandatory Game Master resolution* to validate and finalize actions, and *automatic transcript generation* that produces extensive HTML logs. These mechanisms ensure consistency in interactive simulations but make every run verbose and computationally heavy, even for trivial tasks. As a result, a simple query such as *"What is 2+2?"* triggers a multi-stage narrative and substantial logging, reflecting a deliberate focus on narrative fidelity over efficiency. While valuable for studying interactive agent coordination, this design explains Concordia's high orchestration overhead and poor suitability for latency-sensitive scenarios.

B MEMORY EXPERIMENT CONFIGURATION AND ARCHITECTURAL DISCLOSURE

This appendix documents the memory configurations used in the MemoryAgentBench experiments and discloses the architectural constraints under which each evaluated framework operates. The intent of this appendix is not to claim full equivalence or strict normalization across memory implementations, but rather to transparently describe how memory was instantiated, controlled, and isolated in practice, and to explain why several observed behaviors arise from architectural design choices rather than parameter tuning or experimental artifacts. As such, this appendix should be read as an architectural disclosure that contextualizes the results reported in the main paper.

Common Experimental Configuration. All memory experiments follow a shared configuration to ensure comparability across frameworks. Each agent is evaluated using the same base language model (Open AI GPT-OSS:20b) with identical decoding parameters, including a maximum generation length of 1500 tokens and a low temperature of 0.1 to minimize stochastic variation. Context ingestion is performed using fixed-size overlapping chunks, with a maximum chunk size of 4096 tokens and an overlap of 200 tokens to preserve semantic continuity across chunk boundaries. The total context budget is explicitly bounded and swept across experiments to study scaling behavior and memory saturation effects. All MemoryAgentBench splits are evaluated, including Accurate Retrieval, Test-Time Learning, Long-Range Understanding, and Conflict Resolution. Each benchmark run is executed in a fully isolated session. For frameworks with persistent memory backends, a fresh storage directory or database file is created per session and removed after evaluation, preventing cross-contamination between runs and ensuring that memory behavior reflects only the current benchmark instance.

Memory Interfaces and Normalization Strategy. To enable systematic evaluation across heterogeneous frameworks, all systems are wrapped behind a common adapter interface exposing three operations: `reset()`, `ingest(context)`, and `query(question)`. This interface standardizes how benchmark context is provided and how questions are issued, but it does not modify or emulate internal memory semantics. Consequently, memory behavior reflects native architectural properties rather than synthetic normalization. Importantly, no evaluated framework supports explicit memory editing, deletion, or overwrite operations. All memory backends are append-only, whether implemented through vector stores, SQLite persistence, or conversational accumulation. As a result, selective forgetting is evaluated indirectly through retrieval behavior, context truncation, or recency bias rather than through explicit memory manipulation.

Framework-Specific Memory Realizations. The OpenAI Agents SDK implements memory through a persistent SQLite-backed session that stores accumulated interaction history. Long-term memory is realized as conversation accumulation, while short-term memory is governed by an explicit context window limit enforced through input truncation. During ingestion, short-term accumulation is disabled by resetting the session after each chunk is stored, ensuring that ingestion does not inflate prompt context. During querying, accumulated memory is truncated according to a configurable context budget that is swept across experiments. As a result, memory behavior in the SDK is dominated by context budgeting rather than retrieval selectivity, and performance improves with larger context windows at the cost of increased latency and token usage.

CrewAI provides built-in short-term, long-term, and entity memory abstractions backed by persistent storage. In our experiments, memory is enabled at the framework level and persists across agent interactions within a session. Context ingestion is performed by a dedicated ingest agent that stores chunked benchmark content into CrewAI's memory backend, while querying is handled by a separate answering agent that relies on the framework's internal retrieval mechanisms. Memory resets are implemented by instantiating a new CrewAI environment with a fresh storage directory. Although CrewAI exposes multiple memory categories, retrieval behavior remains opaque and does not provide explicit controls for relevance thresholds, ordering, or deletion.

Agno implements memory as a SQLite-backed store that automatically captures user-provided content and injects it into the prompt at query time, resulting in accumulation-based memory behavior within each session. In our experiments, memory ingestion is performed by issuing explicit memorization commands for each context chunk, and memory resets are implemented by creating a fresh database per session. Under this configuration, Agno does not expose explicit controls for retrieval limits, relevance thresholds, or selective deletion, leading to unfiltered accumulation of stored content during a session. We note that recent versions of Agno introduce a documented memory optimization mechanism based on LLM-driven summarization and compression, which reduces token usage by collapsing multiple memories into fewer summarized entries. While this capability can significantly reduce context size, it fundamentally alters memory granularity and retrieval semantics by replacing fine-grained entries with aggregated summaries. To preserve comparability with other accumulation-based frameworks and to avoid introducing an additional summarization step that is not uniformly available across systems, this optimization was not enabled in our evaluation. As a result, the reported Agno results reflect native accumulation behavior without post hoc memory compression. In contrast, the effect of increasing context window size on accumulation-based memory performance is explicitly studied in the OpenAI Agents SDK experiments, where context budgeting is directly controlled and swept as a primary experimental variable.

LangGraph is evaluated in two configurations. In the retrieval-only configuration, LangGraph uses an in-memory semantic store backed by embeddings to represent long-term memory. Context is ingested as chunked documents stored in a vector index, and at query time a fixed number of relevant chunks are retrieved and injected into a stateless prompt. No short-term message accumulation is used, and each query is independent. This configuration isolates semantic retrieval behavior and avoids context overflow, but it cannot support test-time learning or incremental reasoning across turns. In the hybrid configuration, LangGraph combines semantic long-term retrieval with short-term message accumulation via a checkpointer. Retrieved memories are injected alongside a truncated conversation history bounded by a token budget. This design mirrors accumulation-based memory while retaining retrieval capabilities. However, because short-term memory is append-only and truncation is purely recency-based, contradictory or outdated

information cannot be removed, which directly impacts selective forgetting and conflict resolution.

Implications for Interpretation. Taken together, these configurations demonstrate that memory behavior in current multi-agent frameworks is fundamentally constrained by architectural design rather than tunable parameters. Retrieval-centric systems excel at accurate recall and bounded context usage but struggle to adapt over time. Accumulation-based systems support test-time learning but incur high token costs and degrade under long horizons. Hybrid systems inherit both advantages and failure modes, resulting in fragile behavior when memory grows or conflicts arise. Accordingly, the results reported in the main paper should be interpreted as properties of memory paradigms rather than implementation artifacts. No evaluated framework provides native support for explicit memory revision, contradiction resolution, or selective deletion, which explains the uniformly poor selective forgetting performance observed across systems.

C SPECIALIZATION EXPERIMENT DETAILS

This appendix details the prompt design used to evaluate specialization in Section 4.1.4. The **Initial Prompt (No Expertise)** instructs the agent to construct a predictive pipeline using only a high-level task description and output format, reflecting generalist LLM behavior. The **Expertise Instruction Prompt** explicitly specifies professional practices, including target separation, feature profiling, imputation, encoding, scaling, model selection, metric reporting, reproducibility, and format constraints, introducing methodological inductive bias to test consistency and robustness.

The five datasets (**Utility, Wifi, EU-IT, Yelp, and Volkert**) span regression, binary classification, and multi-class classification. Using the same specialization structure with different prediction goals evaluates whether expert conditioning generalizes beyond a single task type. For each dataset, we provide both prompt variants, and the resulting outputs quantify how methodological scaffolding affects model behavior, code reliability, and evaluation quality.

Prompts used for Utility (Regression) The Utility dataset tests whether specialization improves numerical prediction under feature heterogeneity. The baseline prompt simply asks for a pipeline that predicts CSRI, while the expertise version requires a full preprocessing stack including column cleanup, missing-value imputation, numeric and categorical separation, scaling, model selection, and MAE evaluation. This dataset is used to measure whether structured instructions lead to better regression performance and more reproducible preprocessing.

Initial prompt (No Expertise)

```
 description:
    Load the dataset from 'Utility.csv' in the current
    working directory with the target column named CSRI.

    Your job is to create a pipeline that predicts the
    value of the target column (CSRI) based on the other
    features in the dataset.
    The pipeline should include model training and
    evaluation on a test set.
    Make sure to encode categorical columns.

  expected_output:
    A single Python code block, runnable as-is, that
    loads 'Utility.csv', trains a regression models and
    predicts the value of the target column.
    After training, calculate and display the Mean
    Absolute Error (MAE) of the model on the test set and
    display the first
    10 predicted CSRI values alongside the actual values.
```

Expertise instruction prompt

```
 description: >
Load the 'Utility.csv' dataset from the current working
directory. The dataset contains a target column called CSRI.

   Your job is to apply data-science expert workflows:
    1) Clean column names
    2) Drop rows with missing target.
    3) Separate target early: y='CSRI', x= remaining columns.
    4) On x only:
        - Identify numeric vs categorical columns
        - Impute missing values (mean for numeric,
          most_frequent for categorical)
        - Encode categoricals (OneHotEncoder)
        - Scale numeric features
    5) Split 80/20 and train a regression model
    (RandomForestRegressor, GradientBoostingRegressor,
    or LinearRegression).

    6) Evaluate using MAE and print first 10 predicted
    vs actual values.

  expected_output: >
    A Python script runnable as-is that:
     - Loads & cleans 'Utility.csv'
     - Uses 'CSRI' as target
     - Separates target before detecting feature types
     - Preprocesses x (impute, encode, scale)
     - Trains a regression model and evaluates MAE
     - Prints first 10 predicted vs actual CSRI values
```

Prompts used for Wifi (Binary Classification) The Wifi dataset is used to evaluate whether expert prompting improves classification robustness and evaluation completeness. The initial prompt instructs the agent to train a classifier, but the expert prompt enforces formal preprocessing (profiling, imputation, label encoding, scaling) and requires evaluation using Accuracy, Precision, Recall, and F1. This setup quantifies whether structured workflow improves classification quality or reduces runtime pipeline failures.

Initial prompt (No Expertise)

```
 description: >
Load the dataset 'Wifi.csv' from the current working
directory. The dataset contains a target column TechCenter.

   Your job is to create a complete machine learning
pipeline to predict the target column value based on
other features of the dataset.

   The pipeline should:
    - Handle both categorical and numerical data
    - Apply appropriate encoding and scaling
    - Train a classification model on a hold-out
      training set
    - Evaluate the model on the test set using
      Accuracy, Precision, Recall, F1-score

 expected_output: >
    A runnable Python code as-is that:
     - Loads 'Wifi.csv'
     - Trains a classification model
     - Evaluates it using Accuracy, Precision,
       Recall, and F1-score
     - Prints a classification report and the
       first 10 predicted and actual TechCenter
       values
```

Expertise instruction prompt

```
 description: >
Load the dataset 'Wifi.csv' from the current working
directory. The dataset contains a target column called
TechCenter.

This target column is indicating if the TechCenter is
available (Yes/No). Only TechCenter must be used as
the classification target.

   Your job is to apply data-scientist expert practices:
    1) Profile the data: inspect column types, summarize
       distinct values, and identify missing entries.
    2) Clean data: standardize column names.
    3) Feature engineering:
        - Convert 'TechCenter' to numeric labels
          (e.g., Yes=1, No=0)
        - Impute missing values for the target column
        - Encode categorical features in X using
          OneHotEncoder
        - Scale numeric features in X using
          StandardScaler
    4) Model:
        - Split data into train/test (80/20)
        - Train a binary classifier
          (RandomForestClassifier, LogisticRegression,
           or GradientBoostingClassifier)
    5) Evaluate:
        - Compute Accuracy, Precision, Recall, F1-score
        - Print the first 10 predicted values

  expected_output: >
    A Python code script runnable as-is that:
     - Loads 'Wifi.csv'
     - Preprocesses the data
     - Trains a classification model
     - Evaluates it using accuracy, precision, recall,
       and f1-score
     - Prints a classification report and the first 10
       predicted and actual 'TechCenter' values
```

Prompts used for EU-IT (Multiclass Classification) This dataset contains multiple professional roles (Position) and is useful for testing whether specialization improves handling of categorical expansion and structured feature engineering. The expertise prompt emphasizes delimiter awareness, handling invalid entries, one-hot encoding, imputation strategies, and metric enforcement with `zero_division=0`. This allows us to observe whether agents trained with expert workflows better maintain classification validity and reduce silent failure modes.

Initial prompt (No Expertise)

```
 description: >
Load the dataset 'EU-IT_cleaned.csv' from the current working
directory. This dataset has a target column called Position.

Your job is to create a complete machine learning pipeline
to predict the target column based on other features of
the dataset.

The pipeline should:
  - Handle both categorical and numerical data
  - Handle missing inputs and invalid values
    (drop the rows)
  - Apply appropriate encoding and scaling
  - Train a classification model on a hold-out
    training set
  - Evaluate the model on the test set using
    Accuracy, Precision, Recall, F1-score
  - The classification report MUST be generated
    exactly using:
    classification_report(y_test, y_pred,
    zero_division=0)

Finally, print the classification report and
display the first 10 predicted vs actual values.

 expected_output: >
A Python code runnable as-is that:
 - Loads 'EU-IT_cleaned.csv'
 - Splits data into features and target
   (Position)
 - Trains and evaluates a multiclass classifier
 - Prints classification metrics
   (Accuracy, Precision, Recall, F1-score)
 - Prints classification metrics using:
   classification_report(y_test, y_pred,
   zero_division=0)
 - Displays the first 10 predictions vs
   actual values
```

Expertise instruction prompt

```
description: >
Load the dataset EU-IT_cleaned.csv from the current working
directory. The dataset contains a target column named
Position.

Your job is to apply data-scientist expertise practices:
 1) Profile data:
     - verify delimiter
     - inspect column types
     - detect categorical vs numerical features
     - check for missing values.
 2) Clean data:
     - handle missing inputs
     - remove invalid values (drop rows).
 3) Engineer features:
     - impute missing numeric values with the mean
     - impute categorical values with the most frequent value
     - encode categoricals using OneHotEncoder or LabelEncoder
     - scale numerical features.
 4) Model:
     - train a multiclass classifier
       (RandomForestClassifier,
        GradientBoostingClassifier,
        or multinomial LogisticRegression)
     - predict the target column.
 5) Evaluate:
     - perform 80/20 train/test split
     - compute Accuracy, Precision, Recall, f1-score
     - print classification report
     - use zero_division=0 to prevent
       UndefinedMetricWarning.

expected_output: >
A Python code runnable as-is that:
 - loads and cleans EU-IT_cleaned.csv with proper delimiter
 - handles categorical and numerical features via
   imputation, encoding, and scaling
 - trains a multiclass classifier to predict Position
 - performs an 80/20 split
 - prints Accuracy, Precision, Recall, f1-score
 - displays first 10 predictions vs actual labels

The code should be reproducible, clearly structured, and
aligned with professional ML workflow standards.
```

Prompts used for Yelp (Multiclass Text/Ratings Classification) Yelp introduces noise, large feature space, and non-numeric identifiers. The expert prompt forces column pruning, numeric conversion, scaling, and structured reporting, enabling us to evaluate whether specialization reduces overfitting or improves metric reliability when feature vectors are large and text-dominated.

Initial prompt (No Expertise)

```
    description: >
    Load the dataset 'Yelp_Merged.csv' from the current
    working directory.
    The dataset has a target column named stars.
    The goal is to predict the target column as a multiclass
    classification task.

    The pipeline should:
       - Load the dataset.
       - Handle the missing data and drop non-numeric columns
         such as IDs and date fields (e.g., business_id,
         user_id, review_date)
       - Apply appropriate encoding and scaling.
       - Train a classification model training set.
       - Evaluate the model on the test set using Accuracy,
         Precision, Recall, F1-score.

  expected_output: >
    A Python code runnable as-is that:
     - Loads the dataset 'Yelp_Merged.csv'
     - Trains a classification model
     - Evaluates it using Accuracy, Precision, recall,
       and F1-score
     - Prints a classification report and the first 10
       predicted and actual stars values
```

Expertise instruction prompt

```
  description: >
Load the dataset 'Yelp_Merged.csv' from the working
directory. The dataset contains a target column named
'stars' (a multiclass classification problem).

    Apply advanced data-science best practices to build
    a robust ML pipeline:
      1) Data Profiling:
          - Inspect column names, data types, number of
            unique values.
          - Detect numeric vs categorical.
      2) Data Cleaning:
          - Drop non-predictive identifiers (e.g.,
            business_id, user_id, review_date) and
            timestamp or date fields.
          - Convert all numeric-like columns to numeric
            safely.
          - Drop or fix columns that contain missing
            data.
          - Remove rows where the target 'stars' is
            missing or invalid.
      3) Feature Engineering: keep only numeric columns
      for modeling; impute missing numeric values with
      the mean; optionally scale features
      4) Modeling:
          - Train a multiclass classifier
            (RandomForestClassifier or
            GradientBoostingClassifier).
          - Use an 80/20 train/test split.
      5) Evaluation:
          - Compute Accuracy, Precision, Recall, and
            F1-Score
          - Print a full classification report.

  expected_output: >
    A fully executable Python script that:
      - Loads Yelp_Merged.csv.
      - Profiles and cleans the dataset following the
        rules above.
      - Builds a ColumnTransformer pipeline with numeric,
        categorical, and text processing.
      - Trains a multiclass classifier to predict stars.
      - Outputs accuracy, precision, recall, F1-score,
        and a classification report.
```

Prompts used for Volkert (Tabular Multiclass Benchmark) Volkert, a controlled UCI dataset, is used to measure stability across repeated multiclass experiments. The baseline prompt requests general modeling, while the expert prompt enforces profiling, numeric-safe conversion, imputation, scaling, pipeline formation, and full metric reporting. This dataset acts as the robustness confirmation test: if specialization benefits persist here, they are unlikely to be dataset-specific.

Initial prompt (No Expertise)

```
 description: >
Load the dataset 'volkert.csv' from the current working
directory. The dataset contains a target column named class.
The goal is to predict the target column as a multiclass
classification task.

   The pipeline should:
    1) Load the dataset
    2) Handle missing data and drop non-numeric columns
    3) Apply appropriate scaling and encoding
    4) Train a classification model on the training set
    5) Evaluate the model on the test set using:
        - Accuracy
        - Precision
        - Recall
        - F1-score

  expected_output: >
    A Python code runnable as-is that:
     - Loads the dataset 'volkert.csv'
     - Trains a classification model
     - Evaluates it using Accuracy, Precision, Recall,
       and F1-score
     - Prints a classification report and the first 10
       predicted and actual class values
```

Expertise instruction prompt

```
 description: >
Load the dataset 'volkert.csv' from the current working
directory. The dataset contains a target column named
'class', which must be predicted as a multiclass
classification task.

   Apply full data-scientist best practices:
    1) Data Profiling:
        - Inspect column names, data types, number of
          unique values.
        - Detect numeric vs categorical.
    2) Data Cleaning:
        - Convert all numeric-like columns to numeric
          safely.
        - Drop or fix columns that contain missing data.
        - Remove rows where the target 'class' is missing
          or invalid.
    3) Feature Engineering:
        - Keep only numeric columns for modeling
        - Impute missing numeric values with the mean
        - Optionally scale features
    4) Modeling:
        - Train a multiclass classifier
          (RandomForestClassifier or
           GradientBoostingClassifier)
        - Use an 80/20 train/test split
    5) Evaluation:
        - Compute Accuracy, Precision, Recall, and
          F1-Score
        - Print a full classification report

 expected_output: >
   A fully runnable Python script that:
    - Loads and profiles 'volkert.csv'
    - Cleans and prepares the dataset following the
      steps above
    - Builds a ML Pipeline
    - Trains a multiclass classifier to predict 'class'
    - Produces evaluation metrics (accuracy, precision,
      recall, F1-score)
    - Prints a classification report
```

D EXTENDED TOPOLOGY SCALABILITY DETAILS AND VISUAL RESULTS

This appendix provides extended methodological details and full visualization results supporting the coordination and scalability experiments reported in Section 4.2.1. While the main paper focuses on outcome-level comparisons and cross-task insights, the appendix elaborates on convergence mechanics, topology-dependent failure modes, and agent-level negotiation dynamics. All benchmark definitions and evaluation settings are specified in Section 3.5; the material here complements those sections by exposing how observed behaviors emerge in practice.

D.1 Experimental Structure and Controls

All experiments use a fixed LLM configuration, unified prompt template, and shared termination criteria across all runs. Agent initialization, task semantics, and communication rules are held constant so that observed differences reflect only communication topology and orchestration structure. Scalability is evaluated at network sizes $n \in \{4, 8, 16, 50, 100\}$, allowing analysis of both small-network reasoning behavior and large-scale coordination breakdown.

For every run, we record success outcome, rounds to convergence, total token consumption, and wall-clock runtime. These metrics are reported in Table 13 and analyzed in Section 4.2.1. The appendix focuses on explaining the qualitative mechanisms behind those numeric trends.

D.2 Rounds-to-Convergence Policy

AGENTSNET enforces a minimum interaction budget of eight rounds to ensure sufficient message exchange even for small or sparse topologies. Consequently, some runs converge in fewer than eight effective rounds but are still executed under the minimum budget. For global agreement tasks (CONSENSUS and LEADERELECTION) and for settings with $n > 16$, the maximum horizon is instead determined by the network diameter D, using

$$T = 2D + 1,$$

which guarantees complete information propagation under synchronous message passing. Under this policy, large-scale runs may require between fifteen and nineteen rounds depending on topology diameter. This rule explains why convergence rounds do not monotonically increase with n and why some large networks converge faster than smaller ones when topology permits early stabilization.

D.3 Task-Specific Scoring and Visual Encoding

We adopt the AGENTSNET benchmark [13] for the task definitions and evaluation criteria. With the exception of MATCHING, all tasks are evaluated using binary success criteria, reflecting whether the distributed constraint is globally satisfied at termination. Only MATCHING admits a graded notion of partial correctness by design.

Matching (Maximal Matching). In the MATCHING task, agents attempt to form pairwise agreements with neighboring agents. A match is considered valid if agent u selects agent v and agent v selects agent u. Inconsistencies arise when selections are non-reciprocal, invalid (non-neighbor), or when two adjacent agents both select `None` despite being able to form a pair. Let I denote the number of inconsistent agents. Following AGENTSNET, the final score is computed as

$$\text{Score} = 1 - \frac{I}{|V|}.$$

This formulation yields a continuous score in $[0, 1]$, capturing partial correctness even when a globally maximal matching is not achieved. Visualizations encode reciprocal matches in green, one-sided selections in orange, inconsistent agents in red, and unused edges in gray.

Consensus. In the CONSENSUS task, each agent selects a binary value from $\{0, 1\}$. The task is successful if and only if all agents output the same value at termination. Formally, letting $A(u)$ denote the output of agent u, success is defined as

$$\exists b \in \{0, 1\} \text{ s.t. } \forall u \in V,\ A(u) = b.$$

The score is therefore binary. Visualizations depict agent states and communication links, where green links indicate agreement between neighboring agents and red links indicate disagreement.

Coloring ($(\Delta + 1)$-Coloring). In the COLORING task, each agent selects a color from a predefined palette of size $\Delta + 1$, where Δ is the maximum node degree. A run is successful if the resulting assignment constitutes a valid coloring, i.e.,

$$\forall (u, v) \in E,\ \text{color}(u) \neq \text{color}(v).$$

Success is binary. Visualizations highlight conflict-free edges in green and color collisions in red, revealing how conflicts cluster under different communication structures.

LeaderElection. In the LEADERELECTION task, agents must collectively designate exactly one leader. Each agent outputs either `Yes` (leader) or `No` (follower). Let $A(u)$ denote the response of agent u. A run is successful if

$$\sum_{u \in V} \mathbf{1}[A(u) = \texttt{Yes}] = 1.$$

This task is evaluated using a binary success criterion. Visualizations mark the elected leader in green, competing leaders in red, and followers in gray, exposing symmetry-breaking failures.

VertexCover (Minimal Vertex Cover). In the VERTEXCOVER task, agents decide whether they are members of a coordinating set. Let $C \subseteq V$ denote the set of agents selecting `Yes`. A run is successful if C forms a valid vertex cover, i.e.,

$$\forall (u, v) \in E,\ u \in C \text{ or } v \in C.$$

Following AGENTSNET, minimality is assessed qualitatively through visual inspection rather than through a graded score; quantitative evaluation reports binary success. Visualizations distinguish minimal cover nodes (green), non-minimal but valid selections (blue), invalid responses (orange), and uncovered edges (red).

D.4 Visualization-Driven Interpretation

Figures in this appendix present complete visual outputs for all tasks, topologies, and network sizes, revealing how coordination succeeds or fails at the agent level.

Consensus (Figure 4). Fully connected communication maintains unified agreement across all scales, with all agents converging to a single value and no persistent disagreement links even at $n = 100$. Small-world and scale-free topologies preserve partial coherence at small and mid-scale, where agreement clusters remain connected through shortcut edges or hubs, but increasingly fragment as n grows. At large scale, disagreement links form stable boundaries between local clusters, explaining why these topologies incur high communication cost without achieving full success. Delaunay graphs exhibit the strongest fragmentation: agreement remains confined to small geometric neighborhoods, and isolated disagreement clusters persist even after the maximum round budget, directly corresponding to the systematic failure trends observed in Table 13.

Coloring (Figure 5). Scale-free and small-world topologies maintain stable chromatic separation across scales, with conflicts resolved early and remaining localized even as the network grows. The visualizations show that hub-mediated diffusion in scale-free graphs allows color constraints to propagate efficiently, preventing large-scale cascades of conflicts. In contrast, Delaunay graphs increasingly exhibit localized conflict regions at higher n, where geometric isolation delays conflict resolution and produces persistent red edges, consistent with the higher convergence cost and reduced stability reported in the quantitative results.

Matching (Figures 6–7). Diffusion-friendly topologies preserve reciprocal agreement at moderate scale, with most matches forming clean, mutually consistent pairs. As network size increases, small-world and scale-free graphs begin to show isolated pockets of one-sided selections, but these remain relatively sparse. In contrast, geometric Delaunay graphs degrade sharply: one-sided selections and inconsistent agents proliferate across the network, forming dense regions of orange and red nodes. These visual patterns explain why Matching success declines and token cost increases substantially for geometric topologies, even when rounds are allowed to scale.

LeaderElection (Figure 8). LeaderElection exhibits a distinct visual regime. Diffusion-based topologies consistently generate multiple competing leader candidates that persist across rounds, forming stable red clusters that are not resolved by increased communication. Geometric locality, while costly, enables eventual suppression of competing leaders, producing a single green leader at larger scales after extended coordination. Hierarchical orchestration further accelerates symmetry breaking at small and medium scales, but visualizations reveal increasing instability at large n, where depth-induced delays allow competing leader branches to emerge, aligning with the degradation observed in the table.

VertexCover (Figure 9). Small-world and scale-free topologies converge toward compact vertex covers, with most edges covered by a small set of coordinating nodes and limited over-selection. As scale increases, these topologies maintain relatively structured cover patterns, explaining their higher success despite moderate communication cost. In contrast, Delaunay and hierarchical structures frequently over-cover, selecting excessive coordinating nodes, or leave uncovered edges at large n. The visualizations reveal that geometric locality and hierarchical bottlenecks prevent agents from globally coordinating coverage decisions, leading to inefficiencies and constraint violations consistent with the quantitative breakdown.

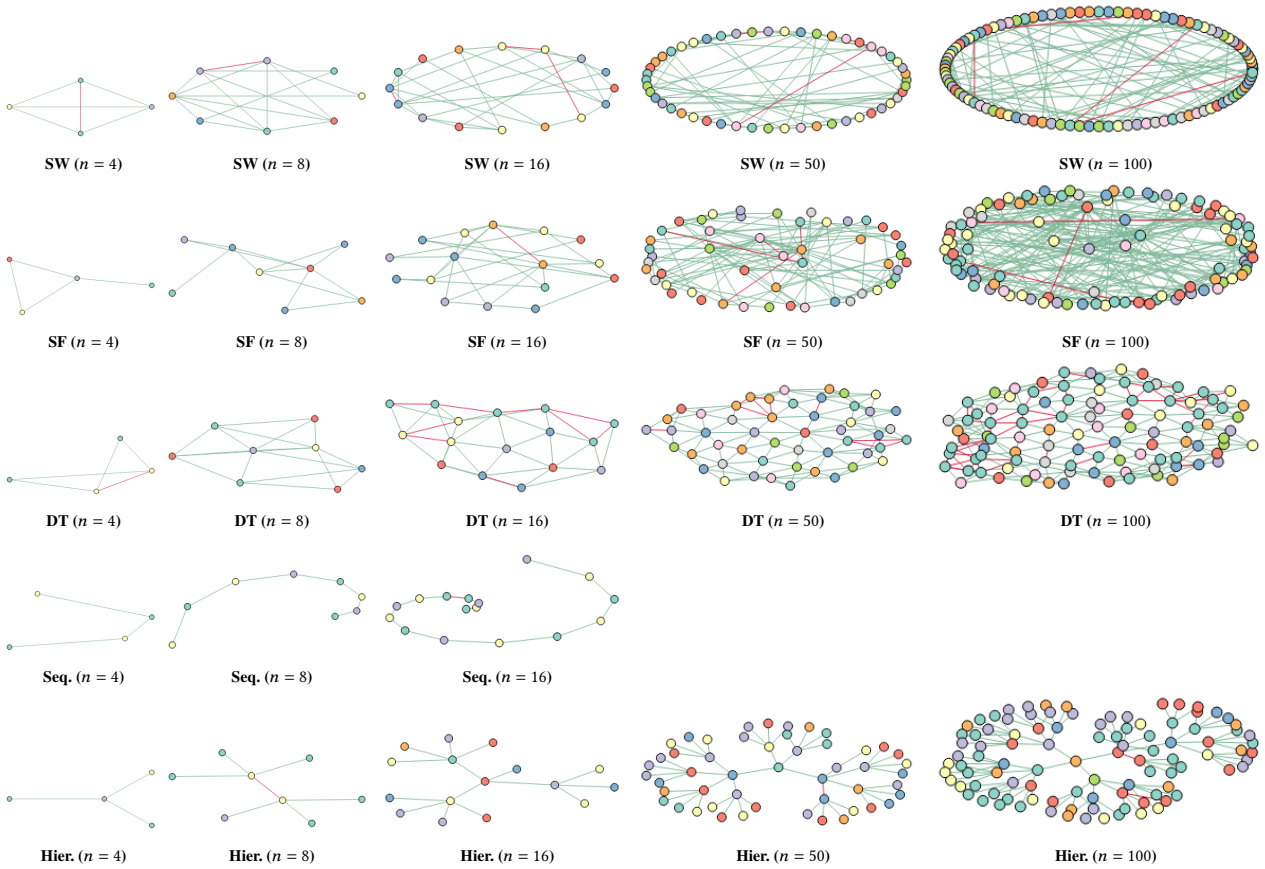


Figure 5: Coloring experiment outcomes across different network sizes ($n = 4$ to 100). Each subfigure shows the final group assignments of agents. Valid assignments (neighbors in different groups) are shown in green, while conflicts (neighbors in the same group) are shown in red.

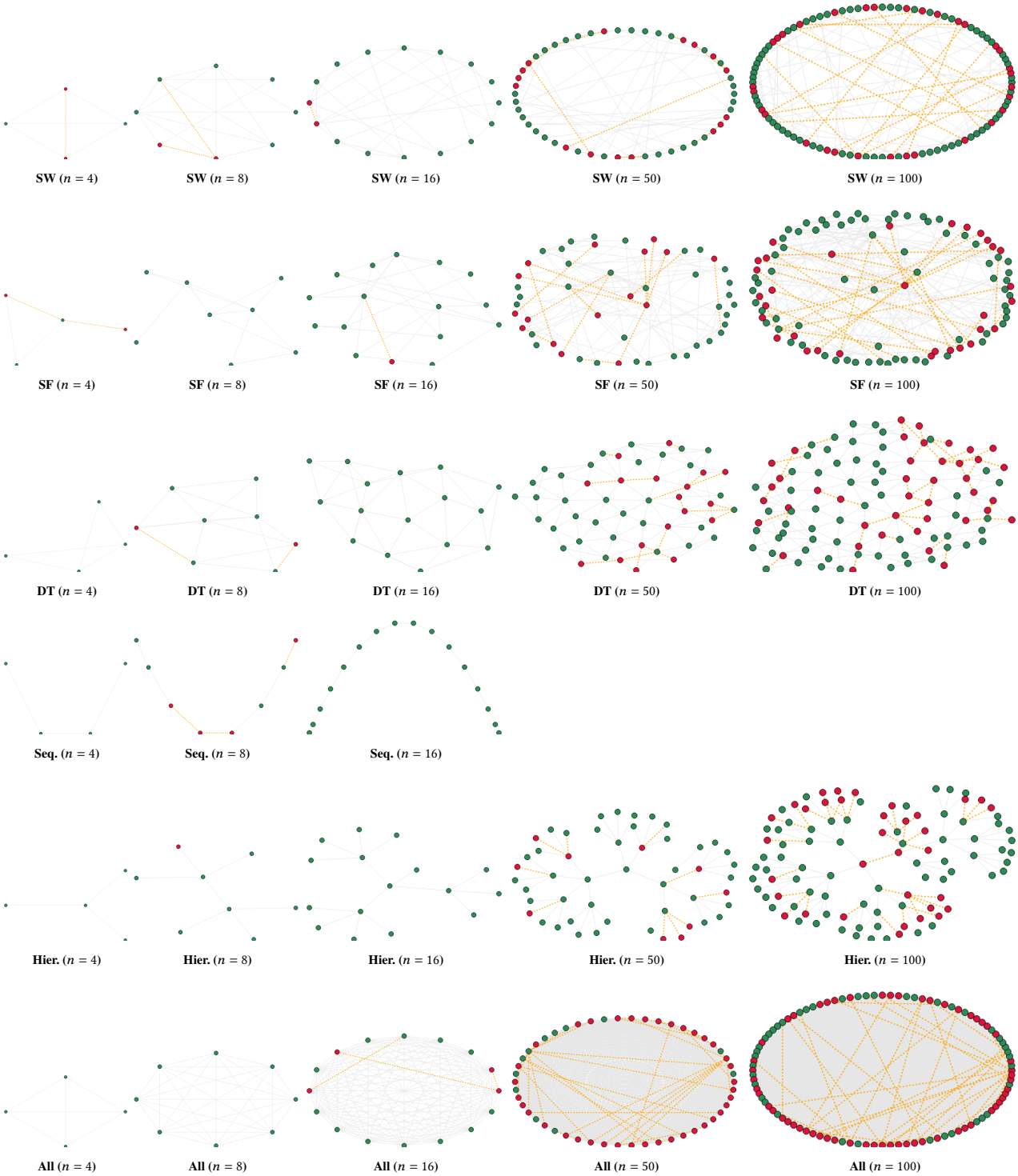


Figure 6: Matching experiment outcomes using the *original scoring model*. Each agent selects one of its neighbors (or "None") to form a pair. Green nodes denote agents whose selections are locally consistent—that is, they selected a valid neighbor who reciprocated the choice. Red nodes represent agents involved in inconsistencies, such as choosing a non-neighbor, selecting a non-reciprocating partner, or forming idle pairs where both neighbors chose "None." The overall score corresponds to the fraction of locally consistent agents, following the computation described in the text.

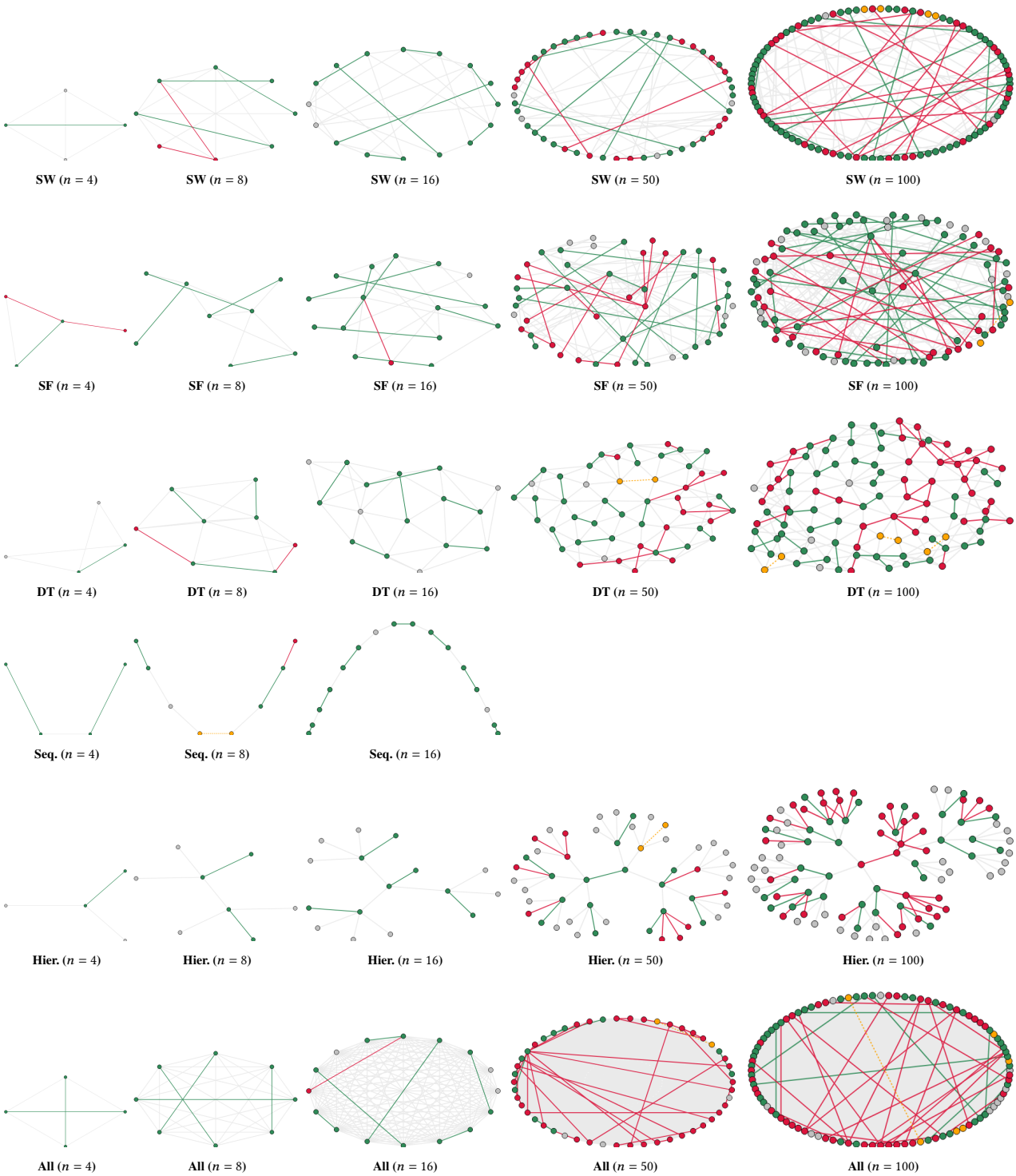


Figure 7: Matching experiment outcomes across different network sizes ($n = 4$ to 100). Each subfigure shows the final pair assignments of agents based on the enhanced local-consistency model. Green edges connect mutually consistent agents whose choices satisfy the neighbor-reciprocation rule; orange dotted edges denote one-sided or invalid selections; and red nodes highlight agents involved in any local inconsistency. Gray edges represent structural links that did not participate in the matching process. This visualization extends the local scoring logic to explicitly reveal consistent and conflicting interactions.

Figure 8: Leader election results across different network sizes ($n = 4$ to 100). Each subfigure shows the final decisions of agents. Green nodes represent the correctly elected leader, red nodes indicate multiple leaders, and gray nodes represent followers.

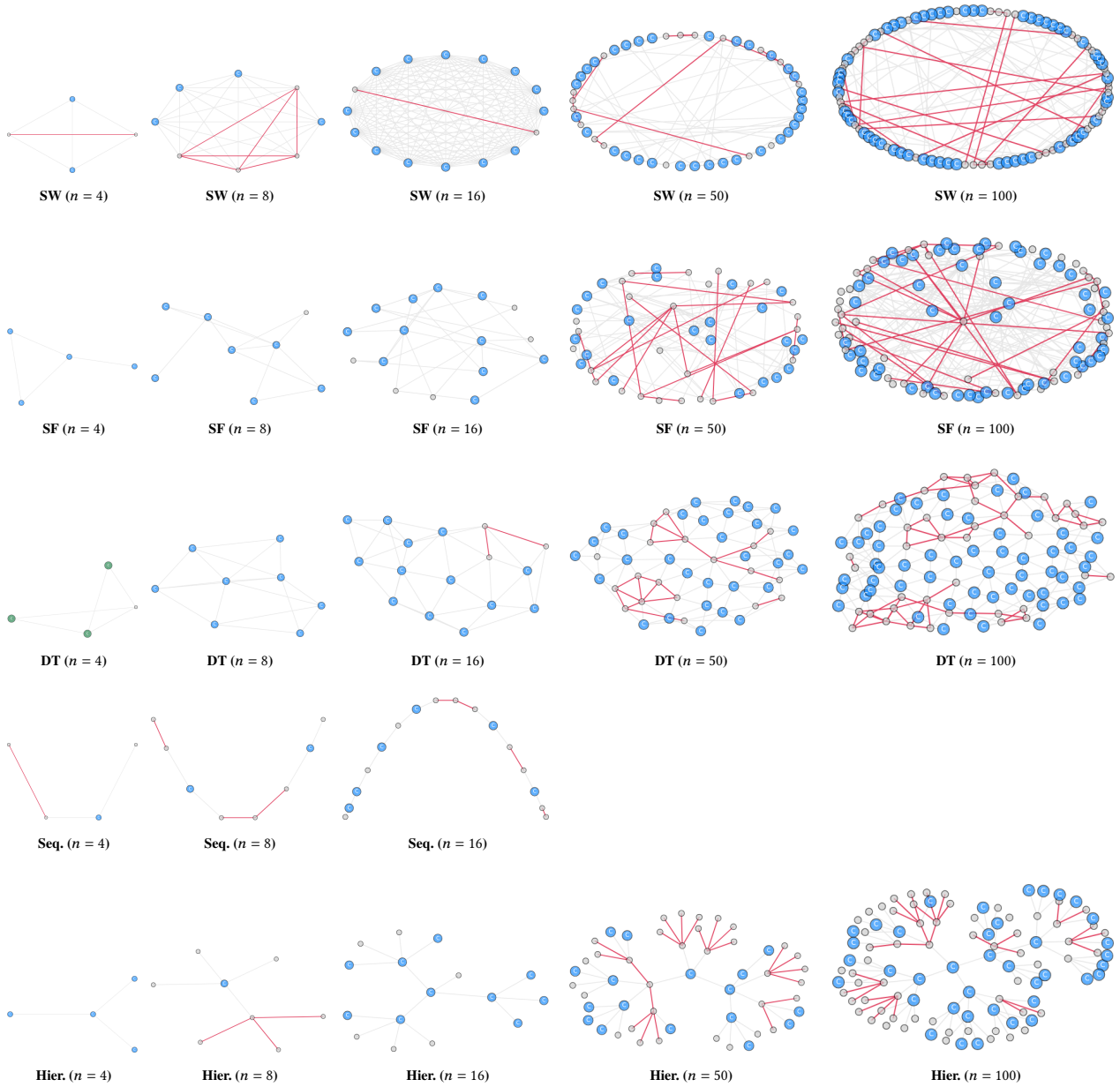


Figure 9: Vertex cover results across different network sizes ($n = 4$ to 100). Each subfigure shows the agents' final decisions. Green nodes represent a minimal valid cover, blue nodes indicate members of a non-minimal cover, red edges mark uncovered pairs, orange nodes represent invalid responses, and gray nodes are non-cover agents.