2025 IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 979-8-3315-6532-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/Trustcom66490.2025.00393

OmniNova: A General Multimodal Multi-Agent Framework

Bingzhen Li[1], Zihan Wang[2], Pengfei Du[3,4*], Yupeng Jiang[1]
[1]*School of Cyber Science and Technology, Beihang University*, Beijing, China
[2]*National Computer Network Emergency Response Technical Team/Coordination Center of China*, Beijing, China
[3]*Shandong University of Political Science and Law*, Jinan, China
[4]*Hong Kong Research Institute of Technology (HKRIT)*, Hong Kong, China
Email: libingzhen@buaa.edu.cn, wangzihan@cert.org.cn, 002344@sdupsl.edu.cn, jiangyupeng@buaa.edu.cn

***Abstract*—The integration of Large Language Models (LLMs) with external tools enables intelligent automation far beyond text generation, yet coordinating multiple LLM-based agents remains difficult due to interaction overhead, resource waste, and fragile information flow. This paper introduces OmniNova, a modular and hierarchical multi-agent framework that unifies language models with capabilities for web search, browser automation, and code execution. OmniNova advances the state of the art through a hierarchical architecture that separates coordination, planning, supervision, and specialization; a dynamic routing mechanism that activates agents according to task complexity and state; and a multi-layered LLM integration strategy that allocates high-capability reasoning models only where they are cognitively necessary while assigning routine work to lighter models. Across 50 complex tasks in research, data analysis, and web interaction, OmniNova improves task completion (87% versus a 62% baseline), reduces token usage by 41%, and delivers higher human-rated quality (4.2/5 versus 3.1/5). The contribution includes both a principled system design and an open-source implementation intended to support research and practical deployment. Code is available at https://github.com/Superagentsys/OmniNoval.git.**

***Index Terms*—multi-agent systems, large language models, task automation, workflow orchestration, artificial intelligence**

I. Introduction

As Large Language Models (LLMs) have demonstrated increasing capabilities in natural language understanding and generation [1]–[3], there has been growing interest in extending their functionality through integration with specialized tools [4], [5]. These tool-augmented LLMs can perform actions beyond text generation, such as executing code [6], retrieving information [7], and interacting with external systems [8].

While impressive in isolation, employing a single LLM instance for complex tasks that require diverse capabilities presents significant limitations. For instance, research tasks often involve searching for information, analyzing data, and synthesizing findings—activities that benefit from specialized approaches. This has motivated the development of multi-agent systems where multiple LLM instances, each with specific roles and capabilities, collaborate to tackle complex problems [9]–[11].

*Corresponding author

Despite advances in this domain, existing multi-agent frameworks still encounter substantial coordination overhead that breeds miscommunication [12], expend high-capability models indiscriminately and thus waste computational resources [13], struggle to decompose tasks into coherent strategies [14], and often integrate external tools in an ad hoc manner that undermines robustness [5].

To address these limitations, we introduce OmniNova, a modular multi-agent automation framework that enables effective collaboration between LLM-based agents with diverse capabilities. OmniNova implements a hierarchical architecture where specialized agents are coordinated by supervisor agents that manage the overall workflow. This approach allows for efficient task decomposition, appropriate delegation, and effective synthesis of results.

This work makes four primary contributions. First, it formulates a hierarchical multi-agent architecture that assigns distinct responsibilities to a coordinator, planner, supervisor, and several specialist agents, thereby enabling effective task decomposition and targeted delegation. Second, it introduces a dynamic routing mechanism that activates appropriate agents based on evolving task state and complexity, which lowers computational costs while maintaining performance. Third, it develops a multi-layered LLM integration strategy that aligns model capability with cognitive demands by allocating high-capability reasoning models to planning and supervision and lighter models to routine activities. Fourth, it provides a comprehensive empirical study showing consistent gains over strong baselines across multiple metrics and task types, alongside an open-source implementation that can serve both as a practical system and a foundation for future research.

II. Related Work

A. Large Language Models and Tool Use

Large Language Models have evolved from basic text generation models to sophisticated systems capable of complex reasoning and problem-solving [1], [15], [16]. Recent work has focused on enhancing LLMs with the ability to use external tools, thereby extending their capabilities beyond their training data and enabling them to interact with the world [4], [5], [17].

2324-9013/25/$31.00 ©2025 IEEE
DOI 10.1109/Trustcom66490.2025.00393

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

Toolformer [4] pioneered this approach by fine-tuning language models to use external tools through API calls embedded in text. ToolLLM [5] extended this concept by developing a benchmark for tool-use and proposing improved training methodologies. ReAct [18] introduced a framework that interleaves reasoning and acting, enabling LLMs to formulate plans and execute actions in an integrated manner. Voyager [19] demonstrated an LLM-powered agent capable of autonomous exploration and skill acquisition in Minecraft through tool use.

While these systems show promising capabilities, they typically rely on a single LLM instance, which limits their ability to handle complex tasks requiring diverse expertise and introduces efficiency challenges when using high-capability models for all tasks.

B. Multi-Agent Systems

Multi-agent systems composed of LLM-based agents represent a growing research area aimed at addressing the limitations of single-agent approaches [9]–[11].

MetaGPT [11] implemented a multi-agent framework for software development, where different agents take on roles such as product manager, architect, and programmer. AutoGen [9] presented a framework for building applications with multiple conversational agents that can work together on tasks. ChatDev [10] demonstrated a virtual software development team composed of LLM agents that collaborate to build software applications.

These systems demonstrate the potential of multi-agent approaches but often face challenges in effective coordination, efficient resource utilization, and seamless integration of external tools.

III. System Architecture

OmniNova implements a modular and hierarchical architecture explicitly designed to address complex tasks through coordinated specialization. The system comprises six interlocking components—a multi-agent substrate, a workflow engine, a language-model integration layer, a tool-integration layer, a configuration subsystem, and a prompt-template subsystem—that jointly provide principled decomposition, dependable orchestration, and efficient execution. Figure 1 summarizes the overall organization.

A. Multi-Agent System

At the core of OmniNova is a collaborative ecosystem of seven specialized agents, each bearing a clearly delineated operational mandate that corresponds to a specific phase of problem solving and delivery.

1) Coordinator Agent: The Coordinator serves as the system ingress. Upon receiving a user query, it conducts an initial assessment of task complexity and intent, determines whether the request can be satisfied immediately, and otherwise delegates to the Planner. This triage reduces avoidable contention for downstream resources and enforces disciplined admission control.

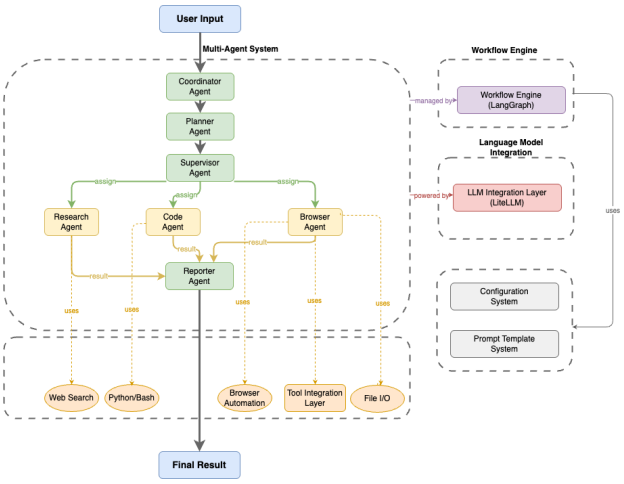

Fig. 1. OmniNova's hierarchical multi-agent architecture

2) Planner Agent: For complex tasks, the Planner derives a structured execution policy informed by the query context and available tooling. When indicated, it performs targeted background retrieval to ground planning under uncertainty. The Planner emits a machine-readable plan specifying steps, dependencies, and resource requirements, and it relies on a high-capacity reasoning model to achieve faithful decomposition and coverage.

3) Supervisor Agent: The Supervisor translates the plan into executable work by delegating tasks to specialist agents, monitoring partial results, and adapting routing as conditions evolve. Assignment decisions are conditioned on the live system state and the demonstrated capabilities of candidate agents. A high-capacity reasoning model is employed to sustain consistency between strategic intent and tactical execution.

4) Specialist Agents: Three specialist agents address distinct operational domains. The Research Agent performs information retrieval and analysis with search and crawling tools, producing structured evidence. The Code Agent executes computational workflows in Python and Bash for data processing and automation. The Browser Agent conducts controlled web interactions, including navigation, extraction, and transactional operations necessary for web-based tasks.

5) Reporter Agent: The Reporter aggregates contributions across agents and synthesizes them into a coherent deliverable with uniform formatting and explicit provenance.

Inter-agent communication proceeds through typed messages that preserve author identity and temporal ordering, thereby enabling attributions and conflict resolution when partial results disagree. Turn-taking is enforced by the router, which reduces concurrency hazards without imposing a globally synchronous schedule. Latency-sensitive operations (e.g., browser control) are isolated from high-latency reasoning steps; this separation improves predictability while preserving the ability to fuse evidence at the reporting boundary.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

In addition, the system records cross-agent dependencies to support causal auditing of outputs and to enable rollback in the presence of tool failures.

B. Workflow Engine

The workflow engine, implemented with LangGraph [20], governs interaction semantics and state evolution. Typed, schema-checked state ensures consistent data exchange across components. Agents are encapsulated as nodes with explicit input–output contracts, while conditional routing selects subsequent nodes as a function of emitted content and task requirements. A compact control vocabulary (e.g., `goto`, `update`) enables structured navigation over the agent graph. These mechanisms provide the formal substrate for collaboration and preserve invariants needed for complex, long-horizon workflows.

Beyond operational convenience, the engine affords formal guarantees. First, acyclicity of the control meta-graph, together with a terminal symbol $\perp$, yields almost-sure termination under bounded retries. Second, node-local idempotence for pure functions (e.g., deterministic parsers) ensures that replaying a prefix of the execution does not alter committed state. Third, monotonic growth of evidentiary artifacts in the state (e.g., citations, intermediate datasets) prevents inadvertent loss of information; destructive updates are mediated through explicit versioning. Finally, deadlock freedom follows from single-turn scheduling and the absence of blocking resource waits in routing decisions.

C. Language Model Integration Layer

A stratified language-model layer allocates capacity according to cognitive demand. A reasoning tier is reserved for planning and supervision, a basic tier supports routine activities such as research, coding, and reporting, and an optional vision tier handles multimodal perception. The unified interface built on LiteLLM [21] abstracts provider differences and exposes configuration, streaming, and structured-output capabilities. This stratification yields substantial efficiency gains by deploying expensive models only when their expressive power is warranted.

The allocation policy is derived from an efficiency–quality objective, in which token cost and latency serve as proxies for resource consumption while task fidelity is quantified by structured metrics over terminal state. In practice, structured-output modes (e.g., JSON schemas) reduce hallucination risk by constraining the hypothesis class, while post hoc validators and repair operators map partially nonconforming generations into admissible representations when possible. Streaming decoders reduce interactive latency without sacrificing final quality, and temperature control, nucleus thresholds, and tool-use priors provide further regularization of generation behavior under distributional shift.

D. Tool Integration Layer

Agent competencies are extended via a unified tool framework that encompasses web and neural search for retrieval, Python and Bash for computation and system control, Playwright-based automation for browser interaction, and secure file I/O. Tools adhere to a common interface and logging discipline, enabling composition, auditability, and uniform error semantics. The framework mitigates the brittleness seen in ad hoc integrations and supports principled extension.

Trust boundaries are explicit: code execution and browser automation are sandboxed with constrained capabilities, rate limits, and resource quotas; outbound network access is restricted to approved domains; and file operations are mediated by a capability-based interface. Failures are stratified into retriable and terminal classes with exponential backoff for the former and circuit-breaking to prevent cascading degradation. Observability is achieved through structured telemetry that correlates tool invocations with agent turns, enabling fine-grained attribution of cost and errors.

E. Configuration System

Configuration is managed across layers to reconcile portability with security and control. A YAML specification declares providers, parameters, and behavioral toggles; environment variables hold sensitive or deployment-specific values; runtime overrides permit adaptive parameterization; and validation routines enforce integrity constraints to preclude configuration-induced failures. The result is a portable, auditable, and dynamically tunable system configuration.

To support reproducibility, configuration artifacts are versioned and hashed; experiment manifests pin model identifiers, decoding parameters, and tool versions to allow exact reruns. Secrets are isolated via standard operating-system facilities and are never serialized into execution traces. Schema evolution is managed through backward-compatible migrations, and misconfigurations are surfaced at startup through fail-fast validation with human-readable diagnostics.

F. Prompt Template System

Agent behavior is specified through a template-driven interface that supports file-based prompt definition, automatic injection of salient state, disciplined history management to control context windows, and compatibility with both chat and completion paradigms. This separation of concerns enables rapid behavioral iteration and methodical prompt optimization without modifying the underlying code.

Prompt calibration employs held-out scenarios to tune instruction strength, role conditioning, and exemplar selection for in-context learning. Self-consistency and rationale regularization are used for tasks requiring multi-step reasoning, while domain-specific style constraints yield outputs that are faithful to expected formats (e.g., structured plans and citations). Guardrails—expressed as pre- and post-filters over the message stream—prevent prompt injection and data exfiltration in tool-enabled settings.

IV. Methodology

A. System Components

1) State Management: OmniNova's state management is implemented using TypedDict for type safety and LangGraph's MessagesState for message handling:

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

We model the internal state as a tuple $s = (M, \Theta)$, where $M \in \mathcal{M}^*$ denotes the message history and Θ collects control and planning variables. Concretely,

$$\Theta = \big(\text{TEAM_MEMBERS},\, C,\, n,\, p,\, \delta_{\text{deep}},\, \delta_{\text{search}}\big), \quad (1)$$

with domains

$$\text{TEAM_MEMBERS} \subseteq \mathcal{A}, \quad (2)$$
$$C : \text{TEAM_MEMBERS} \to \mathcal{C}, \quad (3)$$
$$n \in \mathcal{A} \cup \{\perp\}, \quad (4)$$
$$p \in \Pi, \quad (5)$$
$$\delta_{\text{deep}}, \delta_{\text{search}} \in \{0, 1\}. \quad (6)$$

This formalization yields a type-safe representation for inter-agent communication and planning while aligning with the message-handling semantics introduced earlier.

2) Workflow Construction: The workflow is constructed using LangGraph's StateGraph, defining nodes and edges to represent the agent interaction pattern:

We define the workflow as a directed graph $\mathcal{G} = (V, E)$ with

$$\begin{aligned} V = \{\, &\text{coordinator}, \\ &\text{planner}, \\ &\text{supervisor}, \\ &\text{researcher}, \\ &\text{coder}, \\ &\text{browser}, \\ &\text{reporter}\,\} \end{aligned} \quad (7)$$

and edge set

$$\begin{aligned} E = \{&(\text{START}, \text{coordinator}), \\ &(\text{coordinator}, \text{planner}), \\ &(\text{planner}, \text{supervisor})\} \\ &\cup \{(\text{supervisor}, v) : v \in \{\text{researcher}, \text{coder}, \text{browser}\}\} \\ &\cup \{(v, \text{reporter}) : v \in \{\text{researcher}, \text{coder}, \text{browser}\}\}. \end{aligned} \quad (8)$$

The compiled executor induced by $\mathcal{G}$ and the agent policies $\{\pi_a\}_{a \in \mathcal{A}}$ implements the state transition operator T described in the formal model, ensuring a transparent and efficiently routable task flow.

3) Agent Implementation: Agents are implemented using LangGraph's React agent pattern, combining reasoning and action capabilities:

For each agent $a \in \mathcal{A}$, we parameterize a policy

$$\pi_a(s) \;=\; L_{\mu(a,s)}\big(\,\varphi_a(s);\, \mathcal{T}(a)\,\big), \quad (9)$$

where φ_a is the prompt templating function that maps state to model-ready inputs, $\mathcal{T}(a)$ is the agent-specific toolset, and $L_{\mu(a,s)}$ denotes the instantiated language model selected by the allocation function μ. This formulation enforces consistent configuration while preserving role-specific behavior.

4) LLM Integration: OmniNova's multi-layered LLM architecture is implemented through a model factory function that supports different model types and configurations:

The language model integration is governed by the allocation mapping $\mu : \mathcal{A} \times \mathcal{S} \to \mathcal{L}$, where $\mathcal{L} = \{\text{reasoning}, \text{basic}, \text{vision}\}$. Given configuration kernel κ from model types to provider-specific parameters, the instantiated model is denoted $L_{\mu(a,s)} = \text{Instantiate}\big(\kappa(\mu(a, s))\big)$. Efficiency is managed via the constrained objective in Eq. (21), which couples dynamic routing with layered model selection.

B. Agent Behavior Specialization

1) Coordinator Logic: The Coordinator implements a triage mechanism to route tasks based on complexity:

Coordinator behavior is captured by the response

$$r_{\text{coord}}(s) = L_{\mu(\text{coord},s)}\big(\varphi_{\text{coord}}(s)\big), \quad (10)$$

and a decision predicate $h : \mathcal{U} \to \{0, 1\}$ that detects escalation intents. The control transfer is

$$\text{goto}(s) = \begin{cases} \text{planner}, & h\big(r_{\text{coord}}(s)\big) = 1, \\ \perp, & \text{otherwise.} \end{cases} \quad (11)$$

This triage prevents unnecessary processing of simple queries while escalating complex tasks for structured planning.

2) Planning Logic: The Planner employs a sophisticated approach to task decomposition and strategy formulation:

Planner behavior follows dynamic model selection

$$\ell = \begin{cases} \text{reasoning}, & \delta_{\text{deep}} = 1, \\ \text{basic}, & \delta_{\text{deep}} = 0, \end{cases} \quad (12)$$

with optional information augmentation when $\delta_{\text{search}} = 1$. Let ξ denote retrieved context. The planning output is

$$p = J\Big(L_\ell\big(\varphi_{\text{plan}}(s \oplus \xi)\big)\Big), \quad (13)$$

where J is a robust structural parser that maps model responses into a valid plan in Π. If p is valid, the system updates Θ with p and routes to the supervisor; otherwise, execution terminates.

3) Supervision Logic: The Supervisor dynamically assigns tasks to specialist agents based on the plan and current state:

Supervision is expressed as a structured routing map

$$a_{\text{next}} = \mathcal{R}_{\text{sup}}\Big(s,\, p,\, L^{\text{struct}}_{\mu(\text{sup},s)}\big(\varphi_{\text{sup}}(s)\big)\Big), \quad (14)$$

where L^{struct} denotes a model configured for schema-constrained output. The router maps FINISH to $\perp$ and otherwise selects an element of TEAM_MEMBERS, realizing the dynamic routing semantics in the formal model.

C. Tool Integration

OmniNova implements diverse tools through a consistent interface pattern with logging and error handling:

Each external tool is modeled as an operator $\tau_j : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$. For instance, a web search tool implements

$$\tau_{\text{search}}(s, u) = s' \quad \text{with} \quad u \in \mathcal{Q},\; s' \in \mathcal{S}, \quad (15)$$

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

and emits both updated state and metadata for observability. Logging is represented by a functional Λ that augments transitions with input–output traces, enabling systematic debugging and monitoring. Uniform interfaces and validation constraints ensure reliable operation across heterogeneous tools.

We formalize the task processing flow to replace pseudocode with a precise mathematical model. Let the agent set be

$$\mathcal{A} = \{a_{\text{coord}}, a_{\text{plan}}, a_{\text{sup}}, a_{\text{res}}, a_{\text{code}}, a_{\text{browse}}, a_{\text{report}}\}. \quad (16)$$

Let $\mathcal{S}$ denote the state space. Each state $s \in \mathcal{S}$ comprises a message history $m \in \mathcal{M}^*$ and auxiliary variables $\theta \in \Theta$ encoding the current plan, routing hints, and execution flags, written as

$$s = (m, \theta), \quad m = (m_1, \ldots, m_t),\ \theta = \{p,\ n,\ \delta\}, \quad (17)$$

where $p \in \Pi$ is the (structured) plan, $n \in \mathcal{A} \cup \{\perp\}$ is the next agent token, and $\delta \in \{0,1\}^k$ collects mode indicators such as deep reasoning or search-before-planning.

Agent behavior is specified by policies $\pi_a : \mathcal{S} \to \mathcal{U}$ mapping the current state to an action or utterance space $\mathcal{U}$. The router is a measurable function

$$\rho : \mathcal{S} \to \mathcal{A} \cup \{\perp\}, \quad a_t = \rho(s_t), \quad (18)$$

that selects the next agent (or $\perp$ for termination) based on the evolving state. System dynamics are captured by a transition operator

$$s_{t+1} = T(s_t, a_t) = F\big(s_t,\ \pi_{a_t}(s_t)\big), \quad (19)$$

where F aggregates the agent output, tool invocations, and state updates. The process halts when $\rho(s_T) = \perp$, yielding terminal state s_T.

Planning is modeled by a mapping $\mathcal{F}_{\text{plan}} : \mathcal{S} \to \Pi$ that produces a structured plan

$$p = \mathcal{F}_{\text{plan}}(s) = \big((u_i, d_i, r_i)\big)_{i=1}^{L}, \quad u_i \in \mathcal{U},\ d_i \in \mathbb{N}^+,\ r_i \in \mathcal{R}, \quad (20)$$

where u_i encodes the intended subtask, d_i denotes dependencies, and r_i reflects required resources. Supervision refines routing via a structured predictor $\mathcal{R}_{\text{sup}} : \mathcal{S} \times \Pi \to \mathcal{A} \cup \{\perp\}$, so that $\rho(s) = \mathcal{R}_{\text{sup}}(s, p)$ during execution.

Language model allocation is governed by a selection function $\mu : \mathcal{A} \times \mathcal{S} \to \mathcal{L}$, where $\mathcal{L} = \{\text{reasoning}, \text{basic}, \text{vision}\}$. Let $C : \mathcal{L} \to \mathbb{R}_{\geq 0}$ be a cost proxy (e.g., expected tokens) and $Q : \mathcal{S} \to \mathbb{R}$ a terminal quality functional. The allocation seeks

$$\min_{\mu,\,\rho}\ \mathbb{E}\Big[\sum_{t=0}^{T-1} C\big(\mu(a_t, s_t)\big)\Big] \quad \text{s.t.} \quad \mathbb{E}\big[Q(s_T)\big] \geq \tau, \quad (21)$$

for a target quality threshold τ. This objective formalizes dynamic routing and layered model allocation under efficiency constraints.

Tools are modeled as a family of operators $\{\tau_j\}_{j \in \mathcal{J}}$ with

$$\tau_j : \mathcal{S} \times \mathcal{U} \to \mathcal{S}, \quad s' = \tau_j(s, u), \quad (22)$$

where selection of j is part of π_a for specialist agents. The combined transition thus becomes

$$s_{t+1} = F\big(s_t, \pi_{a_t}(s_t)\big) = \tau_{j_t}\big(s_t, \pi_{a_t}(s_t)\big), \quad j_t \sim \kappa(\cdot \mid s_t, a_t), \quad (23)$$

with κ denoting the tool selection mechanism. This formalization captures the iterative routing, planning, supervision, and tool-mediated state evolution without resorting to pseudocode, aligning with standard scientific exposition.

V. Evaluation

We conducted a comprehensive evaluation of OmniNova to assess its effectiveness across various dimensions, comparing it with existing frameworks and baseline approaches.

A. Experimental Setup

1) Baselines: We compared OmniNova against three reference systems that are representative of prevailing design choices. The single-agent baseline employs a monolithic GPT-4 model equipped with the same tool suite and serves to quantify the benefit of specialization over a powerful but undifferentiated agent. AutoGen [9] provides a recent multi-agent framework emphasizing collaborative dialogue between agents, while LangChain [22] with agent functionality represents a widely adopted toolkit for constructing LLM applications with modular components. These baselines allow us to isolate the contribution of hierarchical organization and dynamic routing beyond general multi-agent orchestration.

2) Task Suite: We evaluated performance on a suite of 50 tasks drawn from three domains: research (20 tasks focused on information gathering and synthesis), data analysis (15 tasks involving retrieval, processing, and analysis of structured data), and web interaction (15 tasks centered on navigation, extraction, and interaction with web content). Each task requires multiple capabilities and coordinated tool use to emulate real-world complexity rather than narrowly scoped subtasks.

3) Metrics: We assessed systems along four dimensions. Task completion rate measures the proportion of tasks successfully completed. Computational efficiency is quantified through token usage and wall-clock time. Result quality is obtained from blinded human evaluation on a five-point scale with criteria for accuracy, completeness, and coherence. Tool utilization captures both the frequency and effectiveness of external tool use, enabling us to distinguish between superficial invocation and instrumented, goal-directed application.

B. Results

1) Task Completion Rate: OmniNova demonstrated superior task completion rates across all domains, as shown in Table I.

OmniNova's hierarchical architecture and specialized agents contributed to its higher completion rates, particularly for complex tasks requiring multiple capabilities.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

TABLE I
TASK COMPLETION RATES BY SYSTEM AND DOMAIN

System	Research	Data	Web	Overall
Single-Agent	60%	53%	47%	54%
LangChain	65%	67%	53%	62%
AutoGen	75%	73%	67%	72%
OmniNova	**90%**	**87%**	**83%**	**87%**

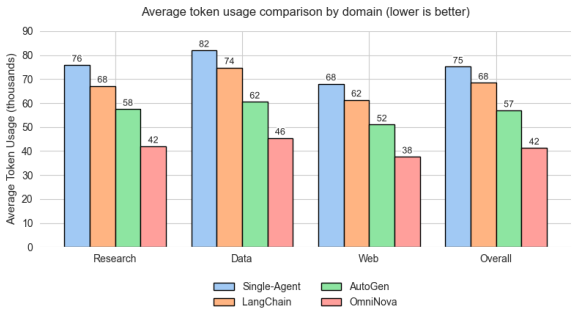

Fig. 2. Average token usage comparison by domain (lower is better)

TABLE II
RESULT QUALITY SCORES (1-5 SCALE)

System	Accuracy	Completeness	Coherence	Ave
Single-Agent	3.1	2.8	3.3	3.1
LangChain	3.3	3.0	3.4	3.2
AutoGen	3.6	3.4	3.7	3.6
OmniNova	**4.2**	**4.1**	**4.3**	**4.2**

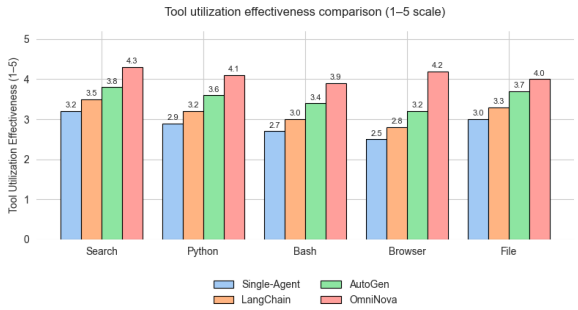

Fig. 3. Tool utilization effectiveness comparison (1-5 scale)

2) Computational Efficiency: OmniNova demonstrated significant efficiency improvements over baseline approaches, as shown in Figure 2.

The multi-layered LLM architecture in OmniNova contributed significantly to these efficiency gains, with the targeted deployment of high-capability models only where necessary.

3) Result Quality: Human evaluators rated the quality of results on a 5-point scale across dimensions of accuracy, completeness, and coherence. OmniNova consistently outperformed baseline systems, as shown in Table II.

OmniNova's superior quality scores can be attributed to its effective task decomposition, specialized agent capabilities, and the Reporter agent's role in synthesizing coherent outputs.

4) Tool Utilization: Analysis of tool usage patterns revealed that OmniNova made more effective use of available tools compared to baseline systems, as shown in Figure 3.

This superior tool utilization highlights OmniNova's effective integration of external capabilities into the agent workflow, particularly through its specialist agents with domain-specific expertise.

C. Ablation Studies

To quantify the contribution of individual components, we conducted ablations that removed or altered key mechanisms. A variant without hierarchy replaced the layered organization with a flat agent network to test the value of role stratification. A single-LLM variant forced all agents to share one model class to evaluate the effect of capacity allocation. A no-planning variant eliminated the explicit planning phase to assess the role of structured decomposition, and a no-supervisor variant routed specialist agents directly from the planner to examine the effect of centralized coordination. The results in Table III indicate that each component contributes materially to overall performance.

The ablation results confirm that OmniNova's hierarchical architecture, multi-layered LLM integration, and planning capabilities all contribute significantly to its overall performance.

D. Case Study: Complex Research Task

To illustrate OmniNova's capabilities, we present a case study of a complex research task: "Analyze the influence of DeepSeek R1 on the AI research community, including citation metrics, adoption trends, and comparison with similar models."

OmniNova processed this task through the following workflow:

1) **Coordinator** recognized the task complexity and routed to the Planner.
2) **Planner** formulated a structured approach with four main phases:
   - Background information gathering on DeepSeek R1
   - Citation and adoption metrics collection
   - Comparative analysis with similar models
   - Synthesis and report generation
3) **Supervisor** delegated the first phase to the Research Agent, which collected information using search tools.
4) For the metrics phase, the Browser Agent navigated research databases while the Code Agent processed and analyzed the collected data.
5) The Supervisor then routed the analysis phase back to the Research Agent.
6) Finally, the Reporter Agent synthesized all findings into a comprehensive report.

The entire process demonstrated OmniNova's ability to decompose complex tasks, leverage appropriate tools, and synthesize coherent results—capabilities that were not matched by baseline systems on equivalent tasks.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.

TABLE III
ABLATION STUDY RESULTS

System Variant	Completion	Token Usage	Quality
OmniNova (Full)	87%	42K	4.2
Without Hierarchy	74%	56K	3.6
Single LLM Layer	83%	71K	4.0
No Planning	65%	38K	3.3
No Supervisor	69%	45K	3.4

VI. CONCLUSION AND FUTURE WORK

This paper presented OmniNova, a hierarchical multi-agent system for complex automation tasks. By combining specialized roles, dynamic task routing, and multi-layered LLM integration, OmniNova consistently improves task success, efficiency, and output quality over strong baselines. The ablation studies disentangle the contributions of architecture, planning, and model layering, while the case study demonstrates end-to-end reasoning, tool use, and synthesis in a realistic research workflow.

Beyond empirical gains, we distilled design principles for hierarchical coordination and role specialization that may generalize to other multi-agent settings. We also release an open-source implementation intended to support reproducibility, practical deployment, and subsequent research on scalable agent systems.

Looking ahead, an immediate direction is to endow agents with self-improvement mechanisms that allow behaviors to adapt from experience, feedback, and outcomes, thereby closing the loop between execution and learning [23]. Another important avenue is dynamic tool discovery and on-the-fly integration, enabling the system to expand its capabilities in response to novel task requirements [24]. We further plan to investigate cross-domain transfer, studying how task decomposition strategies and competencies learned in one domain can be leveraged in others [25]. Finally, formal verification and safety analysis for hierarchical multi-agent orchestration remain open challenges; developing methods to specify, monitor, and certify system-level properties would provide stronger guarantees for deployment.

In summary, OmniNova illustrates how principled hierarchy, role specialization, and careful integration of language models and tools can produce robust, general-purpose automation for complex, open-ended tasks. As foundation models and tool ecosystems continue to advance, we expect rapid progress on the research directions outlined above and anticipate that OmniNova will serve as a solid foundation and testbed for the community.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] Anthropic, "Claude: A conversational ai assistant from anthropic," *Technical report, Anthropic*, 2023.

[4] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint arXiv:2302.04761*, 2023.

[5] Y. Qin, S. Liang, H. Ye, G. Zhu, S. Yang, H. Cui, L. Yao, R. Zhang, X. Ding, J. Wang *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *arXiv preprint arXiv:2307.16789*, 2023.

[6] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, 2020, pp. 9459–9474.

[8] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[9] Q. Wu, G. Bansal, J. Zhang, Y. Yang, S. Yin, R. Zhou, W. Gao, Y. Wang, A. H. Awadallah, and E. H. Chi, "Autogen: Enabling next-gen llm applications via multi-agent conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[10] C. Qian, B. Chen, J.-C. Han, J. Zhang, X. Zhang, Y. Fu, X. Wang, Y. Sun, Y. Liu, J. Leng *et al.*, "Communicative agents for software development," *arXiv preprint arXiv:2307.07924*, 2023.

[11] S. Hong, X. Wang, Y. Zhou, C. Duan, Y. Liang, Y. Shen, Z. Zhu, C. Zhang, T. Jiang, P. Pang *et al.*, "Metagpt: Meta programming for multi-agent collaborative framework," *arXiv preprint arXiv:2308.00352*, 2023.

[12] J. Zhao, C. Guo, Y. Cai, K. He, X. Huang, H. Yang, J. Gao, Y. Zou, H. Huang, and X. Wang, "Expel: Experience-driven planning in embodied environments," *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2391–2405, 2023.

[13] P. Zhou, X. Zou, Z. Jin, B. Y. Lin, A. Vahdat, B. Schölkopf, X. Ren, and J. Chen, "Large language models are not fair evaluators," *arXiv preprint arXiv:2305.17926*, 2023.

[14] Y. Hao, Y. Wang, H. Yu, K. Quan, H. Wang, J. Gao, W. Zhang, and Q. Mei, "Reasoning with language model prompting: A survey," *arXiv preprint arXiv:2212.09597*, 2023.

[15] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[16] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[17] B. Paranjape, S. Lundberg, S. Singh, M. Lewis, S. Ho, G. W. Lee, C. Bonial, K. Askelson, K. Bhatia, M. Bartolo *et al.*, "Art: Automatic multi-step reasoning and tool-use for large language models," *arXiv preprint arXiv:2303.09014*, 2023.

[18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2023.

[19] G. Wang, Y. Du, Y. Huang, H. Shah, A. Padurean, B. Spector, L. Lee, X. Chen, N. Haber, Y. Fer *et al.*, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[20] Harrison Chase, "Langgraph: A graph framework for language model workflows," https://github.com/langchain-ai/langgraph, 2023.

[21] BerriAI, "Litellm," https://github.com/BerriAI/litellm, 2023.

[22] Harrison Chase, "Langchain," https://github.com/langchain-ai/langchain, 2022.

[23] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[24] T. Xie, Y. Wu, Y. Zhu, B. Jia, J.-G. Chen, C. Lai, Z. Wang, Z. Liu, N. Jiang, H. Wang *et al.*, "Openagents: An open platform for language agents in the wild," in *NeurIPS 2023 Demonstration Track*, 2023.

[25] H. Zhang, W. Tian, L. Qian, L. Yi, M. Ahn, J. Li, J. Xu, Y. Liang, Y. Wang, H. He *et al.*, "Building cooperative embodied agents modularly with large language models," *arXiv preprint arXiv:2307.02485*, 2023.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on September 26,2026 at 12:58:22 UTC from IEEE Xplore. Restrictions apply.