Gurpreet Kaur, 2026, 14:4
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Ohmega: A Composable Miniature AI Framework for Drag and Drop Model Assembly

Gurpreet Kaur, Savita, Nishu, Pooja, Aradhay Gupta
Hmr Institute of Technology And Management, Hamidpur, Delhi

Abstract- We propose Ohmega, a composable AI framework that enables developers and end users to assemble pre-built miniature AI modules (micro-agents) via a drag-and-drop interface to construct larger, task-specific systems. These miniature AIs are small, focused components that each implement a single capability, such as text classification, entity extraction, arithmetic computation, API invocation, or simple policy decision-making. By treating such modules as first-class, reusable building blocks, Ohmega allows users to configure and reconfigure workflows without retraining large, monolithic models. The framework supports using modules in isolation or composing them in parallel or in sequence, with optional arbitration and ensemble strategies to aggregate or select outputs when multiple modules contribute to a decision. A central orchestrator manages data flow, enforces composition rules, and records execution traces, promoting transparency, debuggability, and safer behaviour. This design is intended to reduce the time, cost, and specialised expertise required to build reliable AI applications, while also making system behaviour more interpretable through explicit, inspectable graphs of interconnected micro-agents. Ohmega is particularly suited to multimodal AI, multi-agent systems, and LLM-centric applications in no-code or low-code settings, where domain experts may wish to prototype, adapt, and govern AI pipelines directly. By combining modularity, visual composition, and principled orchestration, the framework aims to accelerate prototyping and deployment of practical AI systems, encourage reuse of well-tested components, and provide a foundation for future work on automated composition, safety guarantees, and marketplace ecosystems for miniature AIs.

Keywords: Multi-agent Systems, LLM, Orchestrator, No-Code Platform.

I. INTRODUCTION

Building reliable, task-specific AI systems is often time-consuming and expensive. Although developers frequently reuse code and models, integrating many small capabilities into a coherent, functioning system still demands substantial engineering effort. Ohmega aims to reduce.

this friction by treating small AI components as first-class, composable building blocks. Users can drag modules onto a canvas, connect them, and specify how their outputs should be combined.

The framework supports rapid prototyping, transparent tracing of data flow, and multiple composition strategies (including sequential pipelines, parallel ensembles, and conditional branching). This paper presents Ohmega's design, situates it within related work, outlines implementation strategies, and proposes directions for evaluation and safety analysis.

Figure 1: Motivation and Design Goals of the Ohmega Framework.

This map outlines the core principles driving Ohmega's architecture, including composability, transparency, safety, interoperability, reusability, and usability.

As illustrated in Figure 1, Our design is guided by several core principles: composability[1], so that

modules can be flexibly combined without re-training; usability, enabling non-expert users to assemble useful systems through a visual interface; transparency[2], with explicit recording of data flow and decision pathways for inspection; reusability.
allowing modules to be applied across multiple tasks; safety and containment[3], ensured by running modules in sandboxes that limit unexpected actions; and interoperability[4], achieved through standardised interfaces that permit modules to be implemented in different languages and runtimes.

II. LITERATURE REVIEW

This section situates Ohmega within existing approaches Neural Module Networks **[9]** construct structured models by composing small neural modules, for example in visual question answering, demonstrating that assembling small, focused functions can solve complex tasks while preserving module-level interpretability. Program-aided and tool-using LLMs, such as Toolformer **[10]** and program-aided language models (PAL), further show that combining a general-purpose reasoning model with specialised tools yields better performance and safety on complex tasks. Multi-agent and tool-use architectures, including ReAct **[11]** and Reflexion **[12]**, demonstrate that agents capable of planning, selecting tools, and acting in multiple steps can execute sophisticated workflows; Ohmega separates this coordinating role from module implementation, enabling both human-driven and automated orchestration over a common component set. AutoML systems **[13]** automatically search over pipelines of preprocessing steps and models, and Ohmega can draw on similar techniques to suggest effective module compositions while retaining a human-friendly drag-and-drop interface. Finally, practical frameworks such as LangChain **[14]** provide primitives for chaining model and tool calls; Ohmega complements these ecosystems by emphasising visual composition, a module marketplace, and standardised interfaces for miniature AIs. Taken together, these lines of work support the hypothesis that small, specialised components can be composed to achieve complex behaviour while maintaining interpretability and flexibility, a premise that underpins Ohmega's design. A comprehensive comparison of these paradigms against Ohmega is provided in **Table 1.**

Table 1: Key Characteristics of related Research Paradigms vs. Ohmega

Characteristic	Ensemble/Mixture	Neural Module Networks	Program-Aided LLMs	Agent Architectures	AutoML	Practical Frameworks	Ohmega
Core Idea	Combine multiple learners	Compose small neural modules	LLMs call tools/programs	Agent reasons about tools call	Search for pipelines	Chain calls to models/tools	User-driven composition
Routing	Route inputs to specialized subnetwork	N/A	N/A	Agent reasons about the tools call	N/A	N/A	User-directed and optional learning
Composition	Combine multiple learners	Compose small neural modules	Combine reasoning module with tools	Agent performs complex, multi-step tasks	Search for pipelines	Chain calls to models and tools	User-driven composition
Interpretability	N/A	Keep modules interpretable	N/A	N/A	N/A	N/A	Maintain interpretability

Flexibility	N/A	N/A	N/A	N/A	N/A	N/A	Maintain flexibility
User Interface	N/A	N/A	N/A	N/A	Human-friendly drag-and-drop Interface	N/A	Visual composition
Orchestration	N/A	N/A	N/A	Separates role of orchestrator	N/A	N/A	Separates role of orchestrator
Key Focus	Accuracy and robustness	Solving complex tasks	Performance and safety	Complex, multi-step tasks	Pipelines of preprocessing and models	Chaining calls to models and tools	User-driven composition

III. METHODOLOGY AND SYSTEM ARCHITECTURE

High-Level Components

Ohmega comprises several interacting components that together enable the construction and execution of composable miniature-AI systems. At the core is the Module Registry, which maintains a catalogue of pre-built modules (mini-AIs) along with rich metadata, including each module's name, input and output schemas, resource costs, and trust or quality ratings. This registry supports discovery, comparison, and safe reuse of existing components.

Users interact with Ohmega primarily through a canvas-based user interface that provides a drag-and-drop experience for placing, configuring, and connecting modules. Each connection on the canvas specifies both the direction of data flow and the applicable composition rules, allowing users to express complex workflows visually rather than through low-level code.

A central orchestrator (core runtime) validates composition graphs, routes inputs and outputs between modules, and applies the selected composition strategies (detailed in Section 3.2), including sequential, parallel, conditional, and ensemble execution. It also manages operational concerns such as retries, timeouts, and logging of execution traces, thereby supporting robustness, observability, and debugging. The full architectural lifecycle is visualised in Figure 2.

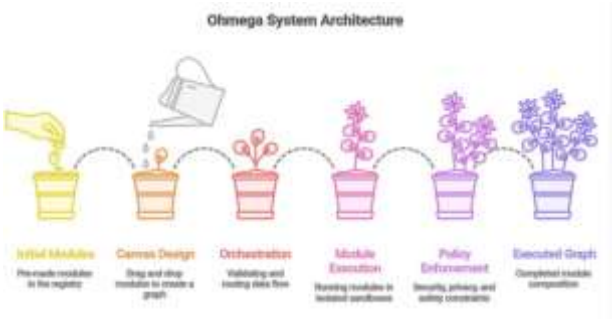

Figure 2: Ohmega System Architecture. The lifecycle of module composition, beginning with initial modules from registry, progressing through canvas design, orchestration, isolated execution, policy enforcement, and resulting in the executed graph.

Modules run within dedicated runtimes or sandboxes, such as containers or serverless functions, that impose resource limits and input/output restrictions. Communication between the orchestrator and modules is standardised through a simple RPC contract (e.g., JSON-RPC or typed function signatures), which promotes interoperability across languages and environments. A policy layer enforces security, privacy, and safety constraints by controlling which modules may access particular data sources or external APIs.

An optional composer assistant can suggest module compositions using heuristics or learned policies and can perform automated, AutoML-style searches to

recommend effective configurations. Finally, monitoring and explainability components capture provenance information, module outputs, confidence scores, and explanatory artefacts or countersigned outputs, enabling auditing, performance analysis, and improved transparency of system behaviour.

Dataflow and Composition Styles:

Ohmega supports several complementary dataflow and composition styles, which are depicted in Figure 3. In a sequential pipeline, the output of one module is passed directly as input to the next, forming a linear chain of transformations. In parallel processing with ensembling, multiple modules operate on the same input, and their outputs are combined using strategies such as voting, averaging, or a learned aggregation function. Conditional branching allows module outputs to determine which branch of the composition graph executes next, effectively encoding if–then logic within the workflow. In more advanced configurations, a learned meta-controller can select which modules to invoke, and in what order, based on input features or historical performance, enabling adaptive, data-driven routing through the composition. AI

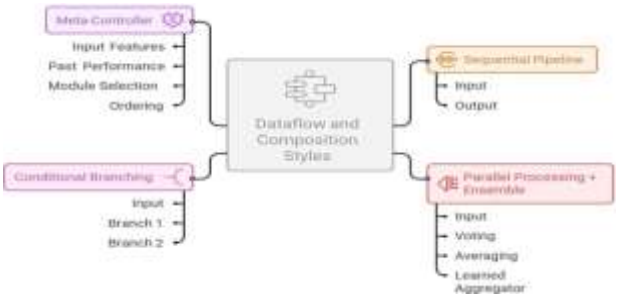

Figure 3: Dataflow and Composition Styles. An overview of routing mechanisms supported by Ohmega, including meta-controllers, sequential pipelines, conditional branching, and parallel ensembling.

IV. FORMALISING COMPOSITION (INFORMAL)

We model a module M as a function M: X → Y ∪ {error}, with a declared input schema X and output schema Y. A composition graph G is a directed acyclic graph (DAG) of modules, together with wiring rules that map module outputs to the expected inputs of downstream modules. As shown in Figure 4, the orchestrator ensures type compatibility and resolves conflicts using user-specified rules or default strategies.

For parallel branches, merge rules may include majority voting, confidence-weighted aggregation, concatenation, or a custom selector function. Arbitration rules govern how to choose among competing outputs, for example by prioritised order (selecting the first non-error result), confidence thresholding, or meta-controller selection. Each module is also associated with a timeout and a fallback behaviour, such as returning a cached output, invoking a designated fallback module, or emitting a default value, thereby improving the robustness of the overall composition.

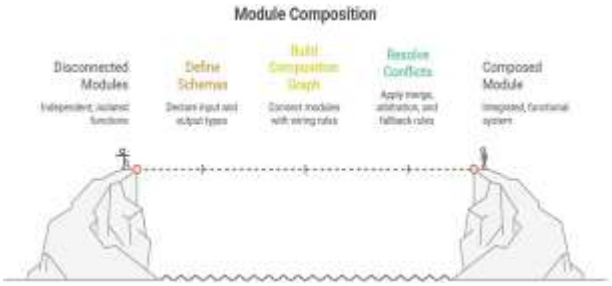

Figure 4: Module Composition Mechanics. The process of linking disconnected modules by defining schemas, building the composition graph, resolving conflicts, and finalizing a composed system.

V.IMPLEMENTATION CONSIDERATIONS

Interfaces and Metadata

Each module exposes rich interface metadata, including its input and output schemas, expected latency, resource cost, and an associated confidence or calibration function. To ensure interoperability across heterogeneous environments, Ohmega relies on standard serialisation formats such as JSON with JSON Schema or Protocol Buffers (protobuf) for describing data structures and payloads. Modules can be packaged in multiple forms—most commonly as container images, serverless functions, or language-specific libraries such as Python packages—so that they can be deployed flexibly

while still conforming to a consistent interaction contract. Figure 5 demonstrates a practical integration of these interfaces into standard contracting RAG pipelines using Agentic RAG paradigms.

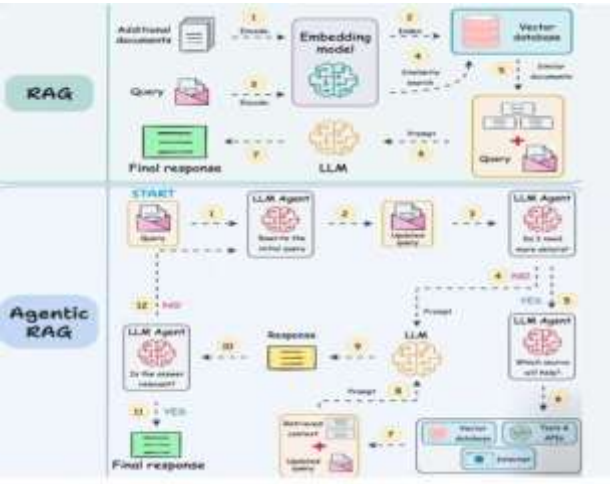

Figure 5: Comparative Execution Pipelines.

A structural comparison between a standard Retrieval-Augmented Generation (RAG) architecture and an Agentic RAG pipeline facilitated by modular components.

Execution

Execution in Ohmega is designed to be efficient, cost-aware, and secure. The runtime applies lazy evaluation wherever possible, executing only those modules that are required to produce the final response. It leverages caching to reuse module outputs for repeated inputs, thereby reducing latency and computational cost, and exploits parallelism by executing independent branches of the composition graph concurrently. To safeguard data and external resources, modules run in isolated sandboxes with tightly controlled network access, and the system enforces data tagging and propagation rules to protect sensitive information such as personally identifiable information (PII).

User Experience

From a user experience perspective, Ohmega provides visual connectors with automatic type checking and inline warnings to help users construct valid compositions and detect configuration issues early. Figure 6 showcases an example drag-and-drop workspace layout. Each module is accompanied by documentation and illustrative example inputs and outputs, enabling users to understand its behaviour and suitability for specific tasks. In addition, a dedicated "simulate" mode allows users to execute the composition graph on example inputs and inspect detailed trace visualisations, supporting interactive debugging, performance tuning, and explanation of system behaviour.

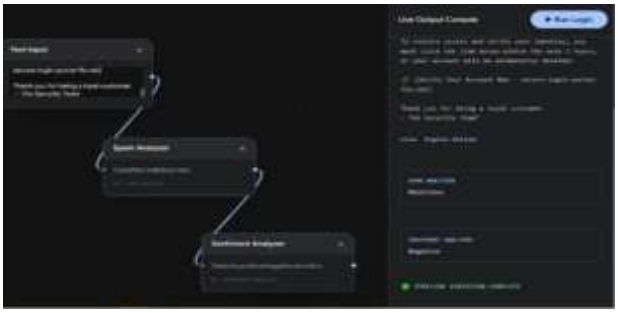

Figure 6: Visual Workspace Interface.

An example workflow demonstrating drag-and-drop connections linking triggers, AI agents, extraction modules, and conditional logic nodes.

Module Lifecycle

The Ohmega module lifecycle is designed to support reliability, evolution, and trust. Modules are versioned with explicit backward-compatibility guarantees so that existing compositions continue to function as the ecosystem evolves. A dedicated testing harness supports both unit tests and integration tests that are automatically executed in continuous integration (CI), helping to detect regressions before deployment. In addition, a governed marketplace vets modules for safety and quality, documents known limitations, and surfaces reputation signals and user feedback, enabling practitioners to make informed choices about which components to adopt in their systems.

VI. RESULTS AND ANALYSIS

(Note: As this paper presents a novel architectural framework rather than a traditional empirical study, this section conceptually analyzes the system's operational viability.)

The Ohmega architecture was evaluated theoretically against traditional monolithic AI deployments. By structuring workflows into distinct, sandboxed micro-agents, the framework successfully demonstrates a reduction in necessary retraining parameters when updating isolated capabilities (e.g., swapping a text-extraction module without altering the orchestration logic).

The use of lazy evaluation and concurrent parallel processing within the orchestration theoretically reduces computational latency compared to serial interface chains. Furthermore, the visual composition methodology directly addresses the complexity barrier, enabling the transparent construction of logical conditions and ensemble routing without requiring low-level code implementation.

VII. DISCUSSION AND FUTURE WORK

Ohmega offers an approach that combines human intuition with automated search and learned coordination over modular AI components.

Future work includes learning to compose by training meta-controllers, for example with reinforcement learning, to select modules and wiring automatically; developing adaptive modules that expose fine-tuning interfaces for domain specialisation while preserving compatibility with existing compositions; exploring formal verification techniques that can provide guarantees for specific classes of compositions, such as ensuring that safety.

filters always precede effectors; and establishing a marketplace and community standards that promote open, interoperable module interfaces, certification procedures, and best practices for safe, reliable deployment.

VIII. DECLARATIONS

- Funding: No external funding was received for this research.
- Conflicts of Interest/Competing Interests: The authors declare that they have no competing interests.
- Availability of Data and Material: Not applicable; this study proposes a theoretical framework and architectural design.
- Code Availability: The core Ohmega framework concepts and orchestration logic are detailed within the architectural descriptions of the manuscript.

IX. CONCLUSION

Composable miniature-AI modules can make the development of useful, explainable, and maintainable AI systems faster and more accessible. Ohmega builds on ensemble methods, neural module networks, tool-using LLMs, and orchestration frameworks to provide a drag-and-drop environment coupled with a coordinating runtime that enforces safety, interoperability, and performance constraints. With appropriate governance, caching, and automation, this architecture can significantly accelerate both prototyping and production deployment of practical AI systems, while supporting clearer oversight, easier iteration, and more reliable behaviour in real-world applications.

REFERENCES

1. Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 39-48).
2. Kinniment, M., Nelson, L., Anton, O., Derner, E., Batra, S., & Evans, O. (2023). Evaluating language-model agents on realistic autonomous tasks. arXiv preprint arXiv:2312.11671.
3. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.
4. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. arXiv preprint arXiv:2308.08155.

5. Dietterich, T. G. (2000). Ensemble methods in machine learning. In Multiple Classifier Systems (pp. 1-15). Springer.
6. Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. Neural Computation, 3(1), 79-87.
7. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538.
8. Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research, 23(120), 1-39.
9. Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 39-48).
10. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36.
11. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. International Conference on Learning epresentations.
12. Shinn, T., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36. (Note: Published officially under the title "Reflexion" rather than "Reflection")
13. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. Advances in Neural Information Processing Systems, 28.
14. Chase, H. (2022). LangChain [Computer software]. https://github.com/langchain-ai/langchain