Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

OpenClaw Echo

Roqaiah Anjum E[1], Mokshita N[2], Ambika[3], Mrs. Syeda Misba[4]

[1,2,3] Department of Computer Science and Engineering, [4]Guide, Assistant Professor
[1,2,3,4]Ghousia College of Engineering, Ramanagaram, Karnataka, India
Visvesvaraya Technological University, Belagavi, Academic Year 2025–2026

Abstract
Agents based on large language models no longer answer a prompt. They plan, call tools, and keep memory across steps. That creates engineering problems that a plain text generator never faces. For example, how much context should be kept? What happens if a provider fails halfway through a task? How can a failed step be recovered from crashing the run? SALLMA is one of the papers that treats this at the software-architecture level, but its validation is only preliminary and qualitative. This paper describes OpenClaw Echo, a local-first agent framework that uses a waterfall router across model providers, a hybrid memory layer that includes relational records, semantic vectors and a small knowledge graph, role-specialized sub-agents and a verification stage that triggers bounded recovery. SALLMA is used as an architectural reference; none of the individual mechanisms is claimed as new. The aim is to see how these mechanisms behave when built and measured together. Project-defined tests gave 96.2 percent intent-classification accuracy, 91.4 percent precision at five, and 88.7 percent recall at five for retrieval. Recovery from injected provider failures was 99.8 percent. The average latency was 0.85 seconds for requests and 8.4 seconds for delegated tasks. These results show that one coordinated pipeline is better than a set of features, but it has a clear latency cost when deeper processing is used. Limitations, dependence on external providers, and the small self-built evaluation sets are discussed. OpenClaw Echo demonstrates this approach.

Keywords: Autonomous Agents, Large Language Models, Multi-Agent Systems, Hybrid Memory, Failover Routing

1. Introduction

Large Language Models (LLMs) were originally designed to generate text, one turn at a time. Now they are often used inside a loop that takes a goal, pulls in information, selects a tool, takes an action, and repeats. Recent studies describe planning, memory and tool selection as the parts of these kinds of agents [2, 3]. When a model is placed inside this loop, the surrounding code has to deal with problems that didn't exist before: keeping track of state across steps, choosing the tool, handling failures, and making sure many steps work together smoothly.

A system built around one model call begins to struggle when a task needs long-term memory, access to external services, or different kinds of specialized work. Long histories make the prompt too big. One bad Application Programming Interface (API) call or a malformed response from a tool can crash the process. Multi-agent research helps by dividing work among workers [4, 5]. Memory research helps by exploring

how stored context should be split or merged [6]. Work on reflection and recovery from tool failure shows that agents perform better when they check results and decide what to do next rather than stopping at the first error [7, 8].

OpenClaw Echo, the system described here, brings all these ideas into one working framework that runs locally. It has a request path that understands intent, a hybrid memory system, a multi-provider router that tries providers in order, a small group of specialized sub-agents, a catalog of tools that can be called, and a verification stage that includes limited recovery options. Users mainly interact with it through a Telegram bot. There are also web dashboards for monitoring what's happening. The design and how it was built come directly from the project's code, configuration files and test logs.

SALLMA (Software Architecture for LLM-Based Multi-Agent Systems) is the reference architecture because it presents a way to build LLM-based multi-agent systems, separating operational and knowledge roles [1]. Its structure is not copied exactly. Its idea is followed—that agent coordination should be clear in the software design—but something much smaller is aimed for: a compact personal assistant that works locally. The focus is on handling provider failures, hybrid memory, registering skills at runtime and measuring end-to-end performance. This is something SALLMA itself says is still missing [1].

To be clear about the contribution: routing, Retrieval-Augmented Generation (RAG), layered memory, reflection and multi-agent delegation are all known ideas, and no credit is claimed for them. What is examined is how they behave when combined in one assistant and whether that behaviour can be measured under a controlled test plan.

1.1 Research Objectives

- Keep track of the conversation context across different sessions using several local memory systems.
- Minimize reliance on a single external thinking tool by using backup methods and cooling-off periods at the model level.
- Deal with complex tasks by breaking them into smaller parts handled by specialized sub-agents, while still giving one unified answer.
- Identify a specific group of issues that happen during execution and try to fix them within limits before giving up on a task.
- Assess performance using tests focused on categorization, searching, dependability, speed, and security.

2. Related Work

Work relevant to OpenClaw Echo falls into five areas: architecture of multi-agent systems, task delegation, memory and retrieval design, reflection and failure recovery, and evaluation of agent systems. Papers were kept when they bore directly on at least one of these.

2.1 SALLMA: Software Architecture for LLM-Based Multi-Agent Systems

SALLMA separates request handling and agent coordination from workflow and deployment configuration, treating each as its own architectural concern [1]. Its proof of concept uses containerized model instances and orchestration tooling to show that the design works. The authors say their evaluation

is preliminary and leave industrial-scale, quantitative testing for future work. This paper partly addresses that gap, at a smaller scale, by reporting measured behavior. SALLMA's separation of coordination logic from configuration data influenced how OpenClaw Echo keeps its reasoning services apart from its persistence layer, although its component boundaries are not adopted.

2.2 A Survey on LLM-Based Multi-Agent Systems

This survey looks at how agents behave by focusing on four key abilities: perception, planning, memory, and action [2]. It doesn't suggest a specific system. Instead, it introduces a common way to talk about autonomy, which comes from a designed control system, not just the model itself. This way of thinking is used to explain why classification, retrieval, and recovery are checked separately instead of treating the whole system as a single process.

2.3 A Survey on LLM Agents

Lee et al. (2026) talk about an agent that uses a language model as its main part, along with memory, planning, and tool-invocation features. Reliability, how to evaluate these agents, and how well they can scale are also mentioned as open issues [3]. The focus on the lack of proper evaluation is why Section 6 includes measurements—describing a system without testing it leaves the reliability question unanswered.

2.4 Master-Agent Orchestrated Multi-Agent Systems

Roy et al. (2025) describe a system that breaks down a request into smaller tasks and creates worker agents as needed, adjusting the number of agents based on the job [4]. OpenClaw Echo doesn't allow unlimited agent creation. Instead, it uses a fixed group of specialized workers, like Researcher, Coder, Analyst, Writer, and Quality Assurance (QA) Engineer, managed by a supervisor with a limited number of steps. This choice gives up some flexibility for more predictable performance, which made it possible to test this approach.

2.5 AutoGen

AutoGen offers tools that let you combine LLM-powered agents, human input, and tool or code execution in a shared conversation, with a manager that handles who speaks next [5]. OpenClaw Echo has a similar goal of separating roles and abilities, but it uses its own manager, skill system, and execution process instead of the conversation-based approach of AutoGen.

2.6 Hybrid, Multi-Layered Memory for Multi-Agent Systems

Ilin et al. (2025) say that agent memory needs its own structure, with different levels ranging from short-term context to long-term stored meaning, along with rules for moving information between these levels [6]. OpenClaw Echo follows the idea but not the layered structure. Its memory includes things like logs of interactions, summaries made by the system, facts pulled out, vector embeddings, and graph triples, all kept locally. The idea is that different types of information need different ways of being stored.

2.7 A Generalized Memory Evaluation Method for Local Language Models

Bednárik et al. (2025) suggest a way to test memory, not create a new memory system. A set of steps is used where different facts are added to a conversation at different times, and then what a system with

memory can still remember is checked [9]. The difference between having a memory system and proving it works is why the results in Section 6 are measured using precision and recall.

2.8 Multi-Scale Adaptive RAG with Semantic and Structural Knowledge Graphs

Ning et al. (2026) use both semantic similarity search and graph walking to handle two problems in RAG: missing important information and including too much irrelevant stuff [10]. OpenClaw Echo has a much smaller graph layer, storing entity relationships as triples next to vector embeddings. But it starts from the same idea that just using similarity is not enough to understand the relationships between stored items.

2.9 Multi-Level Graph Pruning for Graph RAG (MGRAG)

H. Li et al. (2025) show that pruning retrieved subgraphs at the node and subgraph levels before sending them to the language model leads to better results than passing the whole graph [11]. OpenClaw Echo does not use this pruning method. Its graph-aware retrieval is simpler, and MGRAG is mentioned as a possible improvement for the future.

2.10 Self-Reflection in Large Language Model Agents

Renze et al. (2024) tested different reflection strategies across various models and found that allowing a model to rethink a wrong answer leads to much better accuracy [7]. OpenClaw Echo uses a simpler version, focusing on engineering aspects: when a sub-agent step fails, the manager shows the error and lets the system try again up to a set number of retries. It doesn't involve deep reflection on the quality of reasoning in the way they studied.

2.11 Tool-Use Failure Analysis and Recovery

J. Jiang et al. (2026) categorize tool-use failures like timeouts, incorrect outputs, and planning mistakes, and suggest a Monitor–Classifier–Recovery framework to handle these issues during runtime [8]. The categories match the verification and recovery stage in OpenClaw Echo, which also treats a failed tool call as something that can be fixed. However, the recovery system in this work was built separately and is not based on that framework.

2.12 Additional Supporting Literature

Other papers explore personalized task allocation in power systems [13], LLM-enhanced coordination for sixth-generation (6G) communications [14], investor profiling using multiple agents [15], cooperative multi-agent reinforcement learning [16], retrieval-augmented social simulation [17], multi-agent software design [12], and workflow analysis for self-adaptive agent-based simulation [18]. These works show that coordination, memory, and adaptive planning are important topics that go beyond the scope of this paper. They are cited as background information, not as direct precedents.

3. Research Gap and Motivation

When looked at separately, each part of autonomous agents has developed a lot. SALLMA offers a software-architecture way to manage multi-agent teamwork [1]. Memory studies focus on layered and lasting states [6, 9]. Graph-based RAG work helps with better structured retrieval [10, 11]. Research on reflection and recovery deals with failed reasoning and tool calls [7, 8]. Surveys keep highlighting

reliability, observability, and long-term evaluation as problems that are still not solved [2, 3]. So the gap isn't about a missing technique, because each part already exists in some way.

What isn't studied enough is how these parts act together and how they perform in a single running system. OpenClaw Echo keeps application state local, sends inference work to outside providers, and connects memory, delegation, tool execution, verification, and recovery through one request cycle. Its main contribution is the mix of hybrid memory, sub-agent delegation, and provider-level resilience, not just any single part.

Combining these parts has real costs. Simple tasks need to stay fast, while complex ones need more planning and coordination. The challenge is balancing better execution with speed and reliability, which is why the system is tested using separate benchmarks for each concern, not as a single model call.

4. OpenClaw Echo System Architecture

4.1 Overall Architecture

Not all requests follow the same process. There are two execution paths that share the same support services. The lightweight path provides a direct answer once the relevant context is found. The delegated path is used when a query needs to be handled by specialized workers. Both paths use the same services for memory access, provider selection, and output verification. The difference between a simple query and a complex task lies in how much of the system is used, not in having separate implementations.

Three key areas of focus are shared between the two paths. The first is user access, currently implemented as a Telegram bot with web-based monitoring. The second is reasoning: classifying requests, retrieving memory, choosing a provider, and coordinating sub-agents when needed. The third is persistence: using relational storage, semantic vector lookup, graph-based entity relationships, and background scheduling for deferred tasks. Both paths read from and write to the persistence group.

4.2 Request Processing

An incoming message is normalized and passed to intent classification. If the request is simple, it continues on the fast path. If more work is needed, it goes to the Sub-Agent Manager. Relevant stored context is retrieved before any generation. If work is delegated, the sub-agent results are combined and checked before a final answer is given. The answer, along with anything worth storing, is written back to storage. Internally, the stages are: ingest, context retrieval, plan and route, delegate, execute, verify and recover, and persist.

4.3 Multi-Provider Routing

A dedicated model router keeps provider-specific behavior separate from the rest of the application. It tries a primary provider first and switches to a secondary one if there is a timeout, a quota error, or similar availability issues. A cooldown and circuit-breaker mechanism prevents the system from repeatedly calling a provider that is already known to be down. The deployed configuration uses provider-specific timeouts and a five-minute cooldown.

4.4 Specialized Sub-Agents

Complex requests are assigned to a fixed set of role-specialized workers instead of one general agent: Researcher, Coder, Analyst, Writer, and QA Engineer. Each worker runs in its own isolated scratchpad, and a Sub-Agent Manager coordinates them under a bounded iteration budget to prevent infinite loops in a delegated task.

4.5 Skill Registry and Runtime Extension

All tools are accessed through a central SkillRegistry. Each registered skill has a handler, a parameter format, and a description that the model can read to decide which tool to use. The registry currently includes 23 static tools that handle web search, memory access, safe code running, file management, and task handoff. It also allows for runtime skill creation: a new skill can be made, checked for correct syntax, and added without changing the main system code. In this paper, "self-evolving" only refers to this runtime skill creation and adding. It doesn't mean retraining the model, adjusting model settings, or changing the system design.

4.6 Hybrid Memory and Knowledge Graph

Instead of just using raw chat history, the system stores different types of information. SQLite holds structured data like interactions, extracted facts, summaries, graph entities, and graph relationships. A local semantic store helps find similar information using vectors. The knowledge graph keeps explicit relationships between entities as triples, allowing the system to combine semantic matches with structured links. The graph stores these triples, but it isn't a full multi-step reasoning engine.

4.7 Verification and Recovery

Before a step is considered complete, it goes through a verification stage where the result is checked against specific criteria for that task. When the check is straightforward, a deterministic method is used. But when the check involves meaning, the system uses a large language model. If the verification fails, the recovery system can try the step again or change the approach. If a sub-agent fails, it gets a limited number of retries. If that doesn't work, the system provides a partial or best-effort result instead of failing the whole request.

4.8 Network Recovery and Scheduling

If a single provider fails versus a complete loss of connection, the system treats them as separate situations. Regular health checks help distinguish between the two. When there's a total loss of connection, network-based tasks are postponed and restarted once the connection returns. Background tasks are managed through Clockwork. PM2 monitors the Node.js process and restarts it automatically if it crashes unexpectedly.

5. Implementation and Methodology

5.1 Software Environment

OpenClaw Echo is mostly built using TypeScript on Node.js. Express handles the API endpoints, LangChain helps integrate large language models, SQLite3 stores data locally, Telegraf manages Telegram communication, and Software Development Kits (SDKs) from Gemini and Groq are used to connect to those services. It works with Node.js version 20 or newer and doesn't require a local Graphics

Processing Unit (GPU) because all the model inference is handled by external services that are set up in the configuration.

5.2 Configuration and Security

Settings that change based on the environment, like login details, time limits, error handling rules, and resource limits, are stored outside the main code. Checking the logs showed that API keys for the service providers are not saved in the app logs or the metrics file. The coding environment was also checked for any attempts to access files outside its allowed area or open connections to other networks. These are basic checks to protect against some common security risks, but they aren't a complete security review.

5.3 Evaluation Protocol

The prototype was tested using specific tests that were set up by the project. These tests checked things like understanding user intent, retrieving correct information, switching to backup services when main ones fail, response speed, and some security features. The tests for understanding and retrieving info used conversation examples that were either created or carefully selected by the project team. To test how the system handles problems, the tests included fake errors like wrong login details, fake slow responses, and simulated internet cuts. These tests focus on how OpenClaw Echo itself behaves, not on how well it performs across all possible tasks. Any results that involve external services depend on how fast the internet was and how quickly those services responded on the day of testing.

6. Results and Discussion

6.1 Intent Classification

On a test set of 150 samples, the intent classification benchmark achieved 96.2% accuracy.

The routing classifier was generally able to distinguish between requests that could be handled via the fast path and those that required deeper processing. Since the dataset was specifically built for this study, the figure is project-specific and not a general benchmark for intent classification.

6.2 Retrieval Quality

In over 50 multi-turn evaluation conversations, the semantic retrieval subsystem achieved 91.4% Precision@5 and 88.7% Recall@5.

The retrieved context was largely relevant, while still recovering a significant portion of the stored information. These figures also serve as a baseline for comparing future designs using a dedicated vector database or a more advanced graph-based retrieval system.

6.3 Provider-Failover Recovery

In the controlled failover benchmark, the router and its recovery logic managed 99.8% of the injected provider failures.

This reflects the specific faults that were tested. It is not a guarantee of availability during real outages, which may fail in ways not covered by the test setup.

6.4 Latency

The average latency was 0.85 seconds for fast-path requests and 8.4 seconds for the delegated-task path.

The difference comes from the additional work in the delegated path: task decomposition, worker execution, tool calls, and result synthesis. This illustrates the trade-off between processing depth and response time discussed in Section 3.

6.5 Comparative Interpretation

Taken together, the results suggest a cohesive system rather than a collection of independent features.

Intent classification determines how much processing a request undergoes, memory retrieval provides persistent context, failover handles disruptions from external models, and verification-recovery addresses selected execution failures. The design addresses several system-level concerns raised in the literature while remaining much simpler and less distributed than architectures like SALLMA [1].

The central trade-off is clear: the lightweight path is much faster, and the delegated path uses that speed to offer more functionality.

Local persistence keeps the deployment simple and keeps data with the user, though it relies on a flat semantic store instead of a dedicated vector database. The prototype is best viewed as a compact research and engineering platform that needs more infrastructure work before it can support larger deployments.

6.6 System Screenshots

Figures 1 to 5 illustrate the running system.

They show the behaviour and interfaces; they are not a substitute for the measurements in Sections 6.1 to 6.4. Parts of the email address in Figure 2 have been masked.

Figure 1: Real-Time Telemetry Dashboard of OpenClaw Echo

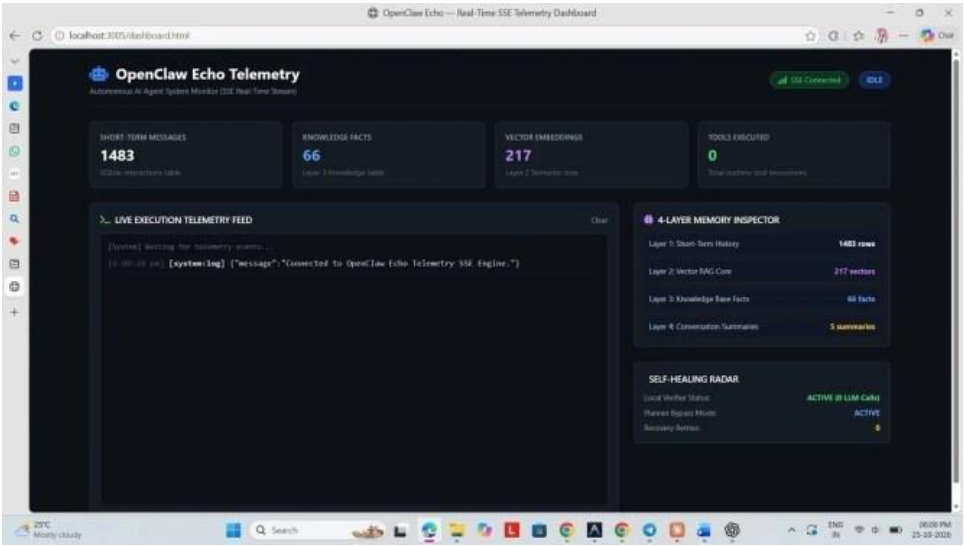


Figure 2: Email Received in Gmail

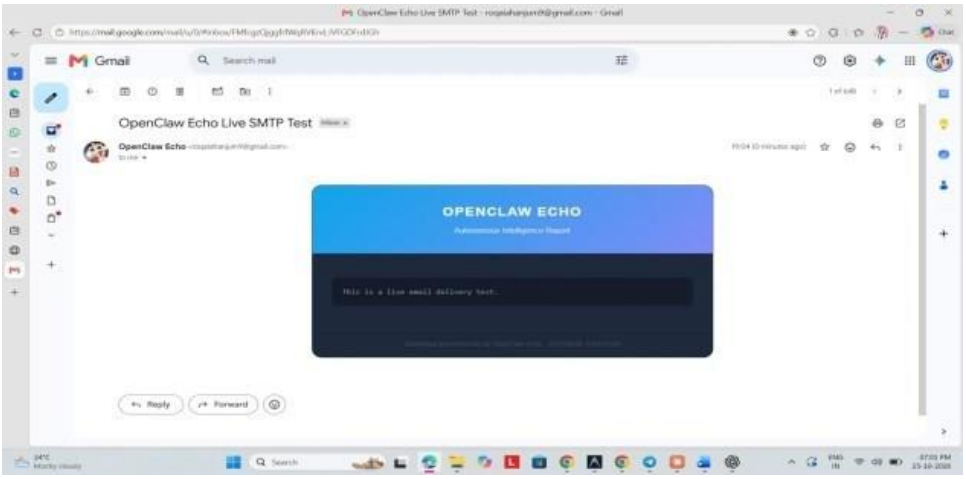


Figure 3: Telegram Interface: Profile Update and a delegate_task Tool Call, Followed by a Recorded Project Goal

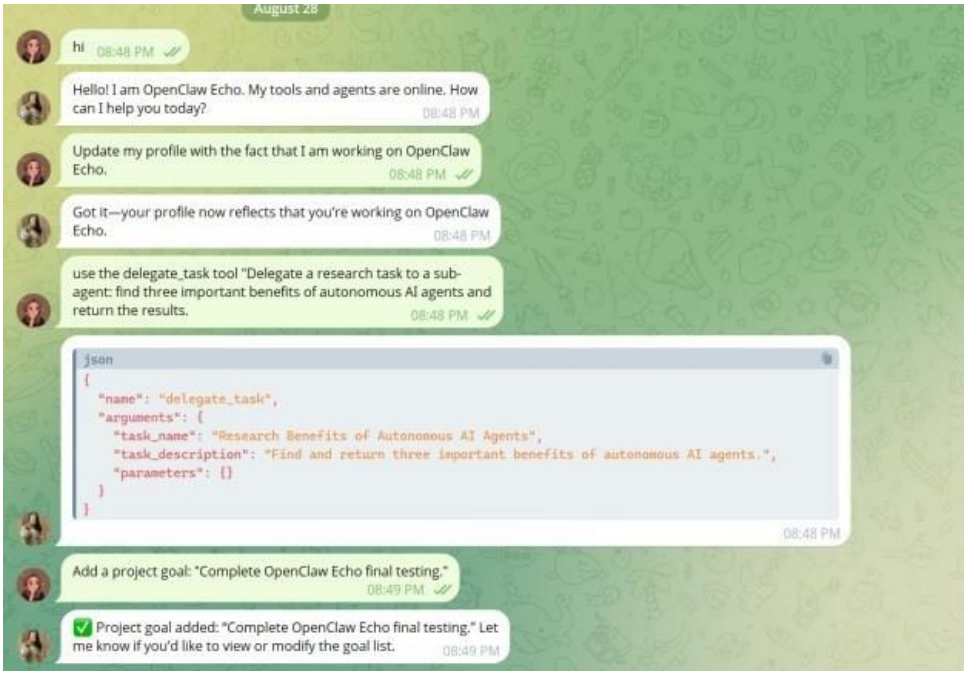


Figure 4: Web Monitoring Dashboard: Connectivity Status, Performance Hub and Chat Panel

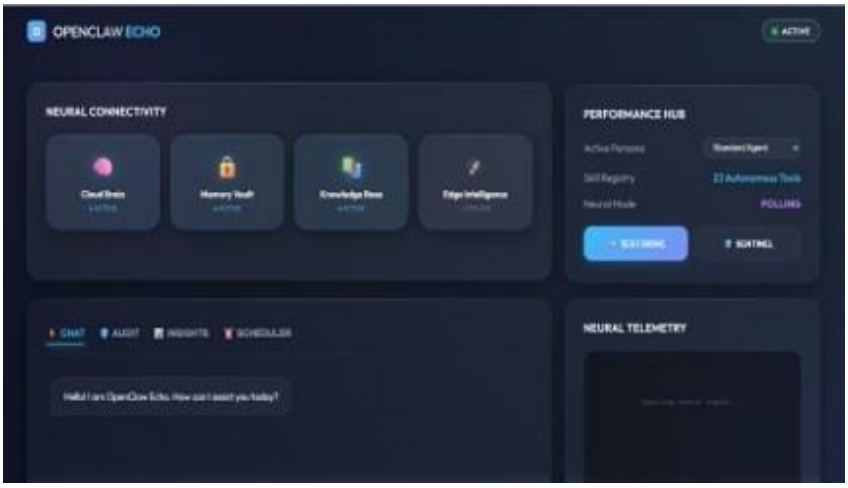


Figure 5: Streamlit Admin Dashboard Interface



7. Limitations and Threats to Validity

- The system still relies on external model providers. While the waterfall router reduces single-provider dependence, it does not offer a fully offline solution.
- The semantic store is based on JavaScript Object Notation (JSON) and may struggle with very large memory collections.
- The evaluation data were prepared or curated by the project team, so the classification and retrieval results may not apply to unrestricted user traffic.
- Failover was tested using controlled fault injection, not real-world, naturally occurring outages.
- Latency was measured in one network environment and could change depending on provider load and location.
- The knowledge graph stores triples but does not support multi-hop graph reasoning.

8. Conclusion

OpenClaw Echo is a functioning LLM agent where memory, task delegation, tool use, provider routing, verification, and recovery are explicit and separable components rather than one opaque pipeline.

SALLMA served as the architectural reference; the implementation emphasizes a compact, local-first deployment and runtime behavior that has been measured.

The reported tests achieved 96.2% intent-classification accuracy, 91.4% retrieval precision@5, 88.7% recall@5, and 99.8% recovery from injected provider failures, with a mean latency of 0.85 seconds on the fast path and 8.4 seconds on the delegated path. These results support the practical feasibility of the integrated design for the workloads tested. The limitations in Section 7 suggest that the findings should not yet be generalized to much larger, distributed, or fully offline deployments.

9. Future Work

- Replace the JSON semantic store with a dedicated vector database for scalability and retrieval management.
- Add a genuinely local model tier to the router for offline operation and reduced reliance on external APIs.
- Extend the graph layer toward multi-hop GraphRAG and graph-aware retrieval.
- Evaluate more specialized agents and adaptive delegation policies on larger, independently prepared datasets.
- Study distributed deployment and stronger observability to determine whether the same reliability holds across nodes.

References

1. M. Becattini, R. Verdecchia, E. Vicario, “SALLMA: A Software Architecture for LLM-Based Multi-Agent Systems”, Proceedings of the 2025 IEEE/ACM International Workshop on New Trends in Software Architecture (SATrends), 2025. https://doi.org/10.1109/SATrends66715.2025.00006
2. X. Li, S. Wang, S. Zeng, Y. Wu, Y. Yang, “A Survey on LLM-Based Multi-Agent Systems: Workflow, Infrastructure, and Challenges”, Vicinagearth, 2024, 1, Article 9. https://doi.org/10.1007/s44336-024-00009-2
3. D. Lee, H. Kim, “A Survey on LLM Agents: Architecture, Applications, and Challenges”, Proceedings of the 40th International Conference on Information Networking (ICOIN), 2026. https://doi.org/10.1109/ICOIN68469.2026.11480623
4. A. Roy, M. Wadhwa, N. Swami, H.H.M. Harshitha, D. Gopalan, P. Selvakumar, “Master-Agent Orchestrated Multi-Agent Systems: An Automated Framework for Task-Driven Dynamic Agent Creation”, Proceedings of the 5th International Conference on Electrical, Computer and Energy Technologies (ICECET), 2025. https://doi.org/10.1109/ICECET63943.2025.11472254
5. Q. Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation”, Proceedings of the International Conference on Learning Representations, 2024.
6. M. Ilin, D. Pavlyuk, “A Hybrid, Multi-Layered Memory Architecture for Collaborative Reasoning in Multi-Agent Systems”, Proceedings of the 2nd International Conference on Foundation and Large Language Models (FLLM), 2025.
7. M. Renze, E. Guven, “Self-Reflection in Large Language Model Agents: Effects on Problem-Solving Performance”, Proceedings of the 2nd International Conference on Foundation and Large Language Models (FLLM), 2024. https://doi.org/10.1109/FLLM63129.2024.10852426

8. J. Jiang, C. Li, J. Ma, K. Chen, “Tool-Use Failure Analysis and Recovery Framework for LLM Agents”, Proceedings of the 9th International Conference on Advanced Algorithms and Control Engineering (ICAACE), 2026. https://doi.org/10.1109/ICAACE69793.2026.11508644
9. B. Bednárik, L. Gogolák, “A Generalized Memory Evaluation Method for Local Language Models with Applications in Critical Infrastructure Security”, Proceedings of the IEEE 23rd Jubilee International Symposium on Intelligent Systems and Informatics (SISY), 2025. https://doi.org/10.1109/SISY67000.2025.11205417
10. F. Ning, Y. Bai, “A Multi-Scale Adaptive Retrieval-Augmented Generation Framework with Semantic and Structural Knowledge Graphs”, IEEE CNML, 2026. https://doi.org/10.1109/CNML68938.2026.11452538
11. H. Li, X. Li, F. Su, Q. Guo, “Multi-Level Graph Pruning-Based Framework for Graph Retrieval-Augmented Generation”, Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), 2025. https://doi.org/10.1109/ICME59968.2025.11209364
12. H.K. Dam, “Towards Multi-Agentic AI for Automated Software Design and Modelling: Challenges and Opportunities”, Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), 2025. https://doi.org/10.1109/ASEW67777.2025.00063
13. Z. Luan, Y. Wang, Y. Lai, L. Tian, Q. Wang, “Multi-Agent Task Planning Method with Personalized Task Guidance in Power Systems”, Proceedings of the Chinese Control Conference, 2025.
14. F. Jiang, Y. Peng, K. Wang, K. Yang, C. Pan, D. Niyato, “Large Language Model Enhanced Multi-Agent Systems for 6G Communications”, IEEE Wireless Communications, 2024, 31 (6), 48–55. https://doi.org/10.1109/MWC.016.2300600
15. H. Wang, Z. Yang, “A Multi-Agent Approach to Investor Profiling Using Large Language Models”, 2025.
16. J. Chai, Z. Zhao, Y. Zhu, D. Zhao, “A Survey of Cooperative Multi-Agent Reinforcement Learning for Multi-Task Scenarios”, 2025.
17. H. Shimadzu, T. Utsuro, D. Kitayama, “Retrieval-Augmented Simulacra: Generative Agents for Up-to-Date and Knowledge-Adaptive Simulations”, 2025.
18. Y. Loo, A. Ahmad, A.Y.C. Tang, A. Mustapha, “Workflow Analysis for Self-Adaptive Agent-Based Simulation Model”, Proceedings of the 2nd International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR), 2016.