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

Marco Becattini, Roberto Verdecchia, and Enrico Vicario
Software Technologies Laboratory, University of Florence, name.surname@unifi.it

***Abstract*—As a new and disruptive technology, the introduction of large language models (LLMs) may be the first step into a paradigm shift of how we develop and deploy software-intensive systems. While the capabilities of LLM agents for software engineering and architecture tasks are currently explored, how to architect LLM-based systems appears to be to date an uncharted territory. Software architectures based on a single LLM agent face inherent challenges, such as lack of task customization, lack of memory, and limited access to ground truth. These challenges become especially pressing in real-world contexts that demand persistent context, validated information, and task-specific flexibility. As a potential solution to overcome these challenges, multiple LLM-agents can be adopted for specialized tasks within a single software-intensive system. In this contribution, we open the discourse on architecting LLM-intensive software products by presenting SALLMA, a Software Architecture for LLM-based Multi-Agent systems. SALLMA leverages two core layers, namely (i) the Operational Layer, responsible for request intent management, handling real-time task execution and dynamic orchestration of agents, and (ii) the Knowledge Layer, used to to store and manage metamodels and configurations for workflows and agents. To primarily assess the viability of SALLMA, we develop a proof of concept leveraging as key technologies Docker, Kubernetes, Python, LangChain, Hugging Face, Mistral, LLaMA, and SQL and NoSQL databases. Currently, SALLMA is deployed to provide information on behalf of public administration offices, and is currently utilized in a business simulation scenario.**

I. Introduction

Rapid advancement of Large Language Models (LLMs) presents promising potential for efficiently tackling complex and dynamic challenges across a wide range of fields, e.g., coordinating enterprise-level business services of heterogeneous nature for personalized customer support. However, single-agent LLM systems face inherent limitations, particularly in memory retention, adaptability, and trade-off between broad generalist capabilities and specialized task configurations. These challenges reduce their effectiveness in real-world, complex and multifaceted scenarios [1].

Research has so far predominantly focused on centralized LLM architectures [2], leaving a gap in approaches that utilize distributed, multi-agent systems. To address this gap, the present paper introduces SALLMA (Software Architecture for LLM-Based Multi-Agent Systems), a software architecture leveraging a network of atomic, task-specific LLM agents, designed for a modular and distributed architecture.

SALLMA is organized into two core layers: the Operational Layer and the Knowledge Layer. The operational layer focuses on real-time task processing and orchestrates agents dynamically to form an efficient workflow in response to each incoming request. It ensures that agents are structured in a way that enables them to work collaboratively and effectively, streamlining the workflow process to handle tasks as they are received. The Knowledge Layer is responsible for storing information about the models used by the Operational Layer in dedicated catalogs. Specifically, it includes a catalog for the metamodel of the agent workflow, detailing workflow structures and relationships; a catalog for the metamodels of possible configurations for each agent; and a catalog for the deployment characteristics of each workflow.

SALLMA is designed as a distributed, modular architecture to potentially support scalability, adaptability, and resilience. Specifically, we hypothesize that container-based deployments can streamline resource expansion under rising workloads (scalability), while task-specific agent configurations facilitate the replacement or integration of new agents without disrupting the entire system (flexibility). Moreover, by isolating agents into separate components, failures can be contained and potentially be localized more effectively (testability). However, we acknowledge that these benefits require further empirical validation.

To assess SALLMA viability, we implemented a concrete instance of the architecture by selecting a range of key technologies, with the most notable being Docker for containerization, Kubernetes for orchestration, and LangChain for workflow management.

This paper seeks to foster meaningful discussion among researchers on multi-agent LLM architectures, with a particular focus on the dual-layer approach of SALLMA. Specifically, we aim to study the advantages of distinguishing between operational and knowledge-managed layers, highlighting how this design can enhance scalability, responsiveness, and the capacity to manage complex, context-driven tasks effectively.

II. Motivation

SALLMA is motivated by limitations observed in real-world applications when using single-agent For instance, a single LLM-based agent may be required to provide technical support, discuss legal matters, and address financial queries. Such systems face (i) suboptimal hyperparameter configuration, which leads to mismatched responses across varied domains; (ii) lack of persistent memory, hindering context retention for multi-step problems; (iii) limited means to validate ground truth, resulting in inaccuracies when reliable information is required; and (iv) static, centralized deployments that impede reliability management [3].

These issues motivate the need an integration of established software engineering principles [4], including modular design, containerized deployment, and dynamic resource management, all reflected in SALLMA's multi-agent strategy. Specifically, each agent can be uniquely configured and maintain state or memory through tools and persistence layers, and integrate verified data for ground truth. A distributed deployment model also addresses reliability by supporting flexible execution across multiple nodes.

III. SALLMA Overview

SALLMA (Software Architecture for LLM-Based Multi-Agent systems) is a modular and adaptive architecture designed to orchestrate multiple large language model (LLM) agents across a cloud-to-edge continuum. The architecture addresses the limitations of single-agent LLM systems by enabling distributed, task-specific agents to operate collaboratively in complex, dynamic environments.

The architecture of SALLMA is designed with two primary layers—the Operational Layer and the Knowledge Layer—proposing an application of Fowler's Reflection pattern [5] to multi-agent system architecture, thereby enabling dynamic adaptability and modularity within the framework.

A. SALLMA Operational Layer

The Operational Layer in SALLMA is responsible for handling real-time interactions and orchestrating task-specific agents to achieve the intended outcome of each request. This layer acts as the dynamic core, where tasks are processed through an integrated sequence of components designed to ensure efficient and adaptive operation. The primary elements of the Operational Layer layer are:

- **Intent Management Agent:** This component acts as the entry point for all incoming tasks, parsing user requests to understand their intent. By doing so, it sets up the system to retrieve workflows and configure agents specific to the request nature, using predefined intents stored in the Knowledge Layer.
- **Workflow Management Agent:** Once the intent is identified, the Workflow Management Agent decomposes the task into subtasks and assigns them to the appropriate specialized agents. This agent-driven workflow ensures that each subtask receives attention based on its specific requirements.
- **Routing Manager:** This component determines if an appropriate instance of the workflow exists to handle the requested execution. If an instance is available, it routes the request to it; if not, it requests the Deployment Manager to create a dedicated instance of the necessary cognitive workflow.
- **Deployment Manager:** The Deployment Manager deploys workflows by leveraging configurations and resources outlined in the Knowledge Layer, ensuring each deployment aligns with preconfigured specifications and can adjust dynamically to changes or failures.
- **Cognitive Workflow Manager:** The Cognitive Workflow Manager orchestrates task execution within a containerized environment managed by Kubernetes. It leverages a chain-of-thought process, managed by a dedicated manager, to invoke specialized agents in form of specific LLM instances, optimized for tasks, as defined in the workflow metamodel catalog. The Cognitive Workflow Manager thus enables interaction with tools for database operations. The database stores two key types of data: Foundational data that informs the LLMs for reliable RAG-based (retrieval-augmented generation) operations, and persistent memory for maintaining context across multiple agents and executions.

Thus, the Operational Layer processes each request through an optimized chain of thought, selecting the most suitable agent with access to reliable data sources and persistent memory. This setup also allows for flexible deployment across distributed resources, including cloud and edge environments. By using a workflow-driven approach, it supports modular task processing, enhancing SALLMA's adaptability to complex, multi-agent scenarios. These workflows, termed "*cognitive workflows*" represent structured tools designed to deliver context-aware, intelligent outcomes.

B. SALLMA Knowledge Layer

The Knowledge Layer serves as the structural foundation for metamodel management of SALLMA, maintaining a catalog of reusable configurations of workflows and agents. Its primary components include:

- **Workflow Metamodel Catalog**: SALLMA's Workflow Metamodel catalog stores predefined workflow configurations tailored for various task categories. These configurations, organized as metamodels, define the sequence of agent interactions and tool usage necessary for each workflow. Specific characteristics of each agent are retrieved from the Agent Metamodel Catalog, enabling the Workflow Management Agent to select and configure the most appropriate cognitive workflow for each request efficiently.
- **LLM Configuration Catalog**: This catalog holds predefined configurations for each LLM agent, allowing for quick access and deployment of optimized agent settings based on task requirements. The catalog includes hyperparameters, resource allocations, and specifications for each agent to ensure efficiency and task alignment.
- **Deployment Metamodels Catalog**: To support flexible deployment across diverse environments, such as cloud or edge systems, the Knowledge Layer maintains a set of deployment models that detail the proper congnitive workflow configurations for each operational scenario. These models ensure that SALLMA can dynamically adapt to varying infrastructure constraints, balancing latency, compute power, and resource availability.

Together, these catalogs and archives enable SALLMA to operate with both high efficiency and adaptability. Specifically, by maintaining a structured yet flexible archive of agent configurations and workflows, the Knowledge Layer serves as the essential backbone of the system, enabling SALLMA to dynamically respond to diverse, complex tasks with optimized configurations.

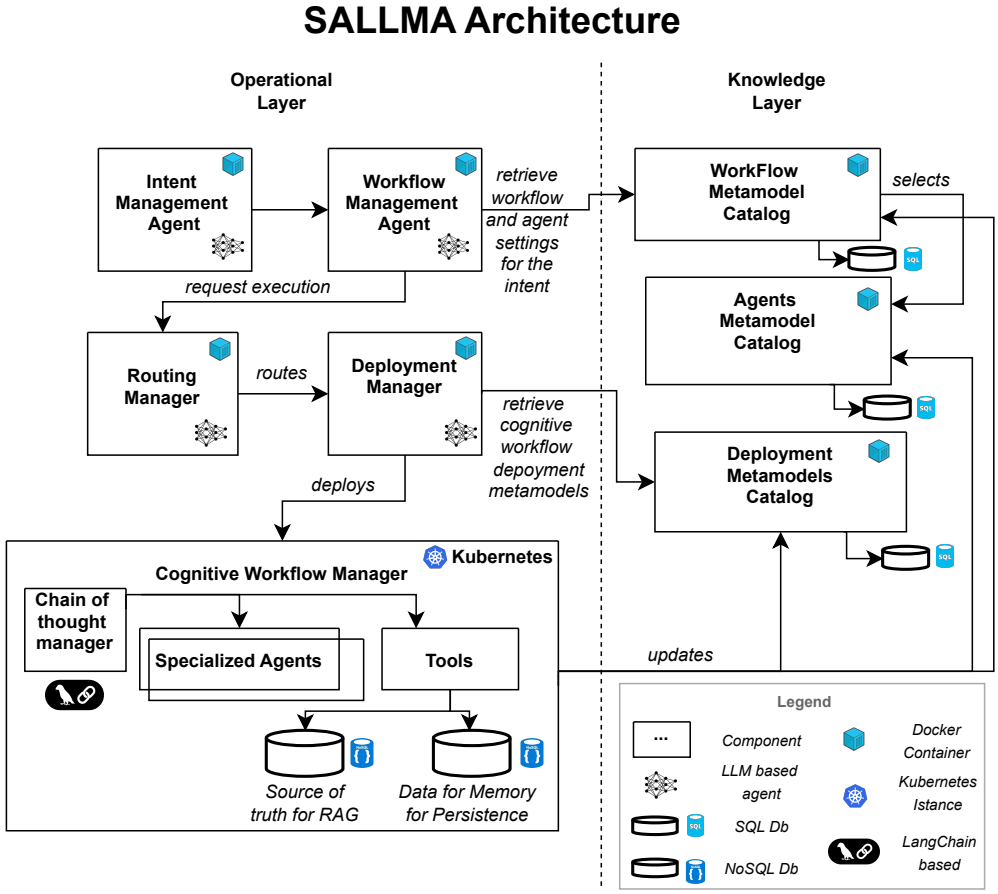

Fig. 1: SALLMA Architecture Overview — A distributed, multi-agent system featuring an Operational Layer for real-time task execution and a Knowledge Layer for workflow and agent configuration meta model management.

IV. Key Architectural Decisions and Assumptions

In this section, we briefly outline the architectural decisions and architectural assumptions we deem most important to report to support the design documentation of SALLMA.

In designing SALLMA's architecture, the most relevant choice was adopting a modular and distributed Multi-Agent System architecture. This approach enables flexible deployment of components essential to cognitive workflows, allowing for the optimal selection and placement of components within each workflow for deployment in the most effective locations.

As additional key architectural decision, each component and its inter-dependencies are positioned based their requirements, such as resource demands and latency constraints. This flexibility ensures resource-efficient deployment and enhances the system's responsiveness to diverse task requirements. This also enables the possibility to add constraints, e.g., in terms of energy consumption, a perspective important in terms of the evergrowing environmental sustainability of LLMs.

The SALLMA architecture presumes that (i) reliable network bandwidth and adequate computational resources is always available, and (ii) all components and agent modules operate within secure environments that fully enable communication between components.

V. SALLMA Proof of Concept Implementation and Key Technologies

In this section, we outline the technologies we used to develop a proof of concept (POC) of SALLMA. The POC was created to explore the feasibility and operational dynamics of the proposed architecture from an exclusively functional suitability point of view. In preliminary in-house experiments focused on a small-scale set of tasks, SALLMA effectively demonstrated multi-agent orchestration and retention of contextual information, showing promise for handling diverse requests. While these results are primarily qualitative, they confirm the viability of a multi-agent, workflow-based approach. Notably, SALLMA provides a flexible blueprint that defines fundamental components and specifies their interactions within a distributed, multi-agent framework. The POC was designed to assess SALLMA's capacity to support scalable

agent collaboration and complex task management, though alternative configurations are possible.

The POC has been implemented using containerized Large Language Models (LLMs) managed with Docker and orchestrated via Kubernetes with Python as the primary codebase. To streamline LLM management, especially for database integrations in operations like retrieval-augmented generation (RAG), SALLMA utilizes LangChain[1] due to its adaptability in managing multi-step workflows. LangChain enables agents to retrieve, process, and act on contextual information in real time, efficiently orchestrating RAG workflows where information from the SALLMA database is dynamically incorporated into responses.

Within SALLMA's modular architecture, we use Docker for containerization, enabling each LLM-powered agent to operate independently within its own dedicated environment. Hugging Face[2], a public repository of open large language models (LLMs), is integrated to provide access to a vast catalog of models, facilitating seamless deployment of diverse options like Mistral[3] and LLaMA[4]. Docker was chosen for its industry-standard status and strong compatibility with Hugging Face, which provide a broad range of choice for agents.

The SALLMA POC utilizes Kubernetes to manage and orchestrate containerized deployments for the Cognitive Workflow Manager (see Section III-B). Kubernetes provides the essential infrastructure for deploying, scaling, and managing containerized applications, enabling dynamic resource allocation across nodes that host LLMs, application tools, and databases. The tools are Python-based applications, with SQL databases used for structured data and NoSQL databases for unstructured data. This setup ensures that the SALLMA POC can achieve balanced load distribution, high availability, and efficient resource utilization. Kubernetes was chosen as the *de facto* standard for distributed architectures, offering robust support for modular and scalable deployment of components within the SALLMA system. The Knowledge Layer in SALLMA is built with Python-based components that access SQL and NoSQL databases to store meta-models and configurations in real time.

As an initial exploration of the architectural benefits afforded by separating the Operational and Knowledge Layers, we conducted a small-scale, in-house evaluation using a set of ad hoc tasks. Specifically, we deployed one cognitive workflow requiring software engineering knowledge—particularly design patterns—and another focused on managing students' bureaucratic needs. Requests were routed to SALLMA's Intent Management Agent, which successfully summoned the appropriate Cognitive Workflow for each domain. In both cases, relevant contextual data was retrieved via retrieval-augmented generation (RAG), enabling more accurate and context-aware responses. Although these observations remain anecdotal and limited in scope, two key insights emerged: (i) that distinguishing between Operational and Knowledge Layers appears to improve how context is managed, resulting in better responses and (ii) that having dedicated agents for specific tasks facilitated independent scaling, replacement, and reconfiguration, ultimately improving responsiveness when handling multiple domains or heavier workloads. We emphasize that these findings are preliminary. Robust evaluations—particularly in large-scale or industrial environments—are needed to quantify improvements in performance, resource utilization, and user-perceived response quality.

[1]https://www.langchain.com. Accessed 11th November 2024.

[2]https://huggingface.co. Accessed 11th November 2024.

[3]https://mistral.ai. Accessed 11th November 2024.

[4]https://www.llama.com. Accessed 11th November 2024.

VI. Conclusion and Future Work

In this contribution we present SALLMA, a architecture designed to overcome single-agent LLM limitations by implementing a distributed multi-agent system panning cloud-to-edge environments. Through modular, task-specific agents coordinated through a cognitive framework, SALLMA can enable efficient, adaptive workflows meeting real-time operational demands.

Concretely, a PoC of SALLMA was implemented by focusing primarily on exploring the functional suitability of the architecture. In a follow up stage, SALLMA was deployed in an industrial setting, namely to provide information on behalf of public administration offices, and is currently utilized in a business simulation scenario.

We aim toward further and collaborative exploration in the evolving field of multi-agent LLM architectures. The intersection of LLMs and software architecture, though still nascent, presents expansive opportunities. As future work, we plan to evaluate and tune SALLMA *via* industrial case studies, study alternative deployment configurations, and explore the integration of diverse LLMs across different sectors.

Acknowledgment

This research paper was developed with assistance of large language models to enhance the clarity of its written text.

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART").

References

[1] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Vision and the road ahead," *arXiv preprint arXiv:2404.04834*, 2024.

[2] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[3] N. R. Jennings, "A survey of agent-oriented software engineering," *Knowledge Engineering Review*, vol. 15, no. 4, pp. 215–249, 2000.

[4] I. K. Aksakalli, T. Çelik, A. B. Can, and B. Tekinerdoğan, "Deployment and communication patterns in microservice architectures: A systematic literature review," *Journal of Systems and Software*, vol. 180, p. 111014, 2021.

[5] M. Fowler, "Dealing with properties." https://martinfowler.com, June 1997. Available at https://martinfowler.com.