ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://doi.org/10.36930/40350412
Article received 23.06.2025
Article accepted 26.06.2025
UDC 004.[896+8]
Correspondence author: A. R. Bidochko andrii.r.bidochko@lpnu.ua
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

A. R. Bidochko, Ya. I. Vyklyuk
Lviv Polytechnic National University, Lviv, Ukraine

LLMAGENTNET: A COLLABORATIVE NETWORK OF AUTONOMOUS AI AGENTS FOR COMPLEX TASK EXECUTION

Recent advancements in Large Language Models (LLMs) have opened new horizons for building autonomous AI agents capable of executing complex, multi-step tasks across a wide range of domains. Despite this progress, existing multi-agent systems often suffer from limited interoperability and extensibility due to rigid communication protocols, closed ecosystems, and monolithic architectural assumptions. To address these limitations, we introduce LLMAGENTNET – a collaborative, modular framework designed for orchestrating multiple autonomous AI agents that leverage LLMs for planning, reasoning, and execution. The LLMAGENTNET framework is based on a decentralized agent architecture in which agents can communicate and collaborate using standardized RESTful API calls. By utilizing the OpenAPI Specification, LLMAGENTNET enables seamless integration of both native and external agents, including those developed using other frameworks or hosted on different platforms. Each agent in the system is capable of interpreting high-level user requests, breaking them down into sub-tasks, selecting appropriate tools or agents for execution, and maintaining contextual memory across task sequences. A distinctive feature of LLMAGENTNET is its hierarchical Agent Manager (Orchestrator), which coordinates interactions, maintains execution trajectories, and optimizes the use of specialized agents. The system supports multiple operational modes – Human-in-the-Loop, Human-on-the-Loop, and Human-out-of-the-Loop – enabling both supervised development and fully autonomous operation. Notably, LLMAGENTNET includes a memory mechanism that allows agents to access and update knowledge bases via vector databases (e.g., ChromaDB), supporting long-term memory, runtime variables, and contextual reasoning. The framework has been benchmarked using standard datasets such as HotPotQA and in practical use cases such as content marketing automation, demonstrating significant improvements in efficiency and quality over manual or single-agent approaches. By enabling collaboration, reflection, and self-improvement among AI agents, LLMAGENTNET contributes to the development of general-purpose intelligent systems and marks a critical step toward the realization of Artificial General Intelligence (AGI).

Keywords: Large Language Models (LLMs); AI Agents; Autonomous Systems; Multi-Agent Framework; Task Planning; Agent Collaboration.

Introduction / Вступ

The rapid advancement of Generative AI [23, 26, 37] has significantly influenced the development of AI agents, enabling them to perform a wide range of complex tasks with remarkable proficiency using Large Language Models (LLMs [3, 5, 24]). These models, such as GPT-4, are highly skilled in understanding and generating natural language, making them indispensable for creating intelligent agents capable of interacting with tools [27, 29, 31] and environments while receiving dynamic feedback [6, 15].

A longstanding goal in AI research is to develop agents that can solve problems at a human level, exhibiting the capacity to generalize and adapt across a wide array of tasks. LLM-powered agents, such as AutoGPT [34] and BabyAGI [21], represent important steps towards this vision. These agents have demonstrated their ability to autonomously perform complex, multi-step tasks [6, 40, 43] by leveraging LLMs to guide their decision-making processes. As a result, they have garnered attention from both technical experts and broader audiences due to their potential to automate workflows and reduce human effort in areas like customer service [30], programming [12, 14, 22, 40], and content creation [19, 38, 47].

One of the key strengths of LLM agents is their ability to reflect [32, 43] and reasoning (ReAct [45]) strategies based on real-time feedback from their environments. This interaction enables them to continuously refine their strategies and improve task performance, mimicking human adaptability in real-world scenarios. However, building truly adaptive agents remains a challenge, as most LLM-based systems still rely on static prompts and predefined behavi-

Інформація про авторів:
Бідочко Андрій Ростиславович, здобувач, кафедра систем штучного інтелекту. **Email:** andrii.r.bidochko@lpnu.ua; https://orcid.org/0009-0001-4083-5040
Виклюк Ярослав Ігорович, д-р техн. наук, професор, кафедра систем штучного інтелекту. **Email:** yaroslav.i.vykliuk@lpnu.ua; https://orcid.org/0000-0003-4766-4659
Цитування за ДСТУ: Бідочко А. Р., Виклюк Я. І. LLMAGENTNET: колаборативна мережа автономних агентів штучного інтелекту для виконання складних завдань. Науковий вісник НЛТУ України. 2025, т. 35, № 4. С. 101–114.
Citation APA: Bidochko, A. R., & Vyklyuk, Ya. I. (2025). LLMAGENTNET: a collaborative network of autonomous ai agents for complex task execution. *Scientific Bulletin of UNFU*, 35(4), 101–114. https://doi.org/10.36930/40350412

ors, limiting their ability to learn from past experiences and evolve dynamically. Unlike humans, who adjust their strategies based on feedback, most agents lack this crucial ability to adapt to changing environments.

The next frontier in AI research involves creating agents that can learn and evolve through interaction, similar to human learning. BOLAA [17] explores various agent architectures and suggests employing a multi-agent framework to break down and manage the complexity of tasks more effectively. Similarly, the Internet of Agents (IoA) [4] framework introduces a flexible platform for dynamic collaboration between diverse agents, further enhancing scalability and adaptability in distributed environments.

This paper aims to address this challenge by proposing an architecture for AI agents in multiagent systems (MAS) [8, 9, 10] that leverages REST API requests as the primary communication method. REST API, being a widely adopted protocol, allows for seamless and scalable interactions between agents in complex environments. Our proposed architecture supports both basic and automated modes, where a single request can generate a multi-step plan, dynamically adjusting during execution to improve decision-making and task outcomes.

Building on frameworks like AgentLite [47] and RestGPT [36], we integrate essential components such as planner, executor, memory system, and communication modules to ensure robust planning, execution, and coordination among agents. These components work together to enable the agents to handle more intricate tasks by continuously refining their strategies based on real-time feedback.

Object of research – the use of a collaborative network of autonomous artificial intelligence agents to perform complex tasks.

Subject of research – is the set of architectural methods, orchestration techniques, and communication tools used to enable the integration and autonomous interaction of artificial intelligence agents in multi-agent systems in order to enhance their scalability, adaptability, and practical applicability in intelligent distributed environments.

The purpose of the research – the develop and validate a collaborative architecture of AI agents within a multi-agent system based on large language models, ensuring its effectiveness through experimental benchmarking and real-world application.

To achieve this purpose, the following *research objectives are identified*:

- to design a modular and extensible agent infrastructure, which will enable seamless integration of external tools and services, thereby supporting system interoperability;
- to implement a hierarchical Agent Manager for task decomposition and delegation, which will provide adaptive control over dynamic task execution.
- to validate the architecture through standardized benchmarks (e.g., HotPotQA) and real-world business applications, which will ensure objective assessment of performance and generalizability;
- to investigate various levels of agent autonomy – from human-in-the-loop to fully self-directed modes – which will demonstrate the potential for workload reduction and productivity enhancement;
- to introduce reinforcement techniques based on agent reflection, which will minimize execution errors and enable self-improvement without human oversight;
- to apply the system in cognitive and business domains, which will contribute to the development of multimodal AI solutions, intelligent interfaces, and hypermedia-driven applications.

These research directions not only demonstrate the feasibility and efficiency of the proposed system but also contribute to the advancement of intelligent interface design, the application of hypermedia technologies, and the development of multimodal AI solutions for business and cognitive computing tasks.

Analysis of recent research and publications. In the study by [40], the authors introduced *AutoGen*, a pioneering framework that demonstrates how Large Language Models (LLMs) can engage in structured multi-agent conversations to solve complex tasks through role-based collaboration. The system enables the definition of agents with specific roles (e.g., planner, executor, coder) and facilitates their communication via scripted or interactive dialogue patterns. However, the framework is limited by its reliance on pre-configured agent roles and static communication templates, reducing its adaptability for dynamic orchestration. Furthermore, integration with external tools often requires custom code rather than standardized interfaces. In contrast, *LLMAgentNet* adopts a more extensible and modular design based on RESTful APIs and OpenAPI schemas, supporting runtime flexibility and plug-and-play interoperability between heterogeneous agents.

In the work of [18], the authors presented *AgentLite*, a lightweight and extensible framework aimed at constructing task-oriented LLM agents with an emphasis on modularity. It separates memory, prompting, and action handling into distinct components, enabling rapid prototyping and reuse. While *AgentLite* promotes developer flexibility and supports hierarchical agents, it does not incorporate standardized communication protocols or schema-driven interaction, and its coordination remains developer-centric. *LLMAgentNet* builds upon this modular foundation by integrating schema-aware RESTful APIs, enabling real-time orchestration and standardized interaction across distributed agents and services.

In their study, [36] proposed *RestGPT*, a framework that bridges LLMs with real-world RESTful APIs using OpenAPI specifications and a coarse-to-fine planning mechanism. This enables LLMs to reason over available APIs, generate valid sequences of actions, and reduce hallucinations. While *RestGPT* enhances the reliability of single-agent systems interacting with external services, it lacks support for multi-agent coordination or inter-agent workflows. *LLMAgentNet* extends the ideas from *RestGPT* to a collaborative, multi-agent environment by using OpenAPI both for API integration and as a unified communication interface between autonomous agents.

The *ReAct* framework, introduced by [45], proposed a method of interleaving reasoning (thoughts) and actions in an iterative loop to enable dynamic problem solving within a single LLM agent. By making intermediate reasoning steps explicit, *ReAct* supports decomposition of tasks into logical phases. However, it remains confined to single-agent settings and often relies on hardcoded in-context prompts. *LLMAgentNet* generalizes the ReAct paradigm by distributing the reasoning – action loop across multiple agents and enabling real-time exchange of outputs through a shared memory mechanism, coordinated via RESTful and schema-based interactions.

In the study by [29], the authors developed *ToolLLM*, a model fine-tuned on tool-related instructions to enable LLMs to autonomously access and utilize over 16,000 APIs. The system relies on a large indexed API database

and performs tool retrieval and structured action generation. While *ToolLLM* demonstrates impressive scale and generalization, it lacks mechanisms for runtime adaptation or inter-agent collaboration. *LLMAgentNet* addresses these limitations by incorporating dynamic feedback loops, enabling agents to refine actions based on execution context, and supporting real-time interaction among multiple agents.

In the work by [12], the authors introduced *MetaGPT*, a multi-agent framework inspired by software development practices. It uses a hierarchical structure where agents are assigned fixed roles such as Product Manager or Developer and collaborate via a shared memory. While this structured workflow supports alignment and consistency, the rigid role templates and lack of runtime extensibility limit its adaptability in open-ended environments. *LLMAgentNet*, by contrast, employs dynamic task composition through OpenAPI-defined agent interfaces, allowing heterogeneous agents to be orchestrated at runtime based on emerging task demands.

In the work by [16], the authors introduced the effectiveness of large language models for query expansion in code search. Language Models (LMs) are deep learning models trained on massive amounts of text data. One of their main advantages is their superior language understanding capabilities. This study explores the application of Large Language Models (LLMs) understanding capabilities in code search query expansion. To this end, we collected a query corpus from multiple data sources and trained multiple LMs (GPT2, BERT) on this query corpus using a self-supervised task. The trained LM models are then used to expand the input query. We evaluate the performance of these LLMs on the CodeSearchNet dataset using two state-of-the-art code search methods (GraphCodeBERT and CoCoSoda) and compare these LLMs with currently popular expansion methods. Experimental results show that LLM-based query expansion methods outperform existing query reformulation methods in most cases.

In their early-stage experiments, [34] and [21] developed *AutoGPT* and *BabyAGI*, which implement goal-driven autonomous workflows using LLMs. These systems decompose tasks iteratively and use memory to guide future actions. However, they lack standardization, modular integration mechanisms, and formal APIs for agent interaction. Their orchestration logic is simplistic and does not scale well for complex, multi-agent scenarios. *LLMAgentNet* formalizes these concepts by introducing schema-based planning and RESTful multi-agent communication, enabling structured, scalable, and interoperable coordination.

In the study by [1, 2, 7, 13, 20], the emergence of agent communication protocols such as MCP, A2A, ACP, and ANP reflects an important effort to standardize agent interoperability. These protocols offer features like peer discovery, schema negotiation, and structured messaging, primarily through JSON-RPC. However, they often rely on rigid schemas or predefined ontologies, limiting real-time adaptability and integration with external tools. *LLMAgentNet* complements these approaches by abstracting communication through OpenAPI and HTTP-based microservice interaction, allowing agents to discover and invoke services dynamically, without being tightly coupled to specific protocol stacks.

In summary, while the emergence of modern communication protocols such as MCP, A2A, ACP, and ANP marks significant progress toward interoperability in multi-agent LLM systems, these efforts often rely on static schemas, rigid messaging semantics, or tightly coupled communication stacks that limit adaptability at runtime. In contrast, *LLMAgentNet* introduces a novel, framework-agnostic approach that leverages RESTful APIs and machine-readable OpenAPI specifications to enable dynamic discovery, orchestration, and plug-and-play integration of heterogeneous agents and tools. Rather than replacing existing protocols, *LLMAgentNet* complements them by abstracting protocol complexity through standardized web interfaces, offering greater flexibility, extensibility, and accessibility for real-world deployment. This distinct capability to support runtime configuration, protocol-agnostic communication, and seamless interoperability across agent ecosystems clearly highlights a gap in current research and underscores the necessity and originality of the proposed framework.

Research results and their discussion / Результати дослідження та їх обговорення

The development of AI agents within multiagent systems (MAS) has been a rapidly evolving field, driven by the advancements in Large Language Models (LLMs) and the need for more sophisticated and cooperative AI solutions. This section reviews the significant contributions to the field, highlighting the frameworks, theories, and practical implementations that have shaped current MAS research (Tab. 1). A comparative overview of these protocols is provided in Table 1, highlighting key differences in communication models, extensibility, and orchestration capabilities.

AI Agent Frameworks. Several frameworks have been pivotal in the development of AI agents, particularly those leveraging LLMs. AgentLite is one such framework that simplifies the creation and orchestration of task-oriented agents. It focuses on lightweight code architecture and hierarchical multi-agent systems, allowing for easy customization and integration of various agent components such as prompts, memory, and actions. RestGPT, another notable framework, integrates LLMs with RESTful APIs to handle complex instructions through a coarse-to-fine online planning mechanism. It addresses the challenges of API understanding, planning, and response parsing, showcasing superior extensibility and flexibility compared to other tool-augmented LLMs.

Theoretical Foundations. Theoretical advancements in MAS include the Extended Coevolutionary (EC) Theory, which incorporates coevolutionary dynamics, adaptive learning, and LLM-based strategy recommendations to model and analyze interactions among heterogeneous agents. This theory goes beyond traditional game theory by addressing diverse interactions and learning capabilities among agents, promoting cooperative behavior and system robustness.

Game theory remains a foundational framework for understanding strategic interactions within MAS. Concepts like Nash Equilibrium and Stackelberg Equilibrium have been applied to optimize task allocation and decision-making processes among agents. However, challenges such as defining appropriate payoff structures and achieving equilibrium states efficiently continue to spur ongoing research.

Research Directions. The integration of feedback mechanisms in MAS is a critical area of research, aiming to enable agents to learn from their interactions and adapt their strategies for better performance. Both inter-agent and self-feedback mechanisms are being explored to enhance the learning and decision-making capabilities of agents within a multi-agent environment.

In summary, the research and development of AI agents in multiagent systems have significantly advanced through frameworks like AgentLite and RestGPT, theoretical models such as EC Theory, and practical implementations in dynamic and collaborative environments. These contributions continue to push the boundaries of what MAS can achieve, paving the way for more sophisticated and resilient AI systems.

Table 1. Comparison Table: AI Agent Frameworks and Protocols (based on the authors' own experimental results) / Порівняльна таблиця: фреймворки та протоколи агентів ШІ (власні узагальнення на основі авторського аналізу джерел)

Model	AI Agent Frameworks						Protocols						
	Extendability	Schema Communication	Context Optimization	Multi-tool Scenario	Real-world API	Num. of Tools	Feedback	Multi-Agent Orchestration	API Interface Generation	Code Generation	Plug-n-Play	Agent Memory	Planning
ReAct	✗	Specialized	✗	✗	✗	3	✓	✗	✗	✗	✓	✗	Online
CrewAI	✓	Specialized	✗	✓	✓	70	Human	✓	✗	✓	✓	✗	✗
HuggingGPT	✓	HuggingFace	✗	✓	✓	28	✗	✗	✗	✗	✓	✓	Offline
API-Bank	✗	Specialized	✗	✗	✓	53	Human	✗	✗	✗	✓	✗	✗
Toolformer	✗	Specialized	✗	✗	✗	5	✗	✗	✗	✗	✗	✗	✗
GPT4Tools	✗	Specialized	✗	✗	✗	31	Human	✗	✗	✓	✗	✓	✗
ToolBench	✓	JSON	✗	✓	✓	232	Human	✗	✗	✗	✓	✗	✓
AutoGPT	✓	Specialized	✗	✓	✓	81	Human	✓	✗	✓	✓	✓	✓
MetaGPT	✓	JSON	✗	✓	✗	90	✓	✓	✓	✓	✓	✗	✓
BabyAGI	✗	JSON	✗	✗	✗	7	✗	✗	✗	✓	✗	✗	✓
MCP (Model-ContextProtocol)	✓	JSON / OAS	✓ (Prompt Linking)	✓	✓	Tool-agnostic	Agent Feedback	✓	✓	External	✓	Optional	✓
A2A (Agent2Agent by Google)	✓	JSON-RPC 2.0	✓ (Linked Contexts)	Protocol-level	✓	Protocol-only	System-level	✓	Protocol Level	External	✓	✗	✓
ACP (Agent Communication Protocol by IBM)	✓	REST	✗	Limited	✓	Protocol-only	✗	✗	✗	✗	✗	✗	✗
ANP (Agent Network Protocol)	✓	JSON + Type	✗	✗	✓	Modular	✗	✓	✓	✓	✓	✗	✗
LLMAgentNet	✓	REST	✓	✓	✓	Tool-agnostic	✓	✓	✓	✓	✓	✓	✓

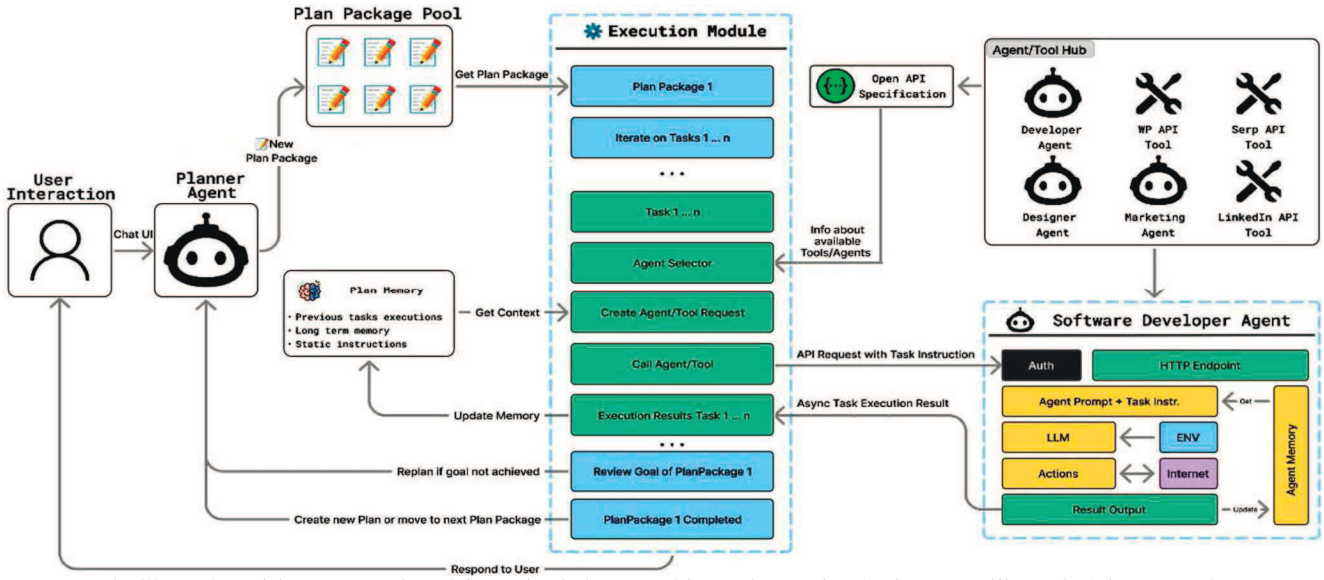

Figure 1. The illustration of the conceptual workflow of task decomposition and execution (authors' own illustration) / Ілюстрація концептуального робочого процесу декомпозиції та виконання завдань (авторська розробка)

Architecture of AI Agent in Multiagent Systems. To construct an effective AI agent within a multiagent system (MAS), several essential components must work together to ensure robust task planning, execution, and communication. Each of these core components plays a critical role in enabling agents to handle tasks autonomously, while still being able to collaborate with other agents. The following sections will outline these key components in more detail (based on the authors' own analysis and system design) (Fig. 1):

1) *Planner*: The Planner is responsible for breaking down complex tasks into a series of smaller, manageable actions. It generates a detailed plan for each task, allowing the agent to follow a structured workflow.
2) *Executor*: Once a plan is created, the Executor carries out each action step-by-step. It ensures that the task is executed in alignment with the plan, interacting with external systems and other agents as needed.
3) *Memory*: The Memory component stores past actions, decisions, and results. This allows the agent to reference histo-

rical data, improving decision-making by learning from previous experiences.

4) *Communication Module*: Effective communication between agents is critical for collaboration. The Communication Module ensures smooth information exchange and coordination, allowing agents to share data and request information from one another.
5) *Agent Manager*: The Agent Manager oversees the operations of multiple agents, assigns tasks, monitors progress, and ensures all agents work efficiently towards the collective goal.

Figure 1 illustrates the task workflow in a multiagent system. It begins with a task request, where the system, using an LLM, creates a plan with multiple action items. The agent then executes each action step-by-step, saving and verifying the results after each one. The plan is updated in real-time if needed. Once all steps are completed, the agent finishes the process and informs the user, with the ability to autonomously generate new plans for continuous execution.

Planner. The Planner serves as the brain of the AI agent, tasked with breaking down complex, high-level user requests into smaller, manageable sub-tasks. It leverages the powerful natural language processing capabilities of Large Language Models (LLMs) to generate detailed and actionable plans, guiding the agent step-by-step through task execution. The workflow of the Planner module is clearly shown in Figure 2, which guides us through each step – from receiving a user request to generating and executing tasks. This visual flow makes it easier to understand how the Planner organizes and manages tasks (Fig. 2).

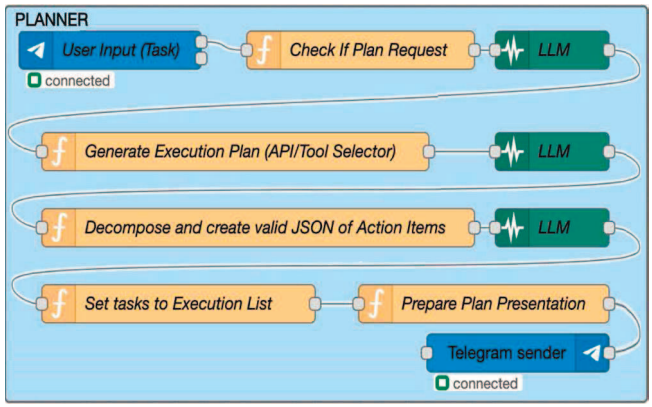

Figure 2. Visual flow of the Planner module, from user request to task execution (author's own illustration) / Візуальний потік модуля Планувальник: від запиту користувача до виконання завдань

To create the plan, the Planner must be aware of all available tools and agents, represented as REST APIs, that assist with each step. These tools are described using the Open API standard, detailing their purpose, usage, and parameters. The Planner uses a custom prompt (see Appendix) that combines the user's request with the list of available tools in JSON format. This standardized approach ensures that the system can easily manage, integrate, and extend new tools or agents as needed.

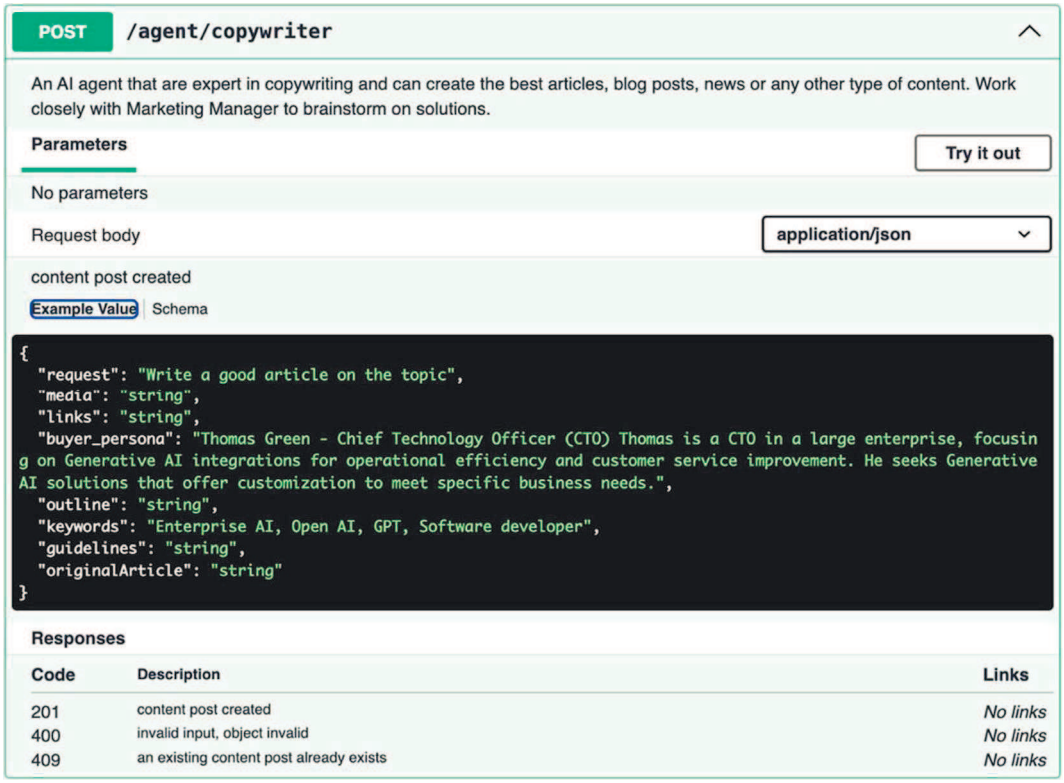

Figure 3. Structure of a Tool/Agent using the OpenAPI Specification (author's own illustration) / Структура інструменту/агента з використанням специфікації OpenAPI (авторська розробка)

Each *Tool/Agent* in the system is defined using the OpenAPI Specification, which provides the necessary metadata for proper functionality within the multi-agent system. This metadata includes details about when a tool should be used and how it operates, ensuring that the LLM can select the appropriate tool for each step in the plan.

Once the tool or agent is chosen, the LLM constructs the request payload by referencing the OpenAPI Specification, which outlines key elements such as the *parameters*, *method*, and *URL* required to interact with the tool. This structured approach ensures seamless integration and efficient task execution by automating how requests are created and sent.

In Fig. 3, we can see a clear example of how a *Tool/Agent* is described using the OpenAPI Specification. It shows the necessary information for selecting and interac-

ting with the tool, including request structure, parameters, and expected responses. This design allows the system to flexibly adapt to new tools and extend its capabilities as needed. Further details on how these interactions are executed will be explored in the *Execution Module*, where we will cover the process of handling requests and responses in greater depth.

After generating the plan, it is converted into an actionable array of steps and saved in JSON format. This ensures that the system can process and execute each step efficiently and in the correct sequence. Once the plan is complete, it is stored in the global state and added to the execution queue, allowing the system to manage when and how each task is executed. Some tasks are performed immediately, while others may be scheduled for future execution based on the plan's meta information, such as timing or specific conditions.

The structure of the Plan Package and its tasks is visually outlined in Figure 4, providing a clear representation of how tasks are organized, executed, and tracked within the system (Fig. 4).

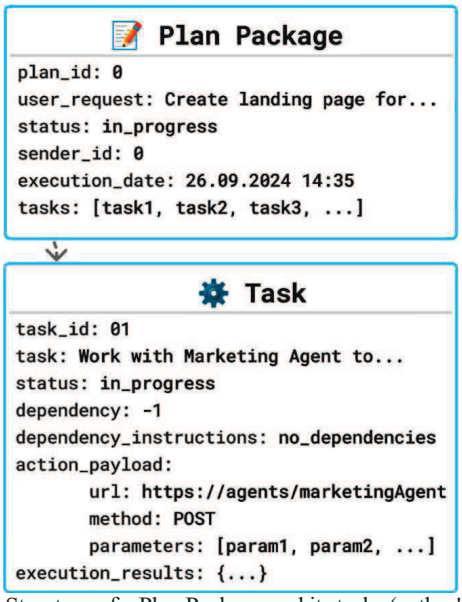

Figure 4. Structure of a Plan Package and its tasks (author's own illustration) / Структура пакету плану та його завдань

This structured approach allows the multi-agent system to manage tasks flexibly, ensuring both immediate execution and future scheduling when necessary. It optimizes task management, making the system scalable and adaptable to real-world applications.

Executor. The Executor plays a crucial role in carrying out the tasks defined by the Planner, executing them by interacting with external APIs and other agents. It is the most computationally demanding part of the system, as it processes each task and ensures that the appropriate Tool/Agent is used for the execution.

Once a plan is generated, the Executor begins by identifying the next incomplete task. It uses the OpenAPI Specification to gather detailed information about the Tool/Agent that was pre-selected during the planning phase. Before proceeding, the Executor validates that the selected Tool/Agent is still appropriate for the task, checking the results of previously executed steps in the current plan. This validation helps the system stay flexible and responsive to changes as tasks are executed (Fig. 5).

With the tool verified, the LLM model generates the payload required to call the Tool/Agent. The model uses the task details and metadata from the OpenAPI Specification to produce a valid JSON request, ensuring all necessary parameters and data are correctly formatted. This payload serves as the input for the API call to the external agent or tool (Fig. 6). The Executor then makes the API call using the generated request. Upon receiving a response – usually in JSON format – the system extracts the relevant information from the response, filtering out any unnecessary or redundant data. The extracted data is used to update the system's memory and inform future tasks in the plan.

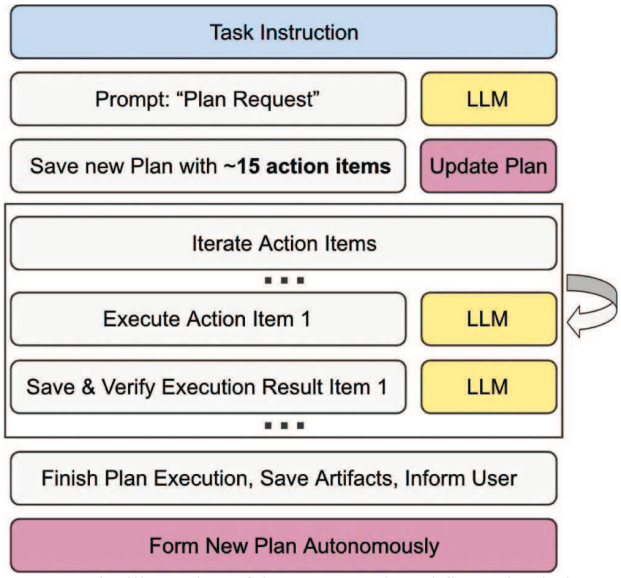

Figure 5. The illustration of the conceptual workflow of task decomposition and execution (author's own illustration) / Ілюстрація концептуального робочого процесу декомпозиції та виконання завдань

This process is illustrated in Figure 6, which shows the step-by-step flow of how the Executor handles each task, from retrieving the plan and generating the API payload, to executing the API call and updating the plan. The figure highlights the integration of LLMs and how they aid in generating requests, handling responses, and managing task execution efficiently. If any issues arise during execution, such as errors in the API response, the Executor can prompt the LLM model to assist in interpreting or correcting the response. This ensures the system can handle unexpected errors or exceptions gracefully.

Once the task is successfully executed, the plan is updated, and the task is marked as completed. The Executor then moves on to the next task, repeating this process until all steps in the plan are completed. By iterating through the tasks and continuously verifying and executing them, the Executor ensures smooth operation and real-time updates, enabling the system to dynamically adapt to new situations as they arise. In essence, the Executor not only handles the task execution but also ensures the proper flow of information, validates responses, and updates the system's memory, making it a critical component for the overall success of the multi-agent system.

Memory. The Memory Module is a crucial component within LLMAGENTNET, providing agents with a persistent and structured memory that allows them to manage complex tasks effectively and learn from past interactions. This module serves multiple critical functions, from tracking previous task results to maintaining a continuously updated registry of available agents and tools within the multi-agent system.

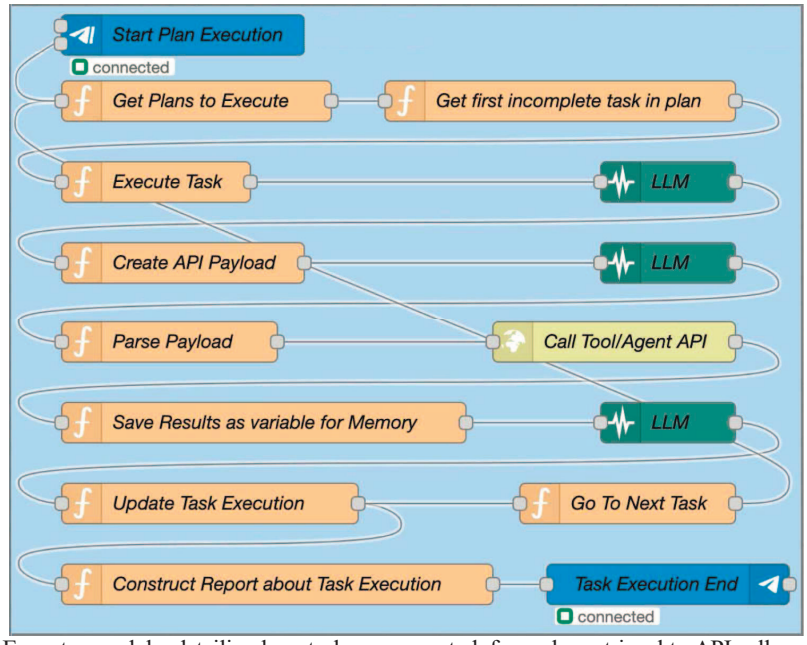

Figure 6. Workflow of the Executor module, detailing how tasks are executed, from plan retrieval to API calls and plan updates (author's own illustration) / Робочий процес модуля Виконавець: від отримання плану до виконання API-запитів і оновлення плану

Task Execution Memory. One of the primary roles of the Memory Module is to maintain continuity across task executions. By storing the outcomes and details of each completed task, agents gain contextual awareness, allowing them to make informed decisions based on previous results. This continuity reduces redundant processing, as agents can recall relevant information rather than reprocessing identical or similar tasks. For complex, multi-step workflows, this memory function ensures that agents can adapt their approach based on historical data, enhancing task efficiency and precision.

Runtime Variables as Memory. A unique feature of the Memory Module in LLMAGENTNET is its use of runtime variables as memory, which addresses token and context limitations common in Large Language Models (LLMs). This approach optimizes communication between AI agents in a Multi-Agent System (MAS) by reducing the data payload during task execution, allowing agents to work efficiently within LLMs' token constraints, as shown in Fig. 7.

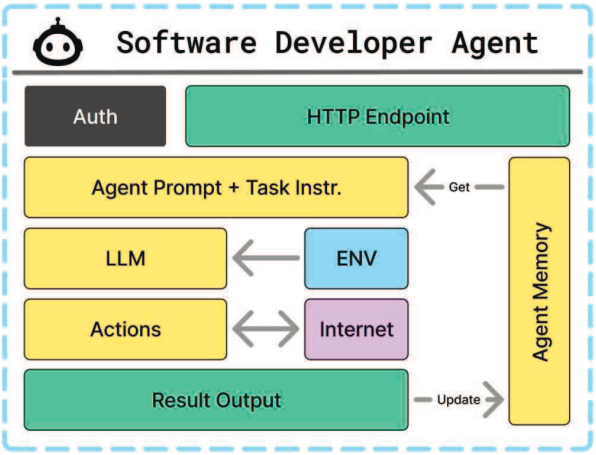

Figure 7. Single Agent architecture and memory modules (author's own illustration) / Архітектура одного агента та модулі пам'яті (авторська розробка)

LLMs, despite their powerful processing capabilities, have fixed token limits that restrict the amount of information they can handle in a single prompt. By using runtime variables, LLMAGENTNET can work around these limitations, efficiently managing data that exceeds the context window (Fig. 8).

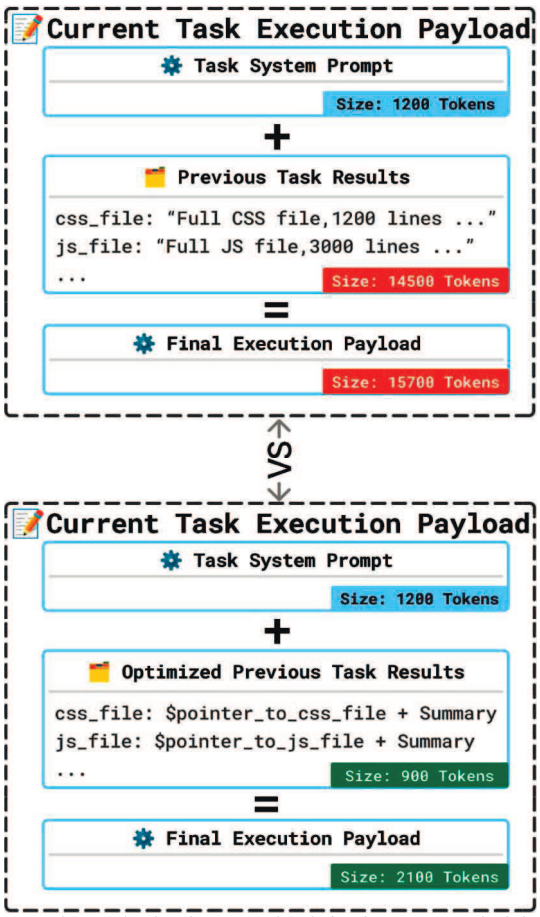

Figure 8. The method reduces payload size by 7.5 x, optimizing data transmission and fitting within context limits (author's own illustration) / Метод зменшує розмір даних у 7.5 разів, оптимізуючи передачу та вміщення у межі контексту

Let T_i represent a specific task in a sequence of tasks $\{T_1, T_2, \ldots, T_n\}$, where each task T_i has an associated result R_i and a set of runtime variables V_i relevant to that task. To retain essential data while minimizing payload size, the memory module follows these steps (authors' own approach):

1. Extract and Summarize Variables – after the execution of each task T_i, a subset of variables $V_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,k}\}$ is extracted. The variables are summarized to form a representation S_i, where $S_i = f(V_i)$ and f is a summarization function that compresses relevant data for efficient use within the token limit.
2. Store Variables in Runtime Memory – each summary Si and the corresponding result Ri are stored in runtime memory M, which is managed by Node.js. The memory state M at any given time can be represented as: $M = \{(T_1, S_1, R_1), (T_2, S_2, R_2), \ldots, (T_i, S_i, R_i)\}$ where each entry (T_i, S_i, R_i) allows agents to recall and reference prior executions efficiently.
3. Select Variables for Payload – before executing the next task T_{i+1}, the AI agent selects relevant variables from memory M using a prompt to retrieve the required summaries. The set of selected summaries $\{S_j\}$ where $j \leq i$ forms the payload P: $P = \{S_j | j \leq i$ and S_j is relevant to $T_{i+1}\}$.
4. Fulfill Payload with Pointers – for efficient data transmission, the payload P uses pointers p_k instead of full data sets, reducing the total payload size. Let p_k represent a pointer to data d_k stored in runtime memory: $P = \{p_k \mid d_k \in M\}$ where d_k is accessed in its entirety only if needed by T_{i+1}, reducing token usage.

This runtime variable method is critical for LLMAGENTNET's performance, as it ensures that agents can handle extensive workflows without exceeding token or context limits. By dynamically managing memory in this way, agents can adapt to complex, multi-step tasks with minimal latency and increased efficiency, paving the way for scalable and high-performance multi-agent collaboration.

Agent Registry Memory. In addition to tracking task history, the Memory Module is also integrated with the OpenAPI Specification to create a real-time registry of all active agents in the system. Each time a new agent is registered or an existing agent is updated, the memory module records these changes. This updated registry allows the Agent Orchestrator to maintain a clear map of available agents, including details on how to communicate with each one, ensuring smooth and coordinated interactions across the network.

In-Agent Memory with RAG Architecture. Each agent also has its own internal memory module powered by a Retrieval-Augmented Generation (RAG) architecture, which allows it to access and retrieve memory data from a vector database such as ChromaDB. This in-agent memory is tailored to the specific requirements of the agent, enabling it to retrieve relevant task information or agent interactions with high efficiency. By leveraging RAG, agents can perform targeted information retrieval from ChromaDB, which stores data in vectorized form for rapid and accurate access.

The Memory Module thus not only enhances the autonomy and adaptability of individual agents but also strengthens the overall coordination and scalability of LLMAGENTNET. By structuring memory in a way that is accessible, persistent, and updatable, the framework allows agents to operate effectively in dynamic, multi-agent environments, continually improving performance with each interaction.

Communication Module. In LLMAGENTNET, communication between AI agents is based on the HTTP protocol, enabling seamless interaction across systems and easy integration with external tools and services via REST APIs. This setup allows agents to not only interact with each other but also connect with agents built on different frameworks, making LLMAGENTNET versatile and adaptable for complex multi-agent systems.

Each AI agent operates at a unique addressable endpoint, which may be hosted on a dedicated virtual machine or shared across multiple agents on a single server, depending on resource needs. This flexibility allows for efficient scaling and independent management of each agent.

Agents within LLMAGENTNET can call other agents or external tools via API, creating an interconnected network capable of executing complex, distributed tasks. When an agent needs to interact with another, it constructs a payload for the API call based on the OpenAPI Specification and History of Executions.

To ensure precision and contextual relevance, LLMAgentNet uses the GPT-4o model from OpenAI to generate payloads for API calls. The LLM references the OpenAPI Specification for each target agent or tool, identifying required parameters, endpoint URLs, HTTP methods, and headers. Additionally, it considers the History of Executions, which provides context from previous interactions, allowing the payload to incorporate relevant data and avoid redundancy.

Below in Prompt Template 1 is a prompt used to instruct the LLM for payload generation, ensuring clarity and adherence to API specifications:

Prompt Template. Prompt for generating payload to run API calls. You are a helpful assistant. Given a user's question and API tool documentation, output parameters according to the API documentation to successfully call the API to solve the user's question.

Note:

1. Use examples in the API tool documentation to understand the API better.
2. Ensure the output contains all required parameters, and add optional parameters if relevant to the question. If no parameters are needed, return { "Parameters": {} }.
3. Only consider the provided API tool documentation, ignoring mentions of other APIs.
4. Task dependencies are provided in logs from previous questions and answers.
5. Use availableDataVariables from previous task logs as parameters, using variable keys that start with $. Avoid inventing keys if no available variables match.
6. Use keySummary to understand how to utilize $key but not as a parameter.
7. Always output valid JSON format, for example:

json

Copy code

{"URL": "API url", "Method": "API HTTP Method", "Parameters": {"input": [1,2,3]}}

Logs of previous tasks: ${msg?.background}

Current question: ${msg.current_task_object.task}

Main question: ${msg.current_execution_plan.user_request}

API Documentation:

- base URL: ${msg.selectedAPIURL};
- endpoint: ${msg.selectedAPIDoc};

Output should be valid JSON.

Output:

Agent Manager. In LLMAGENTNET, the Agent Manager, also known as the Orchestrator, plays a critical role in coordinating tasks across the multi-agent system. This agent is responsible for determining which tasks will be handled by which tools or agents, managing the distribution of responsibilities in a dynamic, multi-agent environment. The Orchestrator routes data, delegates tasks, and ensures

that each agent or tool receives the necessary parameters and context for successful execution.

The Agent Manager is designed to create a hierarchical multi-agent system. It decomposes complex tasks into smaller, manageable sub-tasks and assigns each sub-task to the most suitable agent or tool in the system. This hierarchical design allows the Orchestrator to leverage the capabilities of specialized agents, optimizing efficiency and enabling agents to function as specific "actions" within a larger task execution framework (Fig. 9).

As shown in Figure 9, the system includes multiple specialized agents – JavaScript PRO, Python PRO, and Node-RED PRO – each equipped with its own vector database serving as a memory module. These vector databases store the agent-specific knowledge, allowing each agent to retain, retrieve, and update information relevant to its specialization. For instance, the JavaScript PRO agent stores all JavaScript-related knowledge, while the Python PRO and Node-RED PRO agents retain expertise in their respective fields.

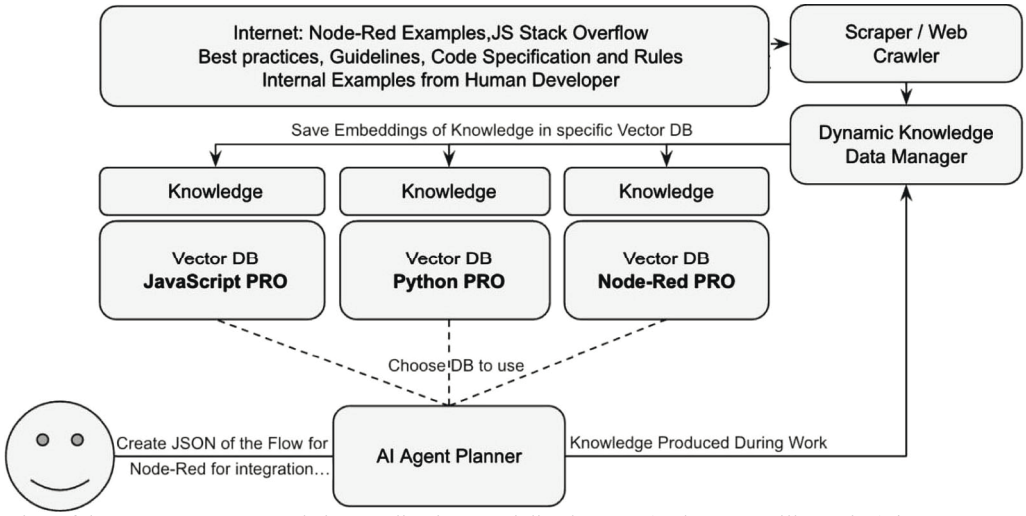

Figure 9. Illustration of the Agent Manager's role in coordinating specialized agents (author's own illustration) / Роль менеджера агентів у координації спеціалізованих агентів (авторська розробка)

The Agent Manager uses these vector databases strategically to enhance system performance. During task execution, the Orchestrator can access the Dynamic Knowledge Data Manager, which compiles knowledge from various sources like online resources, best practices, and internal examples. The Agent Manager then decides which knowledge should be added to a particular agent's vector database, ensuring that each agent's knowledge base remains relevant and up-to-date. This approach allows agents to learn and improve autonomously, as knowledge can be updated continuously based on task outcomes.

Additionally, the Orchestrator evaluates the context of each task and directs the request to the most suitable agent. If the task requires knowledge beyond an agent's existing database, the Orchestrator can instruct the agent to retrieve updated knowledge from external sources or other agents' databases. This system of modular, specialized agents equipped with dedicated knowledge stores makes LLMAgentNet highly adaptable and capable of handling complex workflows with efficiency.

AI Agent work modes. In LLMAgentNet, the concept of work modes is essential for balancing automation with human oversight and ensuring flexibility during both development and deployment. The system supports three primary work modes: Human-in-the-Loop, Human-on-the-Loop, and Human-out-of-the-Loop. Each mode offers a different level of human involvement, which is crucial for testing and refining agents, evaluating their decision-making processes, and managing complex workflows.

These work modes not only allow for greater control and adaptability but also play an important role in debugging and optimizing the multi-agent system. During development, agents can be tested in isolation, at a granular task level, or within a collaborative environment to observe how they interact with one another. This flexibility enables developers to iteratively improve each agent's functionality, ensuring robust performance before full autonomy is deployed (Fig. 10).

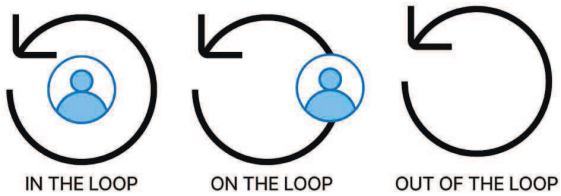

Figure 10. Illustration of the three work modes of Multi-Agent System (author's own illustration) / Ілюстрація трьох режимів роботи мультиагентної системи (авторська розробка)

The work modes also serve as a foundation for scaling up system autonomy. While Human-in-the-Loop and Human-on-the-Loop modes enable supervised and semi-supervised operation, Human-out-of-the-Loop mode allows agents to operate independently, autonomously generating and executing plans aimed at high-level goals. This progressive structure is crucial for building a reliable multi-agent system that can function effectively in both controlled and fully autonomous scenarios.

Human-in-the-Loop. In Human-in-the-Loop mode, human intervention is required at specific points within the task execution process. This mode is ideal for tasks that are either too complex for full automation or require validation to ensure accuracy. In this setup, agents can autonomously process portions of the task but will pause at designated checkpoints, awaiting human approval before proceeding to the next stage.

This approach enhances control and accuracy in situations where the stakes are high or where nuanced decision-making is essential. For example, in a content creation pipeline, an AI agent might draft initial content and then require a human to review and approve it before publishing.

Human-on-the-Loop mode provides a balance between autonomy and oversight, where the system operates autonomously but a human monitors the overall process, ready to intervene if necessary. This mode is designed to reduce the need for constant human supervision while still allowing human operators to step in if errors or unexpected conditions arise.

In LLMAGENTNET, Human-on-the-Loop mode is particularly useful for multi-step tasks where agents collaborate to accomplish a goal. The Agent Manager handles task delegation and decision-making autonomously, but a human operator can observe progress and step in if an agent encounters an unexpected situation or if adjustments are needed. This mode is especially suitable for scenarios that require responsiveness but where tasks are generally well-defined and predictable.

Human-out-of-the-Loop mode is the most advanced and autonomous operational mode within LLMAGENTNET. In this mode, AI agents not only execute tasks based on a predefined plan but also continuously generate new plans to achieve high-level, long-term goals without human supervision. This mode leverages the reward-based method from Reinforcement Learning (RL) to provide each agent with a "north star" objective, guiding the direction and purpose of their autonomous actions.

In Human-out-of-the-Loop mode, agents are equipped with a scoring mechanism to measure progress toward overarching goals. Each action or plan contributes to a cumulative reward, motivating agents to make decisions and create new plans that maximize their reward over time. This reward system serves as an intrinsic feedback loop, helping agents evaluate which strategies bring them closer to their objectives and adaptively refining their approach based on past performance.

By using a Reinforcement Learning reward mechanism, LLMAGENTNET enables agents to work beyond individual user requests, autonomously seeking ways to improve their performance and achieve higher scores. Each agent's behavior is aligned with its reward structure, meaning that every decision is made with the goal of accumulating more points, ultimately driving the agent toward continuous improvement and efficient task execution. This approach allows agents to operate effectively in complex environments where predefined task sequences may be insufficient, encouraging them to innovate and discover new paths toward their objectives.

For example, an agent might start with a high-level goal, such as "increase engagement on social media", and autonomously generate a sequence of tasks aimed at achieving this goal. It could plan, execute, and evaluate multiple strategies – such as posting at different times, experimenting with content styles, or targeting various audience segments – adapting its approach based on feedback and accumulating rewards as it learns what works best.

In this mode, the Agent Manager plays a supervisory role by monitoring the agents' progress, analyzing reward accumulation, and adjusting the agents' objectives as needed to align with broader system goals. The agents, however, operate with full autonomy, continuously iterating on their plans and pursuing actions that maximize their long-term reward.

Human-out-of-the-Loop mode not only enables autonomous task execution but also fosters goal-oriented, self-directed learning within the multi-agent system. Through this mode, LLMAGENTNET demonstrates the potential for agents to evolve independently, learn from each outcome, and make progress toward complex, high-level objectives, setting the stage for increasingly sophisticated and resilient multi-agent collaboration. This ability to pursue high-level goals without human oversight represents a significant step toward realizing scalable, intelligent systems that can adapt to evolving challenges and operate in real-world environments with minimal human intervention.

To evaluate the effectiveness and robustness of LLMAGENTNET, we conducted benchmarking tests across various established datasets and scenarios. These benchmarks, including HotPotQA and TaskBench, provide a standardized way to assess the system's performance in handling complex, multi-step tasks and answering intricate questions. By using these benchmarks, we aimed to measure the system's accuracy, efficiency, and adaptability in diverse environments, highlighting the capability of AI agents to operate autonomously while achieving high-level goals.

To assess the multi-hop reasoning capabilities of LLMAGENTNET, we used the HotPotQA dataset, a benchmark specifically designed for evaluating question-answering systems that require reasoning across multiple documents. This dataset presents a challenging environment for agents, as they must retrieve, integrate, and reason with information from various sources to arrive at accurate answers. Such a process mirrors the complex, multi-step reasoning required in real-world applications.

In our evaluation, LLMAGENTNET was tested on key metrics like F1-Score and Accuracy. These metrics help gauge the precision and correctness of the system's responses. By measuring performance on HotPotQA, we could evaluate how effectively LLMAGENTNET decomposes questions, selects relevant information from multiple sources, and synthesizes this information to generate a coherent answer (Tab. 2).

Table 2. Performance of LLMAGENTNET on HotPotQA (based on the authors' own experimental results) / Результати LLMAGENTNET на HotPotQA (власні експериментальні дані авторів)

LLM	Easy		Medium		Hard	
	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy
GPT-3.5	0.310	0.30	0.230	0.20	0.213	0.20
GPT-4o	**0.525**	**0.42**	**0.544**	**0.49**	0.427	0.33
Claude Sonnet	0.512	0.41	0.510	0.43	0.420	0.32
Claude Haiku	0.490	0.39	0.500	0.42	**0.430**	**0.34**
Gemini 1.5 Pro	0.522	0.40	0.541	0.41	0.455	0.31

Through this benchmarking, LLMAGENTNET demonstrated its capability in handling tasks that require complex, multi-document reasoning, a critical feature for autonomous systems aiming to perform high-level information synthesis and decision-making without human intervention.

AI Agent Benchmarking in Content Marketing. To evaluate the real-world application and efficiency of LLMAGENTNET, we conducted a comprehensive experiment in the field of content marketing. This experiment aimed to benchmark the system's ability to automate the entire content creation pipeline, from topic research to publishing, demonstrating its effectiveness in a practical, high-demand environment.

Experiment Setup. The experiment involved creating around 400 content-rich blog pages on WordPress, simula-

ting a large-scale content marketing campaign. Each page required multiple steps, including topic analysis, content drafting, image generation, content formatting, and publication on WordPress. Traditionally, this workflow would require substantial manual effort and time, even with AI tools like ChatGPT.

Comparison with Manual Content Creation. In a manual setup, content creators typically spend 25-30 minutes on each blog post, even with the help of ChatGPT. This time includes analyzing competitor content, generating unique text, creating an image, structuring the content, and publishing it on WordPress. For an organization aiming to produce hundreds of pages, this time investment becomes a significant constraint, limiting scalability and increasing costs (Fig. 11).

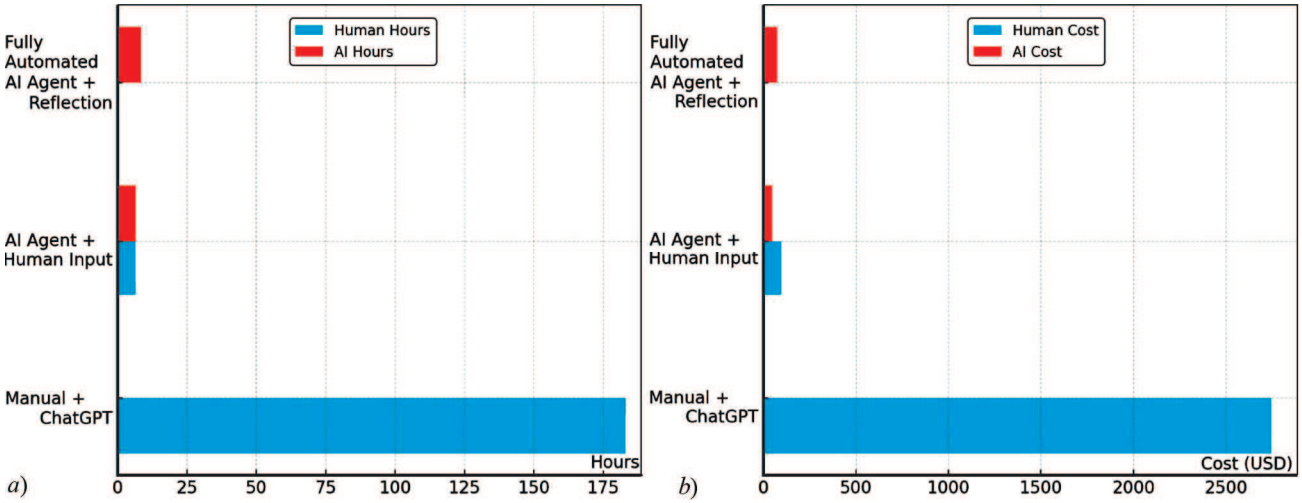

Figure 11. Comparison of working hours (*a*)vand costs (*b*) for 3 scenarios / Порівняння робочого часу (*a*)та витрат (*b*) у 3 сценаріях

In contrast, with LLMAGENTNET, the entire process of creating and publishing a blog post was triggered by a single user request. Once the request was submitted, the multi-agent system autonomously handled each step, completing the full workflow in approximately 30 seconds to 1.5 minutes per blog post. This remarkable time efficiency showcases LLMAGENTNET's ability to scale content production dramatically, reducing time and resource expenditure by automating traditionally labor-intensive tasks.

System Performance Metrics. Throughout the experiment, LLMAGENTNET generated around 20,000 messages containing execution results and operational logs. Each content creation task was broken down into an average of 5-7 steps, with 3-4 specialized agents collaborating to accomplish each task. These agents included those dedicated to content research, text generation, image creation, formatting, and publishing. By dividing responsibilities among multiple agents, LLMAgentNet was able to handle complex tasks that typically require specialized human skills.

Efficiency Gain Calculation: Manual vs. Automated with AI Agent. This experiment demonstrates the efficiency and quality improvements LLMAgentNet brings to content marketing compared to manual methods (even with ChatGPT assistance). By evaluating three setups – manual content creation with ChatGPT, AI agent-driven automation, and AI agent automation with an added reflection step – we assess gains in productivity, quality, and cost-effectiveness (Tab. 3).

Here is a detailed description of each scenario:

1. *Manual Content Creation with ChatGPT:*

- time per Blog Post: 25-30 minutes;
- total Time for 400 Blog Posts: 166-200 hours;
- error Rate: Estimated at 15-20 % due to varying human input and oversight limitations;
- human Work Time: High (over 150 hours for 400 posts);
- cost: ChatGPT subscription of $25/month.

In this setup, manual creation requires substantial time and effort, with ChatGPT assisting in content generation. However, time spent on competitor analysis, image creation, formatting, and publishing limits productivity, making it challenging to scale.

Table 3. Comparison of work hour reduction and costs using LLMAgentNet for content marketer automation (based on the authors' own experimental results) / Порівняння скорочення робочого часу та витрат при автоматизації маркетолога контенту за допомогою LLMAgentNet

Metric	Manual +ChatGPT	AI Agent	AI Agent + Reflection
Time per 1 Blog Post	25-30 min (human)	1 min (human) + 1 min (AI)	0 min (human) + 1.2 min (AI)
Total Human Time for 400 Posts	166-200 hours	6-7 hours	0 hours
Total AI Exec. Time for 400 Posts	N/A	6-7 hours	8-9 hours
Error Rate	15-20 %	17 %	3 %
Human Work Cost	$2,490-$3,000	$90-$105	$0
AI Execution Cost	$25 per month	$40-$56	$64-$88
Total Cost for 400 Posts	$3025	$161	$88

2. *AI Agent-Driven Automation* (*Standard Execution*):

- time per Blog Post: 1 minute of human input to initiate the request, plus 1 minute of AI execution time;
- total Time for 400 Blog Posts: 6-7 hours of AI processing time plus 6-7 hours of human input (1 minute per post);
- error Rate: 17 %, with minor issues such as format inconsistencies or inaccuracies;
- human Work Time: 1 minute per blog post to initiate the request;
- cost per Step: $0.02 per step, with 5-7 steps per post, resulting in $0.10-$0.14 per post; total estimated cost for 400 posts is $40-$56.

In this configuration, LLMAgentNet autonomously managed content creation, formatting, and publishing with minimal human intervention. Each blog post required 1 minute of human time to initiate, after which the AI agent autonomously handled the multi-step workflow. The overall production time was reduced by over 95 % compared to

manual methods, and the total cost per post remained low at $0.10-$0.14.

3. *AI Agent with Reflection Step* (*Enhanced Execution with Self-Review*)*:*
   - time per Blog Post: ~1.2 minutes (with additional 3-4 review steps);
   - total Time for 400 Blog Posts: 8-9 hours (including reflection steps);
   - error Rate: Reduced to 3 % due to self-review and refinement;
   - human Work Time: 1 minute per blog post to initiate the request;
   - cost per Step: $0.02 per step, including an additional 3-4 review steps, adding $0.06-$0.08 per post for review; total estimated cost per post is $0.16-$0.22; total estimated cost for 400 posts is $64-$88.

Adding a reflection step allowed LLMAgentNet to autonomously review and refine each blog post, improving quality and reducing the error rate from 17 % to 3 %. Each reflection step took 3-7 seconds at a cost of $0.02 per step, resulting in an additional $0.06-$0.08 per post for the quality check. This approach maintained high efficiency with no additional human time, delivering high-quality content at a marginal cost increase.

Discussion of research results. In the comprehensive survey by [8], the authors provide a structured analysis of large language model (LLM)-based multi-agent systems, outlining their evolution, typical collaboration patterns, and key limitations. They categorize these systems based on agent autonomy, communication modalities, and orchestration strategies, identifying that most existing frameworks rely on rigid, pre-defined task flows or specialized agent configurations. Their review highlights several open challenges, including dynamic agent registration, inter-agent coordination at runtime, and the absence of standardized schema for capability negotiation.

In the work by [9], the authors present an in-depth exploration of the current challenges and open problems in the development of LLM-based multi-agent systems. Their paper identifies several unresolved issues that hinder the practical deployment of such systems, including the lack of robust dynamic agent discovery, scalable communication protocols, and efficient shared memory architectures. The authors emphasize that while many frameworks showcase promising agentic behavior in isolated scenarios, they struggle to generalize across diverse tasks or integrate heterogeneous tools without significant manual configuration.

In the work by [32], the authors introduce Reflexion, a pioneering approach that enables language agents to iteratively improve their performance through verbal reinforcement learning. The core idea is to allow agents to reflect on their past actions by generating natural language explanations of failures and successes, thus facilitating internal reasoning and adjustment in subsequent iterations. This method significantly enhances task performance, particularly in environments where explicit reward signals or ground-truth answers are sparse or delayed.

The study by [4] introduces the concept of the Internet of Agents, a visionary framework for connecting heterogeneous AI agents across the web to facilitate collaborative intelligence. Their system emphasizes interconnectivity among diverse agents built on different architectures, allowing for distributed decision-making and joint task execution. A key contribution of their work is the introduction of a web-layered infrastructure for agent communication, utilizing standardized endpoints and metadata-based discovery to enable interoperability.

The framework proposed in [44], known as Tree of Thoughts (ToT), offers a novel approach to problem-solving that enhances large language models with deliberate multi-step reasoning. Rather than executing a single chain-of-thought, the model explores multiple reasoning paths as a decision tree, evaluating intermediate thoughts to select optimal trajectories. This enables more systematic exploration of solution spaces, improving outcomes on tasks requiring complex logical reasoning such as mathematical proofs, puzzles, and planning.

According to [27], the *Gorilla* framework connects large language models with a massive collection of real-world APIs, enabling them to generate executable API calls through prompt tuning and supervised fine-tuning. Gorilla focuses on training LLMs to understand the function signatures and descriptions of APIs at scale, allowing for robust code generation across over 1,600 popular APIs. The system demonstrates high accuracy in matching natural language queries to corresponding APIs and their parameters, offering a valuable contribution toward grounding LLM outputs in actionable operations.

So, based on the results of the work performed, it is possible to formulate the following scientific novelty and practical significance of the research results. The analysis of recent publications and the discussion of the obtained findings confirm the relevance and necessity of this study, as no prior research has proposed a comparable framework that enables dynamic, protocol-agnostic orchestration of autonomous AI agents with such flexibility, interoperability, and scalability.

Scientific novelty of the obtained research results – for the first time, a visual method for dynamic orchestration of autonomous artificial intelligence agents has been developed, based on RESTful communication and OpenAPI schema integration, which enables real-time agent coordination, tool invocation, and interoperability across heterogeneous systems within multi-agent architectures.

Practical significance of the research results – the results of the study can be used to develop and implement intelligent agent-based systems across various applied domains, such as customer support, digital content creation, cybersecurity, and enterprise workflow automation. Specifically, the proposed method of visual orchestration and agent coordination can be applied in the construction of scalable, adaptable, and interoperable multi-agent architectures based on large language models and external APIs.

Conclusion / Висновки

A collaborative architecture of artificial intelligence agents within a multi-agent system based on large language models has been developed and validated, ensuring its effectiveness through experimental benchmarking and real-world application. Based on the results of the research the following main conclusions can be drawn.

1. Developed a new framework, LLMAgentNet, which enables seamless and dynamic interaction between autonomous agents through RESTful API communication and OpenAPI-based schema integration. This approach overcomes the static and rigid structures found in earlier multi-agent systems.
2. Presented a detailed comparison of LLMAgentNet with existing frameworks and protocols, including AutoGPT, MetaGPT, ReAct, and emerging agent communication standards such as MCP, A2A, ACP, and ANP. The compa-

rison highlights LLMAgentNet's superiority in modularity, interoperability, and plug-and-play orchestration.
3. Implemented mechanisms for runtime planning, reflection, and memory-based optimization, allowing agents to learn iteratively, reduce execution errors, and dynamically adapt their strategies based on task outcomes and feedback.
4. Validated the framework through quantitative benchmarking and practical use cases that demonstrated significant improvements in agent collaboration, performance efficiency, and the reduction of manual orchestration overhead.
5. Confirmed the potential of LLMAgentNet to serve as a foundational infrastructure for future research and deployment of multi-agent systems in domains such as enterprise automation, intelligent assistants, and AI-based knowledge work.
6. Outlined directions for future enhancements, including protocol harmonization, reinforcement learning integration, and broader ecosystem compatibility to accelerate the development of autonomous systems contributing toward Artificial General Intelligence (AGI).

References

1. Agent Network Protocol Contributors. (2025). Agent Network Protocol (ANP). URL: https://github.com/agent-network-protocol/AgentNetworkProtocol
2. Agent Network Protocol Contributors. (2025). Agent Network Protocol Official Website. URL: https://agent-network-protocol.com/
3. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., et al. (2024). A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology, 15*(3), 1–45. *arXiv*. URL: https://arxiv.org/abs/2307.03109
4. Chen, W., You, Z., Li, R., Guan, Y., Qian, C., Zhao, C., Yang, C., Xie, R., Liu, Z., & Sun, M. (2024). Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv*. https://doi.org/10.48550/arXiv.2407.07061
5. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., et al. (2022). PaLM: Scaling language modeling with Pathways. *arXiv*. https://doi.org/10.48550/arXiv.2204.02311
6. Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., & Su, Y. (2023). Mind2Web: Towards a generalist agent for the web. *arXiv*. https://doi.org/10.48550/arXiv.2306.06070
7. Google. (2025). Agent2Agent (A2A) Protocol Documentation. URL: https://google.github.io/A2A/
8. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., & Zhang, X. (2024). Large language model based multi-agents: A survey of progress and challenges. *arXiv*. https://doi.org/10.48550/arXiv.2402.01680
9. Han, S., Zhang, Q., Yao, Y., Jin, W., Xu, Z., & He, C. (2024). LLM multi-agent systems: Challenges and open problems. *arXiv*. https://doi.org/10.48550/arXiv.2402.03578
10. Harper, J. (2024). AutoGenesisAgent: Self-generating multi-agent systems for complex tasks. *arXiv*. https://doi.org/10.48550/arXiv.2404.17017
11. He, P., Lin, Y., Dong, S., Xu, H., Xing, Y., & Liu, H. (2025). Red-teaming LLM multi-agent systems via communication attacks. *arXiv*. https://doi.org/10.48550/arXiv.2502.14847
12. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., et al. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv*. https://doi.org/10.48550/arXiv.2308.00352
13. IBM BeeAI. (2025). Introduction to Agent Communication Protocol (ACP). URL: https://docs.beeai.dev/acp/alpha/introduction
14. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. R. (2024). SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=VTF8yNQM66
15. Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., et al. (2023). AgentBench: Evaluating LLMs as agents. *arXiv*. https://doi.org/10.48550/arXiv.2308.03688
16. Liu, Xiangzheng, Liu, Jianxun, Kang, Guosheng, Shi, Min, Liu, Yi, & Yin, Yiming. (2025, December). On the effectiveness of large language models for query expansion in code search. *Journal of Systems and Software*, Vol. 230, article ID 112582. https://doi.org/10.1016/j.jss.2025.112582
17. Liu, Z., Yao, W., Zhang, J., Xue, L., Heinecke, S., Murthy, R., Feng, Y., et al. (2023). BOLAA: Benchmarking and orchestrating LLM-augmented autonomous agents. *arXiv*. https://doi.org/10.48550/arXiv.2308.05960
18. Liu, Z., Yao, W., Zhang, J., Yang, L., Liu, Z., Tan, J., Choubey, P. K., et al. (2024). AgentLite: A lightweight library for building and advancing task-oriented LLM agent systems. *arXiv*. https://doi.org/10.48550/arXiv.2402.15538
19. Mirowski, P., Mathewson, K. W., Pittman, J., & Evans, R. (2022). Co-writing screenplays and theatre scripts with language models: An evaluation by industry professionals. *arXiv*. https://doi.org/10.48550/arXiv.2209.14958
20. Model Context Protocol. (2025). Introduction to Model Context Protocol (MCP). URL: https://modelcontextprotocol.io/introduction
21. Nakajima, Y. (2023). BabyAGI. URL: https://github.com/yoheinakajima/babyagi
22. Nguyen, M. H., Chau, T. P., Nguyen, P. X., & Bui, N. D. Q. (2024). AgileCoder: Dynamic collaborative agents for software development based on agile methodology. *arXiv*. https://doi.org/10.48550/arXiv.2406.11912
23. OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Leoni Aleman, F., et al. (2024). GPT-4 technical report. *arXiv*. https://doi.org/10.48550/arXiv.2303.08774
24. OpenAI. (2024). Hello GPT-4o. URL: https://openai.com/index/hello-gpt-4o/
25. OpenAI. (2024). *Swarm: Stateless multi-agent coordination via JSON-RPC routines*. URL: https://platform.openai.com/docs/models/swarm
26. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., et al. (2022). Training language models to follow instructions with human feedback. In Oh, A. H., Agarwal, A., Belgrave, D., & Cho, K. (Eds.), *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=TG8KACxEON
27. Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive APIs. *arXiv*. https://doi.org/10.48550/arXiv.2305.15334
28. Qian, C., Liang, S., Qin, Y., Ye, Y., Cong, X., Lin, Y., Wu, Y., Liu, Z., & Sun, M. (2024). Investigate-Consolidate-Exploit: A general strategy for inter-task agent self-evolution. *arXiv*. https://doi.org/10.48550/arXiv.2401.13996
29. Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., et al. (2023). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv*. https://doi.org/10.48550/arXiv.2307.16789
30. Rome, S., Chen, T., Tang, R., Zhou, L., & Ture, F. (2024). Ask me anything: How Comcast uses LLMs to assist agents in real time. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2827–2831. https://doi.org/10.1145/3626772.3661345
31. 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. *arXiv*. https://doi.org/10.48550/arXiv.2302.04761
32. Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *arXiv*. https://doi.org/10.48550/arXiv.2303.11366
33. Shinn, N., Labash, B., & Gopinath, A. (2023). Reflexion: An autonomous agent with dynamic memory and self-reflection. *arXiv*. https://doi.org/10.48550/arXiv.2303.11366
34. Significant Gravitas. (2023). AutoGPT. URL: https://github.com/Significant-Gravitas/Auto-GPT
35. SmolAgents Project. (2024). SmolAgents: A single-file Python library for multi-modal LLM agent prototyping. URL: https://github.com/smolagents/smolagents

36. Song, Y., Xiong, W., Zhu, D., Wu, W., Qian, H., Song, M., Huang, H., et al. (2023). RestGPT: Connecting large language models with real-world RESTful APIs. *arXiv*. https://doi.org/10.48550/arXiv.2306.06624
37. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv*. URL: https://arxiv.org/abs/2307.09288v2
38. Wang, B., Li, Y., Lv, Z., Xia, H., Xu, Y., & Sodhi, R. (2024). LAVE: LLM-powered agent assistance and language augmentation for video editing. *arXiv*. https://doi.org/10.48550/arXiv.2402.10294
39. Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, H., Peng, & Ji, H. (2024). Executable code actions elicit better LLM agents. *arXiv*. https://doi.org/10.48550/arXiv.2402.01030
40. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv*. https://doi.org/10.48550/arXiv.2308.08155
41. Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., et al. (2023). The rise and potential of large language model-based agents: A survey. *arXiv*. https://doi.org/10.48550/arXiv.2309.07864
42. Xu, Z., Yu, C., Fang, F., Wang, Y., & Wu, Y. (2024). Language agents with reinforcement learning for strategic play in the Werewolf game. *arXiv*. https://doi.org/10.48550/arXiv.2310.18940
43. Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv*. https://doi.org/10.48550/arXiv.1809.09600
44. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv*. https://doi.org/10.48550/arXiv.2305.10601
45. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *arXiv*. https://doi.org/10.48550/arXiv.2210.03629
46. Ye, Q., Axmed, M., Pryzant, R., & Khani, F. (2024). Prompt engineering a prompt engineer. *arXiv*. https://doi.org/10.48550/arXiv.2311.05661
47. Zhang, X., Chen, X., Liu, Y., Wang, J., Hu, Z., & Yan, R. (2024). LLM-driven agents for influencer selection in digital advertising campaigns. *arXiv*. https://doi.org/10.48550/arXiv.2403.15105

А. Р. Бідочко, Я. І. Виклюк

Національний університет "Львівська політехніка", м. Львів, Україна

LLMAgentNet: КОЛАБОРАТИВНА МЕРЕЖА АВТОНОМНИХ АГЕНТІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ВИКОНАННЯ СКЛАДНИХ ЗАВДАНЬ

Досліджено нові можливості, що відкриваються завдяки стрімкому розвитку великих мовних моделей (LLM), особливо у сфері створення автономних агентів штучного інтелекту, здатних до багатокрокового планування та виконання складних завдань. Проаналізовано недоліки наявних мультиагентних фреймворків, зокрема їхню архітектуру, залежність від конкретних інструментів і масштабованість. Для подолання цих обмежень запропоновано LLMAgentNet – модульну мережу взаємодіючих агентів, які співпрацюють завдяки LLM, використовуючи RESTful API як універсальний комунікаційний інтерфейс. Такий підхід забезпечує незалежну взаємодію між агентами та зовнішніми інструментами, дає змогу ділити запити високого рівня на підзавдання та виконувати їх автономно на підставі специфікації OpenAPI. У системі передбачено спеціальний менеджер агента (англ. *Orchestrator*), який відповідає за декомпозицію запитів, вибір виконавців та узгодження потоків даних між агентами. Інтеграція механізмів динамічної реконфігурації та підтримка зовнішніх агентів, розгорнутих на окремих віртуальних серверах, робить LLMAgentNet гнучкою та масштабованою платформою. Окремо охарактеризовано систему пам'яті агентів, реалізовану через векторну базу даних, що дає змогу зберігати знання, формувати змінні під час виконання завдань і використовувати їх як контекст у наступних кроках. LLMAgentNet підтримує кілька режимів роботи: Human-in-the-Loop, Human-on-the-Loop і Human-out-of-the-Loop, що особливо важливо на етапах розроблення та налагодження системи. Проведено як формальні тести (наприклад, HotPotQA), так і практичне застосування системи у сфері автоматизації контент-маркетингу, де було створено понад 400 статей без участі людини. Отримані результати показали зниження рівня помилок до 3 % після увімкнення модуля рефлексії та зростання продуктивності в понад 20 разів порівняно з ручним створенням контенту із використанням ChatGPT. Унаслідок цього доведено ефективність LLMAgentNet як системи для реалізації складних задач із перспективою застосування у майбутніх автономних інтелектуальних системах та поступу в напрямі AGI (англ. *Artificial General Intelligence*).

Ключові слова: великі мовні моделі; агенти ШІ; автономні системи; мультиагентний фреймворк; планування завдань; співпраця агентів.