LUPINE PUBLISHERS

Current Trends in Computer Sciences & Applications

DOI: 10.32474/CTCSA.2024.03.000162

ISSN: 2643-6744

Review Article

Exploring LLM-based Agents: An Architectural Overview

Ayse Arslan*

Department of Computer Science, Oxford Alumni, of Northern California

***Corresponding author:** Ayse Arslan, Department of Computer Science, Oxford Alumni of Northern California, Oxford Alumni, of Northern California

Received: May 09, 2024 **Published:** June 25, 2024

Abstract

Built upon these powerful LLMs, emergent LLM-based agents can present strong task fulfillment abilities in diverse environments, ranging from virtual assistants to more sophisticated systems involving complex and creative problem solving, planning and reasoning. This study explores the architecture of LLM-based agents and provides research directions for the future of LLM-based agents. It concludes that each outlined direction can not only build upon the foundational elements of LLM-based agents but also contribute to the advancement of the field at large.

Introduction and Motivation

Over the last few years, large language models (LLMs) have evolved from mere research artifacts into useful products Open AI [1]. This advent of large language models (LLMs) has brought new possibilities to the agent development. Current LLMs have shown great power in understanding instructions reasoning and solving problems and interacting with human users as well as external environments. Built upon these powerful LLMs, emergent LLM-based agents can present strong task fulfillment abilities in diverse environments, ranging from virtual assistants to more sophisticated systems involving complex and creative problem solving, planning and reasoning.

This study explores the architecture of LLM-based agents and provides research directions for the future of LLM-based agents.

Related Work

Autonomous agents utilizing Large Language Models (LLMs) offer promising opportunities to enhance and replicate human workflows. In real-world applications, however, existing systems [2] tend to oversimplify the complexities. They struggle to achieve effective, coherent, and accurate problem-solving processes, particularly when there is a need for meaningful collaborative interaction [3].

Large language models (LLMs)are becoming a crucial building block in developing powerful agents that utilize LLMs for reasoning, tool usage, and adapting to new observations [4] in many real-world tasks. Given the expanding tasks that could benefit from LLMs and the growing task complexity, an intuitive approach to scale up the power of agents is to use multiple agents that cooperate. Prior work suggests that multiple agents can help encourage divergent thinking [5] improve factuality and reasoning and provide validation [6].

Large language model (LLM) based autonomous agents take natural language instructions as input for complex task solving. Agents consume observations and produce actions. The Generative Agent-Based Models (GABM)s consumes agent actions and creates event statements, which define what has happened in the simulation as a result of the agent's attempted action.

Some scholars [7] posit that human-beings generally act as though they choose their actions by answering three key questions:

1. What kind of situation is this?
2. What kind of person am I?
3. What does a person such as I do in a situation such as this?

In a similar vein, the GABM mediates between the state of the world and agents' actions. The state of th One compelling example of how an LLM-based agent solves real-world tasks can be seen in Figure 1. Given the trip organization request from the user e world is contained in GM's memory and the values of grounded variables (e.g. money, possessions, votes, etc.). To achieve this the GM has to repeatedly answer the following questions:

1. What is the state of the world?
2. Given the state of the world, what event is the outcome of the players activity? What observation do players make of the event?
3. What effect does the event have on grounded variables?

Similar to human-beings, the GABM is responsible for:

1. Maintaining a consistent and grounded state of the world where agents interact with each other.
2. Communicating the observable state of the world to the agents.
3. Deciding the effect of agents' actions on the world and each other.
4. Resolving what happens when actions submitted by multiple agents' conflict with one another.

The components of the GABM describe the state of the world—for example location and status of players, state of grounded variables (money, important items) and so on-––so that GABM can decide the event that happens as the outcome of players' actions. The outcome is described in the event statement which is then added to the GABM associative memory. After the event has been decided the GABM elaborates on its consequences. For example, the event could have changed the value of one of the grounded variables or it could have had an effect on a non-acting player.

Throughout the literature, some concrete recommendations have been made for best practices in generative agent-based modeling:

1. **Measure generalization**: Direct measurement of model predictions on truly new test data that could not have influenced either the model's concrete parameters or its abstract specification is the gold standard. For instance, when a model makes predictions about how human-beings will behave in certain situation then there is no better form of evidence than actually measuring how real people behave when facing the modeled situation.
2. **Evaluate algorithmic fidelity**: Algorithmic fidelity describes the extent to which a model may be conditioned using socio-demographic backstories to simulate specific human groups. Some scholars [14] [8] conclude from this that algorithmic fidelity must be measured for each research question. A finding of sufficient algorithmic fidelity to address one research question does not imply the same will be true for others [11, 12] [9,10].
3. **Model comparison**: It is a lot easier to support the claim that one model is better (i.e. more trustworthy) than another model than to support the claim that either model is trustworthy on an absolute scale without reference to the other.
4. **Robustness**: It will be important to try to develop standardized sensitivity analysis / robustness-checking protocols. For instance, it's known that LLMs are often quite sensitive to the precise wording used in text prompts. Best practices for GABMs should involve sampling from a distribution of "details" and ways of asking questions to show that the factors not thought to be mechanistically related to the outcome are indeed as irrelevant as expected.

The most important responsibility of the GABM is to provide the grounding for particular experimental variables. The GABM determines the effect of the agents' actions on these variables, records them, and checks that they are valid. Whenever an agent tries to perform an action that violates the grounding, it communicates to them that their action was invalid.

A generative model of social interactions (i.e. a GABM) consists of two parts:

- the model of the environment and
- the model of individual behavior.

There are: (a) a set of generative agents and (b) a generative model for the setting and context of the social interaction i.e. the environment, space, or world where the interaction takes place.

As chat-optimized LLMs (e.g., GPT-4) show the ability to incorporate feedback, LLM agents can cooperate through conversations with each other or human(s), e.g., a dialog where agents provide and seek reasoning, observations, critiques, and validation. As a single LLM can exhibit a broad range of capabilities (especially when configured with the correct prompt and inference settings), conversations between differently configured agents can help combine these broad LLM capabilities in a modular and complementary manner. Furthermore, LLMs have demonstrated ability to solve complex tasks when the tasks are broken into simpler subtasks. Multi-agent conversations can enable this partitioning and integration in an intuitive manner.

It is important to bear in mind that each medium has its own unique qualities, and those qualities have a transformative impact on society, culture, and individuals [15] [11]. For instance, the recommender algorithms used in social media have a substantial effect on human culture and society and the fact that LLM-based systems

have analogous properties, affecting both how information is transmitted and how it is valued, implies they are likely to influence human culture and society more and more as time goes on [13] [12].

To overcome LLM vulnerabilities, model validation can be done. The basic principle of model validation is one of similarity between tested and untested samples. A model typically makes a family of related predictions, and perhaps a rigorous experiment tests only one of them.

GABMs constructed for different purposes call for validation by different forms of evidence. A GABM is said to generalize when inferences made on the basis of the model transfer to real life. Evidence of effectiveness in real life (ecological validity) is at the top, rigorous experiments in controlled settings like labs or clinics below that, observational data lower down, and consistency with prior theory lower still. For validation, it also matters what the model will be used for. If it will only be used to guide decisions about where one may most fruitfully focus time, effort, and resources in further research (e.g., in piloting) then the evidence bar should be correspondingly lower than if the model is to be used to guide real world decisions with real consequences.

A major topic of interest is how large-scale "macrosocial" patterns emerge from the "microsocial" decisions of individuals [8] [13], as explored, for example, in assemblage theory (DeLanda, 2016, 2011). For instance, the collective social phenomena of information diffusion emerged in a simulation without specific programming to enable it [11] [9]. The generative agent's ability to copy, communicate, reproduce, and modify behavioral and thinking patterns potentially makes them a substrate for cultural evolution.

Some change in their environment eventually forces their routines to end, and when that happens, they have to engage in sense-making by asking themselves "what is the story here?" and "what should I do now?" by retrospectively connecting their past experiences and engaging in dialogue with other members of the organization. New social facts and routines can emerge from this sense-making process.

It should also be taken into account changes in the social structures constituting the environment deeply change the agents' own "internal" models and categories [19] [14]. Causal influence flows both from agents to social structures as well as from social structures to agents. Agents and structures may be said to co-constitute one another (Onuf, 1989) [15].

Overview of Architecture

The layered architecture ensures a clear delineation of responsibilities across the system. LLM-based single-agent systems (SAS) use a single LLM agent for complex task solving, such as travel planning or personalized recommendation [16]. The agent takes natural language instruction from users as input and decomposes the task into a multistep plan for task solving, where each step may call external tools to be completed, such as collecting information, executing specialized models, or interacting with the external world.

LLM-based multi-agent systems (MAS) leverage the interaction among multiple agents for problem solving [17]. The relationship among the multiple agents could be cooperative, competitive, or a mixture of cooperation and. In cooperative multi-agent systems, each agent takes and assesses the information provided by other agents, thereby working together to solve complex tasks [18].

In competitive multi-agent systems, agents may debate, negotiate and compete with each other. Some multi-agent systems may exhibit both cooperation and competition among agents.

One compelling example of how an LLM-based agent solves real-world tasks can be seen in Figure 1. Given the trip organization request from the user, the travel agent follows the steps sequentially to book flights, reserve hotels, process payments, and update calendars based on the user's preferences [19].

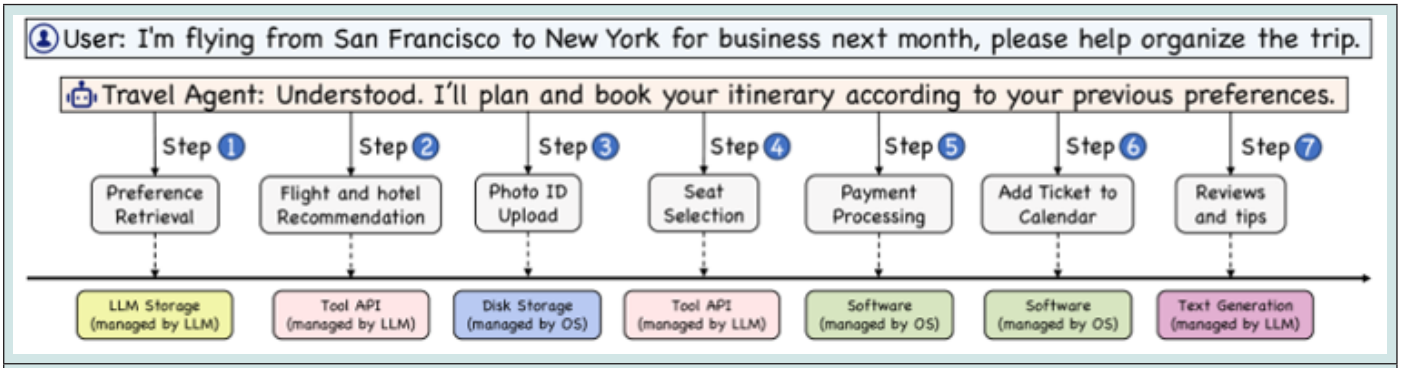

Figure 1: An Overview of LLM-based agent.

During the plan execution, agents show the reasoning and decision-making abilities, which sets it apart from the traditional software applications that are constrained to a pre-defined set of functions or workflow [20]. To realize this travel scenario, the agent needs to interact with both LLM services (e.g, retrieving and understanding user preferences, deciding which tool API to call, generating reviews and responses) and traditional operating system (OS) services (e.g., accessing disk driver and executing software).

Accompanied by the exponential growth in the agent quantity and complexity, there is an increasing strain on the functionalities of LLM. For example, scheduling and prioritizing agent requests in limited LLM resources poses a significant challenge.

Moreover, the LLM's generation process can become time-intensive when dealing with lengthy contexts, occasionally resulting in the generation being suspended by the scheduler. This raises the problem of devising a mechanism to snapshot the LLM's current generation result, thereby enabling pause/resume behavior even when the LLM has not finalized the response generation for the current request.

Within this context, each higher layer abstracts the complexities of the layers below it, facilitating interaction through interfaces or specific modules, thereby enhancing modularity and simplifying system interactions across different layers.

1. **Application Layer:**

At the application layer, agent applications, such as travel agent or math agent, are developed and deployed. This SDK allows for development of agent applications by offering a rich toolkit that abstract away the complexities of lower-level system functions. This enables developers to dedicate their focus to the essential logic and functionalities of their agents, facilitating a more efficient development process.

2. **Kernel Layer:**

The kernel layer is divided into two primary components: the OS Kernel and the LLM Kernel, each serving the unique requirements of non-LLM and LLM-specific operations, respectively. This distinction allows the LLM kernel to focus on LLM specific tasks such as context management and agent scheduling, which are essential for handling LLM-related activities and are not typically within the purview of standard OS kernel functions.

3. **Hardware Layer:**

The hardware layer comprises the physical components of the system, including the CPU, GPU, memory, disk, and peripheral devices. It is crucial to note that the LLM kernel's system calls cannot directly interact with the hardware.

Agent scheduler is designed to manage the agent requests in an efficient way. In the sequential execution paradigm, the agent tasks are processed in a linear order, where steps from a same agent will be processed first. This can lead to potential increased waiting times for tasks queued later in the sequence.

4. **Context Manager:**

The context manager is responsible for managing the context provided to LLM and the generation process given certain context. It primarily involves two crucial functions: context snapshot and restoration, and context window management Figure 2.

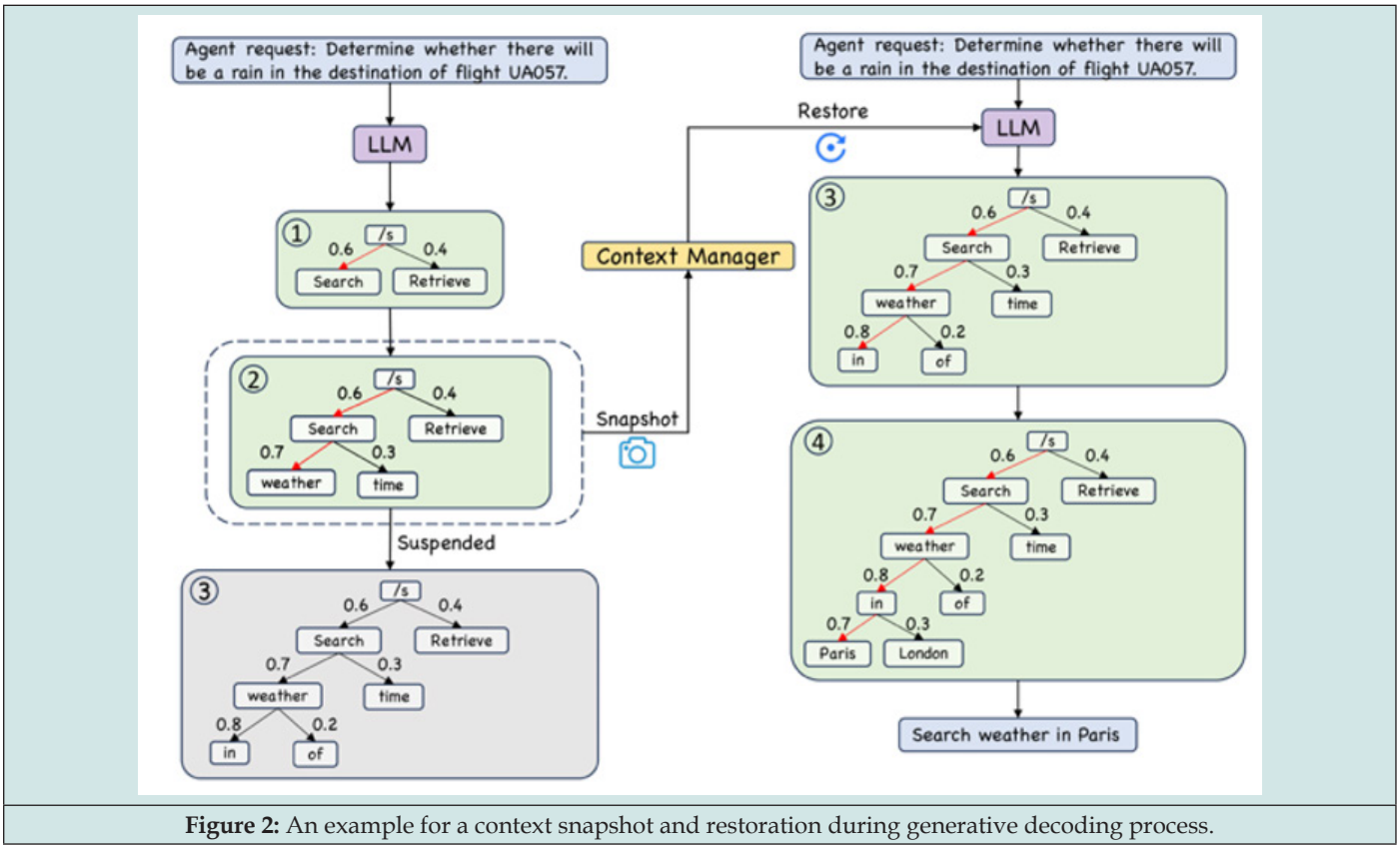

Figure 2: An example for a context snapshot and restoration during generative decoding process.

5. **Context Snapshot and Restoration:**

When such generation process has been suspended by the scheduler at an intermediate step, the context manager uses the snapshot function to capture and store the current state of the LLM's beam search tree, including all intermediate probabilities and paths being explored for generating the response.

The restoration function is employed to reload the saved state from the snapshot, allowing the LLM to continue its generation process exactly from the point of suspension to reach the final answer. In this way, the context manager ensures that the temporary suspension of one agent's request does not lead to a loss of progress, thereby optimizing resource use without compromising the quality and efficiency of response generation.

6. **Context Window Management:**

To address challenges posed by long contexts that surpass the context window limit of LLMs, context manager also needs to manage potential expansion of context window. In this way, it can help enhance the LLM's ability to process and understand extensive contexts without compromising the integrity or relevance of the information.

7. **Memory Manager**

Memory manager manages short-term memory within an agent's lifecycle, ensuring that data is stored and accessible only while the agent is active, either waiting for execution or during runtime. Compared with the storage manager introduced in the following, the memory manager enables rapid data retrieval and processing, facilitating swift responses to user queries and interactions without overburdening the storage of AIOS.

8. **Storage Manager**

In contrast, the storage manager is responsible for the long-term preservation of data, overseeing the storage of information that needs to be retained indefinitely, beyond the active lifespan of any single agent. This is achieved through mediums such as local files, databases, or cloud-based solutions, ensuring data integrity and availability for future reference or analysis.

9. **Tool Manager**

The tool manager integrates commonly-used tools from various sources and classify them into different categories, which covers web search, scientific computing, database retrieval, image processing, etc. In this way, the managed tools can cover different modalities of input and output (image and text), thus facilitating agent development within the ecosystem.

10. **Access Manager**

The access manager orchestrates access control operations among distinct agents by administering a dedicated privilege group for each agent. Those other agents that are excluded from an agent's privilege group are denied access to its resources, such as the interaction history.

To further enhance system transparency, the access manager compiles and maintains auditing logs. These logs capture detailed information about access requests, agent activities, and any modifications to the access control parameters, which help to safeguard against potential privilege attacks.

11. **LLM System Call**

LLM system call interface within the LLM kernel is designed to offer basic LLM call operation functions. This interface acts as a bridge between complex agent requests and the execution of different kernel's modules. This LLM system calls offers a suite of basic functions that span across the kernel's modules, including agent management, context handling, memory and storage operations, and access control.

12. **Implementation Framework**

Below is an example of how the agent class implements three methods:

1. .name()—returns the name of the agent, that is being referred to in the simulation. It is important that all agents have unique names;

2. .observe(observation: str)—a function to take in an observation;

3. .act(action spec)—returns the action (as a string), for example "Alice makes breakfast". The function takes in action spec, which specifies the type of output (free form, categorical, float) and the specific phrasing of the call to action. For example, the call to action could be "what would Alice do in the next hour?", in this case the answer type would be free form. Or it could be "Would Alice eat steak for dinner?" with answer type of binary choice (yes / no).

The agent class constructor is parameterized by a list of components. The components of agent have to implement the following functions:

1. .state()—returns the state of the component, for example "Alice is vegetarian";

2. .name()—returns the name of the components, for example "dietary preferences";

3. .update()—updates the state of the component by implementing; eq. (2). Optional, can pass for constant constructs;

4. .observe(observation: str)—takes in an observation, for later use during update. Optional. Observations always go into the memory anyway, but some components are easier to implement by directly subscribing to the observation stream.

During an episode, on each timestep, each agent calls .state() on all its components to construct the context of its next decision and implements eq. (1) (the components' states are concatenated in the

order supplied to the agents' constructor). .observe() is called on each component whenever it receives observations, and .update() is called at regular intervals (configurable in the constructor). Unlike in RL, we do not assume that the agent will produce an action after every observation. Here the GM might call .observe() several times before it calls .act().

LLM Agent Service Evaluation

In modern systems, data is the new king. A large amount of high-quality data is needed in order to build and evaluate services and models. Yet, collecting and curating user data is often challenging, especially when dealing with personal user data where privacy is of high concern. This creates a chicken-egg scenario, where data is needed for building of modern systems yet users might be reluctant to provide said that without immediate benefit.

Moreover, when considering the case of evaluating personalized services where each instance is specific and tailored to the individual user, it makes the problem even more substantial. How can one A/B test a personalized service at the single user level?

The grounded action space offers a conceptual way to overcome some of these challenges by simulating synthetic users and allowing them to interact with real services. This can allow generation of synthetic user activity by constructing, via simulation, agent digital action logs along with agent reasoning for each action. This data can serve as training data, or evaluation. By repeated simulation with different services configurations, one can perform at the single user level A/B testing of a service.

Nevertheless, it is important to note that this concept is contingent on the ability of the underlying LLM and system to faithfully capture user experience and realistic behavior. Therefore, the viability of this approach is highly dependent on the representation and reasoning power of the LLM, and the use of best practices.

Future Work

Since there is no consensus at present concerning how to interpret results of LLM-based simulations of human populations, the future work will address the critical epistemic question: "by what standard should we judge whether (and in what ways, and under which conditions) the results of in silico experiments are likely to generalize to the real world?".

These are not questions any one group of researchers can answer by themselves; rather these issues must be negotiated by the community as a whole. It should be seen as an invitation to the researchers from various fields that are interested in GABM to come onboard and participate in the creation of validating procedures, best practices, and epistemic norms.

Other topics of interest worth to explore might be:

New environments

1. Integration with different LLMs to see which are more suitable for constructing GABMs (e.g., they act "reasonably", are internally consistent, apply common sense, etc).
2. Improving agents—better associative memory, context-driven and dynamic component assemblage, tool use.

Visualization and audit tools.

3. Snapshot—serializing and persisting the simulation at specific episode, to enable to later resumption and performance comparison of different approaches for a specific scenario.
4. Keyframes—conditioning the agent actions to be consistent with future key action or of narrative. This allow steering the simulation more granularly and addresses an inherent issue that is caused by the fact that there is no guarantee that due to the stochastic nature of GABMs, ongoing simulations might diverge from their intended topic.
5. There are many directions for future research to pursue. This section outlines potential areas of study that expand upon the foundational features of LLM-based agents.

Advanced Scheduling Algorithms

6. Future research could focus on algorithms that perform dependency analysis among agent requests, optimizing the allocation of computational resources. Additionally, some of the tool resources are locally deployed models, which can also be incorporated into the scheduling paradigm.

Efficiency of Context Management

7. More efficient mechanisms can be devised to assist context management. For example, the pursuit of time-efficient context management techniques could significantly augment user experience by expediting the processes of context snapshotting and restoration. Also, context compression techniques can also be leveraged prior to snapshotting, which can yield a more space-efficient solution.

Optimization of Memory and Storage Architecture

8. In the context of agent collaboration and communication, the future design of memory and storage systems can adopt a shared approach, enabling the sharing of memory and storage between agents. Such an architecture would enable agents to access a communal pool of memory and storage, thereby improving the agents' decision-making ability since one agent can benefit from other agents' memory or storage.

Safety and Privacy Enhancements

9. In the realm of privacy, the exploration of advanced encryption techniques is vital for safeguarding data transmission within AIOS, thus maintaining the confidentiality of agent communications. Furthermore, the implementation of watermarking techniques could serve to protect the intellectual property of agent developers by embedding unique identifiers in outputs,

facilitating the tracing of data lineage.

In a nutshell, LLM-based agents stand as a motivating body of work that brings a broad spectrum of research opportunities. Each outlined direction can not only build upon the foundational elements of LLM-based agents but also contribute to the advancement of the field at large.

Conclusions

This paper proposes an architecture, demonstrating the potential to facilitate the development and deployment of LLM-based agents, fostering a more cohesive, effective and efficient agent ecosystem. The insights and methodologies presented herein contribute to the ongoing discourse in both AI and system research, offering a viable solution to the integration challenges posed by the diverse landscape of AI Agents.

Diverse future work can be built upon this foundation, exploring innovative ways to refine and expand the AIOS architecture to meet the evolving needs of developing and deploying LLM agents.

References

1. Johnson P (2019) Innovations in Machine Learning, TechWorld Blogs.
2. Lee K and Robinson J (2021) AI Ethics and the Future.
3. Davis M and Clark S (2020) Artificial Intelligence in Modern Healthcare, AI & Medicine 22(4): 567-580.
4. Patel R and Kim J (2020) Evaluating Neural Network Performance, Proceedings of the 13th International Conference on Data Science.
5. Brown A and White T (2021) Deep Learning Techniques for Image Recognition, WebSci 21: Proceedings of the 2021 ACM on Web Science Conference pp. 405-411.
6. Wilson E, et al. (2022) Understanding Convolutional Neural Networks, Proceedings of the 2022 IEEE International Conference on Computer Vision pp. 1235-1241.
7. Thomas G and Taylor S (2020) AI and Bias: Current Challenges, Journal of Artificial Intelligence Ethics 21: 567-581.
8. Morris R and Walker P (2021) Deep Learning for Text Analysis, Proceedings of the 15th International Conference on Computational Linguistics.
9. Green T (2020) The Evolution of Search Algorithms, Google Tech Blogs.
10. Smith A and Jones M (2021) Artificial Intelligence and Robotics.
11. King S and Lee D (2019) AI for Environmental Monitoring, WebSci '19: Proceedings of the 2019 ACM on Web Science Conference pp. 345-352.
12. Cooper H and Taylor K (2020) AI in Financial Services, Financial AI 30 (2):198-211.
13. Roberts L, et al. (2018) Emotion Recognition Using Recurrent Neural Networks, ACL '18: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics pp. 346-351.
14. Harris P and Wilson R (2018) Data Ethics in AI Research, Frontiers in AI Research.
15. Anderson H and Brown M (2017) Data Security in AI Systems, Frontiers in AI and Applications, 15 March 2017.
16. Chen J and Zhao L (2020) Building Resilient AI Systems for Industry Applications, Journal of Machine Learning Applications 25(1):125-136.
17. Adams G, et al. (2021) Neural Networks in Speech Recognition," Proceedings of the 2021 IEEE International Conference on Speech Processing pp. 223-230.
18. Reed K and Thomas J (2020) Mitigating Bias in AI Algorithms, Journal of Ethical AI 22: 112-124.
19. Brown L, et al. (2020) Predictive Analytics Using Deep Learning, ACL '20: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics pp. 547-552.
20. Young S and Zhang Y (2020) AI for Smart Cities, Journal of Urban AI 33(3): 189-202.

DOI: 10.32474/CTCSA.2024.03.000162