DOI: https://doi.org/10.26565/2304-6201-2025-68-06

УДК (UDC) 004.8

Omelchenko Ihor Valeriiovych
PhD student, Department of Mathematical Modeling and Data Analysis
Karazin Kharkiv National University, Svobody Sq 4, Kharkiv, Ukraine, 61022
e-mail: ihor.v.omelchenko@gmail.com;
https://orcid.org/0009-0007-4474-4916

Strukov Volodymyr Mykhailovych
PhD in Technical Sciences, Associate Professor; Head of the Department of Mathematical Modeling and Data Analysis
Karazin Kharkiv National University, Svobody Sq 4, Kharkiv, Ukraine, 61022
e-mail: volodymyr.strukov@karazin.ua;
http://orcid.org/0000-0003-4722-3159

Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments

Relevance: Large language models (LLMs) can be used as one of the components of autonomous agents that solve sequential decision-making tasks. To improve agent performance, it is necessary to store the history of previous observations and actions, which leads to filling the LLM context window, increasing computational costs, prolonging planning time, and raising memory requirements. A possible approach to addressing this problem is the application of observation reflection methods using LLMs.
Goal: To study the impact of memory reflection methods for autonomous agents based on LLMs. To compare these methods with simpler memory organization approaches.
Research methods: Computational experiments and comparative analysis. Memory organization methods: full episode history, reflection, and reflection with a structured set of memories. The agent performance metrics: task success rate, cumulative reward per episode, and the number of steps required to complete the task.
Results: A memory summarization method based on reflection is proposed for a hierarchical LLM-based agent. The Minigrid ColoredDoorKey environment is used for agent training. Agent code is developed, including components for training the agent in the environment. Computational experiments are conducted to train and evaluate the agent with different memory mechanisms. The performance of different memory mechanisms is evaluated using the following metrics: task completion accuracy, cumulative reward, and the number of steps until episode termination. An analysis and comparison of the results of applying different memory mechanisms to the agent's action planning task in the ColoredDoorKey environment are performed.
Conclusions: The study demonstrates that the use of reflection with a structured set of memories is appropriate for action planning tasks in autonomous agents based on LLMs. The reflection method enables the agent to generalize experience, identify effective rules within large volumes of data with sparse reward signals, and achieve a level of performance comparable to that of a human expert.
Keywords: *artificial intelligence, machine learning, deep learning, artificial neural networks, intelligent information systems, automated information systems, natural language processing, large language model, prompt, decision making, agent, memory, virtual environment, Minigrid.*

How to quote: I. Omelchenko and V. Strukov, "Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments", *Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 68, pp. 62-69, 2025. https://doi.org/10.26565/2304-6201-2025-68-06

Як цитувати: Omelchenko I., Strukov V. Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments. *Вісник Харківського національного університету імені В. Н. Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2025. вип. 68. С.62-69. https://doi.org/10.26565/2304-6201-2025-68-06

1 Introduction

Recent advances in the field of large language models (LLMs) have enabled a transition from using models as static text generation tools to creating autonomous agents capable of perceiving the environment, performing actions, and acquiring experience. According to the systematic survey [1], the key difference between an LLM and an LLM-based agent lies in the agent's ability to accumulate knowledge and modify its behavior through experience obtained via long-term interaction with the environment. To enable experience accumulation, LLM-based agents require a memory module that

addresses the inherent limitations of LLMs, namely fixed parameters at inference time and a limited context window length.

The limitation of the finite context window of modern large language models (LLMs) remains a fundamental barrier to their application in tasks that require long-term planning and memory. Although context extension methods exist, simply increasing the input window leads to quadratic growth in computational costs and does not guarantee effective information utilization, especially in long-context scenarios.

Traditional reinforcement learning (RL) methods often require a large number of training samples and computationally expensive fine-tuning of models. An alternative to this approach is the use of In-Context Learning (ICL), where agents learn from their own mistakes in language space through mechanisms of summarization or reflection.

For tasks that require sequential planning, such as navigation in the Minigrid environment, it is important to distinguish between different sources of memory: intra-episode (inside-trial) and inter-episode (cross-trial) memory [1]. Intra-episode memory includes the history of observations, actions, and other messages that arise during the current episode and enables the agent to reason about the current situation. Inter-episode memory contains experience accumulated across multiple trials and serves as the foundation for long-term learning.

Efficient design of inter-episode memory constitutes a key challenge. The following types of memory in LLM-based agents can be distinguished [1]: parametric memory (LLM weights) and textual memory. In this work, we focus on textual memory, as it provides interpretability of decisions and allows manipulation of the agent's knowledge without modifying the LLM weights.

One of the simplest approaches to designing inter-episode memory is storing the complete history of the agent's interaction with the environment. The drawback of this approach is that prolonged interaction leads to overflow of the LLM context window, increased computational complexity, and degradation of the LLM attention mechanism performance. Therefore, modern approaches introduce memory management mechanisms, including summarization, forgetting, and reflection.

Reflection [1] is the process of generating higher-level abstractions, reasoning, and conclusions based on observations. The effectiveness of this approach has been demonstrated in prior work [2-4], where agents formed generalized plans based on key actions from past successful or unsuccessful episodes.

In this paper, we investigate a self-reflective memory mechanism for a hierarchical agent, in which the LLM serves as a high-level planner, while sequences of low-level actions are implemented as procedural options. Building on the Indirect Evaluation methodology described in [1], we conduct an ablation study by comparing agent performance across four configurations, ranging from the complete absence of memory to a comprehensive system with CRUD-like operations over a list of generalized rules. This approach allows us to isolate the impact of the summarization and reflection mechanisms on task performance.

2 Related Works

In the work [4], an agent interacts with a set of tasks and collects a history of successful and unsuccessful attempts to solve them. Subsequently, through reflection, the agent forms a list of rules based on pairs of successful and unsuccessful attempts. The resulting rules are then used by the agent in subsequent attempts to solve the tasks.

In the work [5], a memory mechanism for chatbots is proposed that hierarchically structures memory, storing the complete dialogue history, summarized information about events, and a description of the user's psychological profile. Information summarization is performed periodically, every few episodes. A key feature of this approach is a dynamic memory cleaning mechanism based on the Ebbinghaus forgetting curve. The MemoryBank approach is primarily oriented toward social interaction tasks, such as psychological counseling.

The architecture of Generative Agents was proposed in [2]. A fundamental component of this architecture is the memory stream, a database that stores the agent's complete experience history in natural language form. Unlike approaches that place the agent's experience solely within the LLM context window, the Generative Agents framework employs a dynamic retrieval mechanism that selects relevant entries to construct the context for the current step. The retrieval mechanism accounts for properties of memory entries such as recency, importance, and relevance. The approach also includes a

reflection mechanism that synthesizes low-level observations into high-level abstract conclusions, which are stored in the memory stream alongside low-level observations in a tree-like structure.

The concept of verbal reinforcement learning is introduced in [6]. In this approach, the agent's policy is parameterized not only by the model weights but also by the state of the agent's memory, which contains reflections from previous episodes. The agent's functioning is organized as a loop in which the agent generates a trajectory that is evaluated by an evaluator model; the agent then receives binary or textual feedback, performs reflection, and stores the resulting verbal feedback in long-term memory, enabling behavioral changes in subsequent episodes. The authors note that further improvements to the approach may be achieved by employing more complex memory structures, such as vector databases or SQL-based databases.

LLMs often fail to generate an optimal output on the first attempt. To address this issue, the Self-Refine approach was proposed in [7], which is based on the idea of iterative feedback and refinement. The core idea of this approach is to use an LLM in different roles: a generator, a critic (feedback provider), and a refiner. The agent operates in a loop, where at each iteration an initial solution to the task is generated, feedback on the solution is produced, and the agent refines the previous result based on this feedback. Self-Refine demonstrates the effectiveness of LLMs in generating feedback for themselves and improving their own outputs, achieving significant performance gains without requiring model training. A limitation of this approach is that the feedback is used to improve the current output but is not stored in long-term memory.

The approach proposed in [8] iteratively analyzes the agent's history of previous trajectories and its beliefs, modifying these beliefs to improve the policy. Instead of accumulating episodic memory, this method distills experience into behavioral guidelines and performs world modeling. This enables the agent to form an improved policy that incorporates rules generalized by the agent itself using an LLM.

The authors of [9] propose the MemGPT approach for self-directed memory managing, which simulates an unbounded context length by dynamically moving data between the LLM context and an external storage. In this approach, the LLM not only generates and consumes text but also acts as a controller that manages memory. MemGPT includes a set of functions that allow the agent to add, modify, or retrieve information from external storage based on the current state.

Early approaches to memory organization, such as MemoryBank [5] and Generative Agents [2], rely on fixed memory storage schemas or simple accumulation of history, which limits the agent's ability to generalize knowledge to new environments. A more dynamic and flexible approach, A-MEM, is introduced in [10]. The authors construct agent memory not as a passive storage but as an active, self-organizing system inspired by the Zettelkasten note-taking method. A key feature of A-MEM is the use of a generative approach without fixed schemas. In this approach, the LLM autonomously structures relationships between "notes" (atomic units of knowledge). Despite its flexibility, A-MEM has significant limitations, including high experience recording costs, the absence of a forgetting mechanism, and sensitivity to hyperparameter values.

3 Methods

3.1. Agent Architecture

In this work, we study an agent composed of several modules: a planner, a mediator, and an actor. The planner is built on the LLM Qwen3:8b and performs planning of sequences of abstract actions represented in natural language over the space of procedural options, given the current observation of the environment. The mediator transforms vector-based environment observations into a language representation. The actor receives the abstract plan and executes it by selecting the corresponding options.

Five options were implemented for Minigrid-type environments: *Explore*, *GoTo*, *PickUp*, *Toggle*, and *Drop*. During planning, the agent is restricted to selecting options only from this predefined list. Each option includes a precondition for initiation, an execution procedure that translates the abstract action into a sequence of concrete actions, and a termination condition. The `Explore` and `GoTo` options are implemented using Dijkstra's algorithm and the A* algorithm, respectively.

The options generate linguistic feedback for the planner and provide important information or errors encountered during option execution. The feedback is represented as a tuple consisting of the option execution stage, error status, number of executed steps, and a message. Feedback from the options is used by the planner to correct errors in the plan.

To prevent the agent from entering infinite loops in cases where the planner repeatedly generates the same non-functioning plan, a limit on the number of consecutive identical generated plan was introduced, with a maximum of five repetitions. When this limit is exceeded, the episode is terminated and the task is considered unsuccessful. This modification accelerates the experimental procedure by avoiding waiting for episode termination due to reaching the environment step limit.

3.2. Two-Level Memory System

The agent's interaction with the environment proceeds in episodes. During each episode, the planner selects options, which produce feedback that is stored in memory. The agent's memory is structured by episodes: messages from options are stored sequentially, along with environment observations and the generated plan. At the end of each episode, the memory is augmented with information about the number of steps taken in the episode, task success or failure, and the cumulative reward for the episode.

During agent functioning, the planner's prompt includes a static fragment with the plannner's instructions, the task description, a memory fragment corresponding to the current episode, and a set of memory elements formed from the complete history of previous episodes.

We define a *Reflective Memory System* as a dynamic system that evolves in discrete time $t \in N$ through the application of reflection. The state of the memory at certain time t is defined by a set of memory elements, denoted as K_t. Let us denote a set of all agent–environment interaction trajectories at the moment t as $H_t = \{\tau_1, \tau_2, \dots, \tau_t\}$. A mapping from a method-specific subset $\hat{H}_t \subset H_t$ to a set of memory elements K_t that improves the expected reward in future episodes, formally represents the reflection process:

$$K_{t+1} \leftarrow f_{\text{reflect}}(\hat{H}_t, K_t),$$

where $f_{reflect}$ is a function that uses LLM to update the set of memories using selected history trajectories.

The set K_t is defined as a collection of memory elements indexed by unique identifiers. Let $I_t \subset N$ denote the set of active indices at time t. Then, the set of memory elements is defined as:

$$K_t = \{m_k^{(t)}\}, k \in I_t,$$

where each memory element $m_k^{(t)}$ is represented as a tuple:

$$m_k^{(t)} = (\phi_k, c_k, \sigma_k),$$

where k is a unique, permanent identifier of the memory element, ϕ_k denotes the category of the memory element, we use single category *RULE*, $c_k \in \Sigma^*$ is the textual content of the memory element (here Σ^* is the set of all possible strings), and $\sigma_k \in [0,1]$ is the importance coefficient of the memory element, initialized as $\sigma_{\text{init}} = 0.5$.

3.3. Reflection and Learning Mechanism

In this study, several approaches to reflect on the history of episodes were studied.

The simplest approach involves pre-collecting the history of episodes, followed by a reflection process using the LLM *Qwen3:30b*. The resulting reflections were added to the planner prompt, and the agent was evaluated on validation and test sets of environments.

The next approach involved sequentially updating the reflections as episodes were completed by the agent. This represents an online update of the planner prompt based on the experience obtained from interactions with the environment.

An enhanced version of reflection was also studied, in which reflections were represented as a list of memories K_t, each assigned an importance score that was updated during agent functioning. This score determines how critical a rule is for the agent's successful performance. A high score prevents the rule from being removed from the memory, serving as a mechanism to mitigate "catastrophic forgetting" during prompt updates.

In this study, the parameters of the LLM were kept fixed. Changes in agent performance were achieved solely by modifying prompts based on the agent's experience obtained from interactions with environment.

The transition from K_t to K_{t+1} occurs through the application of sequence of managing operators to K_t, denoted as a function g_θ. This function uses a LLM to select operators.

The managing function g_θ takes as input a subset of the history and the current memory state:

$$O_t = g_\theta(\hat{H}_t, K_t),$$

where O_t is the sequence of transactional operations on memory generated by the LLM to optimize the memory K_t.

The set of allowed operations is $O = \{\text{CREATE,UPDATE,DELETE,CONFIRM}\}$. Each operation $\text{op} \in O_t$ modifies the memory state K_t. We define the transformation function as $T: K \times O \to K$. Operates for the set O_t are sequetially applied to the memory K_t to produce next state K_{t+1}.

The importance coefficient of a memory element is updated according to an exponential moving function computed from the validation signal received from the LLM-generated transactions *DELETE* or *CONFIRM*:

$$\sigma_k' \leftarrow (1-\beta)\sigma_k + \beta \cdot \tau_{\text{target}},$$

where τ_{target} is the target importance value of the memory element (0 for penalty, 1 for reinforcement) defined by LLM, and β is a balancing coefficient.

The CREATE operator adds a new element to the set:

$$K_t' \leftarrow K_t \cup (\phi_{\text{new}}, c_{\text{new}}, 0.5).$$

The UPDATE operator is intended to correct erroneous information. The DELETE operator should be used if information in memory is incorrect and needs to be removed. This operator functions as a soft delete by lowering the importance coefficient of the element; when the coefficient falls below a critical threshold, the element is permanently removed at the memory pruning stage. The CONFIRM operator increases the importance coefficient of the memory element.

After applying all selected LLM-generated operations O_t, a pruning stage occurs, in which memory elements with importance coefficients below $\epsilon = 0.1$ are removed.

3.4. Experimental Design

In this study, the Minigrid ColoredDoorKey environment was used, which is a two-dimensional, discrete-state and discrete-action environment. The environment presents a task requiring the agent to open locked doors using a key of the matching color. At the beginning of an episode, objects in the environment are hidden, and the agent must first explore the environment. The environment contains two keys of different colors, one of which is required to open the door, while the another serves as a distractor. Due to limited visibility and incomplete information, the agent must plan actions to obtain information about the door color, find the matching key, pick it up, navigate to the door, and open it.

Two LLMs were used in the experiment: Qwen3:8b and Qwen3:30b. The larger model, Qwen3:30b, was utilized to reflect upon episode histories and managing memory. The smaller model, Qwen3:8b, was used in the planning module, leveraging the reflections produced by the larger model to plan in the space of options. Using models of different sizes is advantageous because planning must be fast, as it is executed multiple times per episode. In contrast, history reflection occurs once every few episodes, and thus weaker constraints on runtime are acceptable for this stage.

To evaluate the contribution of the proposed memory mechanism, ablation studies were conducted. The simplest variant is a memoryless agent, which receives only direct environment observations and has no access to either the current episode history or the overall history. This approach provides a lower bound on the accuracy achievable by the agent. The next level of complexity is episodic memory, in which the agent receives the history of the current episode. This allows the agent to correct mistakes made during the episode but does not enable knowledge transfer across episodes. A more advanced approach allows the agent to access the memories of the last N episodes, simulating long-term memory, but this fills the LLM context and dilutes the agent's attention.

The agent-environment interaction was divided into three stages: Train, Eval, and Test. For each stage, a separate fixed set of random seeds was selected to ensure reproducibility. During the Train stage, the agent collected interaction history with the environment, which was then used either for forming offline reflections or for online reflections updates. The Eval stage was executed in parallel with Train every N episodes to assess the dynamics of agent performance. The Test stage served as the final evaluation, in which the agent was tested over a large number of episodes to compute the final performance metrics.

3.5. Evaluation Metrics

The following metrics were used to evaluate agent performance. The primary performance metric is the *success rate*, calculated as the average task completion success of the agent per episode. *Return* is computed as the average of the sum of rewards throughout the episode, taking into account a penalty for the number of steps taken. *Steps per Episode* measures the number of steps the agent took to complete the episode (successfully or unsuccessfully). Number of Re-plans counts the number of times the

planner had to regenerate a plan to produce a working plan. Planning Duration measures the time spent in the planning stage, used to evaluate the increase in computational cost as the agent's memory size grows.

4 Experiments

We conducted computational experiments in the Minigrid ColoredDoorKey environment. All experiments were performed on a computer system with an NVIDIA RTX 3090 GPU (24 GB VRAM). The LLM was run using the Ollama 0.13.5 utility. We used LLMs quantized in the Q4_K_M format. The "temperature" hyperparameter was set to 0.1. The maximum number of generated tokens was limited to 4096.

We compared several approaches to agent memory organization:

1. **Without Memory:** The agent had no memory or instructions.
2. **Baseline E-1:** The agent remembered only the current episode.
3. **Offline Reflection:** The agent remembered the current episode and received instructions from LLM Qwen3:30b, generated in advance on pre-collected data.
4. **Text Reflection E-5:** The agent had memory of the current episode and performed unstructured reflection every 5 episodes on the last five episodes.
5. **Text Reflection E-5 CP-1:** The agent had memory of the current episode and performed unstructured reflection every 5 episodes on the last pair of successful and unsuccessful episodes.
6. **Text Reflection E-5 CP-3:** The agent had memory of the current episode and performed unstructured reflection every 5 episodes on the last three pairs of successful and unsuccessful episodes.
7. **Reflection E-2:** The agent had memory of the current episode and performed structured reflection with managing operators every 2 episodes.
8. **Reflection E-5:** The agent had memory of the current episode and performed structured reflection with managing operators every 5 episodes.
9. **Manual Instructions:** The agent had memory of the current episode and received high-quality instructions formulated by a human-expert.

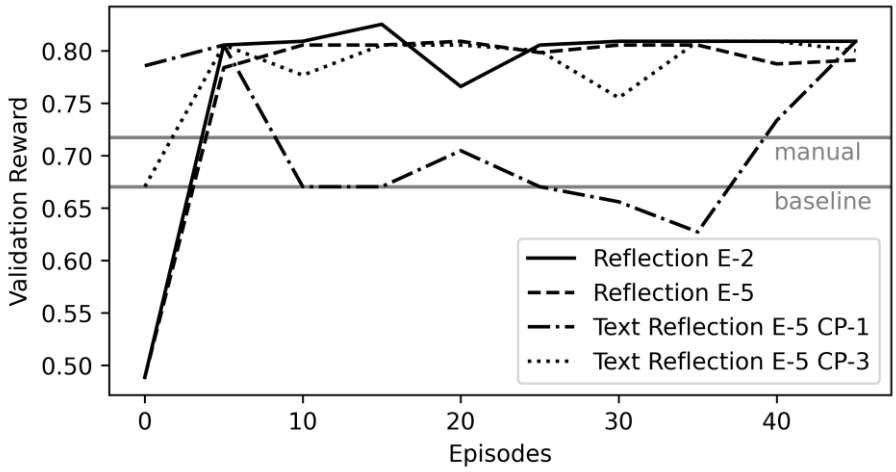

Рис. 1 Залежність сумарної нагороди на валідаційному наборі від епізоду під час навчання
Fig. 1 Dependency of cumulative reward on the validation set as a function of episode during training

Figure 1 shows the dynamics of cumulative reward on the validation set of environments. It can be observed that the cumulative reward for memory with structured reflection (Reflection) reaches maximum values within the first 4 iterations of reflection and achieves better performance than the agent with rules written by a human expert.

An example of the final rule set for the Reflection method:

Model Output
0: Before picking up or toggling an object, move to it to ensure correct interaction, verify its presence, color match, and that the key is held before attempting to toggle. If holding a key and needing to pick up another, drop the current key first. Only move to an object if it is visible or known to be present. Verify key color matches door color before toggling. Move to the object even if visible but not in front to ensure correct targeting. 1: After picking up the key, explore for the door to locate it before attempting to open it. 2: Before toggling the door, verify that the key color matches the door color, move to the door, and ensure the key is held before attempting to toggle.

Табл. 1 Порівняння різних підходів до організації пам'яті агента
Table 1 Comparison of different approaches to agent memory organization

Method	Accuracy	Reward	Steps
Without Memory	0.68	0.4994	52.06
Baseline E-1	0.65	0.4896	52.82
Offline Reflection	0.72	0.5298	48.02
Text Reflection E-5	0.74	0.5470	47.44
Text Reflection E-5 CP-1	0.83	0.6215	40.16
Text Reflection E-5 CP-3	0.96	0.7173	30.96
Reflection E-2	0.99	0.7353	29.29
Reflection E-5	0.98	0.7345	29.27
Manual Instructions	0.99	0.7345	29.62

Table 1 presents a comparison of metric values for agents with different memory mechanisms. The lowest performance is observed for the baseline, corresponding to an agent without long-term memory. As the memory complexity increases, agent performance improves. The most advanced method, Reflexion — structured reflection with a list of memories and managing operators — demonstrates the best results, comparable to or better than the agent with rules provided by a human expert.

6 Conclusions

In this study, the following was accomplished. We proposed using In-Context Learning with reflection via LLM for agent training in virtual environments. The Minigrid ColoredDoorKey environment was prepared for agent training. Agent training code was implemented for the Minigrid ColoredDoorKey environment. Computational experiments were conducted to train and test the agent with different memory mechanisms. The performance of various memory mechanisms was evaluated based on metrics: task completion accuracy, cumulative reward, and number of steps per episode. An analysis and comparison of the influence of memory mechanisms on agent action planning in the ColoredDoorKey environment was performed.

The results indicate that using the reflection method via LLM in an autonomous agent improves agent performance during interaction with a virtual environment. Task completion accuracy shows that the reflective agent can discover rules enabling it to reach human-expert-level accuracy. Learning curves demonstrate that maximum accuracy is achieved by the third reflection step. Additionally, the use of reflection reduces the number of steps required to complete a task and increases the cumulative reward obtained by the agent over the course of an episode.

REFERENCES

1. Zhang Z., Dai Q., Bo X. et. al. A survey on the memory mechanism of large language model-based agents. ACM Transactions on Information Systems. 2025. Vol. 43. P. C. 1—47. https://doi.org/10.48550/arXiv.2404.13501

2. Park J., O'Brien J., Cai C. et. al. «Generative agents: Interactive simulacra of human behavior». In: Proceedings of the 36th annual acm symposium on user interface software and technology. 2023, с. 1-22. https://doi.org/10.48550/arXiv.2304.03442
3. Zhu X., Chen Y., Tian H. et. al. Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory. 2023. arXiv: 2305.17144 [cs.AI]. URL: https://doi.org/10.48550/arXiv.2305.17144.
4. Zhao A., Huang D., Xu Q. et. al. «Expel: Llm agents are experiential learners». In: Proceedings of the AAAI Conference on Artificial Intelligence. Т. 38. 17. 2024, с. 19632—19642. https://doi.org/10.1609/aaai.v38i17.29936
5. Zhong W., Guo L., Gao Q. et. al. «Memorybank: Enhancing large language models with long-term memory». In: Proceedings of the AAAI Conference on Artificial Intelligence. Т. 38. 17. 2024, с. 19724-19731. https://doi.org/10.48550/arXiv.2305.10250
6. Shinn N., Cassano F., Berman E. et. al. Reflexion: Language Agents with Verbal Reinforcement Learning. 2023. arXiv: 2303 . 11366 [cs.AI]. URL: https://doi.org/10.48550/arXiv.2303.11366
7. Madaan A., Tandon N., Gupta P. та ін. Self-Refine: Iterative Refinement with Self-Feedback. 2023. arXiv: 2303.17651 [cs.CL]. URL: https://doi.org/10.48550/arXiv.2303.17651.
8. Zhang W., Tang K., Wu H. та ін. Agent-Pro: Learning to Evolve via Policy-Level Reflection and Optimization. 2024. arXiv: 2402.17574 [cs.AI]. URL: https://doi.org/10.48550/arXiv.2402.17574.
9. Packer C., Wooders S., Lin K. та ін. MemGPT: Towards LLMs as Operating Systems. 2024. arXiv: 2310.08560 [cs.AI]. URL: https://doi.org/10.48550/arXiv.2310.08560.
10. Xu W., Liang Z., Mei K. та ін. A-MEM: Agentic Memory for LLM Agents. 2025. arXiv: 2502.12110 [cs.CL]. URL: https://doi.org/10.48550/arXiv.2502.12110.

Омельченко Ігор Валерійович *Аспірант, кафедра математичного моделювання та аналізу даних Харківський національний університет ім. В.Н. Каразіна. Площа Незалежності, 4, Харків, Харківська область, 61022*

Струков Володимир Михайлович *к.т.н., доцент; завідувач кафедри математичного моделювання та аналізу даних Харківський національний університет ім. В.Н. Каразіна. Площа Незалежності, 4, Харків, Харківська область, 61022*

Архітектура рефлексивної пам'яті для адаптивного планування ієрархічних LLM-агентів у віртуальних середовищах

Актуальність. Великі мовні моделі (LLM) можуть бути використані як один з елементів автономних агентів, що вирішують завдання послідовного прийняття рішень. Для покращення роботи агентів потрібно зберігати історію попередніх спостережень та дій, що призводить до заповнення контекстного вікна LLM, збільшення кількості обчислень, тривалості планування та підвищує вимоги до пам'яті. Можливий підхід до вирішення цієї проблеми полягає у застосуванні методів узагальнення спостережень з використанням LLM.
Мета. Дослідити вплив методів узагальнення пам'яті автономних агентів на основі LLM та рефлексії. Порівняти з простішими методами організації пам'яті.
Методи дослідження. Методи дослідження: обчислювальний експеримент, порівняльний аналіз. Методи організації пам'яті: повна історія епізодів, рефлексія, рефлексія зі структурованим набором правил. Використані метрики якості роботи агента: успішність вирішення завдання, сумарна нагорода за епізод, кількість кроків для вирішення завдання.
Результати. Запропоновано використовувати метод узагальнення пам'яті шляхом рефлексії для ієрархічного агента на основі LLM. Розглянуто середовище Minigrid ColoredDoorKey для навчання агента. Створено код агента, зокрема для навчання агента в середовищі. Проведено обчислювальні експерименти з навчання та тестування агента з різними механізмами пам'яті. Проведено оцінку якості роботи різних механізмів пам'яті на основі метрик: точність виконання завдання, сумарна нагорода, кількість кроків до завершення епізоду. Виконано аналіз та порівняння результатів застосування механізмів пам'яті до задачі планування дій агентом в середовищі ColoredDoorKey.
Висновки. Дослідження демонструє, що застосування методу рефлексії з структурованим набором правил є доцільним в задачах планування дій автономними агентами з використанням LLM. Метод рефлексії дозволяє узагальнювати досвід агента, знаходити ефективні правила в значному об'ємі даних з розрідженим сигналом нагороди та досягати рівня ефективності порівняного з людиною-експертом.

Ключові слова: *штучний інтелект, машинне навчання, глибоке навчання, штучні нейронні мережі, інтелектуальні інформаційні системи, автоматизовані інформаційні системи, обробка природної мови, велика мовна модель, промпт, прийняття рішень, агент, пам'ять, віртуальне середовище, Minigrid.*

Надійшла у першій редакції 03.10.2025; в останній 09.11.2025; опубліковано 30.12.2025.
The first version has been received on 3.10.2025; the final version on 09.11.2025; published 30.12.2025.