Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна

Кваліфікаційна наукова
праця на правах рукопису

Омельченко Ігор Валерійович

УДК 004.8

**ДИСЕРТАЦІЯ**

# МЕТОДИ ТА МОДЕЛІ КЕРУВАННЯ ІЄРАРХІЧНИМИ ІНТЕЛЕКТУАЛЬНИМИ АГЕНТАМИ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ У ЗАДАЧАХ ПЛАНУВАННЯ ТА НАВІГАЦІЇ

Спеціальність 122 Комп’ютерні науки
Галузь знань 12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії
Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Омельченко І.В.

Науковий керівник: Струков Володимир Михайлович, кандидат технічних наук, доцент.

Харків — 2026

# АНОТАЦІЯ

Омельченко І.В. Методи та моделі керування ієрархічними інтелектуальними агентами на основі великих мовних моделей у задачах планування та навігації. — Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 122 Комп'ютерні науки (Галузь знань 12 Інформаційні технології). — Харківський національний університет імені В. Н. Каразіна Міністерства освіти і науки України, Харків, 2026.

Дисертаційна робота присвячена вирішенню актуальної науково-прикладної задачі сучасної галузі штучного інтелекту — забезпечення автономної навігації та взаємодії агентів із динамічними середовищами. У роботі розроблено інформаційну технологію для проєктування та навчання автономних ієрархічних агентів, які базуються на великих мовних моделях (LLM).

У роботі представлено архітектуру ієрархічного агента, яка включає модуль планування на основі великої мовної моделі та рефлексивну пам'ять. Запропоновано метод рефлексивної адаптації агента, який базується на автоматичному виявленні помилок планування в історії епізодів та генерації коригувальних правил природною мовою. Проаналізовано вплив мовних інструкцій на показники ефективності агента у двовимірному середовищі Minigrid. Покращено процес планування дій агента шляхом використання методу декодування з обмеженням граматики.

У дисертаційній роботі представлено результати, які містять наукову новизну та мають прикладну цінність для проєктування та розробки автономних інтелектуальних систем управління, які функціонують у процедурно-генерованих середовищах і можуть бути використані для керування робототехнічними системами.

**Зміст дисертації. Вступ** присвячено обґрунтуванню актуальності обраної теми, визначенню мети та основних завдань дослідження. У розділі також висвітлено наукову новизну та практичне значення дослідження.

У **розділі 1** проаналізовано актуальні наукові джерела, присвячені моделям та методам створення агентів, починаючи від методів навчання з підкріпленням і до сучасних агентів на основі великих мовних моделей (LLM), проблемам

надійності та глаюцинацій у великих мовних моделях, зокрема у задачах планування, моделям пам’яті LLM-агентів та методам керування пам’яттю. На основі аналізу джерел сформульовано завдання дослідження.

**Розділ 2** присвячено методам керування агентами на основі великих мовних моделей за допомогою мовних інструкцій. Розглянуто інструментарій для створення процедурно-генерованих середовищ Minigrid. Проведено експериментальні дослідження з оцінки впливу методів формування мовних інструкцій на показники ефективності агента. Обґрунтовано вибір методу розробки мовних інструкцій для автономних ієрархічних агентів в задачі навігації.

У **розділі 3** розглянуто проблему синтаксичних галюцинацій великих мовних моделей в задачі планування дій ієрархічних агентів. Запропоновано використання методу декодування з обмеженням граматики до процесу планування високрівневих дій LLM-агентом, що дозволяє вирішити проблему синтаксичних галюцинацій LLM. Виконано експериментальні дослідження з виявлення впливу декодування з обмеженням граматики на показники якості планування дій агента.

У **розділі 4** запропоновано модель структурованої рефлексивної пам’яті та архітектуру ієрархічного агента з розробленою рефлексивною пам’яттю. Проведено експериментальні дослідження та порівняльний аналіз показників ефективності агента з різними моделями пам’яті, як з рефлексивною пам’яттю, так і з простішими моделями пам’яті.

У **висновках** відображено головні ідеї та результати дисертаційної роботи.

**Ключові слова:** штучний інтелект, глибоке навчання, нейронні мережі, машинне навчання, стохастична градієнтна оптимізація, навчання з підкріпленням, обробка природної мови, велика мовна модель, transformer, промпт, інтерпретованість, прийняття рішень, агент, віртуальне середовище, Minigrid.

# ABSTRACT

Omelchenko I.V. Methods and models for controlling hierarchical intelligent agents based on large language models in planning and navigation tasks. — Qualification scientific work with the manuscript copyright.

Dissertation for a Doctor of Philosophy Degree by specialty 122 Computer science (Field of knowledge 12 Information Technologies). — V. N. Karazin Kharkiv National University of the Ministry of Education and Science of Ukraine, Kharkiv, 2026.

The dissertation is dedicated to solving an important scientific and applied problem in artificial intelligence field: ensuring autonomous navigation and agent interaction within dynamic environments. The work develops an information technology (framework) for the design and training of autonomous hierarchical agents based on Large Language Models (LLMs).

The dissertation presents a hierarchical agent architecture that includes an LLM-based planning module and reflective memory. A method for reflective agent adaptation is proposed, based on the automatic detection of planning errors within the episode history and the subsequent generation of corrective rules in natural language. The influence of linguistic instructions on agent performance metrics is analyzed within the 2D Minigrid environment. The agent's action planning process is enhanced through the integration of grammar-constrained decoding.

The dissertation presents results characterized by scientific novelty and applied value for the design and development of autonomous intelligent control systems. These systems function in procedurally generated environments and might be applicable to the control of robotic systems.

**Dissertation Content.** The **introduction** justifies the relevance of the chosen topic, defines the research goals and primary objectives, and highlights the scientific novelty and practical significance of the study.

**Chapter 1** provides an analysis of current scientific literature regarding models and methods for agent development, ranging from Reinforcement Learning to modern LLM-based agents. It examines issues of reliability and hallucinations in LLMs, as well as memory models and memory management techniques for LLM agents. Based on this analysis, the specific research objectives are formulated.

**Chapter 2** is dedicated to methods of controlling LLM-based agents via linguistic

instructions. Framework for creating procedurally generated Minigrid environments is reviewed. Experimental studies are conducted to evaluate the influence of instruction-forming methods on agent efficiency. The selection of a linguistic instruction development method for autonomous hierarchical agents in navigation tasks is substantiated.

**Chapter 3** addresses the problem of syntactic LLM hallucinations in action planning for hierarchical agents. The use of grammar-constrained decoding is proposed for the high-level action planning process, which effectively resolves the issue of syntactic hallucinations. Experimental studies show the impact of grammar-constrained decoding on action-planning quality metrics.

**Chapter 4** proposes a structured reflective memory model and a hierarchical agent architecture incorporating this memory mechanism. Experimental research and a comparative analysis of performance metrics are conducted, both for the agent with reflective memory and simplified memory models.

The **Conclusions** section summarizes the key ideas and results of the dissertation.

**Keywords:** artificial intelligence, deep learning, neural networks, machine learning, stochastic gradient optimization, reinforcement learning, natural language processing, large language model, transformer, prompt, interpretability, decision making, agent, virtual environment, Minigrid.

## СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

**Наукові роботи, в яких опубліковано основні результати дисертації:**

*Статті у наукових виданнях, що входять до міжнародних наукометричних баз:*

1. Borysenko O., Bratchenko M., Lukin I., Luhanko M., **Omelchenko I.**, Sotnikov A., Lomi A. Application of Langevin dynamics to advance the Quantum Natural Gradient optimization algorithm. *Physica A: Statistical Mechanics and its Applications*. 2026. Vol. 682. P. 131158. DOI: https://doi.org/10.1016/j.physa.2025.131158 (Scopus, Web of Science, Q2).

*Статті у наукових фахових виданнях України:*

2. **Omelchenko I.**, Strukov V. On the impact of prompts on agent performance in a virtual environment. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 65. P. 83-91. DOI: https://doi.org/10.26565/2304-6201-2025-65-07
   *Особистий внесок здобувача:* проведення аналізу існуючих моделей агентів на основі LLM та методів керування такими агентами, розробка програмного середовища для тестування агента в середовищі Minigrid, розробка мовних інструкцій для керування агентом, виконання обчислювальних експериментів, аналіз та порівняння отриманих результатів, підготовка тексту роботи.
   *Особистий внесок Володимира Струкова:* перевірка отриманих результатів, перевірка тексту роботи.

3. **Omelchenko I.**, Strukov V. Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 67. P. 101-112. DOI: https://doi.org/10.26565/2304-6201-2025-67-10
   *Особистий внесок здобувача:* проведення аналізу методів декодування з обмеженням граматики послідовностей згенерованих LLM, розробка програмного середовища з інтеграцією методу декодування, підготовка датасету на основі середовища Minigrid для тестування модуля планування агента, виконання обчислювальних експериментів, аналіз та порівняння

отриманих результатів, підготовка тексту роботи.
*Особистий внесок Володимира Струкова:* обговорення ідей та методів дослідження, перевірка отриманих результатів, перевірка тексту роботи.

4. **Omelchenko I.**, Strukov V. Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 68. P. 62-69. DOI: https://doi.org/10.26565/2304-6201-2025-68-06
*Особистий внесок здобувача:* проведення аналізу існуючих моделей пам'яті та методів управління пам'яттю автономних LLM агентів, теоретична розробка моделі рефлексивної пам'яті, розробка програмного середовища для дослідження LLM-агента з рефлексивною пам'яттю, проведення обчислювальних експериментів в середовищі Minigrid, аналіз отриманих результатів, підготовка тексту роботи.
*Особистий внесок Володимира Струкова:* перевірка отриманих результатів, перевірка тексту роботи.

**Наукові праці, які засвідчують апробацію матеріалів дисертації:**

1. **Омельченко І.**, Гущин І., Струков В. Аналіз впливу текстових підказок на ефективність навчання агента. *Матеріали X Міжнародної науково-практичної конференції «Комп'ютерне моделювання у наукоємних технологіях» (КМНТ-2024)*. ХНУ імені В. Н. Каразіна. Україна, 2024, С. 160-163.

2. **Омельченко І.**, Струков В. Вплив методів декодування на коректність планування дій LLM-агентів у віртуальних середовищах. *Матеріали Міжнародної науково-технічної конференції «Інтелектуальні технології у міждисциплінарних дослідженнях» (ІТМД-2025)*. ХНУ імені В. Н. Каразіна. Україна, 2025, С. 179-182.

3. **Омельченко І.** Рефлексивна пам'ять в задачі адаптивного планування ієрархічних LLM-агентів у віртуальних середовищах. *Матеріали Міжнародної науково-практичної конференції молодих вчених, аспірантів і студентів «Інформаційні технології в сучасному світі: дослідження*

*молодих вчених»*. ХНЕУ імені Семена Кузнеця. Харків, Україна, 2026, с. 216.

# Зміст

# ПЕРЕЛІК АБРЕВІАТУР І УМОВНИХ ПОЗНАЧЕНЬ

| | |
|---|---|
| LLM | Large Language Model |
| RL | Reinforcement Learning |
| ICL | In-Context Learning |
| ZSL | Zero-Shot Learning |
| FSL | Few-Shot Learning |
| PPO | Proximal Policy Optimization |
| GPT | Generative Pre-trained Transformer |
| MDP | Markov Decision Process |
| POMDP | Partially Observable Markov Decision Process |
| Semi-MDP | Semi-Markov Decision Process |
| BNF | Backus-Naur Form |
| GBNF | GGML Backus-Naur Form |
| GGML | Gerganov’s General Machine Learning |
| GGUF | GPT-Generated Unified Format |
| PTQ | Post-Training Quantization |
| GCD | Grammar-Constrained Decoding |
| UCD | Unconstrained Decoding |
| DFA | Deterministic Finite Automaton |
| JSON | JavaScript Object Notation |
| XML | Extensible Markup Language |

# ВСТУП

**Актуальність теми.** Наразі активно розвиваються методи створення автономних інтелектуальних агентів на основі великих мовних моделей. Такі агенти взаємодіють з певним середовищем, в якому вони здатні отримувати спостереження, виконувати дії та накопичувати досвід.

Попри значний потенціал великих мовних моделей у задачах прийняття рішень, їх застосування до управління агентами у задачах довготривалого планування має суттєві обмеження. Мовні моделі демонструють високу ефективність у декомпозиції задач, але втрачають ефективність на великих часових горизонтах. Інференс мовних моделей є обчислювально-інтенсивним процесом, який також породжує затримки в роботі системи. Це обмежує можливість застосування мовних моделей до вибору низькорівневих дій. Натомість перспективним напрямком є стратегічне планування у просторі абстрактних дій. Це зумовлює необхідність використання ієрархічних архітектур, наприклад, з двох рівнів — планування мовною моделлю у просторі процедурних навичок (опцій) та виконання низькорівневих дій класичною процедурною частиною системи [1–5].

Застосування мовних моделей у задачах прийняття рішень показує значну чутливість таких моделей до формулювання завдань [5–7], тому важливим є дослідження підходів до організації контексту мовних моделей.

Великі мовні моделі працюють на основі статистичних методів і генерують послідовність токенів ітеративно — один токен на кожній ітерації. Токени обираються зі всього доступного словника токенів, що призводить до того, що в прикладних задачах випадково можуть бути обрані продовження вхідної послідовності, які не є рішенням задачі, містять протирічливу інформацію, або порушують очікувану структуру даних. Така ситуація називається проблемою галюцинацій [1, 8]. В задачах, де мовна модель виконує завдання планування, генерація послідовності з випадковими помилками може зробити неможливим використання згенерованого плану через неможливість виконання синтаксичного розбору. Для вирішення цієї проблеми є актуальним застосування методів декодування з обмеженням граматики, які обмежують вибір наступних токенів таким чином, що згенерована послідовність гарантовано відповідатиме очікуваній структурі

даних.

Основною формою пам’яті в нейромережевих системах довгий час були чисельні параметри, для оптимізації яких застосовувався метод стохастичного градієнтного спуску або його модифікації. З розвитком великих мовних моделей з’явилася нова парадигма навчання в контексті (In-Context Learning) [9, 10], яка дозволяє агенту накопичувати досвід у поданні природною мовою та адаптуватися до середовища без зміни параметрів мовної моделі. Історія взаємодії агента з середовищем може містити нерелевантні дані, може бути великою та виходити за межі контекстного вікна мовної моделі. Це зумовлює необхідність створення методів узагальнення епізодичних даних до компактної форми, яка міститиме лише релевантну інформацію. Актуальним підходом до вирішення цього завдання є додавання двох рівнів пам’яті — епізодичної пам’яті, яка містить повну історію, та рефлексивної міжепізодичної пам’яті [11–13], в якій мовна модель обробляє та узагальнює епізодичні дані.

Окрім фундаментального наукового значення, вдосконалення архітектур ієрархічних інтелектуальних агентів має високу прикладну цінність для галузі робототехніки, для розробки автономних систем, здатних до довготривалого планування та навчання у середовищах зі складною динамікою, які функціонують без постійного втручання оператора. Іншим можливим практичним застосуванням є автоматизація багатокрокових бізнес-процесів, моніторинг інформаційних систем та інфраструктури.

З огляду на вищезазначене, наукове завдання полягає у розробці та вдосконаленні методів та моделей управління ієрархічними інтелектуальними агентами, які використовують великі мовні моделі як модуль планування. Вирішення цієї задачі потребує застосування комплексного підходу: покращення підходів подання контексту, впровадження методів декодування з обмеженням граматики, розробки архітектури міжепізодичної пам’яті в задачах планування та просторової навігації.

**Мета дослідження** — розробка та дослідження методів керування ієрархічними інтелектуальними агентами на базі великих мовних моделей, що забезпечують ефективну декомпозицію абстрактних цілей у послідовності дій, підвищення надійності планування та довгострокову адаптацію поведінки агентів без модифікації параметрів мовної моделі.

**Об’єкт дослідження** — це процеси керування інтелектуальними агентами

у середовищах планування та навігації.

**Предмет дослідження** — методи, моделі та алгоритми керування ієрархічними інтелектуальними агентами з використанням великих мовних моделей, структурованого декодування та механізмів рефлексивної пам’яті.

**Основні завдання дисертації:**

1. Провести аналіз сучасного стану досліджень методів керування інтелектуальними агентами на базі великих мовних моделей (LLM) у задачах планування та навігації, виявити обмеження існуючих підходів щодо ефективності, надійності та довгострокової адаптації.
2. Розробити архітектуру ієрархічного інтелектуального агента, яка поєднує модуль стратегічного планування на базі LLM з виконавчим механізмом низького рівня, та забезпечує декомпозицію абстрактних цілей у послідовність атомарних дій.
3. Виконати дослідження впливу різних методів формування контексту LLM на метрики ефективності агента, обґрунтувати вибір методів формування контексту для задач планування.
4. Виконати адаптацію та інтеграцію методу декодування з обмеженням граматики (Grammar-Constrained Decoding) до модуля планування агента для гарантування синтаксичної валідності згенерованих планів та підвищення ефективності планування.
5. Розробити метод рефлексивної пам’яті, який дозволяє агенту узагальнювати досвід попередніх епізодів у вигляді правил, забезпечуючи адаптацію поведінки агента без зміни ваг моделі.
6. Виконати програмну реалізацію розробленого агента у вигляді програмної системи для моделювання агентів у середовищі Minigrid.
7. Провести експериментальні дослідження та порівняльний аналіз роботи агента в різних конфігураціях та порівняти з базовими підходами для підтвердження підвищення ефективності виконання завдань.

**Методи дослідження:**

1. Математичне та комп’ютерне моделювання: розробка концептуальних і комп’ютерних моделей просторової навігації та планування дій агентів у стохастичних середовищах із частковим спостереженням.

2. Експериментальні дослідження: проведення обчислювальних експериментів із застосуванням сучасних інструментальних засобів та високопаралельних обчислювальних систем для верифікації ефективності розроблених моделей.

3. Інтелектуальні методи та інженерія знань: застосування методів глибинного навчання, навчання з підкріпленням та великих мовних моделей (LLM) для апроксимації стратегій управління та репрезентації знань у автономних агентах.

4. Статистичні методи аналізу даних: застосування методів статистичної обробки та аналізу експериментальних даних великого обсягу для кількісної оцінки метрик ефективності.

5. Методи та алгоритми розв’язання прикладних задач комп’ютерних наук: синтез, оптимізація та програмна імплементація алгоритмів ієрархічного управління пам’яттю інтелектуального агента.

**Наукова новизна отриманих результатів.** Наукову новизну в роботі становлять такі практичні та теоретичні результати:

1. Представлено вперше:

   - Розроблено метод адаптивної рефлексивної пам’яті для ієрархічних агентів, який, на відміну від існуючих підходів, здійснює динамічне оновлення множини керуючих правил з використанням операторів модифікації та механізму важливості правил, що забезпечує узагальнення досвіду в процесі не параметричного навчання.
   - Запропоновано архітектуру ієрархічного інтелектуального агента, у якій велика мовна модель використовується як модуль стратегічного планування з формально обмеженим простором дій на основі граматик та застосовано запропоновану модель рефлексивної пам’яті.

2. Набуло подальшого розвитку:

- Методи застосування великих мовних моделей у задачах планування та навігації шляхом поєднання з ієрархічними структурами керування.
- Використання методів формування контексту LLM як інструмента керування поведінкою інтелектуальних агентів у динамічних середовищах.

3. Удосконалено:

- Метод планування в ієрархічних агентах шляхом використання декодування з обмеженням граматики, що підвищує надійність та ефективність генерації планів.

**Практичне значення отриманих результатів.** Практичне значення отриманих результатів полягає у можливості прикладного застосування розроблених методів, моделей та алгоритмів для створення і підвищення надійності ієрархічних інтелектуальних агентів на основі великих мовних моделей. Запропоновані теоретичні підходи, зокрема архітектура агента з рефлексивною пам’яттю та метод планування шляхом застосування декодування з обмеженням граматики, доведено до рівня конкретних алгоритмічних рішень. Розроблені рішення мають перспективу застосування у задачах проєктування автономних інформаційних систем, систем штучного інтелекту та керування робототехнічними системами (зокрема, в задачах просторової навігації та планування з обмеженнями, сформульованими природною мовою).

Розроблені алгоритми та програмні модулі можуть слугувати основою для подальших прикладних розробок та інтеграції інтелектуальних агентів у системи прийняття рішень, які потребують гарантованої коректності синтаксису згенерованих планів та контролю часу планування. Окрім цього, отримані результати доцільно використовувати у навчальному процесі закладів вищої освіти під час підготовки фахівців галузі знань «Інформаційні технології», зокрема при викладанні дисциплін, пов’язаних із машинним навчанням, проєктуванням автономних агентів та обробкою природної мови.

**Особистий внесок здобувача.** Здобувачем самостійно отримано основі результати дисертаційного дослідження. В опублікованих у співавторстві наукових працях здобувачем здійснено огляд та аналіз актуальних робіт,

розробку моделей та методів планування дій агентів у віртуальних середовищах, проведення експериментів з середовищем Minigrid, проведення подальшого аналізу отриманих результатів, розробку програмних інструментів. Співавторам статей належить участь в формулюванні завдань, аналіз результатів та формулювання висновків.

**Апробація результатів дисертації.** Результати дисертації доповідались та обговорювались на:

- X Міжнародній науково-практичній конференції «Комп'ютерне моделювання у наукоємних технологіях» (КМНТ-2024), ХНУ імені В. Н. Каразіна, 2024 року, Харків, Україна.
- Міжнародній науково-технічній конференції «Інтелектуальні технології у міждисциплінарних дослідженнях» (ІТМД-2025), ХНУ імені В. Н. Каразіна, 2025 року, Харків, Україна.
- Міжнародній науково-практичній конференції молодих вчених, аспірантів і студентів «Інформаційні технології в сучасному світі: дослідження молодих вчених», ХНЕУ імені Семена Кузнеця, 2026, Харків, Україна.

**Публікації.** Основні наукові результати дисертаційної роботи повною мірою викладено в 4 публікаціях англійською мовою, з яких: 3 статті опубліковано в фахових виданнях України, 1 проіндексовано в наукометричній базі Scopus.

**Структура та обсяг дисертації.** Дисертаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 173 с., з яких основний текст — 145 с. Кількість матеріалів: бібліографічних найменувань — 110, рисунків — 39, таблиць — 11, додатків — 1.

# 1 Огляд стану проблеми і постановка завдання дослідження

## 1.1 Аналіз особливостей задач навігації та планування у стохастичних середовищах

У [1] досліджується проблема планування дій для втілених агентів (embodied agents — штучних інтелектуальних систем, які взаємодіють з фізичним або віртуальним середовищем). Згідно зі статтею, основна особливість планування в таких середовищах полягає у необхідності середньорівневого заземлення (mid-level grounding) — перекладу високорівневих цілей, сформульованих природною мовою (наприклад, «приготувати сніданок»), у послідовність конкретних, допустимих для виконання дій. У статті зазначається, що згенеровані плани повинні задовольняти синтаксичним нормам симулятора та обмеженням здорового глузду, тобто враховувати передумови та наслідки кожної окремої маніпуляції.

У ряді робіт розглядаються агенти, які оперують у неперервному середовищі. Для спрощення моделі поведінки певні сукупності дій представляються як навички або опції [2–4]. Високорівневе планування дозволяє комбінувати навички або опції для розв’язку поставленої задачі [14–16]. У роботі [4] показано, що оперування абстракцією «навичка» в неперервному середовищі Continuous Playroom дозволило прискорити пошук рішення в 1000 разів порівняно з підходом, у якому план включав розрахунок положень окремих складових елементів робота.

У роботі [17] автори звертають увагу на те, що навчання з підкріпленням використовує фреймворк марковського процесу прийняття рішень. Цей фреймворк використовується для спрощення представлення важливих властивостей штучного інтелекту [17]. Серед таких властивостей автори виділяють зв’язок причини та наслідків, врахування невизначеності, постановку цілей [17].

Minigrid — це бібліотека з набором двовимірних дискретних середовищ, які можуть бути використані для дослідження марковських процесів прийняття рішень (Markov Decision Process [2]) і частково спостережуваних марковських процесів прийняття рішень (Partially Observable Markov Decision Process [18]).

Відносна простота таких середовищ дозволяє досліджувати безпосередньо стратегії прийняття рішень, абстрагуючись від інших деталей, які виникають у більш складних середовищах.

## 1.2 Огляд сучасних підходів до побудови агентів: від навчання з підкріпленням до мовних моделей

Останні роки значний інтерес в науковій спільноті представляють мовні моделі, побудовані на архітектурі Transformer [19]. Такі моделі організовані як глибокі нейронні мережі, які вирішують задачу визначення наступного токену в текстовій послідовності. Токен — це фрагмент слова, який складається з одного або більше символів. Мовні моделі, навчені на великих та різноманітних текстових корпусах, демонструють здібності до обробки довільних текстів, розуміння різних мов, вирішення логічних та математичних завдань. Такі здібності можливо використати для вирішення задачі прийняття рішень в середовищі [20].

Задача прийняття рішень може бути сформульвана через ієрархію дій [2], в якій вищі рівні складаються з послідовностей дій нижчого рівня. Такі високорівневі дії можуть утворювати абстрактний план дій агента.

В недавніх роботах розглядається можливість створити агента, що використовує мовну модель в якості планувальника [5]. Але недостатньо уваги було приділено дослідженню впливу мовних інструкцій (eng. prompt) на навчання та функціонування агента.

У [6] промпт визначається як вхідні дані, які передаються генеративній моделі штучного інтелекту та обумовлюють її вихідні дані. У статті зазначається, що промпт може містити наступні складові: директива (основна ціль промпта); приклади правильного виконання завдання; форматування виводу (вказівки щодо структури вихідних даних); стилістичні інструкції; роль, якої має дотримуватися мовна модель; а також додаткова інформація (контекст), яка необхідна для вирішення задачі.

Промпти створюються з використанням різних технік промптингу [6]. Однією з фундаментальних технік є навчання в контексті (In-Context Learning, ICL) [9, 10]. Ця техніка передбачає покращення виконання завдання за допомогою мовних інструкцій, які можуть додатково містити

приклади вирішення завдання. Здатність мовних моделей використовувати такий контекст усуває необхідність оновлювати ваги моделі градієнтним спуском для покращення її результативності в подальших задачах. У рамках In-Context-Learning існують такі основні парадигми: Zero-Shot [21] передбачає пояснення контексту без надання прикладів, Few-Shot [9] — потребує наведення декількох прикладів правильного рішення задачі. Щодо застосування підходу без прикладів вирішення завдань (Zero-Shot) у задачах планування, у [1] показано, що великі мовні моделі (LLM — Large Language Models, глибокі нейромережеві архітектури, навчені на масштабних текстових корпусах) володіють достатнім обсягом накопичених знань про модель світу для декомпозиції завдань на логічні кроки. Однак кроки, згенеровані LLM, часто неможливо виконати через лінгвістичну неоднозначність, використання вкладених конструкцій та пропуск неочевидних, але обов'язкових проміжних кроків. Подолання цих складнощів вимагає використання додаткових технік.

Поза межами базового ICL виокремлюють складніші стратегії, орієнтовані на багатоетапне логічне виведення. Одним із ключових підходів є генерація «думок» (Thought Generation), найвідомішим представником якої є техніка Chain-of-Thought (CoT) [22]. Цей метод інструктує LLM генерувати проміжні кроки міркувань перед формулюванням остаточної відповіді. Замість безпосередньої генерації виходу, спочатку генерується проміжний ланцюжок думок.

Паралельно виникли два підходи до CoT. Автори роботи [22] передавали LLM декілька прикладів розв'язку завдання. Кожний розв'язок був представлений як послідовність проміжних кроків. Автори [21] показали, що додавання фрази «let's think step by step» до промпту, який не містить прикладів, суттєво покращувало точність моделі GPT-3.

Для вирішення складних проблем застосовуються методи декомпозиції [23, 24]. Ця концепція передбачає розбиття складного завдання на впорядковану множину простіших підзадач. Модель розв'язує їх послідовно, при цьому результат кожної попередньої підзадачі стає частиною контексту для наступної.

З метою підвищення надійності виводу та зменшення фактологічних помилок застосовуються методи самокритики (Self-Criticism / Reflexion) [12, 25]. У таких методах модель виступає у ролі власного валідатора. Після генерації першого варіанту відповіді моделі подається інструкція критично

оцінити першу відповідь, виявити логічні помилки та згенерувати виправлену версію відповіді.

Мовна модель у роботі [26] ознайомлюється з історією текстових ігор, зіграних людьми, формує контекст і модель можливих дій. Дана система не має доступу до характеристик стану симуляції ігрового світу, прихованих від гравців. Незважаючи на це, модель здатна вирішувати завдання завдяки засвоєному контексту. Завдяки здібності до узагальнення система може також без донавчання грати в ігри подібного типу, яких раніше не бачила. Модель отримує апріорні знання з пар контекст-дія. Для узгодження з попередніми діями зберігається обмежена кількість останніх дій і спостережень. Запропоновані мовною моделлю GPT-2 дії проходять фільтрування бінарним нейромережевим класифікатором на допустимість виконання в середовищі. Після цього компонент, навчений методом навчання з підкріпленням, обирає зі списку допустимих запропонованих дій, дію з найвищою оцінкою можливого успіху.

У роботі [27] використано концепцію фонових знань. Велику мовну модель використано в ролі компонента системи, який навчався методом навчання з підкріпленням. Мовна модель викликалася одноразово для збору інформації про принципи функціонування віртуального середовища. Зібрані дані перетворювалися на функцію потенціалу, яка використовувалася для обчислення додаткової винагороди. Цей метод дозволяв агенту правильно вирішувати задачу в середовищі Minigrid як при масштабуванні середовища, так і при появі нового кольору ключа або нового типу об’єкта.

У роботі [28] реалізовано навчання на помилках, яке сприяє вирішенню проблеми розрідженої винагороди. Коли система не досягає поставленої мети, мовна модель аналізує, чи можливо було б досягнути мети замінивши якісь об’єкти на інші (правильні). Якщо це можливо, то система отримає виправлений приклад розв’язаного завдання. Таким чином, розглянута система в середовищі RGB Stacking досягає 80% правильно розв’язаних завдань значно швидше за агентів, які не використовували аналіз помилок.

У роботі [29] запропоновано використовувати мову як латентний параметр, який агент може змінювати, щоб покращити свою ефективність вирішення задач.

Абляційні дослідження в роботі [30] проілюстрували важливість ролі

мовної моделі у формуванні пам’яті про середовище. При довільному відображенні візуальних даних на токени (в якому об’єктам відповідають умовні позначення замість фактичної класифікації) точність планування знижується. Це вказує на те, що зв’язки між поняттями, відображені в мовній моделі, сприяють плануванню. Також заміна мовної моделі на рекурентну нейромережу, яка зберігає дані, але не має зв’язків між поняттями, значно знижує ефективність розв’язання завдання.

В роботі [31] запропоновано ієрархічну архітектуру системи для задачі керування роботом. Задача, поставлена в роботі, розділяється на три рівні: рівень специфікації: неперервний простір конфігурацій робота розбивається на дискретні області, які утворюють граф; рівень виконання — здійснюється вибір конкретного шляху в межах графу; рівень реалізації — дискретні символи перетворюються на конкретні команди керування за допомогою вирішення певних диференціальних рівнянь.

Дослідження в роботі [32] показали, що в середовищі з частковою спостережуваністю агент, навчений імітувати процес мислення людей, розв’язує завдання з вищою ефективністю порівняно з агентом, який навчений імітувати поведінку людей за рахунок вибору найбільш імовірних дій. Агент складається з генератора міркувань і генератора дій, який узгоджує вибір дій, спираючись як на спостереження середовища, так і на вихід генератора міркувань.

В роботі [33] запропоновано архітектуру «планувальник-контролер-мета-контролер». В цій архітектури планувальник використовує символьні знання для довгострокового планування.

В роботі [34] автори використали логічні нейронні мережі (LNN) для ігор з текстовим інтерфейсом. LNN дозволяють вивчати символьні правила всередині нейронної структури.

У [35] стверджується, що на прикладі двовимірних лабіринтів у середовищі Maze2D можна тестувати здатність алгоритмів навчання з підкріпленням з’єднувати частини рішення в повноцінний план. У роботі [36] також пропонується тестувати здатність RL-агентів до узагальнення та переносу досвіду між задачами за допомогою дискретних спрощених середовищ, які мають потенціал для постановки експериментів для виявлення таких здібностей.

Автори [37] продемонстрували, що використання абстракцій дозволило вирішувати задачу прийняття рішень у робототехніці без детального моделювання робота та середовища. Підхід базується на affordances — допустимих діях у певних станах. Такий підхід зменшив вплив візуального шуму. Експерименти показали, що при такому підході агент досягав успішності, близької до рівня агента-оракула, оперуючи одночасно майже 10 об'єктами.

Існує приклад часткового перенесення досвіду, отриманого в модельних середовищах, на рішення більш складних і реалістичних задач. Показано, що агент, навчений у дискретному текстовому середовищі, краще адаптується до виконання завдань у реалістичній кухні ALFWorld [38].

Проект SayCan від Google DeepMind [39] використовує підхід Affordances — дій, допустимих у певному стані агента [40] для управління реальними роботами-маніпуляторами. Підхід Affordances розвивався в тому числі у дослідженнях у середовищах на основі Minigrid [41].

Автори роботи [42] запропонували підхід «Describe, Explain, Plan and Select» (DEPS) на основі великих мовних моделей. Підхід грунтується на взаємодії компонентів архітектури: при отриманні зворотнього зв'язку від середовища про невдалу спробу виконати план, компонент Descriptor формує текст опису поточного стану середовища і передає цей текст компоненту Explainer, який, використовує велику мовну модель і з'ясовує причину невдачі. На основі сформульованої причини планувальник Planner оновлює план. Goal Selector вибирає з множини можливих у поточному стані проміжних цілей таку ціль, шлях до якої найменший (в напрямку виконання поставленого завдання). Агент з такою архітектурою був першим агентом на основі великих мовних моделей, який зміг виконати складне завдання (видобути алмаз) у непередбачуваному середовищі Minecraft без демонстрацій еталонного ігрового процесу.

У [43] системі надається відеодемонстрація або опис траєкторії станів, необхідних для виконання завдання. Відео або опис траєкторії станів конвертується у векторне представлення. Отримуючи завдання, агент розділяє його на кроки. Система виділяє з демонстрації фрагмент, на якому виконується запланована дія. На основі цієї інформації наступний до виконання крок рекурсивно розділяється на ієрархічну послідовність кроків, які аналогічно зіставляються з фрагментами демонстрації. Найглибший рівень цієї вкладеності формує низькорівневі інструкції для керування

роботом. Здійснюючи низькорівневі дії, агент отримує зворотний зв’язок від середовища. На основі зворотнього зв’язку агент може за потреби змінювати план або приходити до висновку, що завдання виконано. Автори показали, що агент з рекурентною пам’яттю може отримувати негативний вплив від пам’яті про останні кроки в ситуації, коли агенту заважають виконувати завдання. Це пояснюється тим, що в рекурентній архітектурі спогади про успішність здійснених кроків зменшує шанс відреагувати на зміну стану середовища під дією незалежних від агента обставин. Тому автори запропонували використовувати «реактивне ядро», яке реагує на фактичний стан середовища для прийняття рішень.

Автори роботи [44] запропонували підхід до навчання ієрархічного агента на основі навчання з підкріпленням HAL (Hierarchical Abstraction with Language), який використовує мову для роботи з абстракціями. Автори запропонували схему перепозначення інструкцій HIR (Hindsight Instruction Relabeling), яка по аналогії з Hindsight Experience Replay (HER) [45], перепозначає початкову інструкцію так, щоб вона відповідала фактично досягнутому результату, якщо послідовність дій призвела до стану, відмінного від заданого спочатковою ціллю. На думку авторів, підхід HER зазвичай призводить до появи шуму із нерелевантних деталей у векторах представлення стану та цілі, що ускладнює навчання агента. На відміну від векторного опису стану, мовне представлення дозволяє описати ціль більш абстрактно без нерелевантних деталей. Запропонований підхід показав вищі показники ефективності розв’язання задачі з маніпуляції об’єктами в середовищі MuJoCo+CLEVR порівняно з попередніми методами ієрархічного навчання з підкріпленням HIRO і Option-Critic, а також методом неієрархічного навчання з підкріпленням DDQN.

У [46] запропоновано архітектуру агента з механізмом уяви IMAGINE. У середовищі сформульовані 3 дії, 32 об’єкти, розділені на 5 категорій, і задано властивості цих об’єктів. Агент може робити спроби застосувати дії з набору до наявних на сцені середовища об’єктів. При цьому агент мав наглядача — зовнішній алгоритм, який називається «соціальний партнер» і має властивості оракула через те, що надає точний опис дії та її результату. HER дозволяє агенту розглядати послідовність дія - неочікуваний наслідок як досвід досягнення нової цілі, дозволяючи агенту вчитися навіть на невдалих спробах. На основі

евристично визначених граматичних конструкцій агент створює нові уявні цілі, шукаючи в реченнях, які відрізняються одним словом, відповідні конструкції, навчаючись здібності заміняти відповідно одні дії на інші дії, властивості на властивості, об’єкти на об’єкти. Таким чином агент отримує можливість сформувати ціль, про яку йому не розповідав «соціальний партнер». При цьому мова сприймається агентом завдяки кодувальнику LSTM, який перетворює мовні конструкції від «соціального партнера» у вектори. Агент здатний оцінити досяжність цілі завдяки тренуванню внутрішньої функції винагороди, яка оцінює потенційний успіх дії на основі отриманого за допомогою «соціального партнера» досвіду. Навчений таким чином агент показав здатність навчатися виконувати дії над новими об’єктами та реагувати на зворотній зв’язок від середовища.

У роботі [47] запропоновано архітектуру LGB (Language-Goal-Behavior) агента DECSTR, яка є ідейним продовженням архітектури LB (Language-Behavior) підходу IMAGINE. На відміну від підходу IMAGINE, агент DECSTR спочатку досліджує середовище не отримуючи зовнішню винагороду, але зіставляючи дії з їхніми наслідками. На наступному етапі навички пов’язуються з відповідним мовним описом. Для досягнення певного стану DECSTR використовує генератор цілей, який для опису цільового стану генерує розподіл семантичних конфігурацій, які описують множину станів середовища, які задовольняють опису цільового стану. DECSTR на відміну від підходу IMAGINE перемикається між стратегіями, спираючись на згенерований розподіл семантичних конфігурацій. Це дозволяє агенту після невдалої спроби продовжити пошук способу досягнення цілі. Такий підхід дозволив виконати завдання зі складання об’єктів у стопку, з чим не зміг впоратися підхід на основі IMAGINE.

Автори роботи [48] запропонували алгоритм RIG (Reinforcement Learning with Imagined Goals), який полягає в тому, що агент вибирає з апріорного розподілу VAE (Variational Autoencoder) вектори в латентному просторі, які відповідають цілі, якої агент має досягти. За допомогою кодувальника візуальна інформація про поточний стан робота перетворюється у вектор у латентному просторі. Метрика винагороди влаштована так, що агент мінімізує відстань між поточним та цільовим станами в латентному просторі. Комбінація підходу HER і перепозначення цілей на основі сформованих

уявних цілей дозволяє отримувати навчальні приклади по мірі того, як агент робить невдалі дії. У середовищі MuJoCo запропонований алгоритм показує кращу ефективність у розв'язанні задачі навігації та маніпуляцій робота порівняно з незміненим підходом HER, а також підходами L&R: Lange and Riedmiller (який використовує автоенкодер для обробки візуальної інформації) та DSAE (глибоким просторовим автоенкодером — підходом, який використовує навчання просторового енкодера та використання керованого пошуку стратегій) [48]. Алгоритм RIG також дозволив навчити агента за порівняно невеликий час.

У роботі [49] представлено CC-RIG (Context-Conditioned RIG) — модифікацію алгоритму RIG, у якій модифікована модель CC-VAE дозволяє виділити з інформації з середовища статичний контекст (колір, освітлення, форма, текстура та інші незмінні властивості об'єктів) і змінні, якими робот може керувати, наприклад, положення об'єктів. Такий підхід є наступним рівнем абстракції відносно RIG, що дозволяє зменшити кількість нерелевантної інформації, яку обробляє агент. У результаті, CC-RIG здатний до узагальнення з першої спроби, коли мова йде про появу нових властивостей об'єктів, якими робот має маніпулювати.

У [50] під час навчання агент вивчає релевантність низькорівневих інструкцій до високорівневих цілей завдяки класифікатору релевантності, який навчається на успішному досвіді: підзадача вважається корисною, якщо вона з'являється в усіх успішних спробах виконати задачу вищого рівня.

У роботі [51] мовна модель використовувалася як планувальник, який перетворював задачу для робота в набір послідовних дій. При цьому планування відбувалося з мінімальною візуальною інформацією або без неї виключно за рахунок загальних знань мовної моделі про світ. Автори стверджують, що на момент створення їхнього планувальника найкращі альтернативні моделі вирішували успішно тільки 5% задач у середовищі ALFRED. Запропонована авторами модель без візуальних даних генерувала правильні плани у 26,1% випадків, при додаванні однієї координати — положення першого об'єкта, з яким треба взаємодіяти — відсоток успішних планів зростав до 58,2%. Такий підхід демонстрував кращу точність порівняно з RNN, а також стійкість до неповних даних у порівнянні з глибоким навчанням з підкріпленням. Таким чином, було показано, що планування з використанням

великих мовних моделей дає переваги порівняно з глибоким навчанням з підкріпленням.

У [52] запропоновано фреймворк, який використовує природну мову для представлення бібліотеки навичок. За допомогою знань, які містяться в мовних моделях агент здатний запропонувати послідовність підзадач для досягнення заданої цілі. Навчання агента відбувалося на даних, які складалися з формулювання цілі, послідовності пар спостереження-дія, 10% таких даних містили додаткове поле — опис етапів виконання завдання. Було показано, що запропонований агент, навчений на такому наборі даних, успішно виконував більше підзадач, порівняно з альтернативними агентами на основі архітектури Seq2Seq [53], навченими на даних, 100% яких містили опис етапів виконання завдання.

У [54] вирішується задача навігації в частково спостережуваному середовищі (POMDP). Модель використовує архітектуру Seq2Seq з увагою, де Encoder трансформує інструкцію в послідовність контекстних векторів. Декодувальник на кожному кроці обробляє конкатенацію візуальних ознак поточного кадру та представлення попередньої дії, застосовуючи механізм уваги до інструкції для формування розподілу ймовірностей над наступними діями.

Автори [55] пропонують фреймворк Neural Modular Control (NMC) для розв’язання задачі Embodied Question Answering (EQA). Процес відбувається як ієрархічне прийняття рішень, де стратегія вищого рівня обирає підцілі, які виконуються стратегіями нижчого рівня. Архітектура об’єднує behavior cloning (імітацію дій експерта на основі багатьох прикладів правильних планів) з A3C (Asynchronous Advantage Actor-Critic) — алгоритмом навчання з підкріпленням. Доведено, що попереднє навчання через імітацію суттєво підвищує ефективність використання даних (sample efficiency), а подальше донавчання за допомогою RL дозволяє агенту адаптуватися до помилок навігації та зіткнень з об’єктами. Порівняння з моделлю PACMAN, яка на основі клонування поведінки та візуальних даних пропонує низькорівневі дії для досягнення цілі, показали, що донавчання з підкріпленням покращує метрику «прогрес наближення до цілі», а також збільшує відносну кількість правильних відповідей, дозволяючи агенту коректно поводитися на довших часових проміжках.

У [56] автори пропонують метод DIAL, де для автоматичної розмітки 80 000 траєкторій використовувалася GPT-3 (для генерації кандидатів інструкцій) та донавчена модель CLIP (для вибору найбільш релевантних описів на основі початкового та кінцевого кадрів процесу виконання дії). На отриманому розширеному наборі даних навчалася імітаційна стратегія керування RT-1. Під час тестування агент отримував виключно текстову команду, демонструючи приріст успішності на 22% у завданнях із просторовими орієнтирами та на 6,7% при зміні формулювань, що свідчить про ефективне узагальнення до раніше невідомих інструкцій.

У статті [57] запропонована чотирирівнева модель пам’яті для впровадження в агента, який спілкується з користувачем за допомогою текстових повідомлень. Однією з задач агента є необхідність утримувати контекст розмови. Перший рівень пам’яті є базою знань, яка містить у векторному представленні необхідну для взаємодії інформацію (ролі, які може відігравати агент; інформацію про світ, з яким взаємодіє агент, тощо). Другий рівень містить векторне представлення повної історії поточної розмови. Третій рівень відстежує сюжетні лінії та розгортання подій у часі, а також аналізує історію взаємодії з користувачем для визначення його емоційного стану та встановлення його ставлення до агента. Четвертий рівень зберігає найбільш актуальну інформацію, надаючи пріоритет новішій інформації. Запропонована модель була здатна утримувати контекст і робити логічні висновки в рамках 60-63 повідомлень.

На думку автора роботи [58], включення великих мовних моделей у промислових роботів може покращити їхню адаптацію до різних умов за рахунок здатності виявляти та виправляти свої помилки, а також здатності отримувати інформацію від людей і інших роботів.

Автори [59] звертають увагу на те, що POMDP може використовуватися в інтелектуальних агентах для моделювання циклу «сприйняття-дія».

У [60] запропоновано оцінювати ефективність алгоритму з пошуку розв’язку задач за наступними критеріями: повнота (гарантія знаходження рішення алгоритмом або впевненості в його відсутності); оптимальність (найменша ціна шляху до цілі); час пошуку рішення (в реальному часі або в термінах розглянутих станів і дій); об’єм пам’яті, необхідний для пошуку рішення.

У [61] автор запропонував багатоагентну систему прийняття рішень для задачі вибору стратегії кібербезпеки корпоративного дата-центру. В основу кожного агента була покладена мовна модель GPT-4, якій за допомогою промпту була призначена певна професійна роль. Прийняття рішень у цій задачі полягало в побудові рейтингу заходів забезпечення кібербезпеки. Усі агенти згідно з відведеною роллю порівнювали попарно надані їм критерії, а також варіанти подальших дій. На основі порівняння агенти заповнювали таблицю попарних порівнянь числами, які відображали пріоритетність, а також надавали окремо текстове обґрунтування вибору. Отримані таблиці оброблялися класичними алгоритмами для прийняття остаточного рішення. Оцінки роботи такої системи авторами показали узгодженість рішень такої системи та стабільність рішення при вилученні окремих агентів. Запропонована системою стратегія узгоджується з загальноприйнятими стандартами галузі.

Автори роботи [62] розробили ієрархічного агента, який планує сценарії кібератаки в комп'ютерних мережах з метою виявлення вразливостей таких мереж. Високорівневий планувальник аналізує опис мережі та її компонентів, бази даних вразливостей, звіти про попередні атаки. На основі цієї інформації планувальник пропонує нову стратегію атаки, використовуючи природну мову. Сформульовані стратегії обробляються компонентом виконання, навченим методом навчання з підкріпленням, на основі моделі марковських процесів прийняття рішень (MDP/POMDP), транслюються в низькорівневі дії та виконуються. Результат виконаних дій повертається високорівневому планувальнику для оновлення плану. Оцінки авторів показали, що такий агент здатний розпізнати 65-78% вразливостей у відкритих сервісах середніх корпоративних мереж.

## 1.3 Аналіз проблем надійності та галюцинацій великих мовних моделей у задачах послідовного прийняття рішень

Основною проблемою впровадження мовних моделей у задачах планування є відсутність граматичних обмежень згенерованих планів. Як зазначається у [1], у деяких ситуаціях це призводить до генерації планів, які неможливо виконати через неоднозначні формулювання в тексті плану, а також через пропозиції дій,

які неможливі в конкретному середовищі. Для рішення цієї проблеми автори [1] пропонують використовувати додаткову мовну модель у якості посередника, який перетворював би згенерований план у послідовність дій, допустимих у середовищі.

Підхід ReAct [63] вирішує проблему галюцинацій в методі Chain-of-Thought. ReAct послідовно чергує дії та міркування. На відміну від навчання з підкріпленням, ReAct не потребує оновлення параметрів моделі, має вищу адаптивність до нових задач і середовищ. У кожний момент агент може виконати дію в середовищі або згенерувати міркування, яке доповнить контекст агента. Таким чином, агент отримує пам’ять і потенційну здатність аналізувати дії та їхні наслідки. Це покращує стратегії, яким слідує агент і зменшує ймовірність появи галюцинацій. Також завдяки представленню міркувань природною мовою така система є інтерпретованою для людини.

У роботі [64] частково вирішується проблема пропущених кроків, яка виникала в підході Zero-shot-CoT, який додавав до промпту фразу «Let’s think step by step». У запропонованому підході модель має виконати декомпозицію плану вирішення завдання і тільки потім його виконати. Модифікований промпт також інструктує модель виконувати перевірку проміжних суджень або проміжних обчислень у математичних задачах. Цей метод працює без надання моделі прикладів, але за ефективністю порівняний з методами, в яких моделям надавали 8 прикладів. Автори припускають, що такий підхід може бути ефективним не тільки для математичних задач, але й символьних і логічних завдань. Проте, як зазначають автори, цей підхід не вирішує проблему нездатності моделі опрацювати семантику задачі, що виникає через нездатність мовної моделі повністю «зрозуміти» умови задачі або встановити правильні зв’язки між елементами задачі.

Для задач, опис яких перевищує розмір контекстного вікна LLM, доцільно використати підхід Recursion-of-Thought [65]. У цьому підході використовується додатковий спеціальний токен, який виділяє з задачі частину, вирішення якої можна ізолювати. Далі відбувається виконання цієї підзадачі в окремому контексті та повернення короткого результату її вирішення в основний контекст. Таким чином, задача вирішується поетапно без втрати проміжних результатів і переповнення контекстного вікна LLM.

У роботі [66] продемонстровано ідею чергування простих питань і

відповідей на них. Послідовна відповідь на такі питання формує розв'язок завдання. У даному підході декомпозиція задачі на окремі питання відбувається окремо від формування відповідей на ці питання, дозволяючи окремо впливати на обидва процеси. При цьому також постійно оновлюється контекст мовної моделі, що зменшує кількість помилок у завданнях, які потребують значну кількість дій.

Нейромережі не однаково добре підходять для різних задач. Наприклад, точність вирішення арифметичних задач сильно знижується при додаванні розрядів чисел, над якими робляться алгебраїчні операції. Для подолання цієї проблеми в роботі [67] автори запропонували модульну систему, в якій нейронний маршрутизатор аналізує задачу та обирає інструмент для її рішення, наприклад, калькулятор для арифметичних задач, велику мовну модель для задач, які потребують аналіз природної мови, або виклик інших API, якщо такі можуть бути корисними для рішення поточної задачі. Це один із прикладів модульної системи, в якій прийняття рішень і їхня реалізація розділені.

У роботі [68] показано, що сократівські моделі — модульні агенти, які складаються з мовної моделі та окремої великої зорової моделі — здатні без донавчання вирішувати задачі опису зображень і відео на рівні та краще за моделі, спеціально навчені під конкретні задачі. При цьому ресурси на донавчання моделей не витрачаються, результат досягається за рахунок взаємодії моделей — обміну текстовою інформацією, яку вони генерують.

У роботі [69] запропоновано метод промптингу Algorithm of Thoughts (AoT), який дозволяє моделі проводити динамічний процес пошуку потенційних розв'язків задачі, оцінювати їхню перспективність і проводити пошук серед них. До промпту мовної моделі додавалися загальні вказівки щодо логіки її поведінки, а також приклади процесу розв'язання задачі людьми, завдяки яким модель формувала внутрішню евристику. Автори стверджують, що Algorithm of Thoughts потребує менше запитів і токенів порівняно з Tree of Thoughts [70]. Tree of Thoughts потребує наявності зовнішнього алгоритму оцінювання потенційного розв'язку. На відміну від Tree of Thoughts, Algorithm of Thoughts фокусується на формуванні внутрішньої евристики, яка заміняє зовнішній алгоритм, що дозволяє проводити пошук рішення в рамках одного запиту. Також автори Algorithm of Thoughts теоретично показують здатність методу вирішувати складні комбінаторні задачі (NP), якщо використати достатньо

велику кількість токенів.

У 2022 році мовні моделі GPT-3, Instruct-GPT3 and BLOOM було застосовано до вирішення логічних задач, в яких були задані обмеження [71]. Рішення приймалося у формі плану дій у заданих ситуаціях. У якості промпта подавався опис завдання і приклад його виконання. Дослідження показало, що задачі, які люди вирішували в 78% успішно, використані мовні моделі мали успішність на рівні 5% та нижче. Також було показано, що досліджені мовні моделі показували низьку здатність генерувати, узагальнювати, модифікувати та перевикористовувати плани.

Автори роботи [72] показали для моделі GPT-3.5-Turbo, що зміна промпту (у статті згадується додавання до промпту точної вимоги до завдання) може призводити до значніших покращень, ніж метод Self-Refine. На думку авторів, успіх Self-Refine в умовах субоптимального початкового запиту пояснюється не внутрішньою логічною корекцією моделі, а наявністю додаткових специфікацій завдання у зворотньому повідомленні. Застосування підхіду Self-Refine до якісного промпта призводило до погіршення точності. Автори показали, що після отримання зворотнього зв’язку мовні моделі GPT-3.5, GPT-4, GPT-4-Turbo, Llama-2 часто не змінюють попередню відповідь. Серед випадків, у яких мовні моделі змінюють відповідь, зміна неправильної відповіді на правильну стається рідше, ніж сума ймовірностей змін неправильної відповіді на іншу неправильну та правильної на неправильну. Автори пов’язують це явище з тим, що мовні моделі ймовірно не здатні оцінити правильність своїх суджень.

Автори статті [7] протестували здатність великих мовних моделей GPT-5-Thinking, Gemini-2.5-Pro, GPT-5-mini і Llama 3.1 8B-Instruct вирішувати логічні задачі «8-puzzle» [60]. Автори показали, що з промптом zero-shot, який містить правила гри «8-puzzle», постановку задачі та формат відповіді, всі мовні моделі успішно вирішували відносно малий відсоток задач (найкращий результат показала GPT-5-Thinking — 12%, інші моделі показали гірші результати). Дослідження показало, що мовні моделі не були здатні коректно обробляти інформацію про зміни в середовищі, через що пропонували неможливі для певних станів дії. Спроба ізолювати навички планування від здатності обробляти інформацію про середовище за рахунок зовнішнього компонента призвело до того, що жодна модель не впоралася з жодним

завданням. Надання зворотнього зв’язку підвищило кількість успішно розв’язаних завдань для всіх моделей, крім Llama 3.1 8B-Instruct. Найкращий результат зі зворотнім зв’язком показала GPT-5-Thinking — 68% успішних розв’язків. Цікавим спостереженням є той факт, що використання CoT і AoT у експериментах без зворотнього зв’язку підвищувало успішність GPT-5-Thinking (успішність з CoT становила 22%, з AoT — 30%, у той час як zero-shot забезпечував успішність 12%); проте успішність всіх інших моделей знижувалася. При використанні CoT і AoT у GPT-5-mini домінуючою помилкою була так звана «graceful failure»: модель відмовлялася пропонувати розв’язок або просила уточнень. Автори припускають, що це свідчить про відсутність універсального методу промптингу і необхідність адаптації промптів до кожної окремої мовної моделі.

У роботі [73] описано проведені експерименти для виявлення обмежень мовних моделей з кількістю параметрів, яка не перевищує 70 мільярдів: Llama 3.1 Instruct, Llama 3.2 Instruct, Gemma-3 Instruct, OLMo 7B Base, OLMo 7B Instruct, OLMo-2 Instruct. Дослідження показує, що здатність цих великих мовних моделей до розрізнення істинних та хибних тверджень суттєво деградує при відхиленні вхідних даних від розподілу тренувальних даних. Автори стверджують, що внутрішні представлення знань у LLM є поверхневими: імовірно мовні моделі значною мірою покладаються на лексичну та орфографічну схожість з навчальним корпусом, а не на стабільні концептуальні структури. Незначні зміни у вхідних даних, друкарські помилки, зміна пунктуації, переформулювання, зміна порядку слів і переклад знижували здатність мовних моделей відрізняти хибні твердження від істинних.

## 1.4 Аналіз методів інтеграції символьних обмежень та пам’яті в архітектуру агентів

В роботі [74] агент Expel вирішує з набором завдань, збирає історію вдалих та невдалих спроб вирішити завдання, після чого шляхом рефлексії формує список правил на основі пар вдалих та невдалих спроб. Отримані правила агент використовує у наступних спробах вирішити завдання.

В роботі [75] запропоновано механізм пам’яті Memory Bank для чат-ботів, який структурує пам’ять ієрархічно, зберігає повну історію

діалогів, узагальнену інформацію про події та опис психологічного портрету користувача. Узагальнення інформації відбувається з періодичністю в декілька епізодів. Ключовою особливістю є динамічний механізм очищення пам’яті заснований на кривій забування Еббінгауза. Підхід MemoryBank головним чином орієнтований на завдання соціальної взаємодії, наприклад психологічне консультування.

Архітектура генеративних агентів (Generative Agents) була запропонована в роботі [11]. Фундаментальним компонентом такої архітектури є «потік пам’яті» (memory stream) — база даних, яка зберігає вичерпну історію досвіду кожного агента у вигляді природної мови. На відміну від підходів, які поміщають досвід агента лише у контекстне вікно LLM, підхід Genenrative Agents використовує механізм динамічної вибірки, який відбирає релевантні записи для формування контексту для поточного кроку. Механізм вибірки враховує такі властивості записів, як недавність, важливість та релевантність. Підхід включає механізм рефлексії, який узагальнює низькорівневі спостереження у високорівневі абстрактні висновки, які зберігаються до потоку пам’яті разом з низькорівневими спостереженнями у деревовидну структуру.

Поняття вербального навчання з підкріпленням вводиться в роботі [12]. У цьому підході стратегія агента параметризується не лише вагами моделі, а й станом пам’яті агента. При цьому пам’ять містить рефлексії з попередніх епізодів. Процес роботи агента організований як цикл, в якому агент генерує траєкторію, яка оцінюється моделлю-оцінювачем. Агент отримує бінарний або текстовий зворотній зв’язок, проводить рефлексію і зберігає результуючий вербальний зворотній зв’язок в довготривалій пам’яті, що дозволяє змінити поведінку в наступному епізоді. Автори вказують, що для покращення підходу варто використовувати складніші структури пам’яті, такі, які векторні або SQL бази даних.

LLM часто не здатні згенерувати оптимальний вихідний результат з першої спроби, для вирішення цієї проблеми було запропонвано підхід Self-Refine [25], який базується на ідеї ітеративного зворотного зв’язку та уточнення. Ідея цього підходу полягає у використанні LLM в різних ролях: генератора, критика (feedback provider) та рефайнера (refiner). Агент працює в циклі, на кожній ітерації генерується початковий розв’язок завдання та генерується зворотній зв’язок щодо рішення. На основі зворотного зв’язку агент уточнює попередній

результат. Self-Refine доводить ефективність LLM для генерації зворотного зв’язку та покращення власних відповідей. Цей підхід продемонстрував значне покращення без необхідності донавчання моделі. Недоліком підходу Self-Refine є те, що зворотній зв’язок використовується для покращення поточного виходу, але не зберігається в довгостроковій пам’яті.

Підхід, запропонований в роботі [13], ітеративно аналізує історію попередніх траєкторій агента та його переконання (beliefs), змінюючи переконання для покращення стратегії. Замість накопичення епізодичної пам’яті, цей підхід проводить дистиляцію досвіду у поведінкові настанови (behevioral guidelines) та виконує моделювання світу. Це дозволяє агенту формувати покращену стратегію, яка включає правила, які були узагальнені самим агентом.

Автори роботи [76] пропонують підхід MemGPT — самокерованого редагування пам’яті, який імітує необмежену довжину контексту шляхом динамічного переміщення даних між контекстом LLM та зовнішнім сховищем. У цьому підході LLM не лише використовує та генерує текст, а і виступає контролером,який управляє пам’яттю. MemGPT включає набір функцій, які дозволяють агенту додавати, змінювати або шукати інформацію у зовнішьому сховищі на основі поточного стану.

Ранні підходи до організації пам’яті, такі, як MemoryBank [75] та Generative Agents [11] використовують фіксовані схеми зберігання пам’яті або просте накопичення історії, що обмежує здатність агента до узагальнення знань у нових середовищах. Більш динамічний та гнучкий підхід A-MEM застосовано в роботі [77]. Автори будують пам’ять агента не як пасивне сховище, а як активну систему, яка самоорганізується за принципом методу організації нотаток Zettelkasten. Ключовою особливістю A-MEM є відмова від фіксованих схем на користь генеративного підходу, в якому LLM самостійно структурує зв’язки між «нотатками» (атомарними одиницями досвіду). Попри гнучкість, підхід A-MEM має суттєві недоліки: висока вартість запису досвіду, відсутність механізму забування, чутливість до значень гіперпараметрів.

У роботі [8] була використана детермінована маска, яка забороняла використання синтаксично недопустимих токенів у системі, яка вирішувала задачу написання програмного коду. Використання цієї маски суттєво покращило ефективність розв’язання задачі.

В роботі [78] розглядається підхід Learning for Reasoning, який передбачає взаємодію нейронної та символьної компонент таким чином, що вони безперервно доповнюють одна одну. Нейронний компонент перетворює неструктуровані дані від середовища в символьне представлення. Символьний компонент оперує структурованими знаннями для формування логічних обмежень для нейромережі.

У DeepPath [79] використовується градієнт стратенії для знаходження найбільш інформативних шляхів у графах знань. У MINERVA [80] додано Long Short-Term Memory для збереження пройденого шляху.

У роботі [81] для системи дворівневого планування запропоновано алгоритм «винайдення» предикатів (predicate invention). У цій статті предикати використовуються як спосіб дискретного опису неперервного світу. Як цільовий, так і поточні стани описуються предикатами. Можливі переходи між станами задаються операторами. Планом у такій системі є послідовність застосування операторів до відповідних станів: з початкового до цільового. Механізм винайдення предикатів допомагає згенерувати необхідні для виконання завдання предикати автоматично. Запропоновані предикати передаються планувальнику для формування плану. На основі експертних демонстрацій алгоритм оцінює ймовірність успішного перетворення плану в послідовність низькорівневих дій.

## 1.5 Постановка задач дисертаційного дослідження

Огляд та аналіз сучасних наукових джерел дозволили сформувати детальне уявлення про стан галузі, проблем та рішень. Розглянуті джерела підтверджують актуальність досліджень методів контекстного керування процесами генерації LLM та процесами прийняття рішень в LLM-агентах. Значний комбінаторний простір можливих контекстів LLM створює складну пошукову проблему, яка наразі вирішується шляхом створення схем та шаблонів, які звужують простір пошуку та аналізу рішень. Проблема галюцинацій в процесі генерації текстових послідовностей з використанням LLM виникла на етапі появи перших LLM та, станом на початок 2026 року, досі залишається відкритим викликом, який значно знижує надійність систем на основі LLM. Активним та важливим напрямком досліджень, який покликаний знизити негативний вплив галюцинацій на якість роботи систем, є підхід

ієрархічної організації, в якому LLM виконує високорівневе планування, а за безпосереднє виконання дій відповідає класичний процедурний код. Однією з найбільш активних галузей є вивчення моделей пам’яті та методів керування пам’яттю в LLM та LLM-агентах. У ході аналізу літератури сформульовані такі кроки для розв’язання знайдених проблем:

1. Провести системний аналіз сучасного стану, проблем та методів керування інтелектуальними агентами, що використовують великі мовні моделі в задачах планування та навігації.

2. Виконати проєктування архітектури ієрархічного інтелектуального агента, який поєднує модуль планування абстрактних дій на основі LLM та модуль виконання низькорівневих дій у середовищі.

3. Провести дослідження підходів формування контексту для ієрархічного LLM-агента, зокрема для використання в середовищі Minigrid. Виконати експериментальні дослідження впливу мовних інструкцій на ефективність ієрархчного LLM-агента в середовищі Minigrid.

4. Розробити модель інтеграції декодування з обмеженням граматики в процес планування абстрактних дій агентом з використанням LLM. Такий підхід дозволить зменшити кількість синтаксичних галюцинацій та забезпечити структурну цілісність команд.

5. Розробити модель та метод керування рефлексивною пам’яттю, що дозволяє узагальнювати досвід агента у вигляді правил та динамічно коригувати його поведінку без необхідності зміни ваг мовної моделі.

6. Виконати програмну реалізацію розроблених архітектури, методів та моделей у вигляді інтегрованого комплексу для моделювання інтелектуальних агентів у віртуальному середовищі на базі Minigrid.

7. Провести експериментальні дослідження для валідації розроблених рішень, оцінки ефективності агента з різними конфігураціями пам’яті та виконання порівняльного аналізу з обраними базовими підходами.

Отримані результати становлять практичну цінність для створення систем навігації у складних середовищах, в яких інструкції формулюються

природною мовою, та необхідно проводити аналіз та міркування над обширним лінгвістичним контекстом зі застосуванням LLM. Результати можуть бути застосовані в автоматизованих або автономних роботизованих системах, в яких, зокрема, необхідна участь та контроль з боку людини, а інтерфейс взаємодії з оператором реалізовано природною мовою. Ієрархічна архітектура агента дозволяє об’єднати сильні сторони LLM, які полягають в можливості виконання довільних завдань, сформульованих природною мовою, та переваги процедурних низькорівневих стратегій, які можуть бути детермінованими та мати ретельно перевірену та прогнозовану поведінку.

# 2 Дослідження методів контекстного керування ієрархічними інтелектуальними агентами в середовищах з частковою спостережуваністю

Даний розділ присвячений дослідженню впливу мовних інструкцій на якість роботи агента, який використовує LLM як модуля планування дій. Ми відштовхуємося від результатів [5], але використовуємо різні промпти та досліджуємо їх вплив на числові показники якості роботи агента.

Ми розглядаємо агента, який функціонує у віртуальному середовищі, здатний отримувати спостереження, обирати та виконувати дії в середовищі. Агент може бути представлений як система з трьох компонентів: Планувальник, Актор та Адаптер [82]. Така структура дозволяє модифікувати кожен з компонентів окремо від інших, спрощуючи застосування до нових середовищ.

У такому агенті мовна модель залишається незмінною, змінюються мовні інструкції, які є специфічними для конкретного середовища та задачі. Мовні моделі здатні аналізувати спостереження, отримані як зворотний зв’язок від середовища у відповідь на дії, обрані мовною моделлю. У роботі [83] продемонстровано, як використання мовних моделей, які генерують внутрішній монолог, покращує функціонування агента, який керує роботизованою платформою у фізичному середовищі.

Побудова плану з використанням мовної моделі може бути обчислювально затратною процедурою. В найгіршому випадку потрібно створювати новий план на кожному кроці. У такому випадку, незалежно від кількості кроків у плані, буде виконано лише одну дію. Для оптимізації обчислювальних затрат план має містити декілька кроків, які послідовно виконуються в середовищі без перебудови плану. Щоб агент був здатний адаптуватися до особливостей конкретного середовища, можна додати додатковий класифікатор, який на кожній ітерації взаємодії з середовищем визначатиме, чи потрібно згенерувати новий план [5].

## 2.1 Математична формалізація задачі навігації як частково спостережуваного марковського процесу прийняття рішень (POMDP)

У [18] марковський процес прийняття рішень представлено як кортеж $\langle S, A, T, R \rangle$, де S — скінченна множина можливих станів середовища, A — скінченна множина доступних агенту дій, $T : S \times A \to P(S)$, $T$ — функція переходу, яка задає для кожної пари стану та дії розподіл ймовірностей переходу у всі можливі стани середовища. Ймовірність перейти в деякий наступний стан $s'$ можна записати як $T(s, a, s')$, де $s$ — початковий стан середовища, $a$ — дія, яку обрав агент. $R : S \times A \to \mathbb{R}$ — функція винагороди, яка визначає очікувану нагроду, яку отримає агент, якщо обере дію $a$ в стані середовища $s$. Множина станів середовища та доступних дій можуть бути нескінченними.

В марковському процесі прийняття рішень наступний стан середовища та очікувана винагорода залежать лише від попереднього стану і обраної дії, але не залежать від більш ранніх станів. Ця властивість моделі називається марковською, що означає, що стан та винагорода в момент часу $t + 1$ залежать лише від стану і обраної дії в момент часу $t$.

Навчання з підкріпленням об'єднує в собі три пов'язані концепції [17]: задачу навчання з підкріпленням, клас методів вирішення цієї задачі, галузь досліджень задачі та методів її розв'язання. Автори [17] вважають, що задача навчання з підкріпленням полягає у пошуку стратегії прийняття рішень, яка максимізує кумулятивний сигнал винагороди, отриманий через взаємодію агента з середовищем. Агент здатний сприймати стан середовища через спостереження, може обирати та виконувати дії, які змінюють стан середовища, і має певну ціль, яка виражається через функцію винагороди [17].

Агент знаходиться в одному з можливих станів $s \in S$ середовища. Взаємодія агента з середовищем відбувається ітеративно, у вигляді послідовних кроків. На кожному кроці агент може обрати дію $a \in A$ з множини доступних дій. Під впливом обраної дії середовище переходить з поточного стану $s_t$ в наступний стан $s_{t+1}$, множина можливих переходів визначається функцією еволюції середовища, яка може бути стохастичною. У відповідь на обрану дію та зміну стану, середовище повертає винагороду $r \in R$.

Вирішення завдання оптимізації прийняття рішень агентом полягає у максимізації деякої величини, що розраховується на основі послідовностей нагород, які отримує агент в процесі взаємодії з середовищем. У випадку, якщо агент може виконувати нескінченно довгі послідовності дій, використовується знецінення з нескінченним горизонтом:

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right], \tag{2.1}$$

де $0 < \gamma < 1$ — фактор знецінювання винагороди.

В цій моделі для агента є більш важливими винагороди які він отримує раніше. Фактор знецінювання гарантує, що сумарна винагорода буде скінченним значенням. Що менше значення $\gamma$, то більший пріоритет агент надаватиме діям, які ведуть до найшвидшого отримання винагороди.

Стан $s$ представляє вичерпний опис властивостей середовища, вся інформація про середовище міститься в стані. Часто, агенту недоступний стан безпосередньо, натомість агент сприймає середовище через спостереження $o_{env,t}$. Спостереження — це частковий опис поточного стану середовища, який містить не повну інформацію.

Специфіка досліджуваного середовища полягає в тому, що агент не має доступу до повної інформації про стан $s_t$ у момент часу $t$. Область зору агента обмежена областю $3 \times 3$ клітинки. На початку епізоду все середовище приховано, але по мірі дослідження агенту відкриваються клітинки, яких він не бачив раніше.

Враховуючи обмежену видимість, задача формулюється як частково спостережуваний марковський процес прийняття рішень POMDP. В задачах, де агент не може з впевненістю визначити поточний стан середовища $s$ застосовується модель POMDP (Partially Observalbe Markov Decision Process). У [18] POMDP представлено як кортеж $\langle S, A, T, R, \Omega, O\rangle$, де $S, A, T, R$ описують ті ж самі множини, що і в MDP, $\Omega$ — скінченна множина спостережень середовища, які отримує агент замість станів, $O : S \times A \to P(\Omega)$ — функція спостереження, яка для кожної пари стану та дії визначає розподіл ймовірностей отримати спостереження $o$. Інакше функцію спостережень можна записати як $O(s', a, o)$, де $s'$ — наступний стан середовища, $o$ — спостереження. Завданням агента залишається максимізація математичного

очікування знецінених майбутніх винагород 2.1.

У контексті даної роботи простір спостережень $\Omega$ представлений тензором, який кодує символьне представлення видимих об'єктів (ідентифікатор об'єкта, колір, стан) та відносну орієнтацію агента. Для часткової компенсації неповної спостережуваності середовища, використовується механізм історичної маски видимості, в якому клітинки, які агент побачив хоча б раз, позначаються як відкриті у внутрішньому представленні до кінця епізоду.

Простір дій $A$ у досліджуваному середовищі є дискретним і складається з набору примітивних команд:

$$A = \{\text{left}, \text{right}, \text{forward}, \text{pickup}, \text{drop}, \text{toggle}, \text{done}\}, \tag{2.2}$$

де дії кодуються числами від 0 до 6 відповідно.

Ключовим елементом навчання є функція винагороди $R$, яка у даному середовищі є розрідженою та модифікованою для стимулювання найшвидшого виконання завдання. Винагорода за епізод $R_{\text{ep}}$ розраховується таким чином:

$$R_{ep} = \begin{cases} 1 - 0.9 \cdot \frac{N_{steps}}{N_{max}}, & \text{якщо ціль досягнута;} \\ 0, & \text{в іншому випадку;} \end{cases} \tag{2.3}$$

де $N_{steps}$ — кількість кроків, витрачених агентом, $N_{max}$ — максимальна дозволена тривалість епізоду (за замовчуванням 100 кроків).

Слід зауважити, що через наявність мінімально необхідної кількості кроків для вирішення задачі (для середовища SimpleDoorKey цей поріг становить близько 21 кроку), максимальна теоретично можлива винагорода обмежена зверху значенням $\approx 0.811$.

Окрім основної винагороди середовища, вводиться штраф за використання обчислювальних ресурсів LLM. Повна функція винагороди на кроці $t$ має вигляд:

$$r_t = r_{\text{env}}^{(t)} - \lambda_{\text{comm}} \cdot \mathbb{1}(\text{call\_LLM}_t), \tag{2.4}$$

де $\mathbb{1}(\cdot)$ — індикаторна функція, яка приймає значення 1 при зверненні до LLM для генерації нового плану, і 0 в іншому випадку, $\lambda_{\text{comm}}$ — коефіцієнт штрафу (було використано значення $\lambda_{\text{comm}} = 0.01$). Така структура винагороди спонукає агента мінімізувати кількість звернень до планувальника, покладаючись на

власну пам'ять та поточний план дій.

Класичні моделі прийняття рішень MDP та POMDP базуються на припущенні, що кожна дія агента $a \in A$ виконується за один неподільний крок часу $t$, після чого середовище синхронно переходить у новий стан $s_{t+1}$. Проте в ієрархічних архітектурах, де поведінка агента керується планом, згенерованим мовною моделлю, Планувальник генерує не окремі низькорівневі дії, а абстрактні макро-дії (плани), виконання яких потребує змінної кількості часових кроків. Для математично строгого опису таких систем застосовується формалізм напівмарковських процесів прийняття рішень (Semi-Markov Decision Process, Semi-MDP) у поєднанні з фреймворком опцій [2].

Стратегія $\pi$ — це функція, яка відображає стан середовища на ймовірністий розподіл над допустимими діями.

Математичне поняття опції формалізує високорівневу дію, яка розгортається у часі над базовим марковським процесом прийняття рішень. У просторі станів $S$ та просторі примітивних дій $A$ опція $\omega \in \Lambda$ визначається як кортеж з трьох компонентів [2, 5]:

$$\omega = \langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle, \tag{2.5}$$

де $\mathcal{I}_\omega \subseteq S$ — множина початкових станів, яка визначає підмножину станів, у яких опція $\omega$ може бути активована, $\pi_\omega : S \times A \to [0, 1]$ — внутрішня стратегія, яка задає ймовірнісний розподіл вибору примітивних дій $a \in A$ на кожному кроці, поки опція $\omega$ активна, $\beta_\omega$ — умова завершення (termination condition), яка визначає ймовірність того, що виконання опції $\omega$ перерветься або завершиться у стані $s$.

За аналогією з агентом в роботі [5], архітектура досліджуваного в цій роботі агента складається з Планувальника, Актора та Медіатора (аналог модуля Адаптер з роботи [82]) та використовує фреймворк опцій. Модуль Планувальник використовує LLM та функціонує як глобальна стратегія над опціями $\mu : S \to \Lambda^*$. Планувальник генерує текстовий план, який містить послідовність опцій. Для забезпечення коректної роботи Планувальника Медіатор виконує функцію перетворення даних між представленнями, специфічними для середовища, та текстовим представленням для планувальника. Компонент Актор імплементує внутрішню стратегію $\pi_\omega$,

послідовно інтерпретуючи та виконуючи команди $a \in A$ із кожної опції $\omega_i$ плану.

Навчання комунікаційної стратегії за допомогою алгоритму PPO [84] необхідне для оптимізації функції завершення $\beta_\omega$. Комунікаційна стратегія є бінарним класифікатором, що вирішує, коли поточний план виконано або коли необхідно перервати його та згенерувати новий.

Зауважимо, що рівняння Беллмана для очікуваної сумарної винагороди у фреймворку Semi-MDP враховує тривалість виконання макро-дії. Якщо опція $\omega$ була ініційована у стані $s$ і тривала $\tau$ кроків до свого завершення у стані $s'$, функція цінності стану $V^\pi(s)$ визначається як:

$$V^\mu(s) = \mathbb{E}\left[r(s,\omega) + \gamma^\tau V^\mu(s')\right], \tag{2.6}$$

де $r(s,\omega) = \mathbb{E}\left[\sum_{k=0}^{\tau-1} \gamma^k r_{t+k}\right]$ — кумулятивна знецінена винагорода, отримана за час виконання опції, а фактор знецінювання $\gamma^\tau$ застосовується до стану завершення $s'$, враховуючи змінний часовий інтервал $\tau$.

Такий формалізм дозволяє довести, що оптимізація комунікаційної стратегії, яка штрафується за виклик Планувальника (відповідно до модифікованої функції винагороди 2.4 ), є математично коректною оптимізацією функції завершення $\beta_\omega$ у просторі напівмарковських процесів.

## 2.2 Архітектура ієрархічного агента: адаптація підходу «Planner-Actor-Mediator»

Агент — це частина контрольованої системи, яка приймає рішення, використовуючи спостереження та винагороди. Агент може взаємодіяти з середовищем для досягнення власних цілей.

Як архітектурну основу для побудови ієрархічного агента в даній роботі використано підхід, запропонований у [5]. Агент складається з трьох компонентів: Планувальник (Planner), Актор (Actor), Медіатор (Mediator). Підхід Planner-Mediator-Actor довів свою ефективність у задачах планування та навігації, проте оригінальне дослідження фокусувалося переважно на навчанні комунікаційної стратегії, залишаючи поза увагою вплив семантичних інструкцій на планувальника.

Компонент Планувальник включає попередньо-навчену мовну модель та

виконує завдання генерації плану абстрактних дій на основі спостереження стану середовища. Цей компонент приймає на вхід та повертає на виході дані в текстовому представленні.

$$p_t = f_{LLM}(\rho(o_t^{(text)}, h_{t-1})), \tag{2.7}$$

де $f_{LLM}$ — функція генерації тексту великою мовною моделлю, $\rho$ — функція формування промпту, $o_t^{(text)}$ — поточне текстове спостереження, $h_{t-1}$ — буфер епізодичної пам'яті.

Для перетворення даних між текстовим представленням та представленням специфічним для спостережень, у нашому випадку — тензорів, та дій використовується компонент Медіатор. Медіатор отримує на вхід спостереження середовища у вигляді тензора $o_{env} \in \Omega$, та перетворює спостереження в текстове представлення $o_t^{(text)} \in \mathcal{T}$ за допомогою функції $f_{Mediator}$:

$$o_t^{(text)} = f_{Mediator}(o_{env}). \tag{2.8}$$

У середовищі Minigrid спостереження, яке отримує агент в момент $t$ (індекс $t$ опускаємо), представлено у вигляді тривимірного тензора $\mathbf{O} \in \mathbb{N}^{W \times H \times C}$, де $W$ та $H$ — ширина та висота просторової сітки двовимірного середовища, а $C$ — кількість каналів ознак для кожної клітинки. У середовищі Minigrid можна виокремити канали, які містять інформацію про окремі властивості об'єктів, з використанням операції слайсингу тензора спостережень по осі каналів ознак за індексом $k$ по третій осі:

$$\mathbf{M}^{(obj)} = \mathbf{O}_{:,:,k}. \tag{2.9}$$

Матриця типів об'єктів $\mathbf{M}^{(obj)}$, яка міститься у нульовому каналі тензора спостережень:

$$\mathbf{M}^{(obj)} = \mathbf{O}_{:,:,0}, \tag{2.10}$$

де нижні індекси у тензорі позначають індекс «:» позначає взяття всіх каналів по відповідній осі.

Відповідно матриця кольору об'єктів $\mathbf{M}^{(color)}$ міститься в першому каналі спостереження $\mathbf{O}_{:,:,1}$. Матриця стану об'єктів $\mathbf{M}^{(state)}$ — у другому та матриця

повороту агента $\mathbf{M}^{(agent)}$ — в третьому каналі стану.

Позиція агента у локальній системі координат $c_a = (i_a, j_a)$ визначається як координата, де значення елементу матриці стану агента $\mathbf{M}^{(agent)}$ відрізняється від константи порожнього простору $C_{\text{empty}}^{(agent)}$ (за замовчуванням $C_{\text{empty}}^{(agent)} = 4$):

$$\mathbf{M}^{(agent)} = \mathbf{O}_{:,:,3}; \tag{2.11}$$

$$c_a = \arg_{(i,j)}(\mathbf{M}_{i,j}^{(agent)} \neq C_{\text{empty}}^{(agent)}). \tag{2.12}$$

Для формування текстового опису необхідно виділити координати цільових об'єктів. Визначимо множину просторових координат ключів $K$ та множину координат дверей $D$ через відповідні числові ідентифікатори типу об'єкту (де $C_{\text{key}} = 5$ позначає ключ, а $C_{\text{door}} = 4$ — двері):

$$K = \{(i, j) \mid \mathbf{M}_{i,j}^{(obj)} = C_{\text{key}}\}; \tag{2.13}$$

$$D = \{(i, j) \mid \mathbf{M}_{i,j}^{(obj)} = C_{\text{door}}\}. \tag{2.14}$$

Стан володіння об'єктом (перебування об'єкта в інвентарі агента) визначається збігом просторових координат об'єкта та локальної позиції агента. Таким чином, ключ вважається таким, що переноситься агентом, якщо його координати еквівалентні $c_a$.

Розділимо загальну множину ідентифікованих ключів $K$ на дві підмножини: множину ключів в інвентарі $K_{carry}$ та множину ключів, що спостерігаються у середовищі $K_{obs}$:

$$K_{carry} = K \cap \{c_a\}; \tag{2.15}$$

$$K_{obs} = K \setminus K_{carry}. \tag{2.16}$$

Визначимо функцію лінгвістичного відображення $T^{(ling)} : \mathcal{O} \to \mathcal{L}$, де $\mathcal{L}$ — простір текстових послідовностей, які описують відповідний стан середовища. Функція $T^{(ling)}$ генерує фінальний лінгвістичний опис $L \in \mathcal{L}$ шляхом конкатенації (послідовного з'єднання) елементів згенерованої множини семантичних висловлювань $S^{(sem)}$:

$$L = \text{«observation: \{»} \oplus \text{Join}(S, \text{«, »}) \oplus \text{«\} »}, \tag{2.17}$$

де оператор Join об'єднує елементи множини $S^{(sem)}$ через кому з пробілом, а сама множина $S^{(sem)}$ формується за такими композиційними правилами: для кожного елемента простору $(i, j) \in K_{obs}$ до множини $S^{(sem)}$ додається висловлювання «observed a key». Для кожного елемента простору $(i, j) \in D$ до множини $S^{(sem)}$ додається висловлювання «observed a door». Якщо потужність множини $|K_{carry}| > 0$, до $S^{(sem)}$ додається висловлювання «carrying the key». У випадку, якщо $K_{obs} \cup D \cup K_{carry} = \emptyset$ (тобто $S^{(sem)} = \emptyset$), множина ініціалізується єдиним базовим висловлюванням: $S^{(sem)} = \{\text{«observed nothing»}\}$.

Таким чином, наведена математична модель однозначно визначає перетворення тензора спостережень середовища у структурований текстовий опис, придатний для подачі до Планувальника для створення наступного плану. Створення нового текстового спостереження $L$ відбувається при кожному виклику Планувальника.

Для визначення синтаксису текстового спостереження, яке передається до модуля Планувальник, структуру текстової послідовності доцільно описати за допомогою нотації Бекуса-Наура (BNF — Backus-Naur Form). BNF є метасинтаксичною нотацією — формальною системою правил, яка використовується для точного математичного опису синтаксису формальних мов.

У наведеній нижче граматиці нетермінальні символи (змінні, які позначають синтаксичні категорії та можуть бути розкриті далі) взяті в кутові дужки $\langle \dots \rangle$, тоді як термінальні символи (константні текстові рядки, які є кінцевими елементами виходу) подані у подвійних лапках. Символ $::=$ означає «визначається як», а вертикальна риска $|$ позначає логічне «або» (альтернативу).

BNF граматика, яка описує результат перетворення Медіатором спостереження середовища в текстове представлення:

Лістинг 1: Граматика формату текстових спостережень середовища Minigrid SimpleDoorKey

```
<observation-string> ::= "observation: {" <state-description> "} "

<state-description> ::= <base-context>
```

```
    | <base-context> ", " <carry-state>

<base-context> ::= "observed nothing"
    | <observation-list>

<observation-list> ::= <observation-item>
    | <observation-item> ", " <observation-list>

<observation-item> ::= "observed a " <object>

<object> ::= "key" | "door"

<carry-state> ::= "carrying the " <carry-object>

<carry-object> ::= "key"
```

На початкових етапах дослідження було використано евристичний алгоритм синтаксичного розбору текстового плану згенерованого LLM. Цей підхід слугує базовою версією, яка передує впровадженню більш стійкого методу з використанням генерації з обмеженням граматики, які розглядаються в розділах 3 та 4.

Процес перетворення текстового виходу планувальника у виконуваний план дій для середовища математично описується як композиція послідовних відображень.

Нехай $\mathcal{U}$ — простір неструктурованих текстових висловлювань, згенерованих високорівневим планувальником з використанням LLM. Нехай $\Pi$ — простір формальних планів агента. План $\phi \in \Pi$ визначається як впорядкована множина (кортеж) параметризованих опцій $\phi = (\theta_1^{(\psi)}, \theta_2^{(\psi)}, \ldots, \theta_n^{(\psi)})$. Кожна опція $\theta_k^{(\psi)}$ розглядається як структурований вектор $\theta_k^{(\psi)} = (\psi, \xi, c)$, де $\psi \in \Psi$ — ідентифікатор опції (наприклад, $EXPLORE$ для дослідження або $PICKUP$ для підняття об'єкта), $\xi \in \Xi$ — числовий ідентифікатор цільового об'єкта у просторі середовища, $c \in \mathbb{N}^2 \cup \{\emptyset\}$ — просторові координати цілі $(i, j)$, які вилучаються з локальної пам'яті медіатора, який зберігає інформацію про виявлені об'єкти.

Загальна функція трансляції $F : \mathcal{U} \to \Pi$ декомпозується на три послідовні етапи:

1. На першому етапі застосовується функція виділення підрядка $E : \mathcal{U} \to \mathcal{V}$ (в

алгоритмі 1 позначена як *MatchPlanWithRegEx*), яка ізолює список дій, які складають план в загальному тексті $\mathcal{U}$, за допомогою апарату регулярних виразів. Символ $\mathcal{V}$ позначає множину усіх можливих планів в текстовій формі. Функція виділення підрядка виділяє символьну послідовність $v \in \mathcal{V}$, обмежену символами { та }.

2. На наступному кроці застосовується функція токенізації $T_{token} : \mathcal{V} \to \mathcal{W}^n$ (в алгоритмі 1 позначена як SplitActions), яка розбиває отриманий рядок $v$ на дискретну множину текстових інструкцій $w_k \in \mathcal{W}$, використовуючи кому як лексичний розділювач; $n$ — кількість кроків в плані.

3. В останню чергу виконується функція лексичного відображення $M : \mathcal{W} \to \Psi \times \Xi \times (\mathbb{N}^2 \cup \{\emptyset\})$, яка виконує семантичне зіставлення підрядків $w_k$ (наприклад, «go to», «pick up») з відповідними елементами простору опцій $\Psi$ та простору об'єктів $\Xi$. Паралельно здійснюється прив'язка абстрактного ідентифікатора об'єкта до його координат $c$ у середовищі на основі стану Медіатора.

Таким чином, для довільного висловлювання $u \in \mathcal{U}$, фінальний план формується як:

$$F(u) = \bigcup_{w_k \in T_{token}(E(u))} M(w_k). \tag{2.18}$$

Алгоритм 1 реалізує найпростіший лексико-синтаксичний підхід до трансляції у дискретний план тексту, згенерованого LLM. Незважаючи на свою обчислювальну простоту, цей алгоритм має суттєві обмеження, які в інженерії програмного забезпечення та машинному навчанні визначаються терміном крихкість (brittleness). Крихкість означає схильність системи до непередбачуваних збоїв при мінімальних відхиленнях або нестандартних варіаціях у вхідних даних. Етап пошуку тексту за допомогою регулярних виразів покладається на формальний шаблон *{(.*?)}*. Це створює жорстку залежність від точності форматування в тексті згенерованому LLM. Враховуючи стохастичну природу генерації тексту мовними моделями, структура тексту може відхилитися від інструкції: пропущена дужка, квадратні дужки замість фігурних або зайві символи форматування. У такому випадку функція *MatchPlanWithRegEx* не знайде збігів, що призведе до генерації

порожнього плану, в результаті чого, виникне необхідність опрацьовувати ситуацію з відсутнім планом.

На етапі визначення опцій та об'єктів алгоритм використовує операцію перевірки входження підрядка (наприклад, «explore» $\sqsubseteq$ $w_k$). Цей підхід не враховує граматичну структуру, семантичний контекст та синонімію. Такий підхід не дозволяє опрацьовувати логічні заперечення. Наприклад, якщо LLM згенерує текст «Do not explore the room», алгоритм хибно ідентифікує рядок «explore» як частину команди $EXPLORE$ і змусить агента виконати дії, прямо протилежні намірам Планувальника.

**Алгоритм 1:** Алгоритм трансляції неструктурованого плану

**Вхідні дані:** Текстова послідовність $u$, згенерована LLM

**Результат:** План $\phi$ як впорядкована множина параметризованих опцій $(\theta_1^{(\psi)}, \theta_2^{(\psi)}, \ldots, \theta_n^{(\psi)})$

$v \leftarrow$ MatchPlanWithRegEx(regex: «{(.*?)}», $u$)

$W \leftarrow$ SplitActions($v$, delimiter: «,») // $W \subset \mathcal{W}^n$ — множина текстових описів опцій для конкретного плану

$\phi \leftarrow \emptyset$

**для кожного** $w_k \in W$ **виконувати**

  // Ініціалізація базового стану для поточної опції

  $\psi \leftarrow$ NOTHING; $\xi \leftarrow$ EMPTY_ID; $c \leftarrow$ null

  // Визначення опції

  **якщо** «explore» $\sqsubseteq w_k$ **тоді** $\psi \leftarrow$ EXPLORE

  **інакше якщо** «pick up» $\sqsubseteq w_k$ **тоді** $\psi \leftarrow$ PICKUP

  **інакше якщо** «go to» $\sqsubseteq w_k$ **тоді** $\psi \leftarrow$ GO_TO_GOAL

  // Визначення ідентифікатора та координат об'єкта

  **якщо** «key» $\sqsubseteq w_k$ **тоді** $\xi \leftarrow$ KEY_ID, $c \leftarrow$ GetCoordinate(«key»)

  **інакше якщо** «door» $\sqsubseteq w_k$ **тоді**

    $\xi \leftarrow$ DOOR_ID, $c \leftarrow$ GetCoordinate(«door»)

  $\theta_k^{(\psi)} \leftarrow$ {option: $\psi$, object: $\xi$, coordinate: $c$}

  $\phi \leftarrow \phi \cup \{\theta_k^{(\psi)}\}$

**кінець**

Фіксований набір умовних перевірок для кожного окремого об'єкта (наприклад, перевірка наявності слова «key» для отримання ідентифікатора ключа KEY_ID) ускладнює розширення системи. В алгоритмі 1 KEY_ID,

DOOR_ID, EMPTY_ID позначають числові ідентифікатори типів об'єктів — ключа, дверей та пустої клітинки, відповідно. При додаванні в середовище десятків нових об'єктів або складних опцій кількість умовних операторів зростатиме лінійно, що ускладнює підтримку такого коду.

Ці обмеження наочно демонструють, що синтаксичний аналіз природної мови на основі правил є недостатнім для забезпечення безпомилкової роботи ієрархічних агентів. Це формує мотивацію для переходу до використання методу генерації з обмеженням грамитики, який розглядається у розділі 3.

Опції, згенеровані компонентом Планувальник, передаються на вхід компоненту Актор, який перетворює опції в послідовність конкретних дій, які будуть виконані в середовищі. Окрім цього, Актор приймає рішення, коли план виконано або коли необхідно викликати Планувальник та згенерувати новий план дій.

**Алгоритм 2:** Цикл виконання плану модулем Актор

**Вхідні дані:** Спостереження $o_{env,t} \in \Omega$, план $\phi$

**Результат:** Атомарна дія $a_t \in \mathcal{A}$, прапорець завершення $d_t \in \{0, 1\}$

$\tilde{\omega} = \text{Null}$

// Вибір наступної опції, якщо поточну завершено або не ініціалізовано:

**поки** ($\tilde{\omega} = \text{Null}$ **or** DoneCheck($\tilde{\omega}, o_{env,t}$)) **and** $\phi \neq \emptyset$ **виконувати**

  $\theta^{(\psi)} \leftarrow \text{Pop}(\phi)$; // Вилучення наступного завдання з плану

  $\tilde{\omega} \leftarrow \text{CreateSkill}(\theta^{(\psi)}, o_{env,t})$;

  **якщо** DoneCheck($\tilde{\omega}, o_{env,t}$) **тоді**

    $\tilde{\omega} \leftarrow \text{Null}$ ; // Пропуск цілей, що відразу є досягнутими або недосяжними

  **кінець**

**кінець**

// Генерація дії або сигналізація про завершення епізоду

**якщо** $\tilde{\omega} = \text{Null}$ **тоді**

  **повернути** $a_{\text{done}}, 1$ ; // Усі завдання виконано або план порожній

**інакше**

  $a_t \leftarrow \text{Step}(\tilde{\omega}, o_{env,t})$;

  $d_t \leftarrow \text{DoneCheck}(\tilde{\omega}, o_{env,t}) \wedge (\phi = \emptyset)$;

  **повернути** $a_t, d_t$;

**кінець**

Алгоритм 2 формалізує цикл управління компонента Актора у структурі ієрархічного агента. Процес виконання базується на обробці плану $\phi$, який концептуально подається у вигляді черги завдань, що функціонує за принципом FIFO (First-In-First-Out / Першим прийшов — першим пішов). Цей план генерується на вищому рівні архітектури — в модулі Планувальник, після чого Медіатор виконує функцію перетворення даних між текстовим представленням для планувальника та представленнями, специфічними для середовища.

На кожному кроці дискретного часу $t$, агент оцінює поточне спостереження $o_{env,t}$ та статус активної опції $\tilde{\omega}$. Якщо опція не ініціалізована ($\tilde{\omega} = \text{Null}$) або функція валідації свідчить про її завершення, підсистема вилучає наступний набір параметрів $\theta^{(\psi)}$ для ініціалізації опції із плану $\phi$ та конструює нову локальну стратегію. Важливим аспектом є обробка крайових станів: якщо цільовий стан є апріорі недосяжним або вже виконаним, опція негайно деактивується. У рамках архітектури з авторефлексивною пам'яттю (розділ 4), такі аномалії не відкидаються, а слугують тригером для оновлення внутрішнього контексту агента. Якщо ж опція $\tilde{\omega}$ є валідною, вона ітеративно генерує атомарну дію $a_t$ для безпосередньої взаємодії з середовищем, аж поки план не буде повністю виконаний, про що сигналізує прапорець завершення $d_t$.

У задачах послідовного прийняття рішень наступні кроки плану зазвичай залежать від успішності завершення попередніх кроків. Ігнорування невдачі, яка виникає на одному з проміжних кроків, може призвести до критичної десинхронізації між очікуваним та фактичним станом середовища. Це, у свою чергу, спричиняє каскад помилок в загальній поведінці агента. Для запобігання таким ситуаціям, модуль Актор має функціонувати не лише як виконавець стратегії, але і як валідатор можливих обмежень виконання стратегії в середовищі. У випадку дострокового завершення навички через неможливість її виконання, сигнал про помилку має передаватися на вищий рівень прийняття рішень — до модуля Планувальник. Отримана інформація про помилку має додаватися до пам'яті агента як негативний досвід, що дозволить коригувати майбутні стратегії з урахуванням властивостей середовища.

Складовою частиною Актора є компонент Decision Policy, який є бінарним класифікатором та вирішує завдання вибору кроку, на якому потрібно згенерувати новий план. Модуль функціонує на основі архітектури Actor-Critic. На вхід системи подається тензор локального спостереження середовища

$o_{env,t} \in \mathbb{N}^{H \times W \times C}$ у момент часу $t$. Процес виділення ознак здійснюється за допомогою згорткової нейронної мережі $f_\theta$, яка проєктує вхідний тензор у прихований простір. Отримані карти ознак трансформуються у плоский вектор $e_t$, який слугує спільним представленням (embedding) для частин нейромережі, які визначають функцію цінності та стратегію [84]:

$$e_t = \text{Flatten}(f_\theta(o_{env,t})) \tag{2.19}$$

Отриманий вектор $e_t$ паралельно обробляється двома повнозв'язними нейронними мережами. Перша мережа RL-Actor апроксимує стохастичну стратегію $\pi_\phi(a_t^{(com)}|o_{env,t})$, формуючи категоріальний розподіл над простором дій $\mathcal{A}^{(com)} = \{0, 1\}$. Дія $a_t^{(com)} = 1$ ініціює комунікацію з Планувальником для генерації нового плану, тоді як $a_t^{(com)} = 0$ означає продовження виконання поточного плану без переривань. Вихід RL-Actor формується через застосування функції Softmax до логітів:

$$\pi_{\theta_{RL-Actor}}(a_t^{(com)}|o_{env,t}) = \text{Softmax}(W_\pi \tanh(W_{e_1} e_t + b_{e_1}) + b_\pi) \tag{2.20}$$

Друга мережа Critic апроксимує функцію цінності стану $V_{\theta_{Critic}}(o_{env,t})$, яка оцінює очікувану сумарну винагороду починаючи з поточного стану. Ця оцінка використовується для розрахунку переваги під час оновлення ваг алгоритмом PPO:

$$V_{\theta_{Critic}}(o_{env,t}) = W_v \tanh(W_{e_2} e_t + b_{e_2}) + b_v \tag{2.21}$$

Процес навчання за допомогою алгоритму PPO вимагає значних обчислювальних витрат та великої кількості ітерацій для стабілізації градієнтів. З огляду на це, у базовій реалізації архітектури Decision Policy навчалася одноразово, після чого її параметри фіксувалися і стратегія повторно використовувалася для вирішення завдань з різними текстовими інструкціями.

Навчання агента відбувається в два етапи. Спочатку мовна модель отримує промпт з описом задачі, який дозволяє мовній моделі вирішувати новий тип задач за допомогою In-Context-Learning. На другому етапі навчання агента відбувається оптимізація параметрів класифікатора Decision Policy.

**Алгоритм 3:** Алгоритм навчання стратегії прийняття рішень (Decision Policy via PPO)

**Вхідні дані:** Середовище $\mathcal{E}$, Планувальник $Q_{\text{Planner}}$, Актор $Q_{\text{Actor}}$

**Результат:** Оптимізовані параметри $\theta_{RL-Actor}$ (Actor) та $\theta_{Critic}$ (Critic)

**для** ітерація $m = 1, \ldots, M$ **виконувати**
  $\mathcal{B} \leftarrow \emptyset$ ; // Очищення тимчасового буфера досвіду
  **для** епізод $k = 1, \ldots, N_{max\ episode}$ **виконувати**
    $o_{env,0} \sim \mathcal{E}, t \leftarrow 0, d_{\text{env}} \leftarrow \text{False}, d_{\text{plan}} \leftarrow \text{True}, \phi \leftarrow \emptyset$;
    **поки not** $d_{\text{env}}$ **and** $t <$ max episode **виконувати**
      $a_t^{(com)} \sim \pi_\phi(\cdot|o_{env,t})$ ; // Семплювання комунікаційного рішення
      **якщо** $a_t^{(com)} = 1$ **or** $d_{\text{plan}} = 1$ **тоді**
        $\phi \leftarrow Q_{\text{Planner}}.\text{plan}(o_{env,t})$ ; // Синхронний генеративний запит
        $Q_{\text{Actor}}.\text{change\_plan}(\phi)$;
      **кінець**
      $a_t, d_{\text{plan}} \leftarrow Q_{\text{Actor}}.\text{get\_action}(o_{env,t})$;
      $o_{env,t+1}, r_t, d_{\text{env}} \leftarrow \mathcal{E}.\text{step}(a_t)$;
      $\tilde{r}_t \leftarrow r_t - (\lambda_{\text{ask}} + \text{Penalty}(\phi)) \cdot (a_t^{(com)} \vee d_{\text{plan}})$ ; // Формування винагороди
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(o_{env,t}, a_t^{(com)}, \tilde{r}_t, o_{env,t+1})\}$;
      $o_{env,t} \leftarrow o_{env,t+1}; t \leftarrow t + 1$;
    **кінець**
  **кінець**
  Обчислити переваги $A_t^{(PPO)}$ за допомогою $V_{\theta\ Critic}$ для всіх транзакцій у буфері $\mathcal{B}$;
  Оновити $\theta_{RL-Actor}$ максимізацією цільової функції PPO з обмеженням $\epsilon$;
  Оновити $\theta_{Critic}$ мінімізацією середньоквадратичної помилки (MSE) функції цінності;
**кінець**

Алгоритм 3 формалізує процес навчання Decision Policy в розглянутому агенті алгоритмом PPO. Процес ітеративно чергує дві ключові фази: збір даних та оптимізацію стратегії. Під час фази збору даних агент взаємодіє

із середовищем протягом $N_{max\ episode}$ послідовних епізодів. На кожному часовому кроці $t$ комунікаційна мережа $\pi_{\theta\ RL-Actor}$ аналізує поточний стан середовища $o_{env,t}$ та стохастично генерує бінарне рішення $a_t^{(com)}$ щодо виклику Планувальника. Якщо мережа ініціює запит ($a_t^{(com)} = 1$) або якщо поточний план завершено ($d_{\text{plan}} = 1$), відбувається виклик планувальника моделі $Q_{\text{Planner}}$.

Для керування поведінкою агента застосовується метод формування винагороди (reward shaping): від базової винагороди середовища $r_t$ віднімається динамічний комунікаційний штраф, який складається з постійного доданка $\lambda_{ask}$ і змінного, який розраховується на основі плану $\phi$. Зібрані транзакції зберігаються в тимчасовий буфер $\mathcal{B}$. Після завершення фази збору даних ініціюється фаза оптимізації, під час якої параметри RL-Actor $\theta_{RL-Actor}$ та Critic $\theta_{Critic}$ оновлюються алгоритмом PPO.

Навчання з підкріпленням не потребує наявності моделі середовища для планування. Натомість, стратегія прийняття рішень вивчається агентом в процесі взаємодії з середовищем. Дослідник має задати метрику ефективності — функцію винагороди.

Загальноприйнятим є поділ алгоритмів навчання з підкріпленням на три великі групи: алгоритми, які вивчають функцію якості (Q-learning); алгоритми, які обчислюють градієнти стратегії (policy optimization) та алгоритми з моделлю (model-based). У підході навчання з підкріпленням агент може містити три компоненти: стратегія, функція цінності, модель середовища. Алгоритми мають різні властивості, від яких залежить вибір в конкретній задачі, такі, як, обчислювальна складність, ефективність використання вибірки.

Одним з найпопулярніших методів онлайнового навчання з підкріпленням без моделі є Proximal Policy Optimization (PPO) [84]. PPO вирішує проблему нестабільної поведніки процедури оптимізації обмежуючи розмір кроку в просторі параметрів нейронної мережі. PPO дає одну з можливих відповідей на питання, як зробити максимально довгий крок оптимізації в просторі параметрів моделі без ризику зіткнутися з деградацією стратегії. Існує дві версії алгоритму PPO: PPO Clip та PPO Penalty. PPO Clip добре працює на практиці та більш обчислювально ефективний, тому застосовується частіше. PPO є алгоритмом оптимізації першого порядку, який відноситься до групи онлайн алгоритмів оптимізації стратегії (policy optimization). Перевагою алгоритму PPO є те, що його можна застосовувати до середовищ з дискретним і

неперервним простором дій. PPO може тренуватися одночасно на багатьох екземплярах середовища, які виконуються паралельно, що дозволяє значно прискорити навчання за рахунок додавання обчислювальних потужностей. Недоліком алгоритму PPO є низька ефективність використання тренувальних прикладів.

## 2.3 Особливості формування контексту для задач планування

Згідно з [85] мовна модель може бути представлена у вигляді функції:

$$output = g(x_0, x_1, ..., x_m; \theta_{LLM}) \tag{2.22}$$

де $output$ — результат обчислення функції, конкретна форма якого залежить від вирішуваної задачі, $x_i$ — токен вхідної послідовності довжиною $m$, $g(\cdot; \theta_{LLM})$ або $g_{\theta\ LLM}(\cdot)$ — нейронна мережа з вектором параметрів $\theta_{LLM}$.

Наприклад, в задачі визначення наступного токену в послідовності, вихідне значення $output$ приймає вигляд статистичного розподілу на всьому словнику допустимих токенів $p(x_{m+1}|x_0, ..., x_m)$. Натомість в задачах векторного представлення даних $output$ приймає вигляд вектора $output \in \mathbb{R}^{n-1}$ з елементами у вигляді дійсних чисел.

У процесі процедури навчання нейронної мережі вектор параметрів $\theta_{LLM}$ змінюється шляхом застосування методу градієнтного спуску так що мінімізується значення функції помилки. Після виконання процедури попереднього навчання, отримується попередньо-навчена мовна модель $g_{\theta\ L\hat{L}M}(\cdot)$. Щоб покращити результати роботи такої мовної моделі, потрібно додатково оптимізувати її на невеликому наборі прикладів вирішення конкретної задачі. У такому разі параметри $\hat{\theta}_{LLM}$ незначно зміняться.

Ми розглянемо навчання лише нейронних мереж з архітектурою Transformer, оскільки це найбільш поширена наразі архітектура для вирішення мовних завдань.

Мовна модель типу декодер визначає ймовірнісний розподіл токенів зі словника на основі послідовності попередніх токенів. Після визначення розподілу застосовується певний метод вибору одного токена з розподілу, найпростіший з них — вибір токену за максимальною ймовірністю. Загальний

підхід до навчаня моделей-декодерів полягає у мінімізації значення функції помилки на наборі послідовностей токенів.

Позначимо модель-декодер як $Decoder_{\theta\ LLM}(\cdot)$. Процес генерації послідовності складається з дискретної послідовності кроків. На кожному кроці $i$ модель-декодер обчислює ймовірнісний розподіл токенів зі словника на основі послідовності попередніх токенів $\{x_0, ..., x_i\}$. Такий розподіл позначається як $Pr_{\theta\ LLM}(\cdot|x_0, ..., x_i)$.

Процедура навчання полягає у виборі такого значення вектору параметрів $\theta_{LLM}$, при якому функція помилки досягає мінімального значення:

$$\hat{\theta}_{LLM} = \arg\min_{\theta\ LLM} \sum_{x \in \mathcal{D}} \text{Loss}_{\theta\ LLM}(\mathbf{x}), \tag{2.23}$$

де $\mathcal{D}$ — набір тренувальних послідовностей токенів.

Ціль оптимізації математично еквівалентна до обчислення оцінки максимуму правдоподібності, та може бути варажена наступним чином:

$$\begin{aligned}\hat{\theta}_{LLM} &= \arg\max_{\theta\ LLM} \sum_{\mathbf{x} \in \mathcal{D}} \log \Pr_{\theta}(\mathbf{x}) \\ &= \arg\max_{\theta\ LLM} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i=0}^{|\mathbf{x}|-1} \log_{\theta} \Pr_{LLM}(x_{i+1}|x_0, \ldots, x_i).\end{aligned}$$

Процедура навчання таких складних систем, як великі мовні моделі, розділяється на декілька проміжних етапів. Перший з них це попереднє навчання (pre-training). На етапі попереднього навчання мовна модель навчається на великому масиві даних, який містить широкий спектр можливих завдань, а ціль такого навчання — досягти загальних здібностей мовної моделі до вирішення мовних завдань. Попереднє навчання застосовується в припущенні, що мовні моделі здатні отримати загальні навички і в подальшому можуть бути адаптовані до конкретних задач, що підвищить якість роботи. У результаті зникає необхідність навчати для кожної конкретної задачі нову нейронну мережу з нуля. Натомість ми можемо використати попередньо-навчену LLM, яка володіє широким спектром навичок, і донавчити її на невеликому наборі даних для конкретної задачі (downstream task).

У цьому дисертаційному дослідженні були використані сучасні підходи

до формування контексту LLM такі, як In-Context-Learning, zero-shot, few-shot, episodic memory.

Для перевірки ефективності часової абстракції дій у формалізмі напівмарковських процесів (Semi-MDP) було протестовано промпт зі штучним обмеженням горизонту планування до одного кроку (лістинг 2.1). Ця інструкція переводить архітектуру агента з режиму генерації послідовності макро-дій (опцій) у режим реактивного управління — одна дія на кожне звернення до Планувальника. Такий формат ізолює здатність моделі до декомпозиції складної задачі та переносить навантаження на механізм епізодичної пам'яті агента, оскільки без запам'ятовування попередніх кроків реактивна навігація в умовах часткової спостережуваності є неефективною.

**Промпт 2.1: Промпт модуля Планувальник з обмеженням на одну дію**

**Task**

```
Your task is to build a sequence of actions (a plan) for an agent in a
minigrid reinforcement learning environment. The task of the agent is to
open a door in the maze using a key. Please, help the agent to build a plan
using agent's current observations and state: carry object or none.
```

**Tools**

```
Available actions: explore, go to object, pick up object, toggle object.
```

**Formatting**

```
The plan should be formated as a list 'action'. You can only select one
action at a time. Do not explain the reasoning.
```

**Context**

```
observation: {observation}
```

Мовні моделі демонструють здатність вчитися вирішувати з першої спроби не бачені при навчанні задачі. Один з важливих методів такого навчання це начання з кількох спроб (few-shot learning). В основі цього методі лежить феномен навчання з контексту (In-Context-Learning, ICL). Суть методу полягає в тому, що до вхідної послідовності додаються декілька прикладів правильного вирішення завдання у вигляді пар вхід-вихід. Такі приклади називаються демонстраціями (demonstrations). Мовні моделі здатні узагальнювати отриманий набір демонстрацій, на основі чого здатні вирішувати більш широкий клас подібних завдань.

Метод створення промптів zero-shot полягає у передачі мовній моделі опису

завдання без прикладів вирішення. Інший підхід, який має назву few-shot, натомість, передбачає додавання набору прикладів успішного вирішення завдання. У випадку few-shot мовна модель може узагальнити приклади, отримавши загальну схему вирішення подібних задач, яка надалі буде застосована до вхідних даних.

З метою дослідження власних евристичних здібностей попередньо навченої мовної моделі та оцінки критичності використання In-Context Learning у обраному середовищі, було сформовано промпт (лістинг 2.2), який не містив прикладів вирішення завдання (zero-shot промпт). На відміну від конфігурацій типу few-shot, цей промпт містить лише загальний опис середовища, ціль (відкрити двері) та просторові обмеження, але повністю позбавлена демонстраційних прикладів. Такий підхід змушує агента покладатися виключно на внутрішнє семантичне розуміння понять «ключ» та «двері» для генерації граматично коректного плану дій.

**Промпт 2.2: Промпт модуля Планувальник без прикладів**

**Role**

```
You are an agent in an environment.
```

**Tools**

```
You can move and interact with objects. In the environment, there are the
following objects: a key and a door. Available actions: explore, go to
object, pick up object, toggle object. You can pick up a key. The door is
locked. The key should be used to unlock the door. At the start of the
episode, objects can be hidden, and you must find them.
```

**Task**

```
Your task is to open the door.
```

**Formatting**

```
The plan should be formatted as a list: action 1, action 2, .... Do not
explain the reasoning.
```

**Context**

```
observation: {observation}
```

Також була протестована схема дизайну промптів, у якій мовна модель отримувала план за замовчуванням, який містив загальну схему дій, яка має бути модифікована мовною моделлю для конкретного випадку.Ця стратегія промптингу інтегрує в контекст моделі концепцію «плану за замовчуванням»

(default plan) (лістинг 2.3). Ця модифікація вводить апріорну евристичну траєкторію (дослідити кімнату, підняти ключ, підійти до дверей, відкрити двері) безпосередньо в інструкцію. Робоча гіпотеза полягає в тому, що наявність базової структури рішення слугуватиме семантичним якорем для формування конкретних планів мовною моделлю. Модель має не створювати план з нуля, а модифікувати запропоновану універсальну послідовність кроків відповідно до поточного спостереження середовища.

**Промпт 2.3: Промпт модуля Планувальник з планом за замовчуванням**

**Role**

```
You are an agent in an environment.
```

**Tools**

```
You can move and interact with objects. In the environment, there are the
following objects: a key and a door. Available actions: explore, go to
object, pick up object, toggle object. You can pick up a key. The door is
locked. The key should be used to unlock the door. At the start of the
episode, objects can be hidden, and you must find them.
```

**Default plan**

```
Default plan is: explore, pick up the key, go to the door, toggle the door.
```

**Task**

```
Your task is to open the door.
```

**Formatting**

```
The plan should be formatted as a list: action 1, action 2, .... Do not
explain the reasoning.
```

**Context**

```
observation: {observation}
```

У деякі конфігурації агента була додана епізодична пам’ять тривалістю в один епізод. З вимкненою пам’яттю, мовна модель отримує лише поточне спостереження середовища. В такому разі агент не здатен запам’ятовувати попередні дії та помилки і робити висновки з власного досвіду. З використанням пам’яті, агент отримує історію взаємодії з середовищем від початку епізоду.

**Промпт 2.4: Початкова версія промпта модуля Planner**

**Role**

```
an agent in a minigrid environment in reinfrocement learning,
```

**Tasks**

```
the task of the agent is toggle the door in the maze with key. please help
agent to plan the next action given agent's current observations and statu:
carry object or none.
```

**Tools**

```
Availabel actions may includes: explore, go to object, pick up object,
toggle object. the actions should be displayed in a list.
```

**Formatting**

```
Do not explain the reasoning.
```

**Few-shot examples**

```
observation: observed nothing, action: explore.
observation: observed a door, action: explore.
observation: observed a key, observed a door, action: go to the key, pick up
the key, go to the door, toggle the door.
observation: observed a door, carry key, action: go to the door, toggle the
door.
observation: observed a key, action: go to the key, pick up the key,
explore.
```

**Context**

```
observation: {observation}
```

Базовою конфігурацією модуля Планувальник виступає промпт (лістинг 2.4) із дослідження [5], отриманий з репозиторію вихідного коду [86]. Ця інструкція використовує парадигму навчання в контексті (ICL) шляхом надання кількох демонстраційних прикладів вирішення задачі навігації. Структура промпту визначає роль агента, його кінцеву мету, доступний простір дій та набір правил форматування. Незважаючи на загальну ефективність методу few-shot, ця початкова версія містить синтаксичні неоднозначності, які можуть призводити до помилок синтаксичного розбору на етапі трансляції тексту в дискретний план.

**Промпт 2.5: Промпт модуля Планувальник з виправленою граматикою**

**Task**

```
Your task is to build a sequence of actions (a plan) for an agent in a
minigrid reinforcement learning environment. The task of the agent is to
open a door in the maze using a key. Please, help the agent to build a plan
using agent's current observations and state: carry object or none.
```

**Tools**

```
Available actions: explore, go to object, pick up object, toggle object. The
plan should be formatted as a list 'action 1, action 2, ...'.
```

**Formatting**

```
Do not explain the reasoning.
```

**Few-shot examples**

```
Examples:
Input: observation: observed nothing
Output (a plan): explore
Input: observation: observed a door
Output (a plan): explore
Input: observation: observed a key, observed a door
Output (a plan): go to the key, pick up the key, go to the door, toggle the
door
Input: observation: observed a door, carry key
Output: go to the door, toggle the door
Input: observation: observed a key
Output: go to the key, pick up the key, explore
```

**Context**

```
observation: {observation}
```

Для забезпечення дотримання синтаксису плану та мінімізації помилок генерації було розроблено виправлену версію промпту (лістинг 2.5), яка була створена на основі промпту 2.4. Змінена версія промпту усуває недоліки базової версії. Зокрема, було модифіковано лінгвістичну репрезентацію порожніх станів та скориговано граматичні конструкції (наприклад,опис стану володіння ключем). Ці зміни критично важливі для коректної роботи компонента Медіатор, який виконує функцію перетворення даних між представленнями, специфічними для середовища, та текстовим представленням для Планувальника. Оновлена структура демонстраційних

прикладів підвищує ймовірність генерації плану з коректним синтаксисом.

Агент функціонує у «реактивному» режимі: на кроці $t$ LLM отримує лише поточне спостереження $o_t^{(text)}$. В умовах часткової спостережуваності середовища це призводить до критичних помилок навігації, оскільки $o_t^{(text)}$ не містить інформації про об'єкти, які знаходяться поза розвіданою областю середовища. Для вирішення цієї проблеми було застосовано короткострокову епізодичну пам'ять довжиною в один епізод.

Пам'ять реалізовано як буфер history у класі Planner, який імплементує модуль Планувальник. Управління пам'яттю здійснюється через конфігураційний прапорець use_memory у класі PlannerConfig. Життєвий цикл пам'яті виконується за наступною логікою.

На початку кожного епізоду (або при примусовому скиданні через метод Planner.reset()) буфер очищується. У ньому залишається лише системне повідомлення (system prompt), яке містить інструкції та опис завдання. Це гарантує, що агент не переносить помилкову інформацію («галюцинації») або застарілий контекст з попередніх епізодів у наступні.

Якщо агент генерує валідний план дій, то трійка (спостереження, міркування, план) додається до списку history. Важливо зазначати, що зберігається не лише дія, а й спостереження, що дозволяє LLM розуміти структуру та зміни в середовищі.

При генерації наступного запиту до LLM, вміст буфера history конкатенується з поточним спостереженням. Це дозволяє моделі використовувати механізм уваги для виявлення залежностей між минулими подіями та поточним станом середовища.

Експериментально встановлено, що наявність пам'яті є критичною для успішної роботи промптів типу «one-step» — генерації по одній дії за один виклик Планувальника.

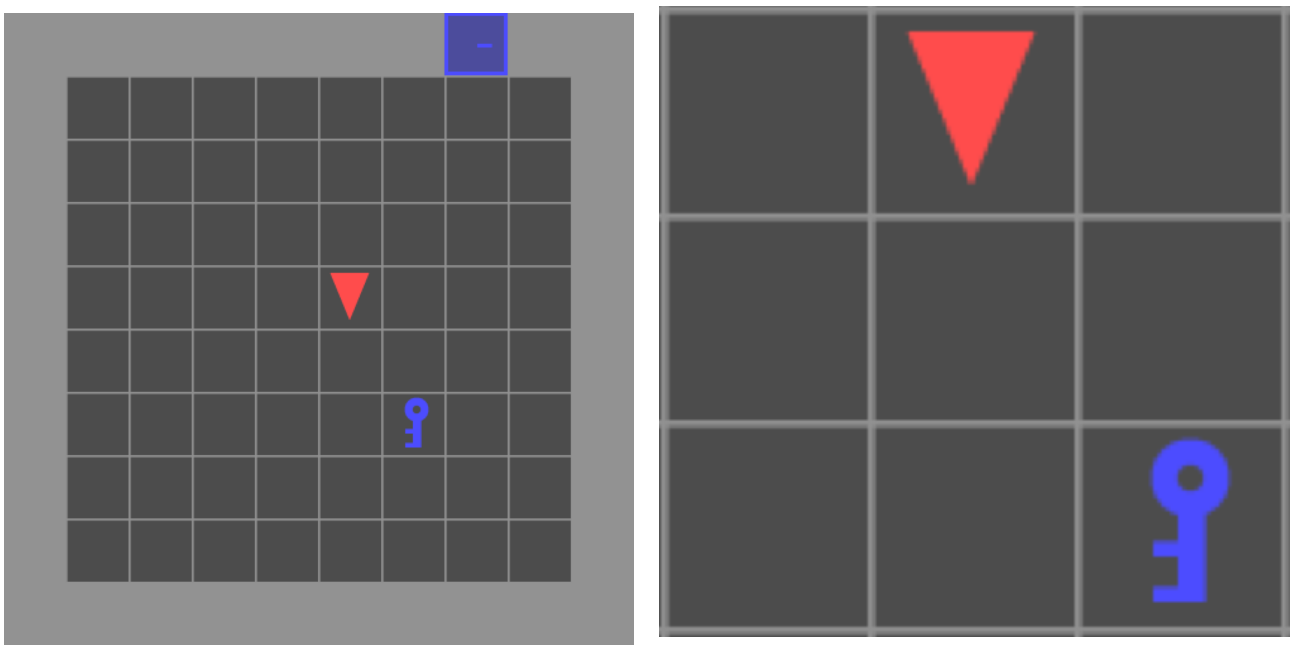
Використання пам'яті дозволило агенту успішно вирішувати завдання, що вимагають послідовності дій над об'єктами, оскільки в контексті зберігалася інформація про стан об'єктів та агента.

## 2.4 Програмна реалізація, інтеграція з середовищем Minigrid та процедура навчання

В дослідженні була використана бібліотека Minigrid [87] — бібліотека модульних середовищ, які можуть бути налаштовані для навчання агентів навчання з підкріпленням вирішенню різних задач, які потребують досягнення цілі у двовимірному просторі. Бібліотека має мінімалістичний дизайн, що дозволяє швидко розробляти нові середовища та проводити експерименти з високою обчислювальною ефективністю. На рисунку 2.1а показано візуальне представлення початкового стану середовища Minigrid — SimpleDoorKey. В середовищах Minigrid агент має обмежену область видимості, тому щоб дізнатися про структуру та предмети в середовищі, агент мусить спочатку дослідити середовище. На рисунку 2.1б показано поле зору агента розміром 3 на 3 клітинки.

Середовище є двовимірною сіткою з $n \times m$ клітинок. Кожна клітинка може бути або пустою, або містити один об'єкт, наприклад стіну, ключ, скриньку та ін. Шляхом вибору розміру середовища та розміщення об'єктів в різних конфігураціях, можливо створювати завдання різної складності. За замовчуванням середовища Minigrid детерміновані. Спостереження середовища представляє собою кортеж з двох елементів $\{\text{image}, \text{mission}\}$, де image — матриця $o_{env}$, яка містить опис клітинок середовища, mission — текстовий опис завдання. Опис завдання може змінюватися при кожному перезапуску середовища. Для інтерпретації завдання агент може використовувати мовну модель. Агент має дискретний набір дій, який включає: повернути ліворуч (turn left); повернути вправоруч (turn right); зробити крок вперед (move forward); підібрати предмет попереду (pickup); викласти з інвентарю предмет, який тримає агент, (drop); взаємодіяти з предметом попереду (toggle); завершити виконання (done). Середовище Minigrid містить розріджені винагороди, винагорода видається в кінці епізоду, якщо агент успішно виконав завдання. Minigrid передбачає можливість змінювати всі компоненти середовища для створення необхідних завдань.

Середовище може містити стіни, які обмежують переміщення агента, двері, які можуть бути закритими, обмежуючи видимість, та потребують застосування ключа для їх відкриття, ящиків, які можуть містити інші предмети, круги,



(а) Приклад початкового стану середовища Minigrid - SimpleDoorKey.

(б) Область зору агента.

Рисунок 2.1: Початковий стан середовища та область зору агента.

які слугують перепоною, яку агент може самостійно пересунути. Предмети відрізняються кольорами, що додає варіативності в формулювання завдання, та додає комбінаторну складність в простір можливих рішень. На рисунку 2.2 три послідовні візуалізації стану середовища SimpleDoorKey на яких зображено процес дослідження середовища агентом. На першому зображенні показано початковий стан середовища, на другому — послідовний рух агента в напрямку початкового повороту, на третьому — агент знаходить двері, але ключ залишається не знайденим.

Середовище Minigrid реалізовано мовою програмування Python з використанням бібліотеки NumPy, що забезпечує високу швидкість симуляції. Інтерфейс середовища відповідає стандарту Gymnasium [88] (раніше OpenAI Gym [89]), що спрощує інтеграцію з сучасними бібліотеками глибокого навчання.

На рисунках 2.3 та 2.4 зображено проблемні ситуації в усіх чотирьох використаних середовищах. На рисунку 2.3а агент першим серед двох об’єктів знайшов двері та помилково намагається відкрити двері не маючи необхідного ключа. На рисунку 2.3б агент в середовищі KeyInBox знайшов скриньку з ключем, але проігнорував її та, не маючи ключа, помилково намагається відкрити двері.

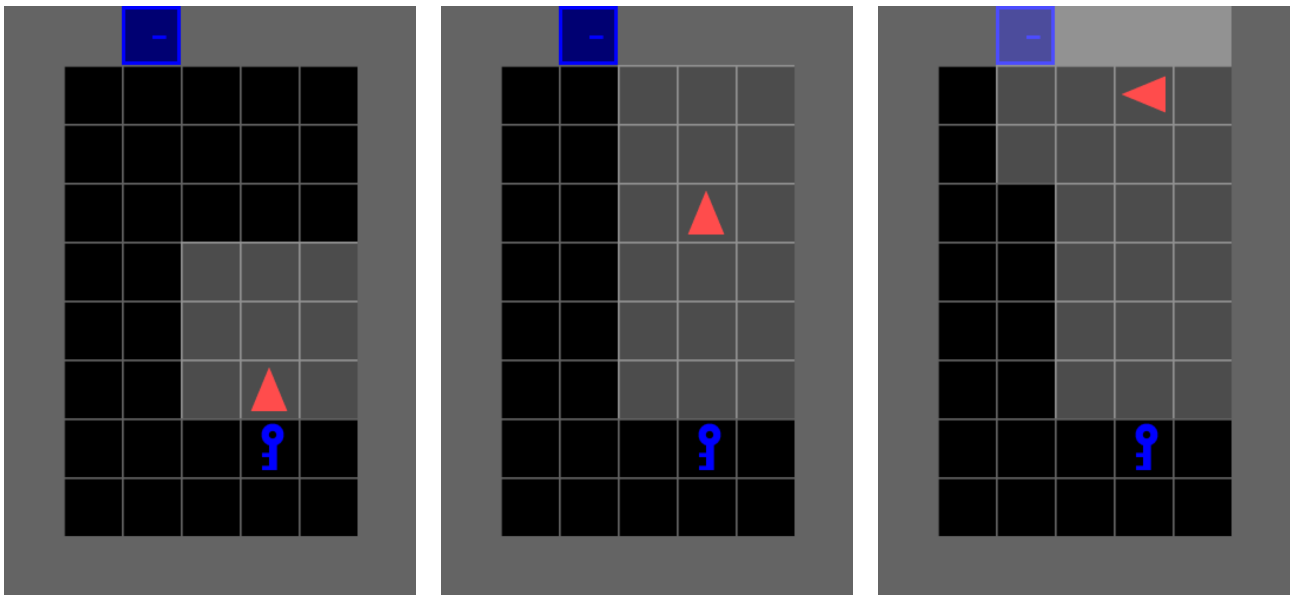
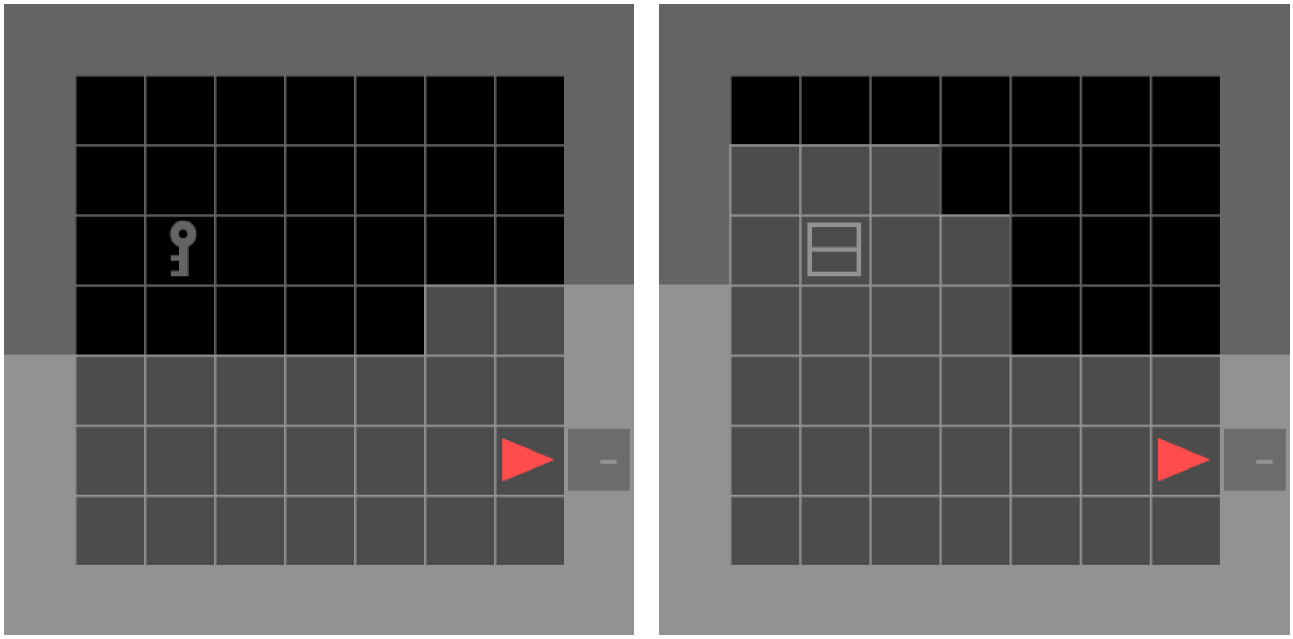


Рисунок 2.2: Процес дослідження середовища агентом.



(а) SimpleDoorKey: Агент намагається відкрити двері не маючи ключа.

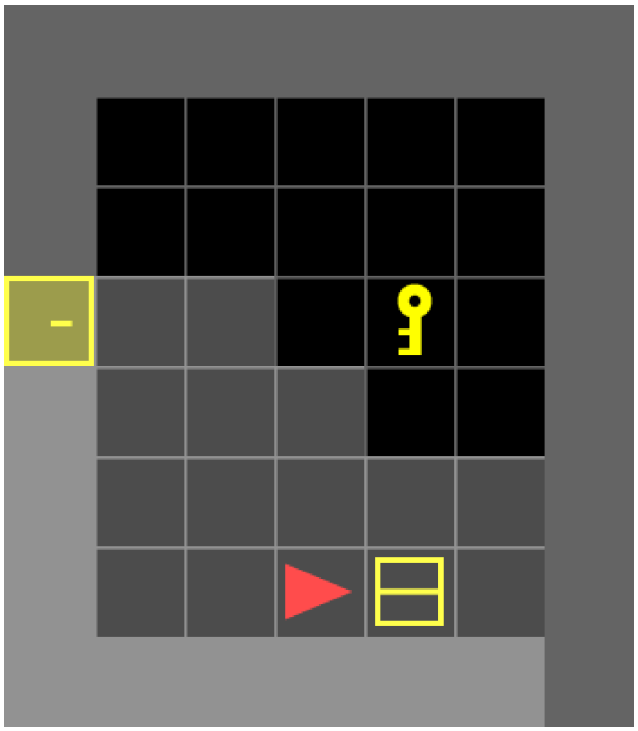
(б) KeyInBox: Агент знайшов скриньку з ключем, але проігнорував її.

Рисунок 2.3: Приклади середовищ Minigrid: SimpleDoorKey, KeyInBox.

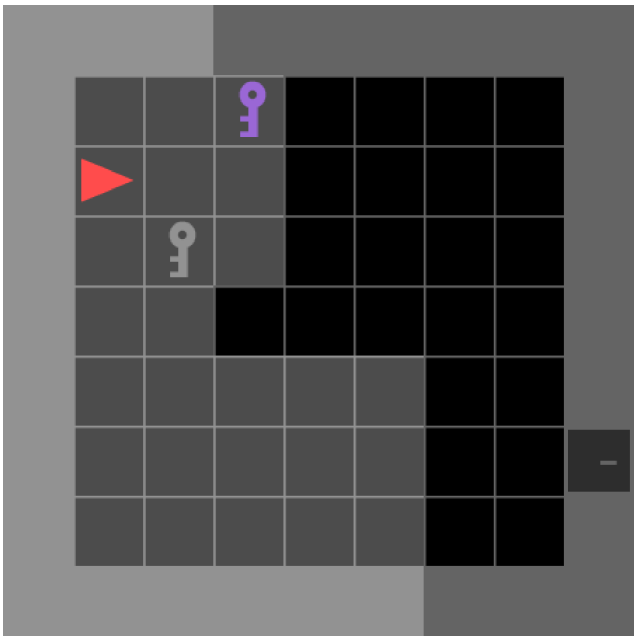
На рисунку 2.4а агент спочатку знайшов скриньку, та намагаєтсья її відкрити, оскільки не має інформації, що ключ лежить окремо. На рисунку 2.4б агент знайшов одразу два ключі, але не маючи інформації, який з них відповідає кольором дверям, які поки що не знайдені.

Для спрощення завдання, ми розглядаємо середовище без перепон. Агент отримує завдання знайти ключ та відкрити двері за найменшу кількість кроків в середовищі. Агент взаємодіє з середовищем в дискретному часі,

шляхом ітерацій. Кількість ітерацій обмежена в 100 кроків, при вичерпанні який епізод завершується, а завдання вважається не виконаним. На початку епізоду середовище ініціалізується обраним завданням, агент поміщається у випадкове положення в лабіринті. Агент може виконувати послідовності кроків, як можуть призвести до успішного виконання завдання, або вичерпати ліміт в 100 кроків.



(а) RandomBoxKey: Агент не знайшов ключ та відкриває скриньку в якій відсутній ключ.



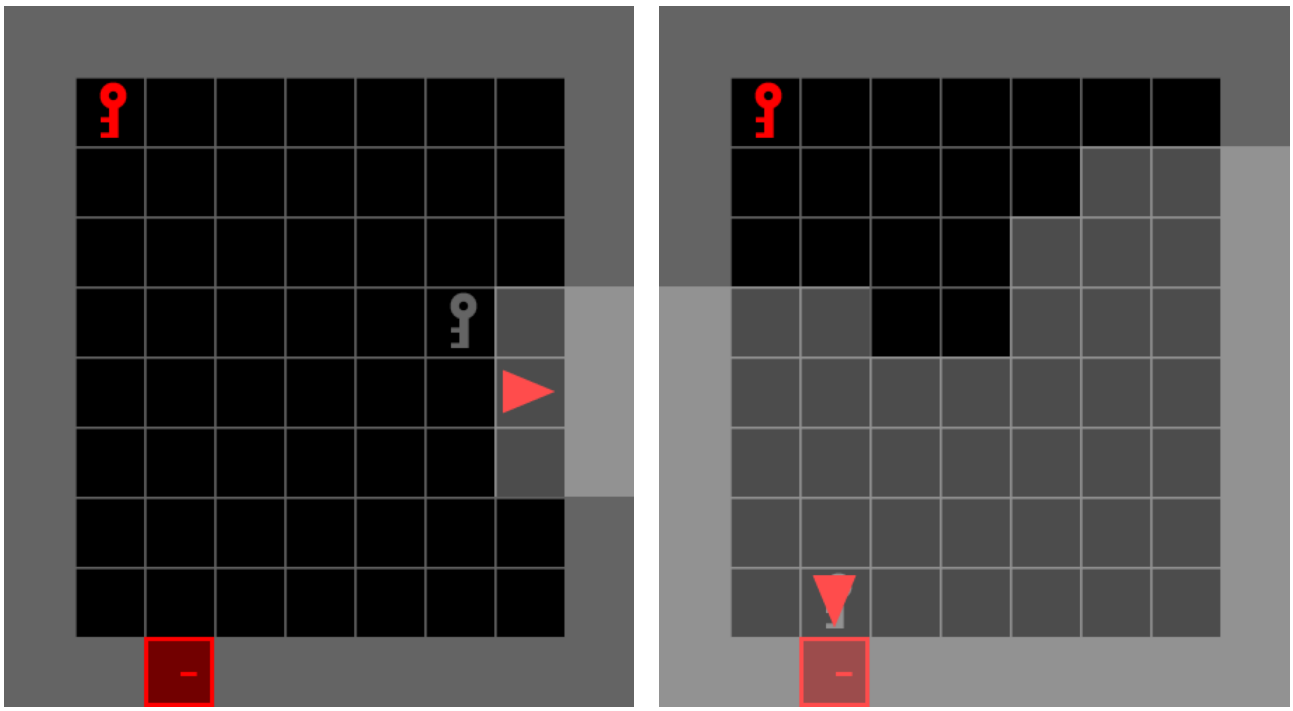
(б) ColoredDoorKey: Агент не знайшов двері, тому не має інформації, який ключ потрібно підібрати.

Рисунок 2.4: Приклади середовищ Minigrid: RandomBoxKey, ColoredDoorKey.

На рисунку 2.5 зображена характерна помилка, якої припускається агент в середовищі ColoredDoorKey. Якщо агент першим бачить ключ не правильного кольору, то підбирає його, знайшовши двері, агент може проігнорувати не відповідність кольорів та спробувати відкрити двері. Інколи агент не здатен визначити власну помилку і повторно намагається відкрити двері не відповідним ключем.

На рисунку 2.6 показано приклад відповідності зображення стану середовища та текстового спостереження, яке отримує на вхід модуль Планувальник. Агент не отримує інформацію про об’єкти які агент покищо не знайшов. Опис знайдених об’єктів формулюється у вигляді висловлювань природною англійською мовою.

На зображенні 2.6а агент знайшов два предмети в середовищі



(а) ColoredDoorKey: Початковий стан середовища при значенні seed = 10.

(б) ColoredDoorKey: Агент тримає ключ сірого кольору і намагається відкрити червоні двері.

Рисунок 2.5: Приклад помилки в середовищі ColoredDoorKey.

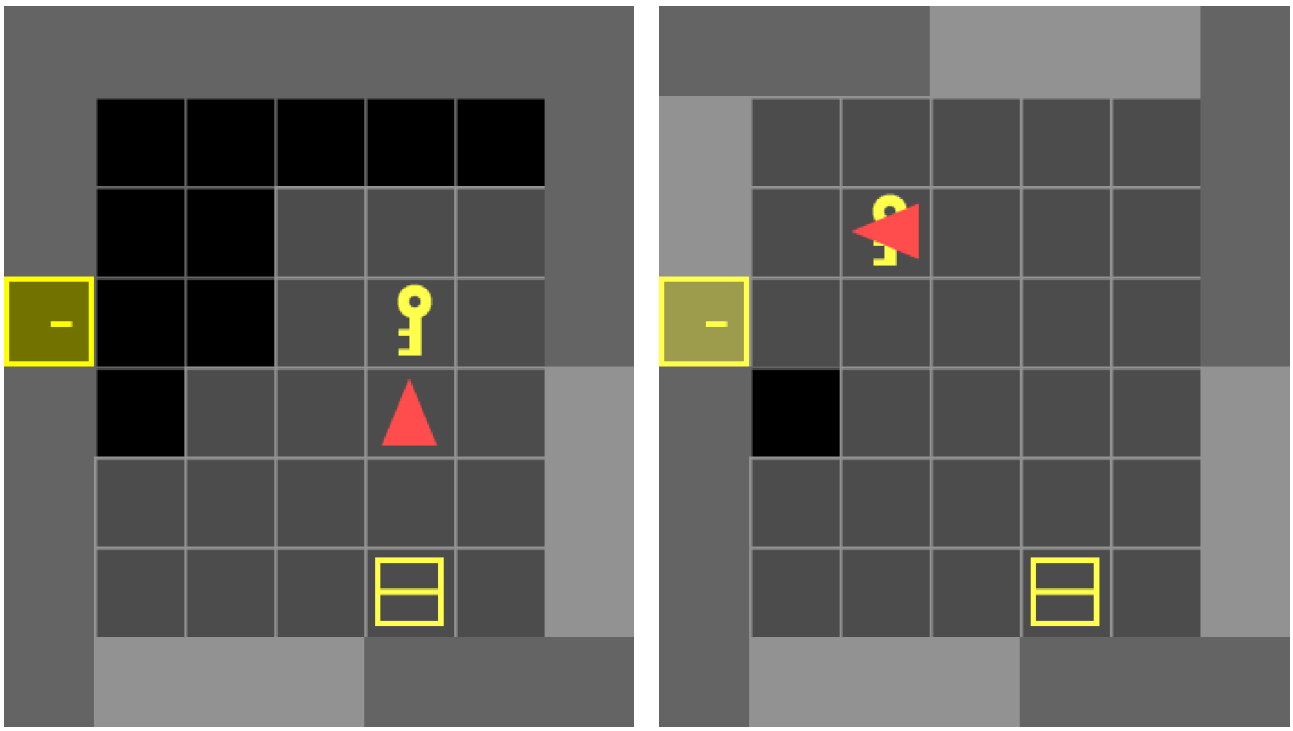
RandomBoxKey — скриньку (box) та ключ (key). Агент покищо не знайшов двері (door). При перетворенні спостереження в текстове спостереження за допомогою модуля Медіатор, отримуємо наступне: *observation: [the yellow key is visible, the yellow box is visible]*.

На зображенні 2.6б агент знайшов три предмети в середовищі RandomBoxKey — агент тримає ключ (key), бачить скриньку (box) та двері (door). При перетворенні спостереження в текстове спостереження за допомогою модуля Медіатор, отримуємо наступне: *observation: [the yellow key is held, the yellow box is visible, the door is visible]*.

Для реалізації агента та процедури начання, використовувалися наступні бібліотеки та технології:

1. *Python* — це інтерпретована мова програмування, яка широко використовується для вирішення проблем штучного інтелекту, дозволяє швидко реалізовувати прототипи алгоритмів. Python — це легка для вивчення та розуміння мова програмування.

2. *Numpy* — це бібліотека з ефективною реалізацією багатьох числових типів, багатовимірних масивів та операцій над ними. Використовується



(а) RandomBoxKey: Стан та спостереження середовища у якому агент знайшов скриньку та ключ.

(б) RandomBoxKey: Стан та спостереження середовища у якому агент тримає ключ, знайшов скриньку та двері.

Рисунок 2.6: RandomBoxKey: Відповідність зображення стану середовища та текстового спостереження.

для симуляції середовища, обробки спостережень та інших математичних обчислень.

3. *Ollama* — це інструмент з відкритим вихідним кодом, який дозволяє виконувати великі мовні моделі (LLM) на локальному обладнанні користувача, що дозволяє уникнути витрат на використання хмарних API та знизити затримку генерації тексту.

4. *Matplotlib* — це бібліотека для створення статичних, анімованих та інтерактивних візуалізацій даних. Бібліотека надає низькорівневий програмний інтерфейс для контролю над окремими деталями графічних об’єктів. Matplotlib є універсальним інструментом для побудови графіків та є де-факто стандартом для побудови графіків для наукових публікацій.

5. *Pandas* — це бібліотека для обробки та аналізу структурованих, зокрема табличних, даних. Бібліотека містить інструментарій для обробки числових рядів, агрегації даних, виконання реляційних операцій, подібних до SQL, та відображення даних з використанням бібліотеки *Matplotlib*.

6. *Seaborn* — це бібліотека для статистичної візуалізації даних, яка базується на Matplotlib та інтегрована зі структурами даних Pandas. Бібліотека надає високорівневий інтерфейс для побудови деталізованих графіків, містить реалізацію статистичних методів для обробки даних та за замовчуванням створює якісні стилі графіків.

7. *Gymnasium* — це бібліотека з відкритим вихідним кодом на мові Python, яка надає стандартизований інтерфейс прикладного рівня для задач навчання з підкріпленням. Бібліотека є відгалудженням проєкту OpenAI Gym, забезпечує уніфіковану взаємодію між агентом та середовищем за циклом «стан - дія - винагорода» та математично формалізує задачу як Марковський процес прийняття рішень (MDP).

8. *Minigrid* — це бібліотека з відкритим вихідним кодом, яка містить набір мінімалістичних двовимірних середовищ типу «сітковий світ» (gridworld) для задач навчання з підкріпленням. Бібліотека містить інструменти для створення нових середовищ та нових завдань. Особливістю середовищ Minigrid є часткова спостережуваність, розріджені винагороди та низькі вимоги до обчислювальних ресурсів. Прстота та низькі вимоги до обчислювальних ресурсів дозволяє проводити швидкі ітерації експериментів, що є критично важливим для тестування нових моделей штучного інтелекту перед подальшим масштабуванням.

9. *PyTorch* — це бібліотека з відкритим вихідним кодом, що надає інструменти для машинного навчання, зокрема створення тензорів, виконання математичних операцій над тензорами, виконання автоматичного диференціювання, прискорення обчислень з використанням графічних прискорювачів. PyTorch реалізує концепцію динамічного графа обчислень, що дозволяє змінювати структуру моделі під час виконання програми. Зокрема, PyTorch є найпоширенішим інструментом для створення та навчання нейронних мереж, але також надає інтерфейс для загальних математичних обчислень, подібний до *Numpy*.

10. *SciPy* — це бібліотека з відкритим вихідним кодом на мові Python для наукових та технічних обчислень, яка додає нові можливості для обробки даних створених в *Numpy*. Бібліотека містить модулі для

чисельного інтегрування, оптимізації, обробки сигналів та статистичного аналізу. *SciPy* використовується в задачах, де потрібні готові, швидкі реалізації класичних методів математичної статистики, які доповнюють сучасні методи машинного навчання. Недоліком бібліотеки є неможливість виконання обчислень на апаратних прискорювачах.

11. *TensorBoard* — це інструмент для візуалізації та моніторингу процесів навчання моделей машинного навчання. Інструмент дозволяє в реальному часі відстежувати зміну метрик моделі, аналізувати гістограми вагів нейронних мереж та візуалізувати структуру графа моделей.

Критичним аспектом проведення експериментів у задачах навчання з підкріпленням є контроль стохастичності. У розробленій системі джерелами випадковості виступають: (1) процедурна генерація середовища Minigrid (розташування стін, ключів, дверей та інших об'єктів), (2) ініціалізація ваг нейронних мереж, (3) семплювання дій агента та (4) стохастична генерація токенів LLM.

Було впроваджено стратегію роздільної генерації псевдовипадкових чисел для ініціалізації середовища. На етапі навчання, на якому збираються дані траєкторій та навчається алгоритм PPO, параметр seed для ініціалізації середовища не фіксується. Це гарантує, що агент на кожному епізоді взаємодіє з новою конфігурацією кімнати, що змушує його вивчати загальні правила взаємодії, а не запам'ятовувати маршрут. На етапі оцінки, для коректного порівняння різних архітектур та промптів необхідно забезпечити ідентичні умови тестування. Для цього було впроваджено окремий параметр конфігурації eval_seed. На етапі тестування, замість оцінки на одній випадковій траєкторії, що дає високу дисперсію результатів, було згенеровано фіксований набір із $N = 100$ унікальних початкових значень значень (seeds). Цей набір було отримано за допомогою генератора numpy.random.default_rng з фіксованим початковим станом, що забезпечує відтворюваність тестового набору на будь-якій обчислювальній машині.

Такий підхід дозволив розділити вплив навчання агента та випадкової генерації середовища на фінальні результати. Фінальна оцінка ефективності розраховується як середнє значення метрик по 100 фіксованим епізодам.

Для уникнення втрати даних та конфліктів при паралельних експериментах було реалізовано автоматизовану систему індексації запусків. Система аналізує

наявні директорії логів і автоматично призначає унікальний ідентифікатор експерименту, що гарантує збереження повної історії навчання для подальшого аналізу.

Однією з ключових методологічних проблеми при дослідженні впливу промптів у гібридних системах LLM+RL є висока стохастичність процесу навчання з підкріпленням. При навчанні агента з кожним новим промптом важко визначити, чим зумовлена зміна ефективності: якістю самого промпта чи випадковими факторами під час градієнтного спуску (ініціалізація ваг, порядок семплювання батчів). Крім того, повний цикл навчання PPO є обчислювально витратним, що сповільнює перевірку гіпотез.

Для вирішення цієї проблеми та забезпечення сталості умов для порівняння різних промптів було використано методику Fixed Decision Policy (FDP). Суть методу полягає в декомпозиції навчання на два незалежні етапи. На першому етапі відбувається тренування компонента Актор. На цьому етапі обирається базовий промпт та фіксована архітектура LLM. Агент навчається у середовищі протягом фіксованої кількості ітерацій ($N = 1000$), оптимізуючи ваги комунікаційної стратегії за допомогою алгоритму PPO. Мета цієї фази — отримати Актора, який здатний ефективно виконувати план, який генерує Планувальник. Після завершення навчання ваги стратегії $\pi_{\theta\ RL-Actor}$ зберігаються та, в подальшому, використовуються без змін. На другому етапі відбувається валідація планувальника. У цій фазі завантажується збережена стратегія $\pi_{\theta\ RL-Actor}$. Ваги нейронної мережі залишаються незмінними. Зміни відбуваються в модулі Планувальник, в якому змінюється мовні інструкції або гіперпараметри генерації LLM.

Застосування цього підходу дає наступні важливі переваги. Оскільки параметри модуля RL-Actor на етапі валідації залишаються незмінними, то будь-які зміни у метриках агента є прямим наслідком зміни якості планування, зумовленої новим промптом. По-друге, це дозволило суттєво скоротити час валідації однієї гіпотези. Замість повного циклу навчання, який триває близько трьох годин, тестування фіксованої стратегії триває 30 секунд для FDP на GPU Nvidia RTX 3090. По-третє, цей підхід дозволяє багаторазово використовувати одну й ту ж саму навчену модель для тестування широкого спектру промптів без ризику деградації стратегії.

Реалізація FDP вимагала модифікації архітектури запуску системи,

розділення режимів роботи на тренування (навчання та збереження комунікаційної стратегії) та валідацію (завантаження комунікаційної стратегії та тестування). Експериментально підтверджено, що стратегія, навчена на одному базовому промпті, успішно узагальнюється на інші типи інструкцій, за умови збереження загальної семантики команд високого рівня.

## 2.5 Експериментальне дослідження впливу стратегій формування контексту LLM на ефективність агента

Протягом навчання підлягали оптимізації параметри однієї з нейромереж агента — стратегії комунікації. Ця стратегія приймає рішення, чи потрібно згенерувати новий план дій, а для генерації плану викликається мовна мережа. Обчислення мовної мережі потребує значних обчислювальних ресурсів та часу, порівняно з іншими компонентами системи, тому зниження кількості викликів мовної моделі призводить до зниження тривалості кожної ітерації. Комунікаційна стратегія була навчена за допомогою методу навчання з підкріпленням PPO [84]. У результаті навчання, мовна модель викликалася лише в тих випадках, коли генерація нового плану призводила до зростання очікуваної винагороди. У той же час параметри LLM залишалися незмінними.

Навчання тривало протягом 1000 ітерацій, де кожна ітерація відповідала одному епізоду середовища. З рисунка 2.7 бачимо, що середня винагорода протягом епізоду з ходом тренування зростала до середнього значення 0.6. На рисунку 2.8 показана тривалість навчальних епізодів. Зазначимо, що епізод не може тривати більше ніж 100 ітерацій, тому після 100 ітерацій в одному епізоді агент вважається таким, що не виконав завдання. З цього рисунка ми бачимо, що середнє значення коливається навколо тривалості епізоду у 30 кроків, що свідчить про здатність агента успішно виконувати завдання у більшості випадків менш ніж за 100 ітерацій. На рисунку 2.9 показано середню кількість викликів мовної моделі для кожного епізоду (ітерації) під час навчання. З цього рисунка видно, що кількість викликів мовної моделі швидко зменшується і до кінця навчання досягає значення у три виклики на епізод.

Навчений агент був використаний для тестування різних технік створення промптів. Тестування проводилося на 100 екземплярах середовища «Simple Door Key», кожному екземпляру відповідає унікальне випадкове початкове

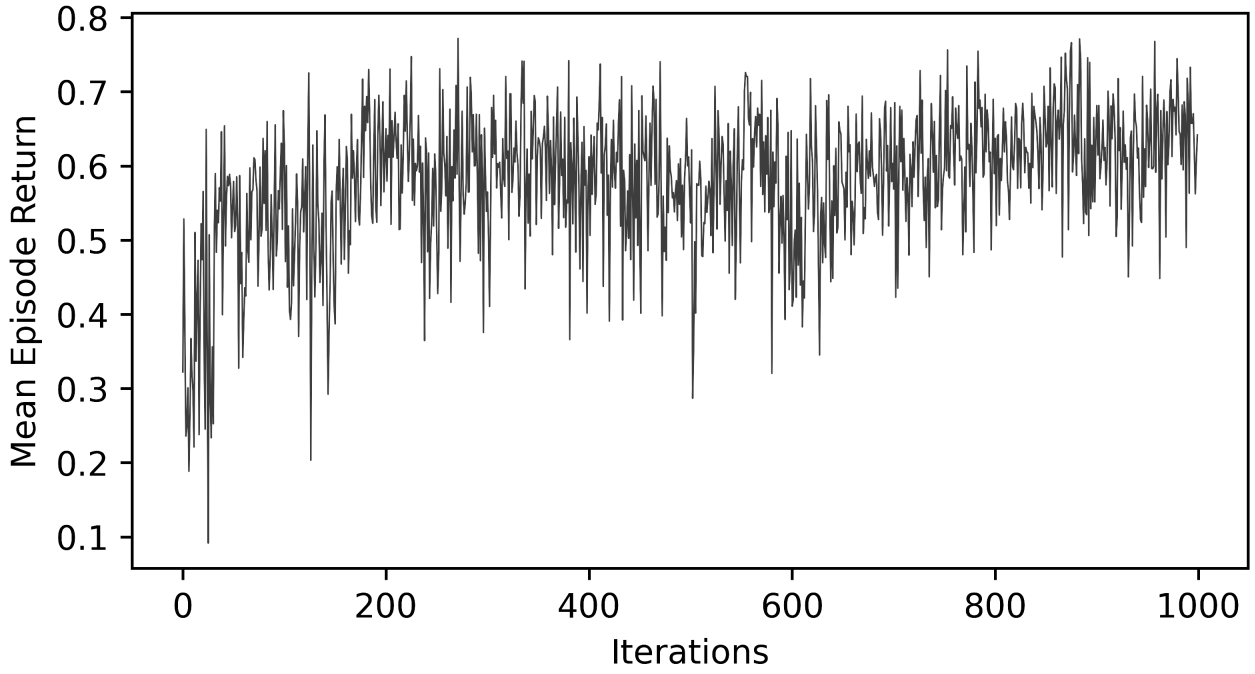



Рисунок 2.7: Графік залежності середнього повернення від кількості кроків тренування

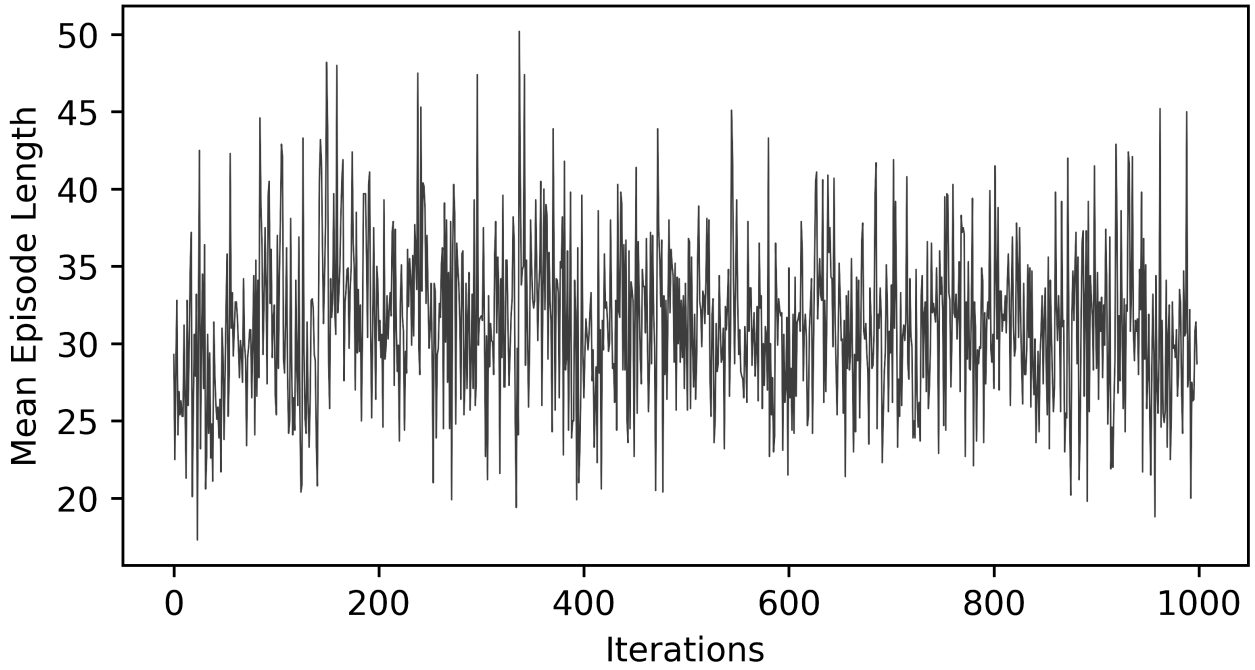



Рисунок 2.8: Графік залежності середньої тривалості епізоду від кількості кроків тренування

значення. Були використані наступні метрики: повернення, повернення з урахуванням штрафу за виклик мовної мережі, кількість викликів LLM, середня тривалість епізоду.

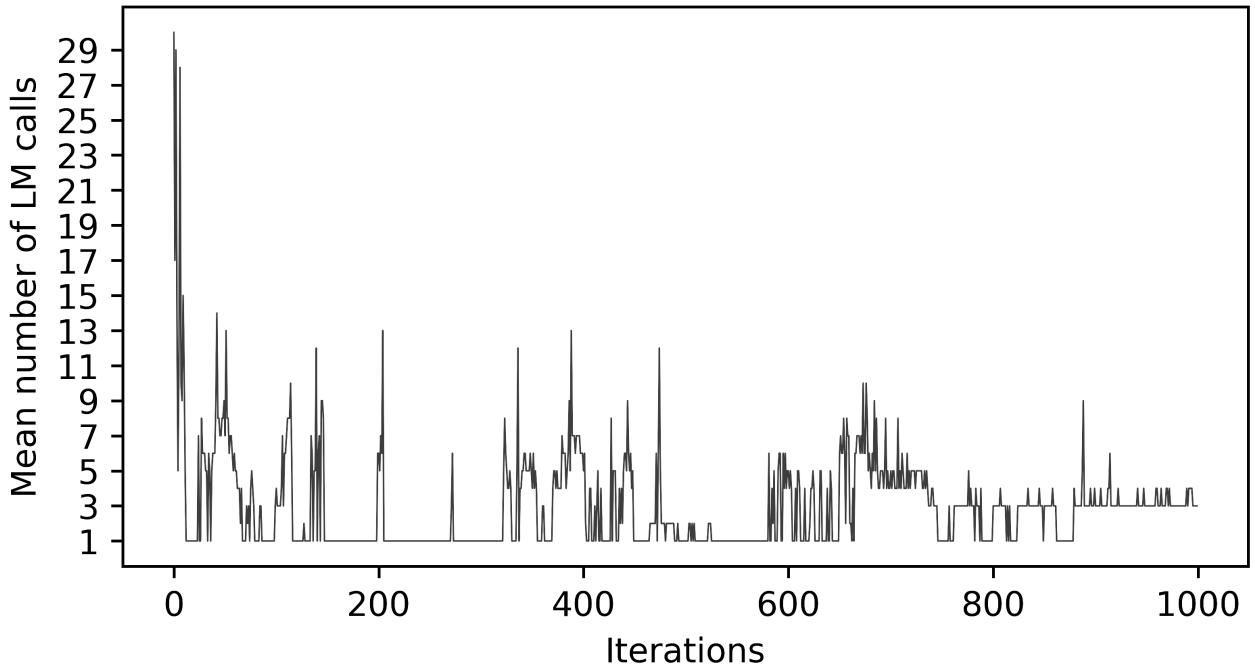



Рисунок 2.9: Графік залежності кількості викликів LLM за епізод від кількості кроків тренування

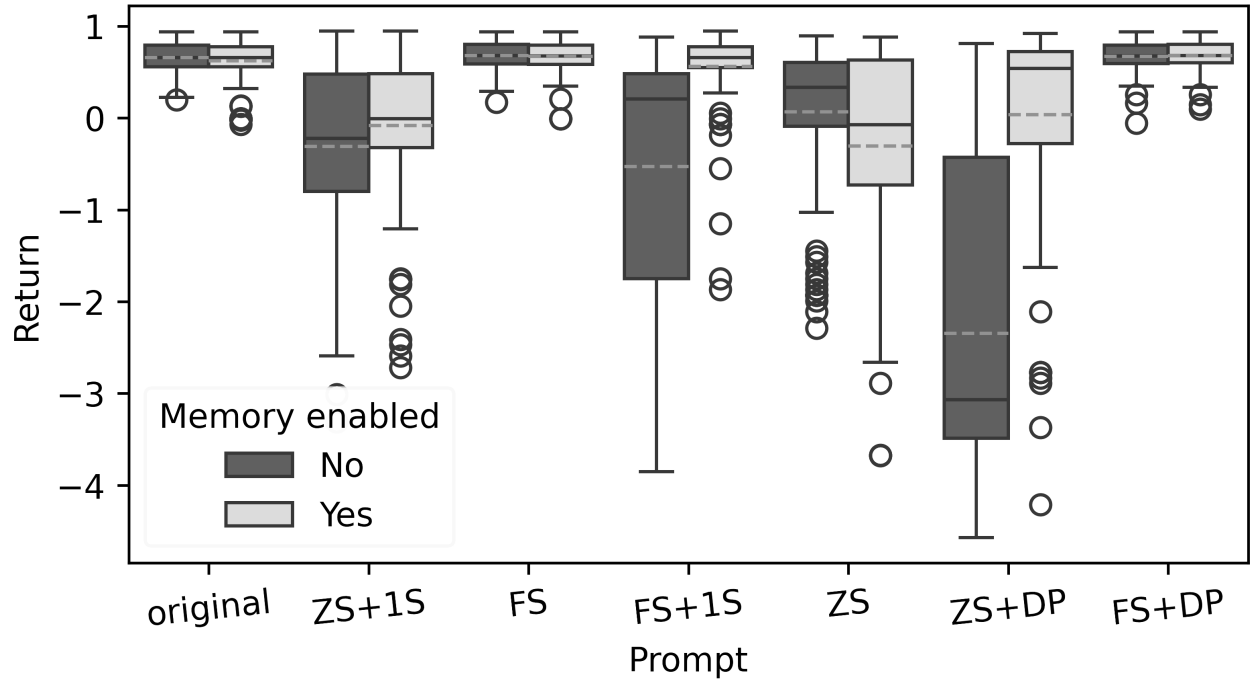



Рисунок 2.10: Залежність квартильних значень повернення від промпту. Умовні позначення: original — промпт з роботи [5], ZS — zero-shot, FS — few-shot, 1S — one step, DP — default plan

На рисунку 2.10 зображено квартильні значення повернення для різних промптів з увімкненою та вимкненою пам’яттю. З рисунку бачимо, що для

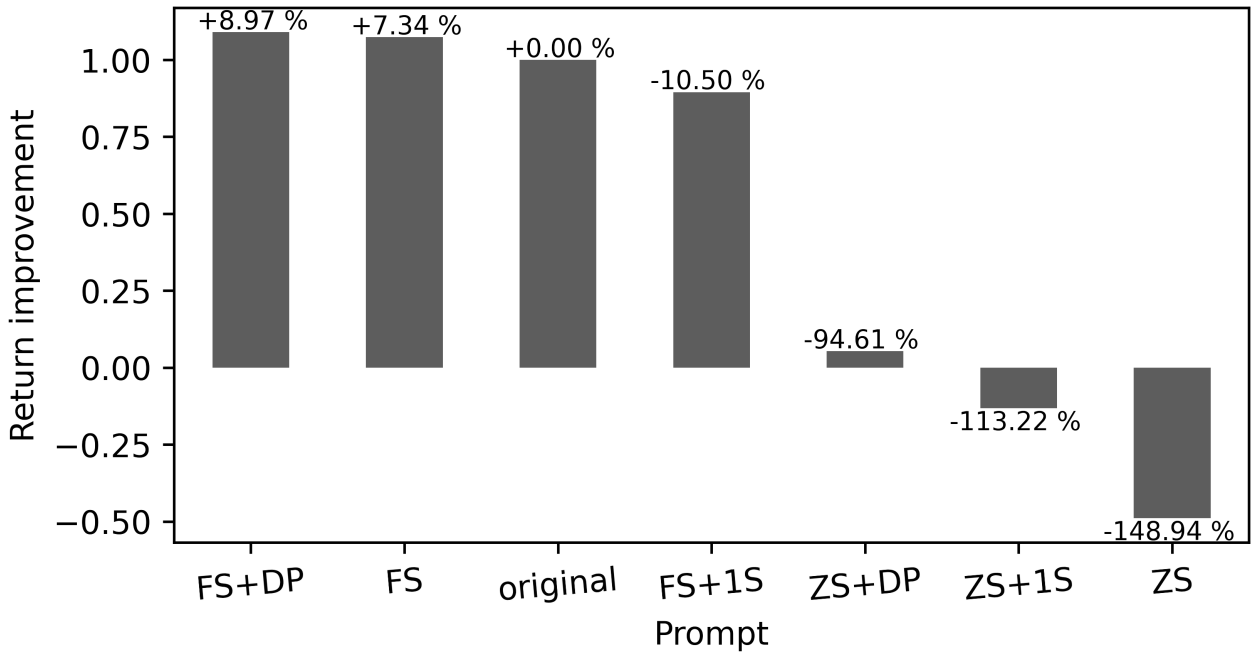



Рисунок 2.11: Залежність середнього повернення від промпту для агента з увімкненою пам’яттю. Умовні позначення такі ж, як на рис. 2.10

промптів original, few-shot, few-shot default plan застосування пам’яті не дає помітних покращень. У випадку з промптом «few shot one step» пам’ять дозволяє агенту запам’ятовувати стан середовища між окремими ітераціями, на яких агент може обрати лише одну дію. Значне покращення з увімкненою пам’яттю спостерігається у випадку промпта «zero shot default plan», що пояснюється тим, що агент отримує доступ до невдалих дій в поточному епізоді, що допомагає правильно застосовувати план за замовчуванням.

На рисунку 2.11 бачимо порівняння повернення для різних промптів з увімкненою пам’яттю з відсотковим покращенням відносно оригінального промпту [5]. Виправлення граматики промпту та додавання плану за замовчуванням дало покращення на 7.34% та 8.97% відповідно. У випадку, коли агент міг обирати лише одну дію за раз, погіршення становило -10.5%. Промпти категорії zero shot показали значне падіння сумарної винагороди аж до негативних значень через численні виклики LLM і отримання штрафів за них. Ці результати показують, що приклади рішення завдання значно підвищують повернення, отримане агентом.

Графік середньої тривалості епізоду (рис. 2.12) також демонструє перевагу промптів типу few-shot над оригінальним промптом та промптами без прикладів (zero-shot).

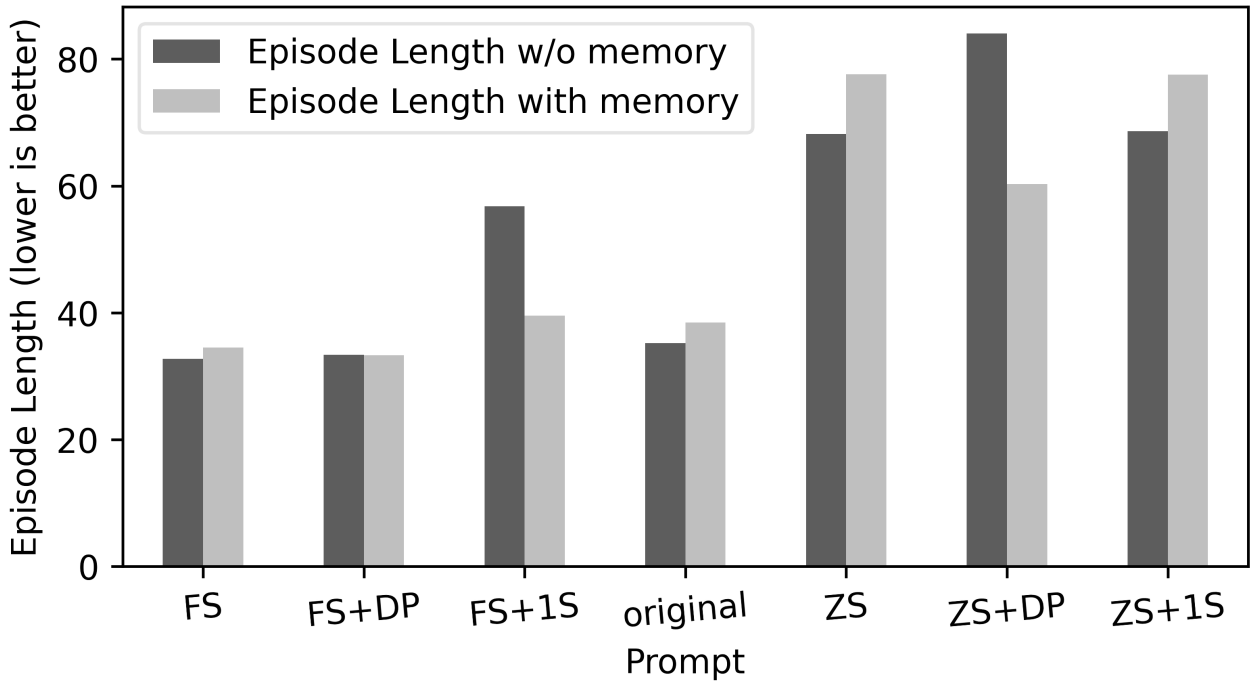



Рисунок 2.12: Залежність середньої довжини епізоду від промпту для увімкненої та вимкненої пам’яті. Умовні позначення такі ж, як на рис. 2.10

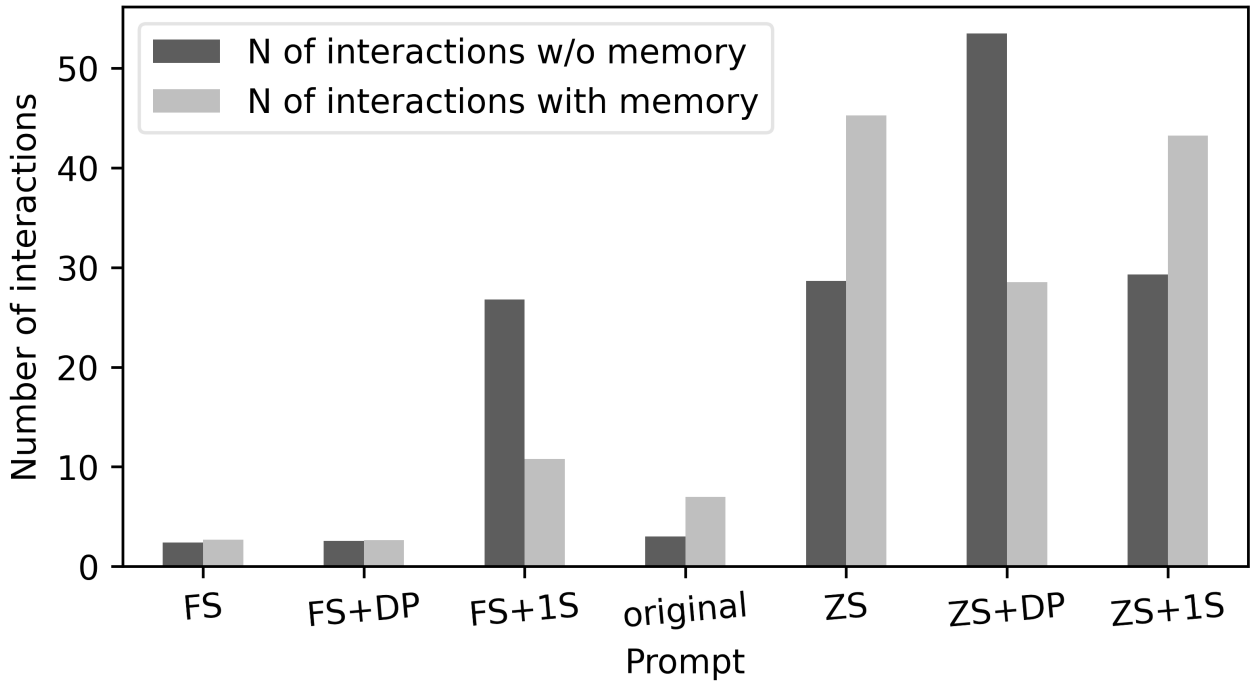



Рисунок 2.13: Залежність середнього числа викликів LLM від промпту для увімкненої та вимкненої пам’яті. Умовні позначення такі ж, як на рис. 2.10

Рисунок 2.13 показує залежність кількості викликів мовної моделі від промпту. За відсутності прикладів (zero-shot) агент набагато частіше викликає мовну модель ніж у випадках, коли агенту доступні приклади (few-shot).

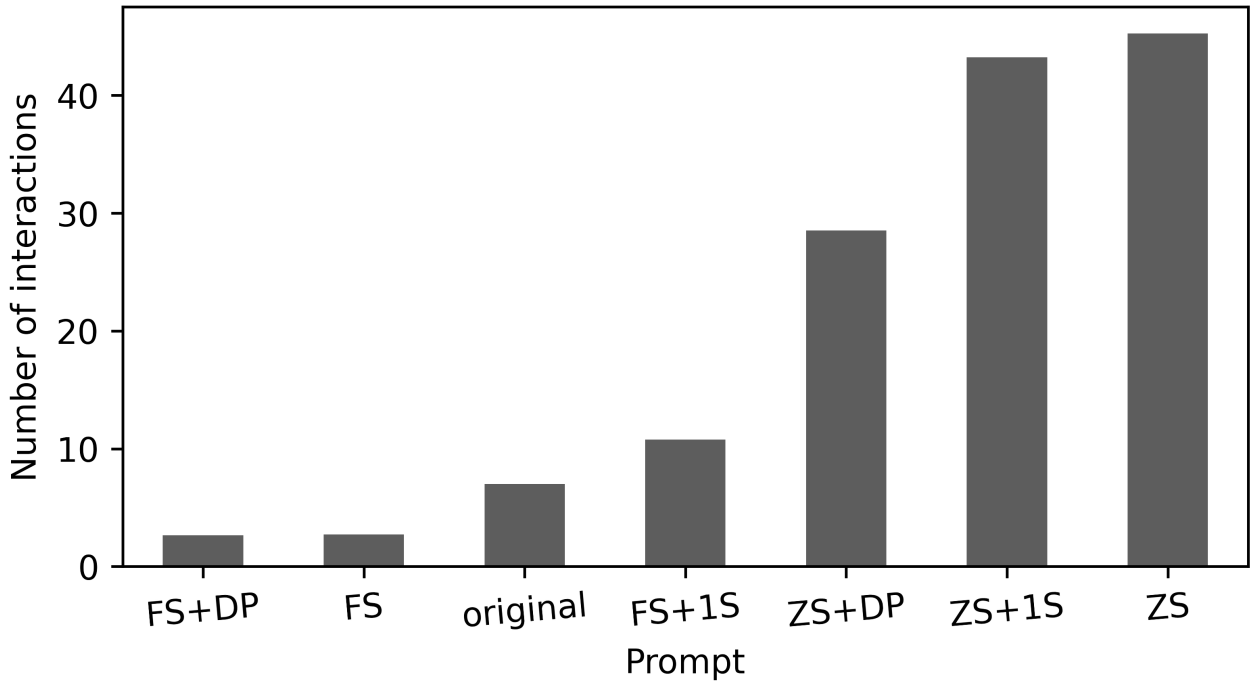



Рисунок 2.14: Залежність середнього числа викликів LLM від промпту. Умовні позначення такі ж, як на рис. 2.10

На рисунку 2.14 показано залежність кількості викликів мовної моделі в залежності від промпта, відранжована за зростанням кількості викликів. Можемо побачити, що найкращого результату досяг агент з промптом, який містить приклади вирішення завдання і план за замовчуванням.

## 2.6 Якісний аналіз поведінкових патернів та обмежень стохастичної генерації

Кількісний аналіз, наведений у попередньому підрозділі, демонструє загальну ефективність використаних методів, проте не розкриває причин прийняття агентом тих чи інших рішень. Для глибшого розуміння обмежень великих мовних моделей у задачах планування було проведено якісний аналіз логів взаємодії.

У ході експериментів було ідентифіковано та класифіковано три основні групи помилок, що виникають при генерації планів: синтаксичні, семантичні та логічні помилки.

У випадку синтаксичних помилок LLM генерує текст, який не відповідає очікуваному формату, що призводить до неможливості синтаксичного аналізу модулем Медіатор. Помилки, в яких LLM генерує дії над об’єктами, які відсутні

у поточному контексті — це семантичні галюцинації. Нездатність LLM знайти в контексті попередній стан виконання задачі (наприклад, інформацію про предмет в інвентарі) породжує логічні помилки.

Дослідження показало високу чутливість LLM до дрібних деталей форматування промпту. В оригінальному промпті відсутність об'єктів позначалася як `«observed: 'nothing'»`. Модель інтерпретувала слово в лапках як іменник, що позначає фізичний об'єкт. Це призводило до генерації плану `«action: go to nothing»`. Оскільки об'єкта з ідентифікатором `«nothing»` не існує в базі об'єктів Медіатора, система повертала координати за замовчуванням `(0, 0)`, за якими завжди знаходиться стіна. Це викликало помилку алгоритму пошуку шляху. Видалення лапок у промпті усунуло цей клас помилок.

Виявлено, що використання фрази `«carry key»` (імперативна форма) замість `«carrying key»` (дієприкметник стану) у вхідному промпті часто призводило до того, що модель ігнорувала наявність ключа в інвентарі. Виправлення граматичного часу дієслова знизило кількість повторних спроб підняти предмет, який агент вже тримає.

Навіть при коректному сприйнятті стану, стохастична природа генерації інколи призводить до зациклювань. Найпоширенішою помилкою (близько 15% невдалих планів для малих моделей) є спроба виконати дію `«pick up key»`, коли статус агента вже містить інформацію, що ключ підібрано: `«carrying: key»`. Це свідчить про те, що менші мовні моделі (менше 7B) мають труднощі з перевіркою передумов дій. Додавання до промпту окремого блоку передумов (наприклад, `«Do not pick up if already carrying»`) або використання few-shot прикладів з негативними обмеженнями дозволило частково компенсувати цей недолік, змушуючи модель звертати увагу на стан інвентаря перед плануванням.

## Висновки до розділу 2

У другому розділі досліджено вплив структури мовних інструкцій на ефективність прийняття рішень автономним LLM-агентом. Для проведення експериментів підготовлено програмну систему на основі віртуального середовища Minigrid SimpleDoorKey, яке імітує задачі просторової навігації та взаємодії з об'єктами. Сформовано набори мовних інструкцій з різною

структурою для керування поведінкою агента. Параметри LLM залишалися незмінними, проводилася оптимізація виключно комунікаційної стратегії агента. Навчання комунікаційної стратегії дозволило знизити частоту звернень до LLM, досягнувши показника три виклики на епізод.

Проведено експериментальну перевірку навченого агента на фіксованому тестовому наборі зі 100 епізодів у середовищі, під час яких агент отримував різні мовні інструкції. Використання методу навчання в контексті (In-Context Learning), зокрема промптів з прикладами (few-shot), показало результати вищі, ніж у агента з промптами без прикладів (zero-shot). Агент з промптами без прикладів продемонстрував значне падіння сумарної винагороди, аж до негативних значень, що спричинено надмірною кількістю викликів LLM та отриманням за це штрафи. Результати демонструють високу чутливість агента до граматичної коректності інструкцій, що підтверджується покращенням показника середньої винагороди на 7.34% при переході від початкового промпту з контекстними прикладами до варіанту з виправленими граматичними помилками. Інтеграція плану дій за замовчуванням до промпту з контекстними прикладами дала додатковий позитивний ефект, збільшивши середню винагороду на 1.63%. Також показано, що генерація багатокрокових планів є значно ефективнішим підходом ніж покрокове планування, обмеження горизонту планування до одного кроку призвело до погіршення ефективності агента на 10.5%. Наявність епізодичної пам’яті виявилася вагомою перевагою лише для окремих конфігурацій, зокрема за умови обмеження планування одним кроком та за використання базового промпту з планом за замовчуванням. Слабкий вплив пам’яті у інших випадках пояснюється тим, що промпт агента вже містив інформацію про стратегію вирішення задачі, що мінімізувало потребу у використанні історії.

У процесі дослідження було виявлено принциповий недолік генерації плану дій агента за допомогою LLM без додаткового контролю. За такого підходу є можливим випадкове порушення синтаксису плану, що робить неможливим подальший аналіз і використання згенерованого плану. Цю проблему дозволяють вирішити додаткові методи контролю генерації плану, які досліджуються у розділі 3.

Основні наукові результати розділу опубліковані в роботі [90].

# 3 Дослідження впливу декодування з обмеженням граматики у мовних моделях на коректність планування дій агентів у віртуальних середовищах

## 3.1 Математична модель обмеженого декодування для простору дій агента

Нехай $V$ — скінченний словник токенів. Мовна модель з параметрами $\theta$ визначає умовний розподіл

$$p_{\theta\ LLM}(w_{i+1} \mid w_{1:i}),$$

який описує ймовірність наступного токена $w_{i+1} \in V$, який слідує за префіксом $w_{1:i} \in V^i$, де $V^i$ — множина усіх можливих префіксів довжиною $i$, $w_{1:i} = (w_1, \ldots, w_i) \in V^i$.

У випадку необмеженої генерації мовна модель обчислює розподіл імовірностей токенів як функцію softmax від логітів [91]:

$$p_{\theta\ LLM}(w_{i+1} \mid w_{1:i}) = \frac{\exp(\ell_{\theta\ LLM}(w_{i+1} \mid w_{1:i}))}{\sum_{v \in V} \exp(\ell_{\theta\ LLM}(v \mid w_{1:i}))},$$

де логіт $\ell_{\theta\ LLM}(w \mid w_{1:i})$ — вихід останнього шару мовної нейромережі для довільного токена $w_{i+1}$. Зауважимо, що права частина наведеної рівності є визначенням функції $\mathrm{softmax}(\boldsymbol{\ell})$ для вектора логітів $\boldsymbol{\ell}$.

На кожному кроці $i$ з умовного розподілу $p_{\theta\ LLM}(\cdot)$ обирається один з токенів. Існують різні способи вибору токенів, найпростіший спосіб — вибір токена з максимальною ймовірністю. Процедура вибору токену також може включати температуру $\tau$ — параметр, який визначає «розмивання» розподілу імовірностей токенів, що призводить до зростання імовірності вибору малоймовірних токенів, та зниження для високоймовірних. Нижче значення температури призводить до генерації більш детермінованих послідовностей, натомість вища температура призводить до генерації більш різноманітних послідовностей. При граничному значенні температури $\tau = 0$ ненульова

ймовірність залишається лише для вибору початково найбільш ймовірного токена/токенів і генерація стає детерміністичною.

Декодування на кроці $i$ можна записати у такому вигляді:

$$\tilde{w}_{i+1} \sim p_{\theta\ LLM}^{(\tau)}(w_{i+1} \mid w_{1:i}) \quad \text{(випадковий вибір)}, \tag{3.1}$$

$$\tilde{w}_{i+1} = \arg\max_{w \in V} p_{\theta\ LLM}^{(\tau)}(w_{i+1} \mid w_{1:i}) \quad \text{(детерміністичний вибір)}. \tag{3.2}$$

Для задач зі строгою структурою, таких, як задачі планування, нижче значення температури зменшує ймовірність генерації малоймовірних токенів, які можуть порушити структуру. У випадку ймовірнісного вибору токенів, мовну модель можна зробити детерміністичною за рахунок фіксування ініціалізації генератора випадкових чисел, при цьому для фіксованого входу, мовна модель породжуватиме фіксований результат. В такому випадку мовна модель може бути представлена як детерміністична математична функція $s_{\text{out}} = g_{\theta\ LLM}(s_{\text{in}}; r_{rand})$, де $s_{\text{in}}$, $s_{\text{out}}$ — довільні вхідна та вихідна послідовності токенів відповідно, $r_{rand}$ — значення ініціалізації псевдовипадкового генератора чисел. Для фіксованого $r_{rand}$ та фіксованого вхідного префіксу модель генерує детермінований результат.

Для вирішення завдання в середовищі агенту потрібно послідовно обирати та виконувати дії, які ведуть до необхідної цілі. Дії в середовищі мають послідовну природу — успішність виконання наступних дій залежить від результату виконання попередніх дій. Задачу планування можна представити наступним чином. Нехай агент працює у середовищі з дискретним часом $t = 0, 1, 2, \ldots$. Мовна модель в момент часу $t$ приймає на вхід промпт $s_{\text{prompt}}$ та спостереження $o_t^{(text)}$ і обчислює нову текстову послідовність $s_{\text{out},t}$ яка є кортежем $(s_{\text{reasoning},t}, s_{\text{plan},t})$, де $s_{\text{reasoning},t}$ — рядок, який містить міркування моделі та може бути пустим, $s_{\text{plan},t}$ — рядок, який містить план дій, сформований на основі інструкцій в промпті $s_{\text{prompt}}$ та спостереження $o_t^{(text)}$.

У випадку, коли мовна модель чітко слідує граматиці плану, згенерована послідовність виглядатиме наступним чином:

$$s_{\text{plan},t} = (a_{t,1}, a_{t,2}, \ldots, a_{t,n_t}),$$

де кожна дія $a \in \mathcal{A}$, де $\mathcal{A}$ — множина дій, допустимих в середовищі, $n_t$ — довжина плану на кроці t. Для кожної дії визначена сигнатура (типи параметрів)

$$\text{sig} : \mathcal{A} \to \Sigma, \qquad \Sigma \text{ — множина сигнатур (шаблонів параметрів).}$$

Задача генерації плану накладає структурні обмеження на згенерований результат $s_{\text{out}}$. По-перше, план має бути представлений як послідовність окремих дій, для яких важливе врахування порядку. По-друге, в кожному конкретному середовищі множина допустимих дій може відрізнятися і кожна дія має власну сигнатуру — назву та набір параметрів. Таким чином з'являється вимога до мовної моделі, щоб процес генерації породжував послідовність з структурою, яка включає послідовність дій, а кожна дія відповідала одній з сигнатур допустимих дій.

При виборі наступного токену відповідно до розподілу імовірностей, обраний токен може порушити схему плану. Для вирішення цієї проблеми застосуємо метод Grammar-Constrained Decoding (GCD), який обмежує вибір токенів лише з множини, яка не порушує граматику.

У цій роботі ми досліджуємо вплив методу декодування з обмеженням граматики Grammar-Constrained Decoding (GCD) на планування дій агента за допомогою мовних моделей у віртуальному середовищі Minigrid.

Позначимо через $G_{gram}$ контекстно-вільну граматику, яка задає допустимі текстові послідовності планів, позначимо мову як $L(G_{gram}) \subseteq V^*$, де $V^*$ — множина усіх можливих рядків, утворених з токенів з множини $V$. На кроці $i$ для префіксу $\pi = w_{1:i}$ введемо множину «дозволених» токенів

$$\mathcal{C}(\pi) \;=\; \{\, w \in V \mid \exists \sigma \in V^* :\; \pi \cdot w \cdot \sigma \in L(G_{gram}) \,\},$$

тобто токен $w$ дозволений, якщо існує деяке продовження послідовності $\sigma$, таке що конкатенація послідовностей $\pi \cdot w \cdot \sigma$ належить мові граматики $L(G_{gram})$.

Тоді Grammar-Constrained Decoding (GCD) обмежує можливі токени на кроці $i$ множиною $\mathcal{C}(w_{1:i})$. Після застосування температури отримаємо масковані логіти:

$$\ell'^{(\tau)}_{\theta\ LLM}(w \mid w_{1:i}) = \begin{cases} \ell^{(\tau)}_{\theta\ LLM}(w \mid w_{1:i}), & \text{якщо } w \in \mathcal{C}(w_{1:i}) \\ -\infty, & \text{інакше.} \end{cases}$$

Розподіл імовірностей для токенів отримуємо шляхом застосування функції softmax до маскованих логітів [92]:

$$p^{(\tau,\mathcal{C})}_{\theta\ LLM}(w \mid w_{1:i}) = \frac{\exp(\ell'^{(\tau)}_{\theta\ LLM}(w \mid w_{1:i}))}{\sum_{v \in V} \exp(\ell'^{(\tau)}_{\theta\ LLM}(v \mid w_{1:i}))}, \tag{3.3}$$

Декодування виконується шляхом вибору наступного токена з маскованого розподілу (3.3) випадково (3.1) або детерміновано (3.2).

При необмеженому декодуванні (Unconstrained Decoding, UCD) обчислювальна складність вибору наступного токена на етапі інференсу є порівняно низькою. Нейронна мережа генерує вектор логітів $\boldsymbol{\ell} \in \mathbb{R}^{|V|}$, після чого застосовується функція Softmax та операція вибірки або пошуку максимуму (arg max). Ці операції підлягають розпаралелюванню і виконуються безпосередньо на тензорних ядрах графічного прискорювача.

У випадку GCD, алгоритм має обчислити вектор маски $\mathbf{m}_i \in \{0, -\infty\}^{|V|}$ для кожного кроку генерації $i$. Елемент маски $m_{i,j}$ дорівнює 0, якщо токен $w_j \in V$ не порушує граматику в поточному кроці $i$, і $-\infty$ у протилежному випадку.

У [92] зазначено, що для того щоб визначити валідність токена $w_j$, GCD має перевірити послідовність символів, з яких складається цей токен. Нехай $L_j$ — кількість символів у токені $w_j$. У найгіршому випадку перевірка одного токена вимагає $\mathcal{O}(L_j)$ операцій. Відповідно, асимптотична складність формування маски для всього словника на одному кроці генерації становить:

$$\mathcal{O}\left(\sum_{j=1}^{|V|} L_j\right) \approx \mathcal{O}(|V| \times \bar{L}), \tag{3.4}$$

де $\bar{L}$ — середня довжина токена у символах.

Різні сімейства мовних моделей використовують різні алгоритми токенізації (наприклад, Byte-Pair Encoding, BPE), що призводить до суттєвої різниці у розмірах їхніх словників. Серед досліджуваних моделей існує значна

розбіжність за цим параметром. Моделі сімейства Llama 3.2 використовують словник розміром $128256$ токенів. Моделі сімейства Qwen3 оперують значно більшим простором — $151936$ токенів.

Оскільки складність алгоритму лінійно залежить від розміру словника $|V|$, генерація маски для моделей Qwen3 вимагає приблизно на 18% більше обчислень порівняно з Llama 3.2.

## 3.2 Адаптація алгоритму Grammar-Constrained Decoding для задачі послідовного прийняття рішень

На вищому рівні абстракції архітектури агента простір допустимих дій та структура плану описуються за допомогою об'єктно-орієнтованої парадигми. Для цього використовується бібліотека Pydantic, яка дозволяє описувати та валідувати дані шляхом використання анотацій типів в мові програмування Python. На цьому етапі задається початковий простір допустимих структур, який легко читається людиною та придатний для статичного аналізу. Цей етап повністю ізольований від особливостей роботи LLM.

Оскільки система інференсу LLM Ollama імплементована на низькорівневих мовах програмування (C/C++, Go) та не підтримує пряму роботу з об'єктами Python, декларативна модель даних автоматично транслюється у JSON-схему. Схема діє як строгий контракт обміну даними, які передаються через API до локального сервера інференсу.

JSON-схема є занадто високорівневою абстракцією для посимвольного аналізу під час генерації тексту, тому рушій інференсу (на рівні імплементації *llama.cpp*) динамічно перетворює отриману JSON-схему у граматику GBNF (GGML Backus-Naur Form), яка розширює класичну форму Бекуса-Наура підтримкою регулярних виразів. На цьому етапі логічні обмеження (наприклад, масив від 1 до 7 елементів типу *enum*) перетворюються на синтаксичні правила валідації рядків. Система інференсу локальних LLM Ollama за замовчуванням підтримує такий механізм трансляції JSON-схем у GBNF.

На основі отриманої GBNF-граматики, система будує детермінований скінченний автомат або автомат з магазинною пам'яттю (Pushdown Automaton, PDA) для обробки вкладених структур JSON. Нехай автомат формалізовано як $A = (Q, \Sigma, \delta, q_0, F)$, де $Q$ — множина станів парсера, $\Sigma$ — алфавіт символів, а

$\delta : Q \times \Sigma \to Q$ — функція переходів [93].

На кожному кроці авторегресивної генерації $i$, автомат перебуває у стані $q_i$. Словник токенів моделі $V$ проектується на граф переходів автомата. Якщо токен $w \in V$ ініціює перехід, для якого функція $\delta$ не визначена (тобто токен порушує граматику JSON-схеми), він виключається з множини допустимих токенів $\mathcal{C}(w_{1:i})$.

Математично це реалізується через модифікацію вихідного вектора логітів $\boldsymbol{\ell}$ перед застосуванням функції Softmax. Невалідному токену $w$ примусово присвоюється значення від'ємної нескінченності:

$$\ell'_{\theta\ LLM}(w \mid w_{1:i}) = \begin{cases} \ell_{\theta\ LLM}(w \mid w_{1:i}), & \text{якщо } w \in \mathcal{C}(w_{1:i}) \\ -\infty, & \text{інакше.} \end{cases} \tag{3.5}$$

Це гарантує, що ймовірність генерації синтаксично хибного токена строго дорівнюватиме нулю, гарантуючи надійність формату виводу без необхідності пост-обробки тексту.

Деякі моделі, зокрема ті, які оптимізовані за допомогою навчання з підкріпленням (наприклад, сімейство DeepSeek-R1), схильні генерувати надмірно довгі міркування перед формулюванням остаточного результату. При використанні методу необмеженого декодування (UCD), навіть за наявності прямих інструкцій у промпті не генерувати міркування, моделі часто ігнорують цю вказівку. Ця поведінка може призвести до перевищення обмежень на довжину контексту (наприклад, 4096 токенів), що призводить до примусового переривання генерації до того, як модель встигне сформувати план дій.

Розгортання LLM вимагає значних обчислювальних ресурсів та обсягу пам'яті. Методи квантування LLM [94, 95] дозволяють зменшити об'єм пам'яті необхідний для завантаження моделей ціною зменшення точності параметрів моделі та загальної якості роботи. У цій роботі були використані моделі квантовані методом Q4_K_M, який належить до загального методу Post-Training Quantization (PTQ). У методі PQT квантування застосовується до заздалегідь навчених LLM та не потребує подальшого донавчання. Інструмент Ollama, який використовує *llama.cpp* для обчислення LLM, за замовчуванням підтримує завантаження LLM квантованих методом PTQ.

Для експериментів було обрано метод квантування Q4_K_M, оптимальний

за співвідношенням зменшення розміру LLM та збереження точності роботи. Метод Q4_K_M належить до сімейства методів K-quants — методів які використовують ієрархічну структуру блоків для оптимізації зберігання параметрів квантування та зменшення втрати точності. Метод Q4_K_M працює наступним чином. Ваги моделі об'єднуються у супер-блоки розміром 256 параметрів в кожному. Кожен супер-блок поділяється на 8 підблоків (32 параметри в кожному). На рівні супер-блоку зберігаються глобальні параметри у форматі FP16 — масштабний множник (scale factor) $\Delta_{sb}$ та зміщення $m_{sb}$. На рівні підблоку $j \in [1, N_{sub}]$ зберігаються локальні квантовані параметри — 6-бітний масштаб $q_{\Delta,j}$ ($k_s = 6$) та 6-бітне зміщення $q_{m,j}$ ($k_m = 6$). Значення параметрів моделі квантуються як 4-бітні цілі числа. Завдяки такому підходу, в середньому на збереження одного параметру моделі припадає 4.5 біт, порівняно з 16 або 32 біт на параметр, необхідних для збереження не квантованої моделі.

Нехай $W \in \mathbb{R}^{256}$ — це вектор параметрів моделі який позначає супер-блок. Розіб'ємо його на $N_{sub} = 8$ неперервних підблоків $w^{(j)} \in \mathbb{R}^{32}$, де $j \in \{1, \ldots, 8\}$. Для всього вектора $W$ обчислюються та зберігаються у форматі FP16 глобальний масштабний множник (scale) $\Delta_{sb} \in \mathbb{F}_{16}$ та глобальне зміщення $m_{sb} \in \mathbb{F}_{16}$. Для кожного підблоку $w^{(j)}$ обчислюються два значення: локальний дійсний масштаб $s_j$ та мінімум $m_j$. Вони квантуються до 6-бітних цілих значень за допомогою глобальних параметрів масштабу підблоку: $q_{\Delta,j} \in [0, 63]$ ($k_s = 6$ біт) та міщення підблоку: $q_{m,j} \in [0, 63]$ ($k_m = 6$ біт).

Кожен параметр моделі $w_i \in w^{(j)}$ відображається у 4-бітний дискретний простір. Оскільки алгоритм асиметричний, використовуються беззнакові цілі числа:

$$q_{w,i} = \text{clamp}\left(\left\lfloor \frac{w_i - M_j}{S_j} \right\rceil, 0, 15\right), \tag{3.6}$$

де $M_j$ та $S_j$ — відновлені значення мінімуму та масштабу для даного підблоку, а оператор $\lfloor \cdot \rceil$ означає математичне округлення. Кожна вага $q_{w,i}$ займає рівно $b_w = 4$ біти.

На етапі інференсу рушій *llama.cpp* не виконує деквантування в оперативній пам'яті, натомість використовуються оптимізовані обчислювальні ядра, які виконують деквантування разом з іншими матричними обчисленнями.

Відновлення наближеного дійсного значення ваги $\hat{w}_i$ відбувається у два етапи. Збережені 6-бітні параметри $q_{\Delta,j}$ та $q_{m,j}$ перетворюються назад у дійсні числа за допомогою нелінійних функцій $f_S$ та $f_M$, які комбінують їх з глобальними FP16 параметрами супер-блоку:

$$S_j = f_S(q_{\Delta,j}, \Delta_{sb}, m_{sb}) = q_{\Delta,j} \cdot \Delta_{sb}, \tag{3.7}$$

$$M_j = f_M(q_{m,j}, \Delta_{sb}, m_{sb}) = q_{m,j} \cdot \Delta_{sb} + m_{sb}. \tag{3.8}$$

Використовуючи значення масштабу $S_j$ та зміщення $M_j$ для підблоку, до квантованих параметрів застосовується формула:

$$\hat{w}_i = q_{w,i} \times S_j + M'_j \,, \tag{3.9}$$

де $M'_j$ — фінальне ефективне зміщення для підблоку $j$. Отримана матриця $\hat{W}$ далі використовується у стандартних операціях лінійної алгебри з векторами активацій.

Квантування значно зменшує вимоги до пропускної здатності пам'яті, яка найчастіше є головним обмежуючим фактором при інференсі LLM. Квантування дозволяє завантажити моделі, наприклад, Qwen3-30b, які у повній точності займають понад 60 ГБ, у відеопам'ять обсягом 24 ГБ графічного прискорювача класу RTX 3090.

Застосування методу Grammar-Constrained Decoding з підтримкою міркувань дозволяє вирішити проблему некотрольованої довжини ланцюжків міркувань шляхом математичного обмеження бюджету обчислень на етапі інференсу. У термінах теорії формальних мов, генерація довільного тексту міркувань відповідає операції замикання Кліні над алфавітом термінальних символів $\Sigma$. Нехай $\Sigma^*$ — множина всіх можливих скінченних рядків над алфавітом $\Sigma$. У методі UCD міркування $r_{reason}$ належить до цієї нескінченної множини:

$$r_{reason} \in \Sigma^*. \tag{3.10}$$

Процес генерації міркувань зупиняється лише тоді, коли модель генерує спеціальний токен завершення послідовності (End-Of-Sequence, EOS). Натомість, при використанні GCD з міркуваннями, ми накладаємо обмеженння за допомогою регулярних виразів у граматиці. Математично це означає перехід від необмеженого замикання Кліні до обмеженої конкатенації. Множина допустимих рядків міркувань $\mathcal{R}_{GCD}$ звужується до:

$$\mathcal{R}_{GCD} = \{r_{reason} \in \Sigma^* \mid L_{min} \leq |r_{reason}| \leq L_{max}\}, \tag{3.11}$$

де $|r_{reason}|$ — довжина рядка міркувань, $L_{min}$ та $L_{max}$ — мінімальна та максимальна допустима кількість символів.

Інтеграція цієї умови у граматику методу GCD змінює динаміку вибору токенів. Як тільки довжина згенерованого префікса міркувань досягає значення $L_{max}$, GCD забороняє подальші текстові символи, дозволяючи лише ті токени, які ініціюють перехід до наступного структурного блоку JSON-схеми (наприклад, перехід до масиву `plan`). Логіти всіх інших токенів маскуються значенням $-\infty$. Це обмеження має прямий вплив на обчислювальну ефективність. Шляхом відсікання надлишкових гілок у просторі пошуку, алгоритм GCD примусово завершує фазу генерації міркувань. У проведених експериментах це дозволило уникнути помилок переповнення контексту для моделей DeepSeek-R1 у складних середовищах (RandomBoxKey) та забезпечило суттєве скорочення часу генерації: для моделі Qwen3-4b тривалість планування зменшилася у 17-25 разів, а для Qwen3-30b — у 6-8 разів порівняно з методом UCD.

## 3.3 Програмна реалізація модуля планування з синтаксичним контролем виходу

Для реалізації системи планування з обмеженням граматики, використовувалися наступні бібліотеки та технології:

1. *Pydantic* — це бібліотека для валідації даних, яка базується на анотаціях типів мови програмування Python. Pydantic дозволяє перетворювати моделі даних в JSON-схеми.

2. *Tyro* — це бібліотека, яка містить інструменти для автоматичної валідації інтерфейсів командного рядка та об’єктів конфігурації безпосередньо з типів даних Python та, зокрема, Pydantic, та є зручнішою альтернативою до стандартної бібліотеки *argparse*. Tyro усуває необхідність написання надлишкового шаблонного коду для парсингу аргументів командного рядка. Шаблонний код дублює визначення параметрів функцій та полів класів. Tyro спрощує процес керування гіперпараметрами моделей, що важливо для відтворюваності експериментів шляхом збереження вхідних конфігурацій.

3. *MLFlow* — це платформа з відкритим вихідним кодом, призначена для керування повним життєвим циклом моделей машинного навчання. Платформа включає компоненти для відстеження та збору параметрів, метрик та артефактів з експериментів, веб-сервера, який надає графічний інтерфейс для керування експериментами. MLFlow є ключовим інструментом у межах методології MLOps, оскільки дозволяє систематизувати ітеративний процес навчання моделей штучного інтелекту та надає інструменти аналізу та порівняння різних версій моделей.

Для визначення моделей даних, які використовуются для комунікації модулів ієрархічного агента, було обрано формат JSON (JavaScript Object Notation) [96]. Синтаксис формату JSON дозволяє будувати ієрархічні структури, які є ізоморфними до базових типів мови Python таких, як *dict, list, str, float/int*, що спрощує відображення даних.

Для валідації та анотування даних використовується JSON Schema [97] — декларативна мова, яка визначає допустимі типи, обов’язкові поля (через ключове слово *required*) та обмеження на додаткові атрибути (*additionalProperties*). У даній роботі JSON-схема використовується не лише як інструмент валідації згенерованих даних, а як мета-опис простору допустимих станів виходу LLM.

Використання JSON-схем у поєднанні з системами інференсу LLM (наприклад, *Ollama*) дозволяє реалізувати методи декодування з обмеженням граматики. На відміну від валідації після генерації, GCD інтегрує обмеження схеми безпосередньо у процес вибору токенів. Це гарантує, що кожен згенерований токен відповідає заздалегідь визначеній граматиці, що

унеможливлює виникнення синтаксичних помилок.

Процес автоматизації створення таких граматик у дослідженні реалізовано шляхом використання бібліотеки *Pydantic* та інструменту *Ollama*. Бібліотека *Pydantic* дозволяє динамічно генерувати JSON-схеми на основі класів даних Python, які далі трансформуються з використанням *Ollama* у формальні граматики (наприклад, у форматі GBNF).

Вибір JSON як основного формату представлення даних при генерації LLM, попри наявність альтернатив (XML [98], Protocol Buffers [99]), зумовлений компромісом між обчислювальною ефективністю та особливостями функціонування LLM. Хоча Protobuf демонструє вищу продуктивність у високонавантажених системах завдяки бінарній серіалізації, його використання в архітектурі LLM-агентів є недоцільним, оскільки мовні моделі оперують виключно текстовими токенами. XML, у свою чергу, вносить значну надлишковість, що прискорює заповнення контекстного вікна LLM та підвищує ймовірність синтаксичних помилок при закритті тегів.

Отже, JSON є збалансованим рішенням. По-перше, JSON підтримує сувору типізацію через JSON-схеми, для якої існують готові реалізації, та які легко інтегруються в програмні системи мовою Python. По-друге, JSON дозволяє здійснювати аналіз логів планування без десеріалізації, оскільки дані зберігаються в текстовому представлення, що є важливим для відлагодження агента на тривалих епізодах.

На основі описаного підходу була розроблена JSON-схема, яка породжує GBNF-граматику для модуля Актор. Лістинг 2 визначає формальну GBNF граматику, яка задає множину допустимих планів, які очікує отримати модуль Актор. Кореневий елемент `<plan-output>` задає чітку послідовність: спочатку модель зобов’язана згенерувати текстові міркування (поле `reasoning`), і лише після цього — масив дій (поле `plan`).

Замість того, щоб дозволити моделі генерувати довільний текст у полі reasoning (що математично відповідало б $\Sigma^*$), граматика накладає строгі границі на оператор повторення: рядок повинен містити від 128 до 1024 символів. Це архітектурне рішення безпосередньо усуває проблему «нескінченних міркувань», яка спостерігалася під час необмеженого декодування (UCD) у моделях сімейства DeepSeek-R1. Обмеження довжини ланцюжка міркувань гарантує, що розмір міркувань не перевищить контекст

LLM, а генерація завершиться за передбачуваний час.

Правило `<plan-array>` обмежує довжину згенерованого масиву макро-дій (опцій) інтервалом від 1 до 7 елементів. Формально, якщо $m$ — це кількість дій у плані, то граматика гарантує виконання умови $1 \leq m \leq 7$. Це запобігає спробам моделі згенерувати надмірно довгі (і найчастіше помилкові) плани.

Правила від `<explore-option>` до `<drop-option>` однозначно формалізують дискретний простір доступних опцій $\Lambda$. Разом із правилом `<object-name>`, яке чітко задає множину допустимих цільових об'єктів (`door`, `key`, `ball`, `box`), граматика повністю унеможливлює помилкові дії такі, як, наприклад, взаємодію зі «стіною», яка відсутня в списку.

Лістинг 2: GBNF-граматика формату виводу LLM

```
<plan-output> ::= "{" "\"reasoning\":" <reasoning-string> ","

"\"plan\":" <plan-array> "}"

<reasoning-string> ::= "\"" [^"]{128,1024} "\""

<plan-array> ::= "[" <option> ("," <option>){0,6} "]"

<option> ::= <explore-option>
    | <goto-option>
    | <pickup-option>
    | <toggle-option>
    | <drop-option>

<explore-option> ::= "{" "\"name\":" "\"explore for\"" ","

"\"searched_objects\":" "[" <object-list> "]" "}"

<goto-option> ::= "{" "\"name\":" "\"go to\"" ","

"\"target\":" <object-scheme> "}"

<pickup-option> ::= "{" "\"name\":" "\"pick up\"" "}"

<toggle-option> ::= "{" "\"name\":" "\"toggle\"" "}"

<drop-option> ::= "{" "\"name\":" "\"drop\"" "}"
```

```
<object-list> ::= <object-scheme>
    | <object-scheme> "," <object-list>

<object-scheme> ::= "{" "\"name\":" <object-name> "}"

<object-name> ::= "\"door\"" | "\"key\"" | "\"ball\"" | "\"box\""

<string> ::= "\"" <characters> "\""
```

В якості задачі прийняття рішень було обрано набір середовищ з бібліотеки Minigrid, яка надає інструментарій для створення двовимірних середовищ, які вимагають послідовного планування дій. Ми використали три середовища зі зростаючою складністю:

- **SimpleKeyDoor:** Агенту необхідно знайти і підібрати ключ, положення якого не відоме заздалегідь, знайти двері, підійти до них і відчинити їх. Це базове завдання на планування послідовності дій.
- **KeyInBox:** Ключ знаходиться всередині скриньки. Агенту спочатку потрібно знайти і відчинити скриньку, забрати ключ, а потім знайти і відчинити двері. Це збільшує довжину плану.
- **RandomBoxKey:** Ключ може знаходитися або в скринці, або поза нею. Агент має або знайти та підібрати ключ, або знайти та відкрити скриньку. Це створює розгалуження варіантів вибору.

Об’єкти в середовищі мають такі атрибути, як тип об’єкта, положення та колір. Спостереження середовища містить кольори, але опції, які обирає Планувальник, кольорів не містять. У використаних середовищах колір дверей та ключа завжди збігаються, тому колір в схемі плану не використовується.

## 3.4 Експериментальна верифікація методу: оцінка валідності планів та порівняння з базовим підходом

При застосуванні мовних моделей в агентах, виникають такі вимоги, як висока швидкість генерації послідовностей для забезпечення швидкодії агентів, можливість виконувати мовні моделі на пристроях з обмеженою потужністю для застосування в роботизованих системах. Серед мовних моделей, які

знаходяться у вільному доступі та можуть бути виконані на прискорювачах з низькою обчислювальною потужністю, можна вказати наступні. Сімейство мовних моделей Qwen3 [100] включає моделі з кількістю параметрів від 0.6 до 235 міль'ярдів. Ці моделі підтримують режим міркувань, який дозволяє динамічно нарощувати використовувані обчислювальні потужності для покращення результатів виконання завдань. Мовні моделі сімейства Qwen3 показують високі результати на багатьох бенчмарках для генерації коду, міркувань в математичних задачах, агентних завданнях. Сімейство мовних моделей Gemma 3 [101] створено для можливості запуску на доступних прискорювачах з не великим об'ємом пам'яті, такі, як персональні комп'ютери з графічними картами. Сімейство мовних моделей Llama [102] також знаходяться у вільному доступі та можуть бути виконані на обмежених обчислювальних потужностях. Моделі сімейства DeepSeek R1 [103] навчені методом дистиляції з великої версії DeepSeek R1 на моделі з сімейства Qwen 2.5 та Llama 3. Особливістю цих моделей є те, що вони навчені за допомогою навчання з підкріпленням генерувати довгі ланцюжки міркувань.

Отже, було відібрано низку сучасних мовних моделей з відкритими вагами, які відповідають наступним критеріям: можливість виконання на графічних прискорювачах з VRAM до 24 ГБ, наявність квантованих версій та донавченість на виконання інструкцій (Instruction-tuned). Моделі, використані в дослідженні:

- **Qwen3:** qwen3-1.7b, qwen3-4b, qwen3-8b, qwen3-30b;
- **DeepSeek-R1:** deepseek-r1-1.5b, deepseek-r1-8b;
- **Gemma3:** gemma3-4b, gemma3-12b, gemma3n-e4b;
- **Llama3.2:** llama3.2-3b.

Для виконання моделей та застосування методу декодування з обмеженням граматики (GCD) було використано програмний інструмент Ollama, який підтримує генерацію тексту відповідно до JSON-схеми.

Для всіх мовних моделей був використаний модифікований промпт з розділу 2, який включав опис завдання, JSON-схему, інструкції щодо міркувань, приклади вирішення задачі.

Були використані наступні методи декодування послідовностей в мовних моделях. Першим методом було необмежене декодування Unconstrained

Decoding (UCD), в якому на кожному кроці обирається найбільш імовірний токен. Промпт містив інструкцію генерувати лише JSON без міркувань, що, однак, не гарантує синтаксичної коректності згенерованого тексту. Використовувалася додаткова пост-обробка згенерованого тексту з метою видалити фрагмент міркувань за його наявності та виділити під-рядок який містить JSON об'єкт. Другий метод — декодування з обмеженням граматики (GCD) без міркувань. В цьому випадку модель генерувала рядок, який чітко відповідає схемі плану, що гарантує синтаксичну коректність та не потребує додаткової пост-обробки перед синтаксичним розбором JSON-рядка. Третій метод, декодування з обмеженою граматикою GCD з міркуваннями, в JSON-схему було додано необов'язкове поле «reasoning», що дозволило моделі генерувати текстові міркування перед формулюванням остаточного плану дій.

Для забезпечення детермінованості результатів температура генерації була встановлена на 0.

Ми визначаємо конфігурацію плану як структуру, яка складається з текстового поля «reasoning» та послідовності дій «plan». План, $\phi = (\omega_1, ..., \omega_n)$, що містить від 1 до 7 дій ($1 \leq n \leq 7$). Кожна дія $\omega_i$ обирається з набору опцій: «explore for objects», «go to object», «pick up», «drop», «toggle». Параметр «object» є одним елементом, а «objects» — це один або більше елементів із множини $\{\text{door}, \text{key}, \text{box}\}$.

У задачах генерації тексту якість роботи великих мовних моделей традиційно оцінюється за допомогою метрик, які вимірюють лексичний або семантичний збіг між згенерованим текстом та еталоном (наприклад, BLEU, ROUGE або відстань Левенштейна). Проте застосування таких метрик до оцінки планів дій у середовищах на основі Марковських процесів прийняття рішень (MDP) та частково спостережуваних Марковських процесів (POMDP) мають обмеження.

Для оцінки коректності плану ми використали метрику середня точність префіксу плану (Mean Exact-Prefix Accuracy, MEPA), яка вимірює частку кроків у префіксі згенерованого плану, які повністю збігаються з істинним планом.

Нехай, істинний план $\phi^{(t,n)} = (\omega_1^{(t)}, ..., \omega_n^{(t)})$, де $n$ — кількість кроків в істинному плані, згенерований план $\phi^{(p,m)} = (\omega_1^{(p)}, ..., \omega_m^{(p)})$, де $m$ — кількість кроків в згенерованому плані. Припустимо, що кожна дія $\omega_i^{(t)}$ переводить середовище через послідовність станів $(s_0, s_1, \ldots, s_n)$ відповідно до функції

переходу $T_{env}(s_{i-1}, \omega_i^{(t)}, s_i)$. Планувальник генерує план $\phi^{(p,m)} = (\omega_1^{(p)}, ..., \omega_m^{(p)})$.

Розглянемо крок $k$, на якому агент вперше робить помилку, тобто $\omega_k^{(p)} \neq \omega_k^{(t)}$, тоді як для всіх $i < k$ виконується $\omega_i^{(p)} = \omega_i^{(t)}$. Виконання помилкової дії $\omega_k^{(p)}$ ініціює перехід середовища у новий стан $s'_k$:

$$s'_k \sim T_{env}(s_{k-1}, \omega_k^{(p)}, \cdot) \tag{3.12}$$

Оскільки $\omega_k^{(p)} \neq \omega_k^{(t)}$, у загальному випадку $s'_k \neq s_k$. У рамках формалізму POMDP агент отримує нове спостереження $o'_k \sim O(s'_k, \omega_k^{(p)}, \cdot)$, яке відрізняється від очікуваного $o_k$.

Наступні дії згенерованого плану $(\omega_{k+1}^{(p)}, \ldots, \omega_m^{(p)})$ були обрані мовною моделлю за умови перебування середовища у стані $s_k$ (відповідно до очікуваної траєкторії). Оскільки фактичний стан змінився на $s'_k$, застосування дії $\omega_{k+1}^{(p)}$ до $s'_k$ є семантично беззмістовним. Навіть якщо згенерована дія випадково збігається з еталонною на цьому кроці ($\omega_{k+1}^{(p)} = \omega_{k+1}^{(t)}$), її виконання у непередбаченому стані $s'_k$ не наближає агента до цілі, а метрики на кшталт відстані Левенштейна зарахували б це як частковий успіх, що є некоректною оцінкою якості роботи стратегії.

Введемо індикаторну функцію правильної дії $c_i$:

$$c_i = \begin{cases} 1, & \text{якщо } i \leq n \wedge i \leq m \wedge \omega_i^{(t)} = \omega_i^{(p)} \\ 0, & \text{інакше.} \end{cases}$$

Позначимо індикатор точного співпадіння префіксу довжини $i$ як

$$\sigma_i = \prod_{j=1}^{i} c_j \tag{3.13}$$

Тобто $\sigma_i = 1$ тоді й тільки тоді, коли всі перші $i$ кроків збігаються.

Тоді MEPA для одного прикладу визначається як середнє значення цих індикаторів по всіх префіксах істинного плану:

$$\text{MEPA}(\phi^{(t,n)}, \phi^{(p,m)}) = \frac{1}{n} \sum_{i=1}^{n} \sigma_i \tag{3.14}$$

Зауважимо, що випадок $n = 0$ не реалізується, оскільки наш датасет не включає приклади з пустим планом.

Така формалізація строго відповідає ймовірності проходження траєкторії в MDP: корисність плану вимірюється виключно довжиною безперервного ланцюжка коректних дій від початкового стану $s_0$. Таким чином, вибір інструкцій для мовної моделі за метрикою MEPA є математично еквівалентним до збільшення ймовірності утримання агента на оптимальній траєкторії розв'язання задачі.

Наведемо приклад обчислення значення метрики MEPA для можливого плану. Якщо $\phi^{(t,3)} = (\text{explore}, \text{go to}, \text{toggle})$ та $\phi^{(p,3)} = (\text{explore}, \text{go to}, \text{drop})$, то $\sigma_1 = 1, \sigma_2 = 1, \sigma_3 = 0$. Метрика буде дорівнювати $(1 + 1 + 0)/3 \approx 0.67$.

Для набору з $K_{example}$ прикладів обчислюємо макро-усереднену MEPA:

$$\text{MEPA}_{\text{macro}} = \frac{1}{K_{example}} \sum_{k=1}^{K\ example} \text{MEPA}_k.$$

де $\text{MEPA}_k$ — значення MEPA на $k$-му прикладі.

Ми провели обчислювальні експерименти у трьох середовищах з набору Minigrid: SimpleKeyDoor, KeyInBox, RandomBoxKey.

Усі експерименти проводилися на апаратному забезпеченні з графічним прискорювачем NVIDIA RTX 3090 (24 ГБ VRAM). Ми використовували фреймворк Ollama 0.11.10 для запуску квантованих версій моделей (Q4_K_M). Для забезпечення відтворюваності результатів параметр температури генерації був встановлений рівним 0, обмеження вибору найбільш імовірних токенів «top_k» встановлено рівним 1. Максимальна довжина згенерованої послідовності була обмежена 4096 токенами.

Ми порівнювали три методи декодування:

1. **UCD (Unconstrained Decoding):** модель отримувала інструкцію генерувати лише JSON. Результат піддавався пост-обробці для отримання JSON-об'єкта.

2. **GCD (Grammar-Constrained Decoding):** генерація обмежувалася JSON-схемою плану без поля для міркувань.

3. **GCD+R (GCD with Reasoning):** JSON-схема включала необов'язкове поле «reasoning», яке дозволяло моделі генерувати міркування перед планом.

Для середовища SimpleKeyDoor існує шість унікальних абстрактних станів середовища, а з врахуванням кольорів, отримуємо 31 приклад з коректним

планом. Завдання мовної моделі — на основі спостереження згенерувати план, який максимально відповідає істинному плану.

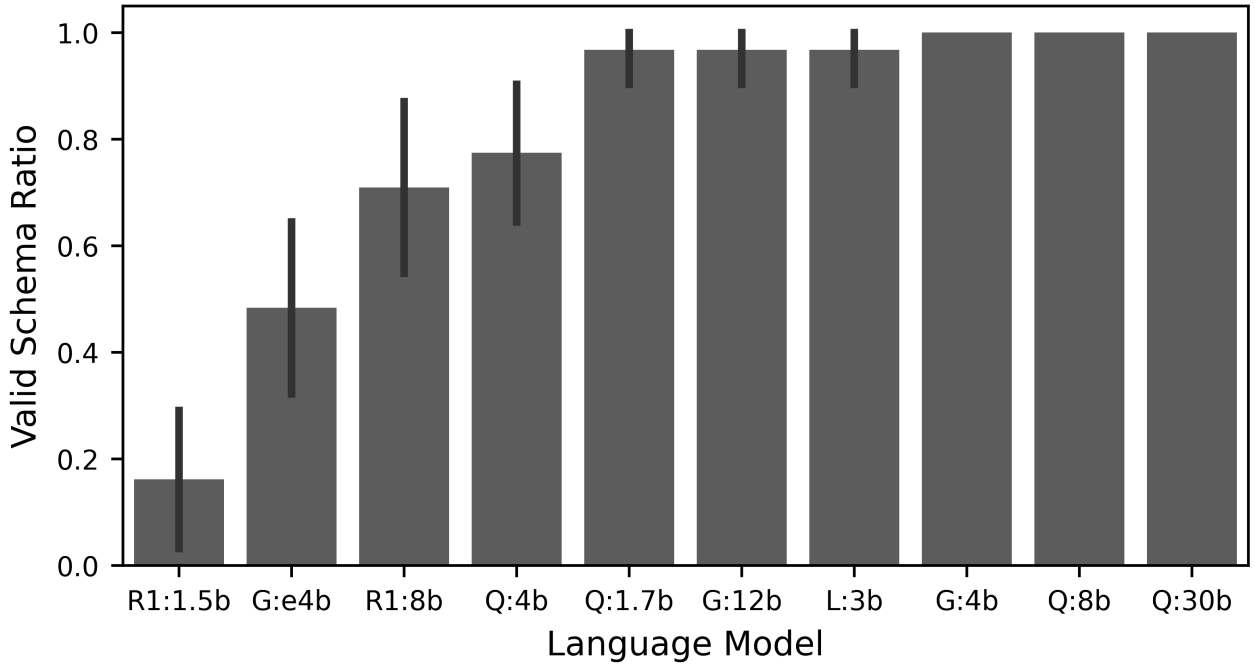



Рисунок 3.1: Значення частки коректно згенерованих планів відповідно до схеми для різних мовних моделей. Чорні вертикальні відрізки позначають 95% довірчий інтервал. Умовні позначення мовних моделей: Q:1.7b — Qwen3:1.7b, Q:4b — Qwen3-4b, Q:12b — Qwen3-12b, Q:30b — Qwen3-30b, G:4b — Gemma3-4b, G:12b — Gemma3-12b, G:e4b — Gemma3n-e4b, L:3b — Llama3.2-3b, R1:1.5b — DeepSeek-R1-1.5b, R1:8b — DeepSeek-R1-8b.

При використанні методу UCD (рис. 3.1) не всі моделі змогли згенерувати синтаксично коректний JSON, що унеможливило його подальшу обробку. Найменш надійними виявилися моделі «DeepSeek-R1-1.5b» та «Gemma3n-e4b». Метод GCD, за визначенням, гарантує 100% коректність схеми.

Рис. 3.2 показує, що для цього порівняно простого завдання, найпотужніші моделі (Qwen3-30b, Gemma3-12b) досягають майже ідеальної точності незалежно від методу декодування. Це свідчить про те, що це завдання є для них порівняно простим. Водночас, для менших моделей таких, як «Gemma3n:e4b», метод GCD дещо покращує результат. Загалом, для 7 з 10 моделей результат планування не змінився або покращився.

На вертикальній осі рис. 3.3 відкладено час генерації плану на логарифмічній шкалі. Як видно з цього графіку, метод UCD для моделей Qwen3-4b та Qwen3-30b виявився значно повільнішим, а саме, для Qwen3-4b в $\approx 25$ разів, для Qwen3-30b в $\approx 8$ разів. Це пов’язано з тим, що ці моделі, попри

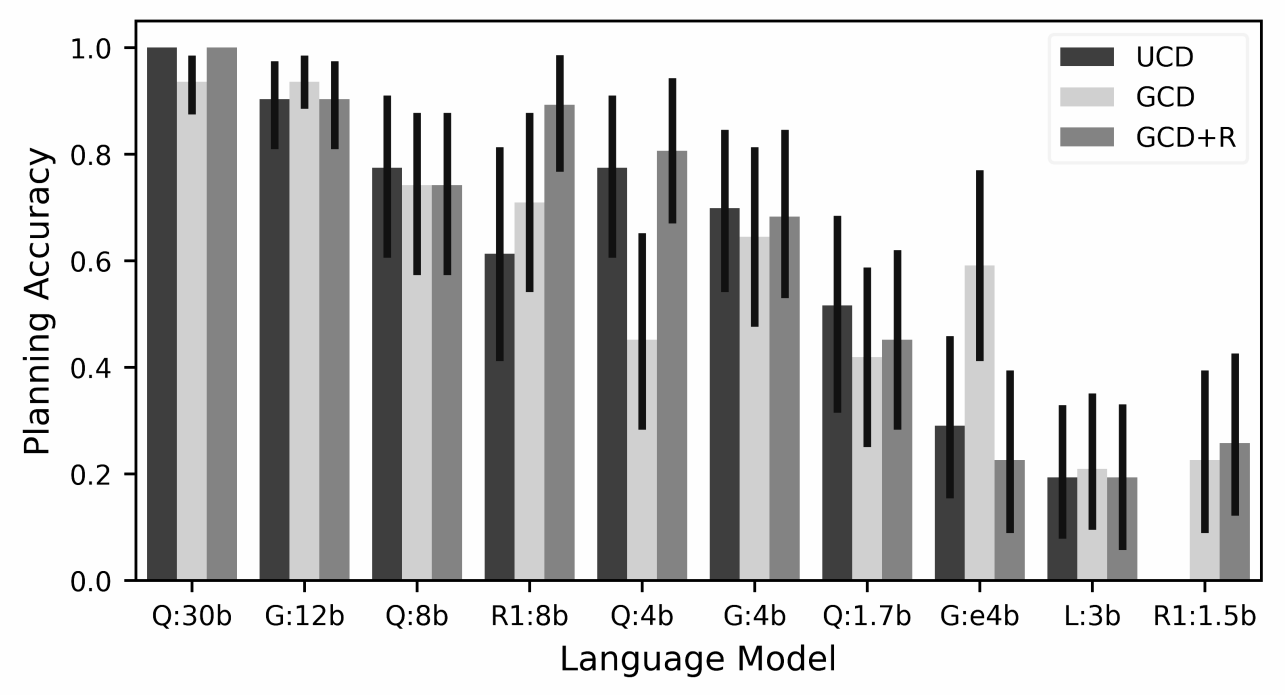



Рисунок 3.2: Значення частки коректно згенерованих планів для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

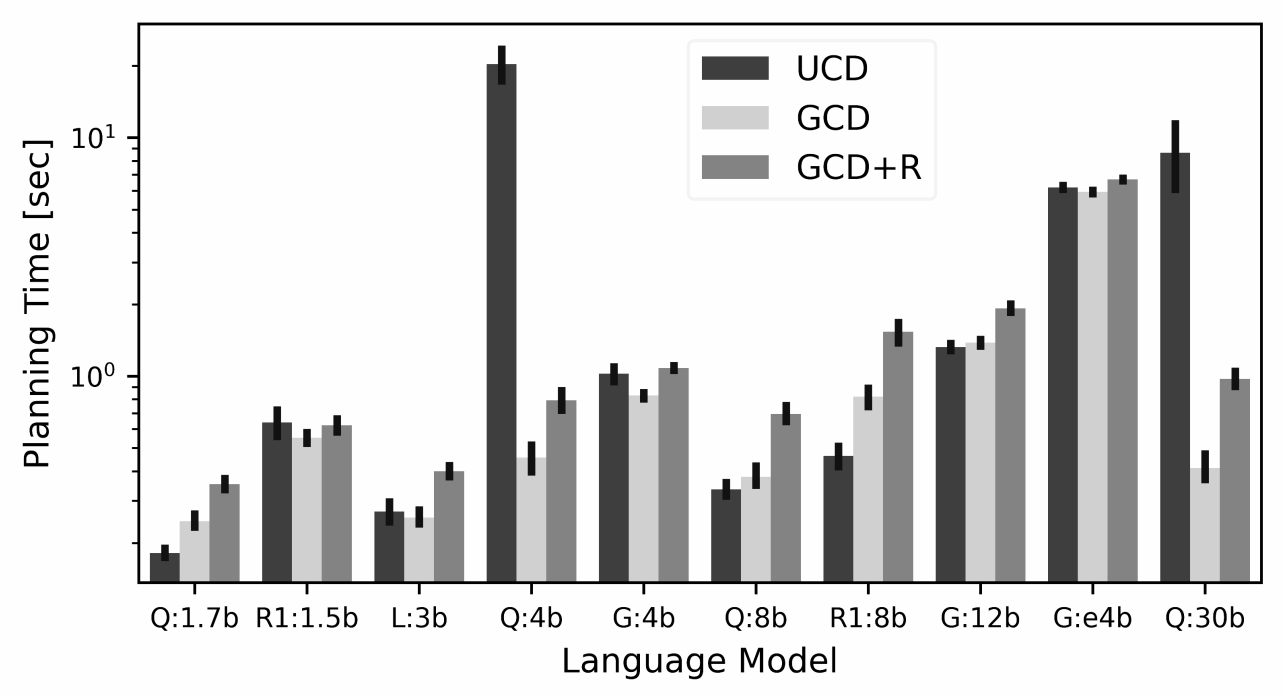



Рисунок 3.3: Середній час генерації одного плану для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

інструкцію не генерувати міркування, генерували довгі ланцюжки міркувань перед JSON-відповіддю. Метод GCD призводить до того, що моделі Qwen3-4b

та Qwen3-30b одразу генерують структурований результат, що значно скорочує час планування і робить ці моделі більш придатними для агентних систем, які потребують високої швидкодії.

У середовищі KeyInBox складність планування зростає, оскільки з'являються додаткові кроки: знайти скриньку, відкрити її, і лише потім взяти ключ та відкрити двері.

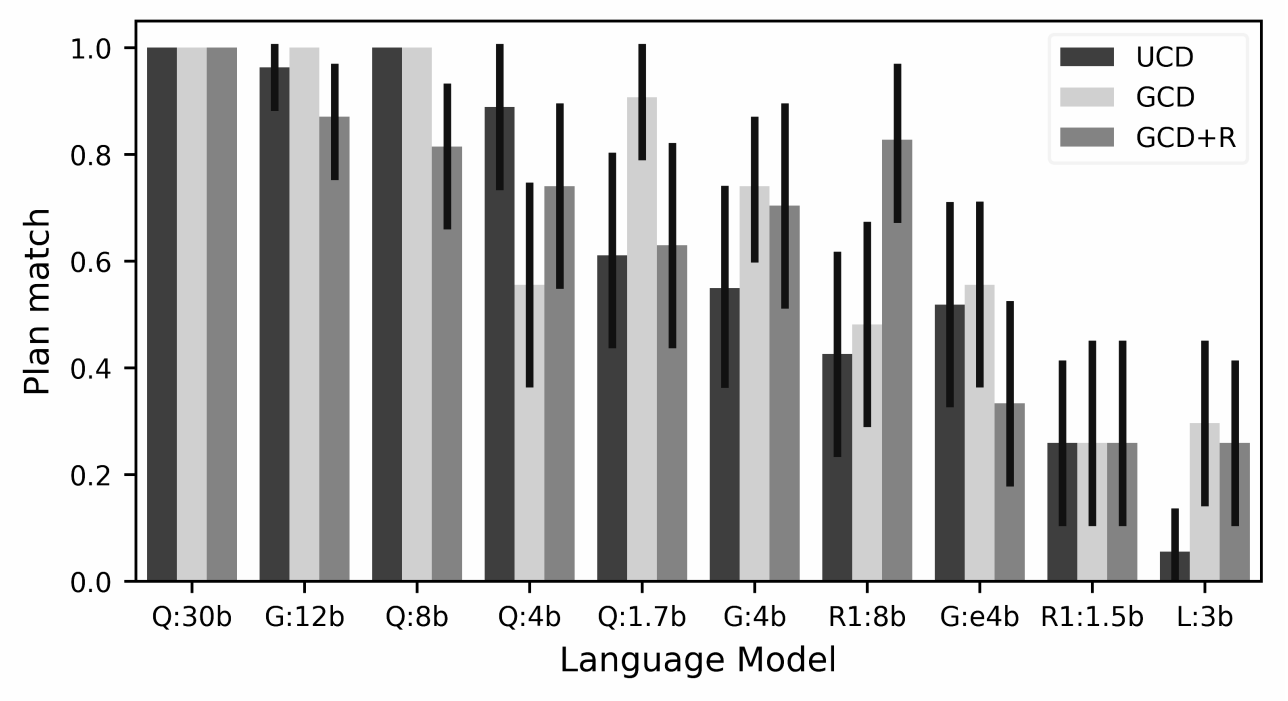



Рисунок 3.4: Значення частки коректно згенерованих планів для середовища KeyInBox для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

Рис. 3.4 демонструє, що такі моделі, як Qwen3-30b, продовжують демонструвати високу точність. Особливо цікавим є результат для моделі DeepSeek-R1-8b: її точність значно зростає при використанні GCD з міркуваннями. Це може свідчити про те, що для моделей, навчених генерувати довгі ланцюжки міркувань, надання спеціального поля «reasoning» в рамках JSON-схеми дозволяє покращити якість фінального плану. Загалом, для 9 з 10 моделей результати планування не змінилися або покращилися.

Рис. 3.5 підтверджує тенденцію, виявлену в попередньому експерименті: моделі Qwen3-4b та Qwen3-30b витрачають значно більше часу на генерацію методом UCD через генерацію зайвих міркувань, тоді як GCD забезпечує швидкий та передбачуваний час відповіді. Для цього середовища, на відміну від попереднього, таке явище спостерігається і для моделі DeepSeek-R1-8b.

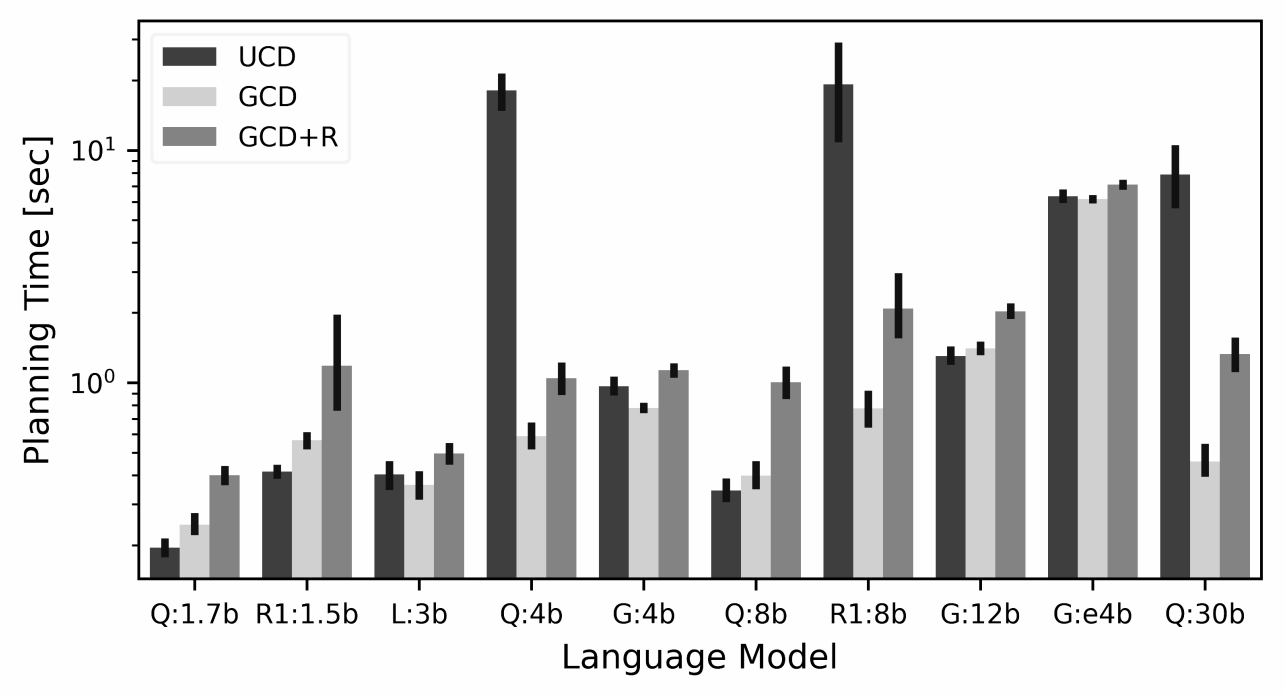



Рисунок 3.5: Середній час генерації одного плану для середовища KeyInBox для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

Неструктурована генерація для моделей Qwen3-4b виявилася повільнішою в $\approx 17$ разів, для Qwen3-30b в $\approx 6$ разів, для DeepSeek-R1-8b в $\approx 9$ разів.

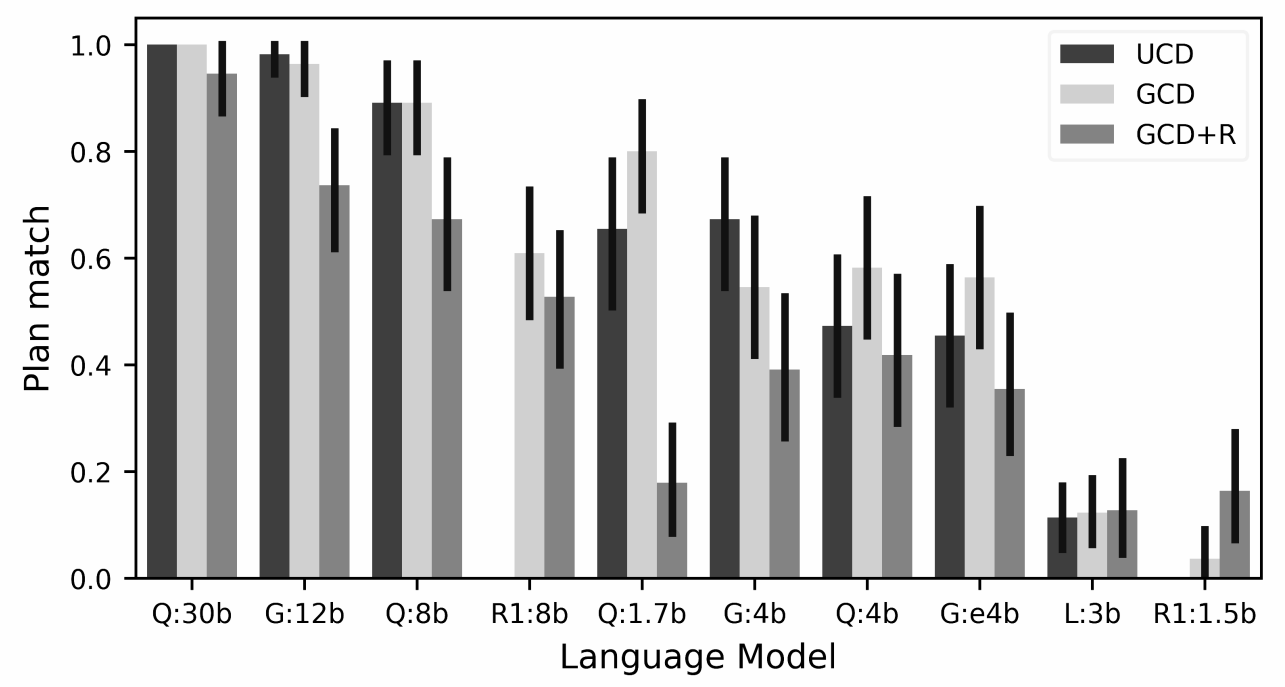



Рисунок 3.6: Значення частки коректно згенерованих планів для середовища RandomBoxKey для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

Результати на рис. 3.6 підкреслюють важливість GCD у складних завданнях. Обидві моделі DeepSeek-R1 зазнали повної невдачі з неструктурованою генерацією, оскільки перевищили ліміт у 4096 токенів, генеруючи неконтрольовані міркування. GCD не тільки дозволив їм згенерувати відповідь, але й значно покращив точність. Для таких моделей, як Qwen3-30b, точність залишається високою, але перевага GCD у швидкості стає суттєвою. Загалом, для 8 з 10 моделей точність планування не змінилася або покращилася.

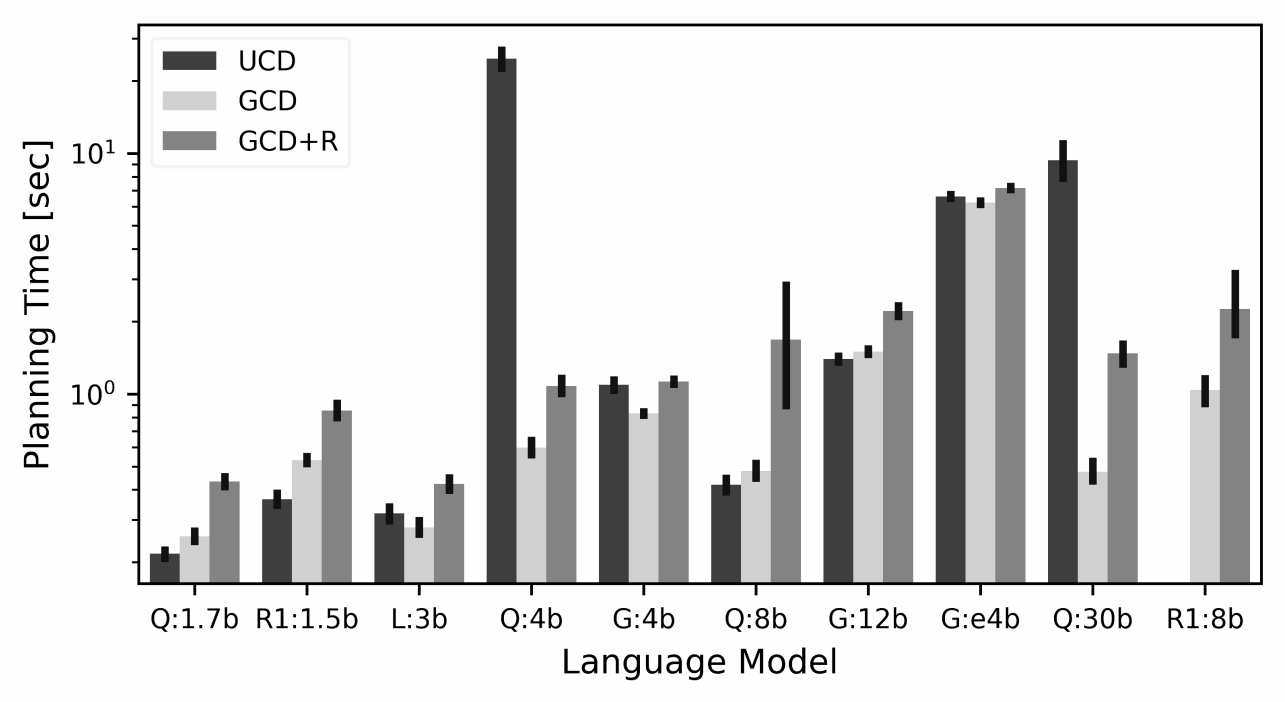



Рисунок 3.7: Середній час генерації одного плану для середовища RandomBoxKey для різних мовних моделей та методів генерації. Умовні позначення та довірчі інтервали ті ж самі, що і на рис. 3.1.

Аналіз результатів, наведених у таблиці 3.1, демонструє загальну тенденцію до зменшення впливу алгоритму декодування з обмеженням граматики (GCD) на точність планування зі збільшенням кількості параметрів мовної моделі. Зокрема, значний позитивний вплив спостерігається для порівняно невеликих моделей, таких як llama3.2-3b (покращення на 151.2%) та deepseek-r1-8b (покращення на 69.9%), що компенсує їхню схильність до відхилення від граматики плану. Натомість для LLM з великою кількістю параметрів, таких як qwen3-30b (приріст 0%), впровадження синтаксичного контролю не призводить до покращення точності планування, оскільки такі моделі досягають майже ідеальної точності з використанням необмеженого декодування. Незначна негативна зміна для qwen3-8b (-1.4%) може бути розглянута як випадковість.

| LLM | Покращення середнього значення MEPA, % |
|---|---|
| deepseek-r1:1.5b | - |
| llama3.2-3b | 151.2 |
| deepseek-r1-8b | 69.9 |
| gemma3n-e4b | 44.9 |
| qwen3-1.7b | 19.4 |
| gemma3-4b | 4.5 |
| qwen3-4b | 3.5 |
| gemma3-12b | 1.8 |
| qwen3-30b | 0 |
| qwen3-8b | -1.4 |

Таблиця 3.1: Порівняння відсоткових покращень середнього значення MEPA для різних LLM

На основі цих результатів, можна зробити висновок, що застосування методу GCD у середньому покращує точність генерації плану за метрикою MEPA для більшості моделей.

Подібно до результатів для попередніх двох середовищ, рис. 3.7 демонструє значно більшу тривалість планування для моделей Qwen3-4b та Qwen3-30b при використанні методу UCD, що робить цей підхід непрактичним для складних завдань. Неструктурована генерація для моделі Qwen3-4b виявилася повільнішою в $\approx 22$ рази, для Qwen3-30b в $\approx 6$ разів.

## Висновки до розділу 3

У третьому розділі досліджено вплив методу декодування з обмеженням граматики (GCD) на ефективність планування дій LLM-агентом. В рамках дослідження було запропоновано використовувати декодування з обмеженням граматики (GCD) в задачах планування послідовності дій агента у віртуальних середовищах, на заміну класичному методу необмеженого декодування (UCD). Підготовлено датасет з прикладами коректних планів дій для трьох середовищ Minigrid. Проведено обчислювальні експерименти з генерації планів дій агента в середовищах з набору Minigrid з використанням методів UCD, GCD та GCD з міркуваннями. Проведено оцінку якості роботи методів на основі наступних метрик: точність слідування граматиці, тривалість планування та точність генерації плану. Виконано аналіз та порівняння результатів застосування методів UCD, GCD та GCD з міркуваннями до задачі планування дій агента

для трьох середовищ Minigrid (SimpleKeyDoor, KeyInBox, RandomBoxKey).

Результати дослідження показали, що на відміну від класичного методу UCD, застосування методу GCD дозволяє гарантувати відповідність згенерованої послідовності до заданої граматики плану, що виключає синтаксичні помилки, та гарантує успішний синтаксичний розбір згенерованого JSON-плану. Цей результат особливо важливий для мовних моделей порівняно не великого розміру, таких як DeepSeek-R1-1.5b та Gemma3n-e4b, які при застосуванні класичного методу UCD генерують граматично некоректні послідовності у більш ніж 50% випадків, натомість, застосування GCD гарантує відповідність граматиці.

Результати вимірювань тривалості планування показують, що класичний метод UCD не гарантує обмеження кількості токенів в згенерованій послідовності, зокрема значна кількість токенів генерується в ланцюжках міркувань. Окремі мовні моделі ігнорують інструкції, які забороняють генерацію довгих ланцюжків міркувань. При застосуванні методу GCD вдалося досягнути скорочення часу планування порівняно з методом UCD для різних середовищ для наступних моделей: тривалість планування моделі Qwen3-4b скоротилася в 17-25 разів, для моделі Qwen3-30b у 6-8 разів, а для моделі DeepSeek-R1-8b у 9 разів (в середовищі KeyInBox). Застосування методу GCD призвело до значного скорочення часу планування завдяки обмеженню довжини ланцюжків міркувань.

Застосування методу GCD в середньому покращує точність генерації плану за метрикою MEPA для більшості моделей. Зокрема найбільший приріст точності був отриманий для моделей DeepSeek-R1, які при використанні методу UCD генерували занадто довгі міркування, що призводило до перевищення ліміту токенів та невдалої генерації плану, з методом GCD та GCD з міркуваннями ці моделі гарантовано генерували план.

Дослідження показало, що застосування GCD в середньому покращує точність генерації плану для більшості використаних моделей і гарантує синтаксичну валідність згенерованого плану відповідно до JSON схеми. Для деяких систем важливою є не зміна точності, оскільки точність вже висока, а скорочення часу планування за рахунок використання GCD. Отже, використання методу GCD є доцільним для покращення роботи мовних моделей у задачах планування.

У дослідженні було використано модуль планування з мовними інструкціями, які включають вичерпний опис середовища, а роль LLM зводиться до правильного вибору плану відповідно до спостереження середовища. Для підвищення автономності агента є необхідним дослідження ситуації, у якій агент володіє обмеженою інформацією про середовище та здатен самостійно аналізувати досвід та перетворювати його у знання, які зберігатимуться до пам’яті. Для вирішення цієї проблеми, у розділі 4 досліджується агент з дворівневою пам’яттю. Одним з рівнів пам’яті є узагальнена, рефлексивна пам’ять, керування якою виконує LLM.

Основні наукові результати розділу опубліковані в роботі [104].

# 4 Розробка методу рефлексивної адаптації агента на основі узагальнення досвіду

## 4.1 Концептуальна модель рефлексивної пам’яті та механізму самокорекції

У цьому розділі дисертаційної роботи ми досліджуємо авторефлексивний механізм пам’яті для ієрархічного агента, який було розглянуто в розділах 2 і 3. У цьому агенті LLM виступає в якості планувальника високого рівня, а послідовності низькорівневих дій реалізовані як процедурні опції. Спираючись на методологію непрямого оцінювання (indirect evaluation), описану в [105], ми проводимо дослідження методом виключення (ablation study), порівнюючи ефективність агента в чотирьох конфігурація: від повної відсутності пам’яті до комплексної системи з транзакційними операціями управління станом пам’яті. Такий підхід дозволяє ізолювати вплив механізму узагальнення та рефлексії на успішність виконання завдань.

Останні досягнення у сфері великих мовних моделей (LLM) дозволили перейти від використання моделей як статичних інструментів генерації тексту до створення автономних агентів, здатних сприймати середовище, виконувати дії, отримувати досвід. Згідно з систематичним оглядом [105], різниця між LLM та агентом на основі LLM полягає в тому, що агент здатний накопичувати знання та змінювати власну поведінку шляхом використання досвіду отриманого через довготривалу взаємодію з середовищем. Для реалізації здатності накопичувати досвід, LLM-агенти потребують модуль пам’яті, який має вирішувати обмеження властиві LLM — фіксовані параметри та обмежена довжина контекстного вікна.

Обмеження фіксованого контекстного вікна сучасних великих мовних моделей (LLM) залишається фундаментальним бар’єром для їх застосування у завданнях, які вимагають довгострокового планування та запам’ятовування. Хоча існують методи розширення контексту, просте збільшення розміру вхідного контекстного вікна призводить до квадратичного зростання обчислювальних витрат і не гарантує ефективного використання інформації, особливо у довгому контексті.

Традиційні методи навчання з підкріпленням (Reinforcement Learning, RL)

часто вимагають значної кількості зразків для навчання та обчислювально витратного донавчання (fine-tuning) моделей. Альтернативою цьому підходу є використання навчання в контексті (In-Context Learning) [9], в якому агенти навчаються на основі власних помилок через механізм узагальнення або рефлексії.

Для задач, які вимагають послідовного планування, таких як навігація в середовищі Minigrid, важливим є розрізнення джерел пам'яті: внітрішньо-епізодична (inside-trial) та міжепізодична пам'ять (cross-trial) [105]. Внутрішньоепізодична пам'ять включає історію спостережень, дій та інших повідомлень, що виникають під час поточного епізоду, та дозволяють агенту орієнтуватися в поточній ситуації. Міжепізодична пам'ять містить досвід, накопичений протягом багатьох спроб виконання завдання під час багатьох епізодів, та є основою для тривалого навчання.

Саме ефективна організація міжепізодичної пам'яті є ключовим викликом, оскільки вона потребує обробки та узагальнення досвіду з багатьох епізодів. Можна виділити наступні види пам'яті LLM-агентів [105]: параметрична (ваги LLM) та текстова. У даній роботі ми фокусуємося на текстовій пам'яті, оскільки вона забезпечує можливість інтерпретації рішень і дозволяє маніпулювати знаннями агента без зміни вагів LLM.

Одним з найпростіших підходів до організації міжепізодичної пам'яті є збереження повної історії взаємодії агента з середовищем. Проблемою такого підходу є те, що тривала взаємодія призводить до переповнення контекстного вікна LLM, зростання кількості необхідних обчислень, погіршення якості роботи механізму уваги LLM [76]. Тому сучасні підходи впроваджують механізми керування пам'яттю (Memory Management), які включають узагальнення (sumarizing), забування (forgetting) та механізм рефлексії (reflection) [12].

Рефлексія [105] — це процес генерації міркувань та висновків вищого рівня абстракції на основі спостережень. Ефективність даного підходу була продемонстрована в роботах [11, 74, 106], де агенти формували узагальнені плани на основі ключових дій з минулих успішних або невдалих епізодів.

Формально процес рефлексії визначається як відображення множини історичних траєкторій взаємодії агента з середовищем $H_t = \{\tau_1, \tau_2, \ldots, \tau_k\}$ отриманих до часового кроку $t$ та поточного стану пам'яті $\mathcal{K}_j$ у нову множину

правил або евристик $\mathcal{K}_{j+1}$. Кожна траєкторія $\tau \in H_t$ є послідовністю станів, дій та отриманих винагород. Оператор рефлексії $f_{\text{reflect}}$ знаходить та узагальнює релевантні знання, та повертає оновлену множину правил:

$$\mathcal{K}_{j+1} \leftarrow f_{\text{reflect}}(H_t, \mathcal{K}_j) \tag{4.1}$$

На початковому кроці $t = 0$ стан пам'яті $\mathcal{K}_0$ ініціалізується залежно від наявності апріорних знань. При холодному старті множина правил є порожньою: $\mathcal{K}_0 = \emptyset$. У випадку, якщо необхідно додати експертні знання або перенести досвід, застосовується ініціалізація заздалегідь визначеним набором евристик $\mathcal{K}_0 = \mathcal{K}_{\text{init}}$, де $\mathcal{K}_{\text{init}} \neq \emptyset$. Це формує початкове індуктивне зміщення цільової стратегії, звужуючи простір пошуку та прискорюючи збіжність на ранніх етапах взаємодії.

Оновлена множина правил $\mathcal{K}_{j+1}$ змінює поведінкову стратегію $\pi(a|s, \mathcal{K})$ агента. Критерієм ефективності застосованого механізму рефлексії є гарантування незменшення очікуваної сумарної винагороди за епізод при використанні оновлених правил порівняно з попередніми. Математично умова покращення стратегії записується через нерівність математичних очікувань сумарної винагороди без використання коефіцієнта знецінювання:

$$\mathbb{E}_{\tau \sim \pi(\cdot|\cdot, \mathcal{K}_{j+1})} \left[ \sum_{t=0}^{T} r_t \right] \geq \mathbb{E}_{\tau \sim \pi(\cdot|\cdot, \mathcal{K}_j)} \left[ \sum_{t=0}^{T} r_t \right], \tag{4.2}$$

де $r_t$ — винагорода на кроці $t$, $T$ — фактична кількість кроків в епізоді, а математичне очікування береться за всіма можливими траєкторіями $\tau$, згенерованими стратегією $\pi$, яка безпосередньо керується поточним станом рефлексивної пам'яті.

Оскільки обчислення точного математичного очікування вимагає обчислення суми на всьому просторі можливих траєкторій, на практиці воно апроксимується емпіричним середнім за скінченною вибіркою з $N$ епізодів. Відповідно, умова емпіричного покращення стратегії набуває вигляду:

$$\frac{1}{N} \sum_{i=1}^{N} G_i^{\pi_{\mathcal{K}_{j+1}}} \geq \frac{1}{N} \sum_{i=1}^{N} G_i^{\pi_{\mathcal{K}_j}}, \tag{4.3}$$

де $G_i^{\pi_{\mathcal{K}}} = \sum_{t=0}^{T_i} r_t$ — сумарна винагорода в $i$-му епізоді, згенерованому стратегією $\pi$ за умови використання набору правил $\mathcal{K}$.

З огляду на розріджену функцію винагород середовища Minigrid, в якому агент отримує ненульовий сигнал винагороди виключно при термінальних станах епізоду, сумарна винагорода $G_i^{\pi_\mathcal{K}}$ дорівнює значенню фінальної винагороди $r_{T_i}$, яка включає штраф за тривалість епізоду. Таким чином, покращення очікуваної сумарної винагороди зводиться до максимізації ймовірності успішного завершення завдання за мінімальну кількість кроків.

У середовищі Minigrid з розрідженою функцією винагороди , сумарна винагорода $G_i$ набуває значень $0$ (невдача) або $r_T > 0$ (успіх). Множина правил $\mathcal{K}$ діє як індуктивне зміщення (умовна апріорна інформація) для поведінкової стратегії $\pi(a|s, \mathcal{K})$. Оператор рефлексії $f_{\text{reflect}}$ аналізує помилкові рішення (де $G_i = 0$) та генерує операції CREATE або UPDATE для додавання або оновлення правил, з ціллю подальшого уникнення поведінки, яка призвела до невдачі. Статистично це приводить до зниження ймовірності переходу до станів, які призводять до $G_i = 0$ (невдачі).

Нехай $P(\tau|\mathcal{K})$ — ймовірність генерації траєкторії $\tau$ за умови правил $\mathcal{K}$. Середовище Minigrid моделюється як простір з дискретними станами (координати та орієнтація) та дискретною множиною дій. Оскільки час $t$ також є дискретним, простір усіх можливих траєкторій скінченної довжини $\mathcal{T}$ є зліченною множиною. Відповідно, $P(\tau|\mathcal{K})$ є функцією маси ймовірності (PMF):

$$P(\tau|\mathcal{K}) = P(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t, \mathcal{K}) p(s_{t+1}|s_t, a_t). \tag{4.4}$$

Оновлення $\mathcal{K}_j \rightarrow \mathcal{K}_{j+1}$ відсікає підмножину термінальних траєкторій $\mathcal{T}_{\text{fail}}$. При зменшенні ймовірності вибору траєкторій, які ведуть до невдачі, ймовірність перерозподіляється на множину успішних траєкторій $\mathcal{T}_{\text{success}}$, що математично гарантує незменшення математичного очікування:

$$\frac{1}{|\mathcal{T}_{\text{success}}|} \sum_{\tau \in \mathcal{T}_{\text{success}}} P(\tau|\mathcal{K}_{j+1}) \geq \frac{1}{|\mathcal{T}_{\text{success}}|} \sum_{\tau \in \mathcal{T}_{\text{success}}} P(\tau|\mathcal{K}_j), \tag{4.5}$$

де $P(s_0)$ — розподіл початкових станів, а $p(s_{t+1}|s_t, a_t)$ — функція переходів середовища. Це рівняння демонструє, що оновлення пам'яті $\mathcal{K}_j \rightarrow \mathcal{K}_{j+1}$ модифікує поведінкову стратегію агента $\pi(a_t|s_t, \mathcal{K})$, знижуючи ймовірності переходу в стани, які належать множині $\mathcal{T}_{\text{fail}}$.

Агент обирає набір $\mathcal{K}$ у дискретному просторі всіх можливих варіантів.

Механізм оновлення коефіцієнта важливості $\sigma'_k \leftarrow (1-\beta)\sigma_k + \beta \cdot y_{\text{target}}$ є формою стохастичної апроксимації (алгоритм Роббінса-Монро) для оцінки функції цінності $V(m_k)$ конкретного елемента пам'яті $m_k$.

Сигнал валідації $y_{\text{target}}$, який позначає результат виконання завдання, відіграє роль TD-помилки (temporal difference). Зниження коефіцієнта важливості та подальше видалення елементів при $\sigma_k < \epsilon = 0.1$ еквівалентне градієнтному спуску у просторі тексту: система монотонно видаляє правила, які мають негативну кореляцію з цільовою функцією $G_i^{\pi_\mathcal{K}}$.

Виконаємо знаходження нерухомої точки оператора математичного сподівання в умовах стохастичного шуму через ітераційну процедуру першого порядку методом Роббінса-Монро [107–109].

Наша мета — знайти корінь $\theta^*$ рівняння $f(\theta) = \alpha$, де $f = \mathbb{E}[\hat{f}(\theta)]$ — невідома регресійна функція від змінної $\theta$, при цьому для функції $f(\theta)$ відомі тільки шумні спостереження $\hat{f}(\theta)$; $\alpha$ — параметр рівняння. Для задачі оцінювання важливості правила (середньої успішності) ми хочемо знайти таке значення $\sigma_k^*$, яке відповідає очікуваному результату $y_{\text{target}}$. Визначимо цільову функцію як математичне сподівання помилки:

$$\mathbb{E}[\sigma_k - y_{\text{target}}] = 0. \tag{4.6}$$

У рівнянні 4.6 функції $f(\theta)$ відповідає $\mathbb{E}[\sigma_k - y_{\text{target}}]$, змінній $\theta$ відповідає $\sigma_k$, $\alpha = 0$.

Вирішення рівняння 4.6 математично еквівалентно мінімізації середньоквадратичного відхилення (MSE):

$$J(\sigma_k) = \frac{1}{2}\mathbb{E}[(\sigma_k - y_{\text{target}})^2], \tag{4.7}$$

де оптимальна точка $\sigma_k^*$ задовольняє рівнянню $\nabla J(\sigma_k^*) = \mathbb{E}[\sigma_k^* - y_{\text{target}}] = 0$.

Згідно з алгоритмом Роббінса-Монро, оновлення параметра $\theta$ відбувається за правилом:

$$\theta_{j+1} = \theta_j - a_j \cdot \hat{f}(\theta_j), \tag{4.8}$$

де $\hat{f}(\theta_j)$ — шумне спостереження функції $f$ у точці $\theta_j$, $a_j$ — послідовність позитивних констант.

У нашому випадку $\theta_j = \sigma_k^{(j)}$, $\hat{f}(\sigma_k^{(j)}) = \sigma_k^{(j)} - y_{\text{target}}^{(j)}$ - помилка на кроці $j$.

Підставляємо в загальну схему алгоритму Роббінса-Монро:

$$\sigma_k^{(j+1)} = \sigma_k^{(j)} - a_j(\sigma_k^{(j)} - y_{\text{target}}^{(j)}). \tag{4.9}$$

Для отримання формули експоненційного ковзного середнього, розкриємо дужки та згрупуємо доданки при $\sigma_k^{(j)}$:

$$\sigma_k^{(j+1)} = \sigma_k^{(j)} - a_j\sigma_k^{(j)} + a_j y_{\text{target}}^{(j)}, \tag{4.10}$$

$$\sigma_k^{(j+1)} = (1 - a_j)\sigma_k^{(j)} + a_j y_{\text{target}}^{(j)}. \tag{4.11}$$

Для отримання остаточної формули виконаємо заміну кроку $a_j$ на константу $\beta$:

$$\sigma_k^{(j+1)} = (1 - \beta)\sigma_k^{(j)} + \beta y_{\text{target}}^{(j)}, \tag{4.12}$$

де компонент $(1 - \beta)\sigma_k^{(j)}$ визначає, яку частку оцінки важливості правила ми зберігаємо, компонент $\beta y_{\text{target}}^{(j)}$ визначає ступінь довіри до останнього отриманого результату (нового досвіду).

В дослідженні було використано декілька підходів до узагальнення історії епізодів.

У найпростішому підході історія була зібрана заздалегідь на окремому наборі епізодів, після чого був проведений процес узагальнень з використанням LLM Qwen3-30b, отримані узагальнення були додані до промпту Планувальника, а ефективність агента була протестована на валідаційному та тестовому наборі середовищ.

Наступний підхід включав послідовне оновлення узагальнень по мірі завершення епізодів. У цій ситуації відбувається online оновлення промпту планувальника на основі досвіду отриманого зі взаємодії з середовищем.

Також було досліджено покращену версію механізму узагальнень, в якому узагальнення були представлені у вигляді списку правил, кожне правило мало коефіцієнт важливості, який оновлювався в процесі роботи агента та визначав наскільки правило важливе для успішної роботи агента. Високе значення цього коефіцієнту унеможливлювало видалення правила зі списку узагальнень, що працювало як механізм упередження «катастрофічного забування» при оновленні промпту.

У даному дослідженні параметри LLM не змінювалися та були фіксованими. Зміна ефективності агента відбувалася за рахунок зміни промптів на основі досвіду агента отриманому з взаємодії з середовищем.

Перехід від $\mathcal{K}_t$ до $\mathcal{K}_{t+1}$ відбувається шляхом застосування LLM, яка позначена як $G_{\theta\ LLM}$, до $\mathcal{K}$.

Позначимо історію останніх епізодів як $H_t$. Функція рефлексії отримує на вхід історію та поточний стан пам'яті:

$$\mathcal{O}_t = G_{\theta\ LLM}(H_t, \text{stringify}(\mathcal{K}_t)), \tag{4.13}$$

де $\mathcal{O}_t$ — це послідовність транзакційних операцій до пам'яті згенерованих LLM щоб покращити набір правил, stringify — функція перетворення $\mathcal{K}_t$ у текстовий рядок.

Набір допустимих операцій $\mathcal{O} = \{\text{CREATE}, \text{UPDATE}, \text{DELETE}, \text{CONFIRM}\}$. Кожна операція $op \in \mathcal{O}_t$ змінює стан пам'яті $\mathcal{K}_t$. Ми визначаємо функцію перетворень як $T_{op} : \mathcal{K} \times \mathcal{O} \to \mathcal{K}$.

Коефіцієнт важливості елемента пам'яті змінюється за експоненційною ковзною функцією обчисленою на сигналі валідації, якісний надходить від транзакції згенерованої LLM:

$$\sigma'_k \leftarrow (1 - \beta)\sigma_k + \beta \cdot y_{\text{target}}, \tag{4.14}$$

де $y_{\text{target}}$ — це цільове значення важливості елементу пам'яті (0 для штрафу, 1 для підкріплення), $\beta$ — балансуючий коефіцієнт.

Оператор CREATE додає новий елемент до набору:

$$\mathcal{K}' = \mathcal{K} \cup \{m_{k_{new}}\}. \tag{4.15}$$

Оператор UPDATE призначений для зміни помилкової інформації. Оператор DELETE має бути використаний якщо інформація в пам'яті помилкова та має бути видалена. Цей оператор застосовує м'яке видалення шляхом зниження коефіцієнту важливості елементу. При зниженні важливості до критичного значення, елемент підлягає остаточному видаленню на етапі прорідження пам'яті. Оператор CONFIRM підвищує значення коефіцієнту важливості елемента пам'яті.

Після застосування всіх вибраних LLM операторів, відбувається етап

прорідження, на якому видаляються елементи пам'яті з коефіцієнтом важливості нижчим за $\epsilon = 0.1$.

**Промпт 4.1: Промпт модуля рефлексії пам'яті**

**Role**

```
You are summarizer for history of interactions of an agent with an
environment.
```

**Tasks**

```
Analyse given episode history and update the knowledge base using the
provided transactional operations. Knowledge base must contain only general
solution algorithm for the tasks. Analyze the history of interactions of
the agent with the environment. Identify patterns where the agent's plan
failed due to preconditions or logical errors. Find what problems was the
agent faced. Find which successful decisions was the agent made. Formulate a
general algorithm of solving the task. Update the given algorithm with found
information. Be concise and clear.
```

**Tools**

```
Use CREATE operation for creation a new step in algorithm. Use UPDATE to fix
wrong information (updating decreases importance of the row). Use DELETE to
when the row is totaly wrong. Use CONFIRM to increase importance of the row
and prevent it from deletion in future.
```

**Critical Instruction**

```
CRITICAL INSTRUCTION: The 'Knowledge Base' uses stable Integer IDs (e.g.,
[ID: 1]). When updating, confirming, or deleting, use these IDs exactly. Do
not guess IDs.
```

**Context**

```
The history of episodes: {history_text} Knowledge Base: {memory_text} Format
the output according to the JSON schema: {output_schema}
```

В програмній реалізації агента, перехід від історії взаємодії $H_t$ до набору транзакційних операцій $\mathcal{O}_j$ здійснюється за допомогою промпту до великої мовної моделі. Промпт модуля рефлексії (Лістинг 4.1) формалізує завдання моделі як аналіз помилкових та успішних рішень агента. Для маніпуляції базою знань $\mathcal{K}_j$ використовуються базові операції управління даними CREATE, UPDATE, DELETE, CONFIRM. Модель отримує інструкцію оперувати виключно існуючими ідентифікаторами правил, що забезпечує стабільність функції перетворень $T_{op}$. На вхід моделі передаються поточна історія `history_text` та стан пам'яті `memory_text`. Для забезпечення

програмної інтеграції результатів рефлексії з планувальником, вихідні дані LLM форматуються згідно з чітко визначеною схемою JSON.

## 4.2 Модифікація та реалізація опцій Explore та GoTo

У роботі було досліджено агента, який складається з декількох модулів: Планувальник, Медіатор, Актор. Планувальник побудований на основі LLM Qwen3-8b та виконує завдання планування послідовності опцій в мовному представленні на просторі процедурних опцій для поточного спостереження середовища. Медіатор виконує перетворення спостереження середовища з векторного в мовне представлення. Актор отримує абстрактний план та виконує його шляхом вибору відповідних опцій.

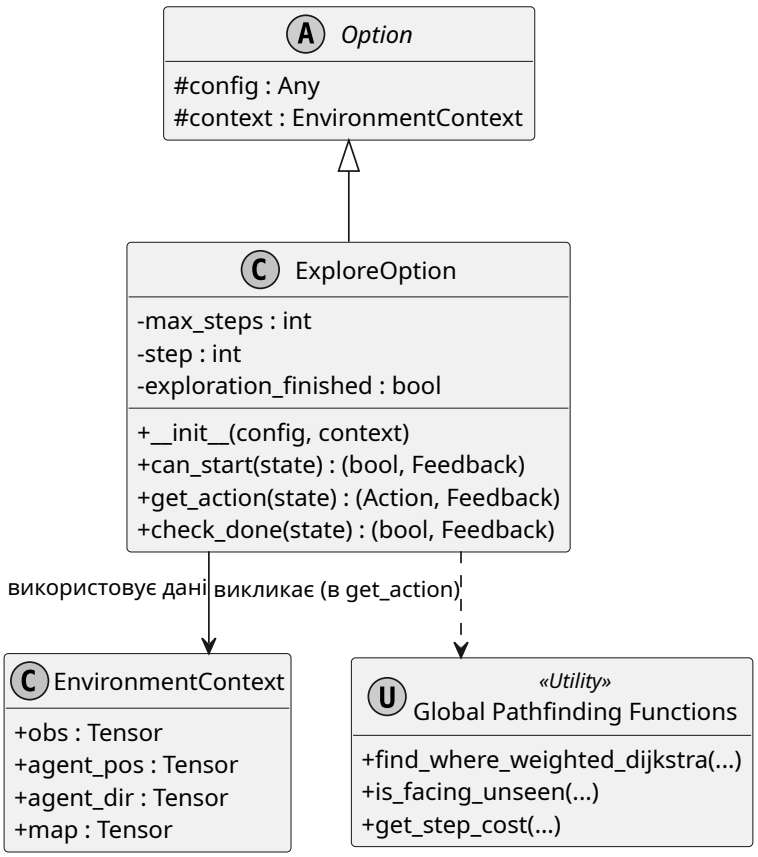



Рисунок 4.1: Діаграма класів опції Explore.

Було реалізовано п’ять опцій для середовищ типу Minigrid: Explore, GoTo, PickUp, Toggle, Drop. Під час планування агент може вибирати опції лише з вказаного списку. Кожна опція включає передумову для початку виконання опції, процедуру виконня, яка перетворює опцію в послідовність конкретних

дій, та умову завершення. Опції Explore та GoTo реалізовані на основі алгоритмів Дейкстри та A*, відповідно.

У більшості випадків взаємодія агента з середовищем починається з дослідження прихованих клітинок шляхом використання опції `Explore` (Рис. 4.1). На відміну від класичного завдання пошуку найкоротшого шляху до заздалегідь відомої цілі, завдання дослідження вимагає динамічного визначення цільових вузлів на основі поточного стану видимості середовища. Для вирішення цього завдання було розроблено модифікацію алгоритму Дейкстри зі зваженим пошуком та умовою раннього завершення (Рис. 4.2).

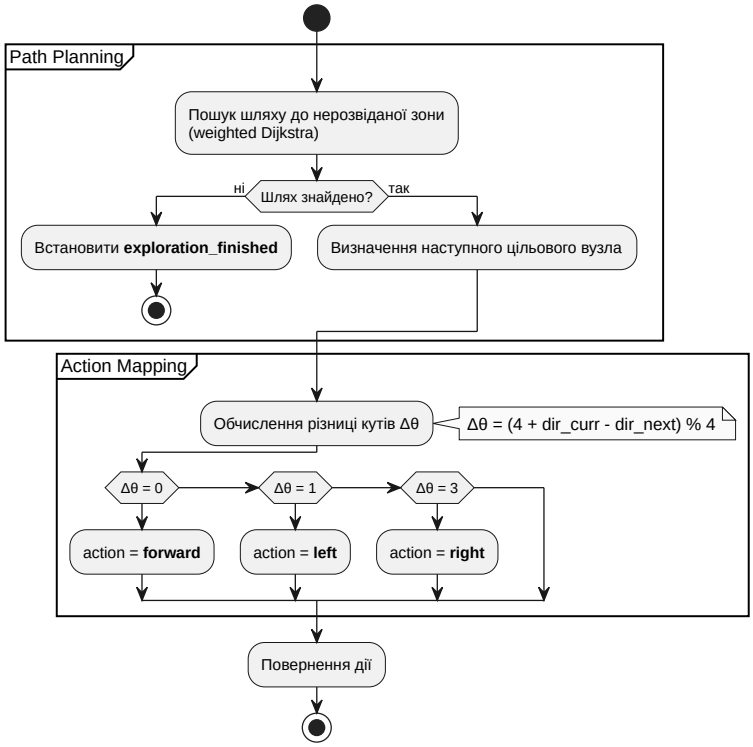



Рисунок 4.2: Діаграма активності опції Explore.

Середовище Minigrid моделюється як орієнтований граф простору станів $G = (V, E)$. Кожен вузол $v \in V$ визначається кортежем просторових координат та орієнтації агента: $v = (x, y, d)$, де $x, y \in \mathbb{N}$ — координати на двовимірній сітці, а $d \in \{0, 1, 2, 3\}$ — дискретний напрямок повороту агента. Множина ребер $E$ відображає доступні низькорівневі дії: крок вперед (`forward`), поворот ліворуч (`left`) та поворот праворуч (`right`).

Для забезпечення пріоритетності руху вздовж стін в середовищі, була введена спеціальна зважена функція вартості переходу між суміжними вузлами $v_{curr}$ та $v_{next}$. Вартість кроку $C(v_{curr}, v_{next})$ обчислюється як базова вартість дії плюс сумарний штраф за конфігурацію оточення цільової клітинки:

$$C(v_{curr}, v_{next}) = 1 + \sum_{dx=-1}^{1} \sum_{dy=-1}^{1} f_{cell}(x_{next} + dx, y_{next} + dy), \tag{4.16}$$

де індекси $dx, dy$ визначають околицю з 8 сусідніх клітинок навколо цільової позиції та цільову клітинку ($dx = 0, dy = 0$). Функція $f_{cell}(x, y)$ визначає евристичну вартість (штраф) конкретної клітинки на основі її типу: порожні клітинки (`EMPTY`) та клітинки з ціллю (`GOAL`) мають найвищий штраф (вартість 2), невідомі клітинки (UNSEEN) та відкриті двері (`DOOR_OPEN`) мають вартість 1, тоді як перешкоди (наприклад, `WALL`) мають нульову вартість. Така конфігурація мінімізує накопичений штраф при русі поблизу перешкод.

Алгоритм пошуку не має фіксованого цільового вузла, натомість використовується динамічна умова зупинки $T_{stop}(v)$, яка перевіряє, чи знаходиться хоча б одна невідома клітинка (`UNSEEN`) у полі зору $\mathcal{F}(v)$ агента при досягненні вузла $v$:

$$T_{stop}(v) = \begin{cases} \text{True}, & \text{якщо } \exists c \in \mathcal{F}(v) : \text{Map}(c) = \text{UNSEEN}; \\ \text{False}, & \text{в іншому випадку.} \end{cases} \tag{4.17}$$

Поле зору $\mathcal{F}(v)$ у даній реалізації визначається як множина з трьох клітинок безпосередньо перед агентом. При виконанні умови $T(v) = \text{True}$ алгоритм негайно перериває пошук та реконструює знайдений шлях, що суттєво зменшує обчислювальну складність порівняно з повним обходом графа.

На відміну від опції `Explore`, опція `GoTo` (Рис. 4.3, 4.4) вирішує задачу навігації до цільового об'єкта з заздалегідь відомим положенням. Ця процедурна опція ініціалізується Планувальником у випадках, коли агент вже має інформацію про положення об'єкта і потребує побудови оптимального маршруту для переходу до клітинки, яка знаходиться поряд з цільовим об'єктом.

Перед початком побудови маршруту алгоритм перевіряє принципову можливість досягнення цілі, перевірка відбувається у методі `can_start`. Для цього використовується бінарна маска яка обчислюється на основі поточного стану видимості середовища. Нехай $M_{obj}$ — це матриця ідентифікаторів

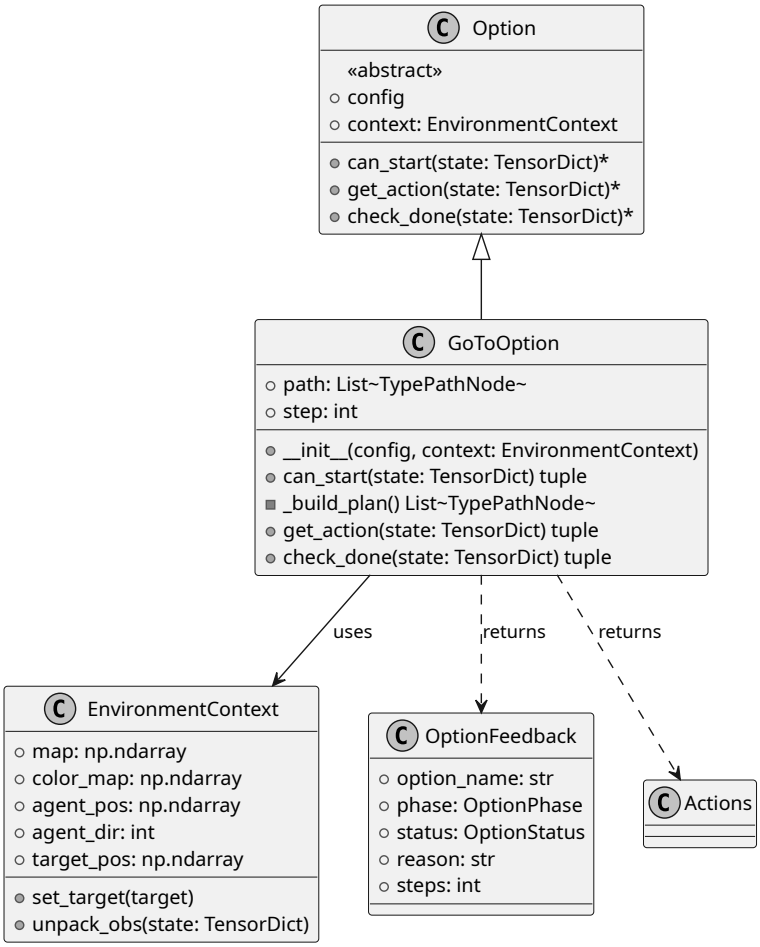



Рисунок 4.3: Діаграма класів опції GoTo.

об'єктів у полі зору агента, а $M_{color}$ — матриця кольорів об'єктів. Цільовий об'єкт задається ідентифікатором $id_{target}$ та кольором $c_{target}$. Матриця наявності цільового об'єкта $M_{target}$ обчислюється як початкова логічна кон'юнкція:

$$M_{target(x,y)} = \begin{cases} 1, (M_{obj(x,y)} = id_{target}) \wedge (M_{color(x,y)} = c_{target}); \\ 0, \text{інакше}. \end{cases} \tag{4.18}$$

Опція може почати своє виконання тоді і тільки тоді, коли існує хоча б одна координата $(x, y)$, для якої $M_{target(x,y)} = 1$.

У середовищі Minigrid агент не може займати одну просторову клітинку з цільовим об'єктом (наприклад, стіною або закритими дверима). Опція `GoTo` виконує завдання пошуку шляху та вибір дій для переміщення до цільового об'єкта для подальшої взаємодії з ним. Цільовим станом для навігації вважається позиція на будь-якій суміжній вільній клітинці за умови, що агент

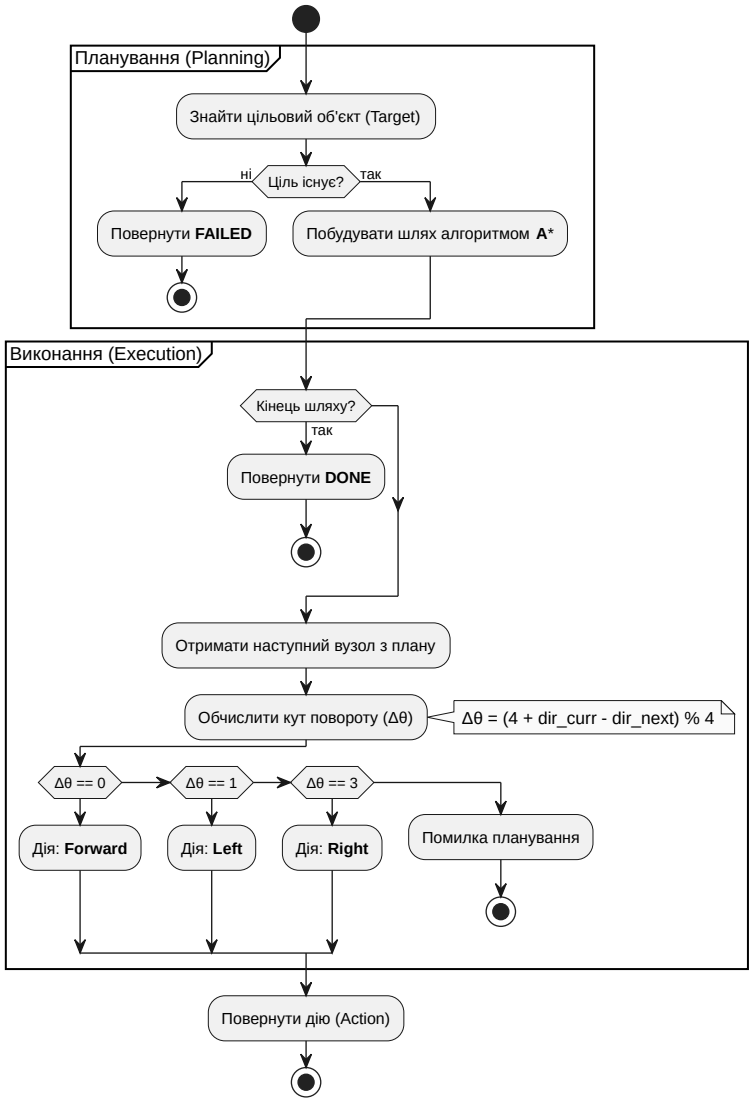



Рисунок 4.4: Діаграма активності опції GoTo.

повернутий в напрямку об'єкта. Якщо координати цілі $\mathbf{p}_{target} = (x_t, y_t)$, то множина допустимих цільових станів $S_{target}$ визначається як:

$$S_{target} = \{(x_t - dx_i, y_t - dy_i, i) \mid i \in \{0, 1, 2, 3\}\}, \tag{4.19}$$

де $i$ — дискретний індекс напрямку, а $(dx_i, dy_i)$ — відповідний йому одиничний вектор орієнтації простору, $\{(dx_i, dy_i)\}_{i=0}^{3} = \{(-1, 0),\ (1, 0),\ (0, -1),\ (0, 1)\}$.

Оскільки координати цілі та поточна позиція агента відомі, для генерації маршруту застосовується алгоритм A* — алгоритм пошуку на графі, який використовує евристичну функцію для оптимізації обходу графу, надаючи перевагу вузлам, які лежать на найкоротшому шляху до цілі. Порівняно з алгоритмом Дейкстри, який використовується в опції `Explore`, A* значно швидше збігається в детермінованих умовах.

В якості евристики використовується Мангеттенська метрика, яка визначає відстань між двома точками на сітці як суму абсолютних різниць їхніх декартових координат. Для довільного вузла з координатами $\mathbf{p} = (x, y)$ мінімальна відстань $h(\mathbf{p})$ до множини цільових станів обчислюється як:

$$h(\mathbf{p}) = \min_{\mathbf{p}_t \in S_{target}} \left( |x - x_t| + |y - y_t| \right). \tag{4.20}$$

Згенерований маршрут на графові $\mathcal{P} = \{v_0, v_1, \ldots, v_n\}$ є послідовністю просторових станів агента. На кожному кроці симуляції середовища опція `GoTo` транслює різницю між поточним станом $v_k = (x_k, y_k, d_k)$ та наступним $v_{k+1} = (x_{k+1}, y_{k+1}, d_{k+1})$ у дискретну дію. Зміна орієнтації $\Delta\theta$ обчислюється з урахуванням циклічності напрямків:

$$\Delta\theta = (4 + d_k - d_{k+1}) \pmod 4. \tag{4.21}$$

На основі $\Delta\theta$ агент виконує дію з множини $A = \{\text{forward}, \text{left}, \text{right}\}$. Якщо $\Delta\theta = 0$, генерується дія руху вперед, якщо 1 — поворот ліворуч, якщо 2 — два повороти праворуч, якщо 3 — поворот праворуч.

## 4.3 Реалізація циклу зворотного зв'язку в архітектурі агента

У процесі подальших змін ієрархічного агента потреба в окремому навчальному модулі бінарної класифікації зникла. Цей перехід відбувся у два етапи. Спочатку впровадження методів структурованої генерації підвищило надійність формування планів. Згодом інтеграція механізму авторефлексивної пам'яті дозволила побудувати прямий, детермінований зворотний зв'язок між виконавчими опціями та Планувальником. У цій оновленій парадигмі, якщо обрана низькорівнева опція не може бути ініційована або завершується станом помилки, це автоматично перериває поточний план і генерує запит на новий план. Такий підхід повністю усунув необхідність тривалого навчання окремої нейронної мережі для керування комунікацією Актора з Планувальником.

Взаємодія агента з середовищем відбувається епізодами. Протягом кожного епізоду Планувальник обирає опції, які надсилають зворотній зв'язок, який зберігається в пам'ять. Пам'ять агента структурована за епізодами, повідомлення від опцій зберігаються послідовно, також зберігаються спостереження та згенеровані плани. У кінці епізоду до пам'яті зберігається

інформація про кількість кроків в епізоді, успішність/невдача виконання завдання, сумарна винагорода за епізод.

У процесі роботи агента в промпт Планувальника включається статичний фрагмент з загальними вказівками стосовно завдання, фрагмент пам'яті про поточний епізод та набір правил, сформований однією з технік узагальнення на повній історії попередніх епізодів.

Ми визначаємо рефлексивну систему пам'яті (Reflective Memory System) як динамічну систему, яка змінюється в дискретному часі $t \in \mathbb{N}$ шляхом застосування рефлексії [12]. Стан пам'яті визначається набором правил, що позначені як $\mathcal{K}_t$.

Набір правил $\mathcal{K}$ визначається як множина елементів пам'яті, які індексуються унікальними ідентифікаторами. Нехай $\mathcal{I}_t \subset \mathbb{N}$ набір активних індексів на часовому кроці $t$. Тоді набір правил:

$$\mathcal{K}_t = \{m_k^{(t)} | k \in \mathcal{I}_t\}, \tag{4.22}$$

де кожен елемент пам'яті $m_k^{(t)}$ представлений як кортеж:

$$m_k^{(t)} = (k, \phi_k, c_k, \sigma_k), \tag{4.23}$$

де $k$ — унікальний, постійний ідентифікатор елемента пам'яті, $\phi_k \in \Phi = \{\text{FACT}, \text{RULE}, \text{REFLECTION}\}$ визначає категорію елементу пам'яті, $c_k \in \Sigma^*$ текстовий вміст елементу пам'яті (тут $\Sigma^*$ це множина всіх можливих рядків), $\sigma_k \in [0, 1]$ — коефіцієнт важливості елемента пам'яті, який ініціалізується як $\sigma_{\text{init}} = 0.5$.

Для забезпечення зворотнього зв'язку між планувальником та актором, який виконує послідовність опцій, було реалізовано вербальний звортній зв'язок від опцій до планувальника. Опції мають три етапи роботи: початок (перевірка передумов), крок роботи, перевірка умови завершення. На етапі перевірки передумов може бути виявлено, що опція не може почати виконання, що відповідає помилці, текст якої відправляється Планувальнику. Якщо опція успішно почала виконання, то на кожному кроці може бути відправлений зворотній зв'язок про особливості виконання або помилку, якщо така виникла. При завершенні опції буде відправлено зворотній зв'язок про параметри завершення опції.

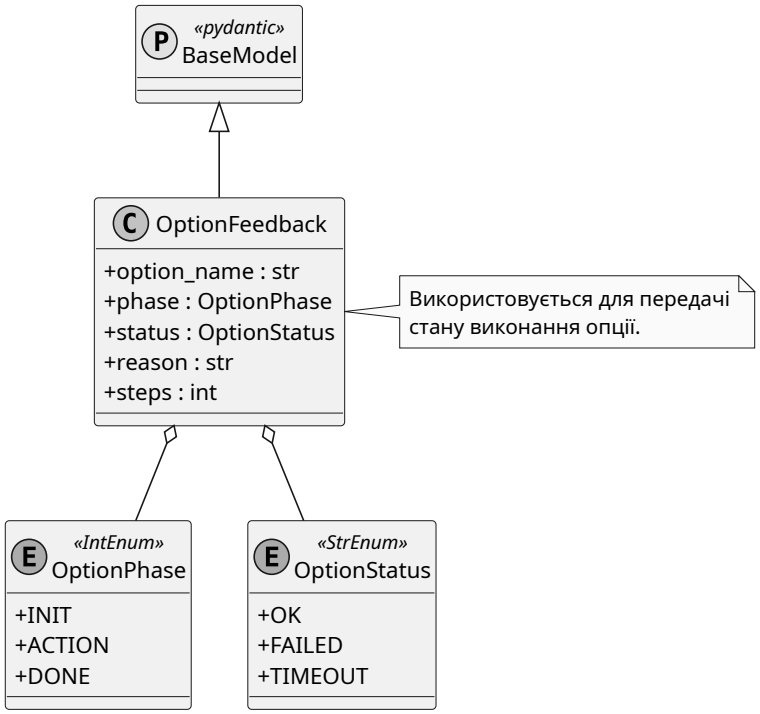



Рисунок 4.5: Діаграма класів компонента OptionFeedback.

Опції створюють мовний зворотній зв’язок, який надходить до планувальника та повідомляє про важливу інформацію або помилки в роботі опцій. Зворотній зв’язок представляє собою кортеж: етап роботи опції, статус помилки, кількість виконаних кроків, повідомлення. Зворотній зв’язок від опцій використовується планувальником для виправлення помилок в плані.

Щоб уникнути зациклювання агента у випадках, коли планувальник знову і знову генерує один і той же не працюючий план, було додано обмеження на кількість послідовних генерацій однакового плану, максимум 5 повторень. При перевищенні цього обмеженння епізод завершувався, а завдання вважалося не виконаним. Це дозволило прискорити проведення експериментів та не чекати завершення епізоду через вичерпання обмеження по крокам в середовищі.

## 4.4 Проєктування та імплементація системи навчання агента з рефлексивною пам’яттю

Розроблена програмна система побудована за модульним принципом, що перетворює її на зручний фреймворк для проведення експериментів. Щоб уникнути фіксації значень гіперпараметрів і забезпечити гнучкість, уся конфігурація симуляції здійснюється через JSON-файли та інтерфейс

командного рядка. Це дозволяє досліднику в момент запуску симуляції змінювати віртуальне середовище, вибирати базову LLM, обирати текстові інструкції та налаштовувати гіперпараметри без втручання у вихідний код системи.

Програмна система інтегрована з платформою логування експериментів MLFlow, що гарантує збереження конфігурації запуску та відтворюваність результатів. Зокрема, чітке розділення генераторів псевдовипадкових чисел (seeds) для етапів навчання, валідації та тестування дозволяє ізолювати вплив стохастичності середовища від реальної ефективності алгоритмів планування.

Керування життєвим циклом експерименту здійснюється через наступний набір конфігураційних параметрів:

1. `config (str)`: Назва json-файлу з аргументами запуску, які матимуть нижчий пріоритет за значення, передані через командний рядок.
2. `environment (str)`: Назва середовища для навчання та тестування агента.
3. `render (bool)`: Логічний прапорець значення якого вмикає/вимикає візуалізацію середовища.
4. `experiment_name (str)`: Спільна назва серії запусків (експеримент) для логування до MLFlow.
5. `run_name (str)`: Назва окремого запуску навчання/тестування агента.
6. `llm_name (str)`: Ідентифікатор LLM яку буде використано для генерації плану.
7. `prompt (str)`: Назва файлу з промптом для Планувальника.
8. `options_preset (str)`: Набір опцій для використання в Планувальнику та Акторі.
9. `train (bool)`: Логічний прапорець, значення якого вмикає/вимикає навчання агента.
10. `reflection_period (int)`: Період, з яким виконується рефлексія.
11. `train_seed (int)`: Значення ініціалізації псевдовипадкового генератора чисел для етапу навчання.

12. `train_n_episodes (int)`: Кількість епізодів, на яких буде проведено навчання агента.
13. `test_eval (int)`: Значення ініціалізації псевдовипадкового генератора чисел для етапу валідації.
14. `eval_n_episodes (int)`: Кількість епізодів, на яких буде проведено валідацію агента.
15. `eval_period (int)`: Періодичність (у епізодах), з якою буде проводитися валідація під час навчання.
16. `test_seed (int)`: Значення ініціалізації псевдовипадкового генератора чисел для етапу тестування.
17. `test_n_episodes (int)`: Кількість епізодів на яких буде проведено тестування агента.

Важливою особливістю цього підходу до конфігурації є ієрархія пріоритетів — параметри, передані безпосередньо через інтерфейс командного рядка, мають вищий пріоритет за відповідні значення, зчитані з JSON-файлу. Кожний повноцінний експеримент має свій фіксований JSON-файл конфігурації, що забезпечує відтворюваність. Водночас, для швидкого тестування чи зневадження системи дослідникy не потрібно створювати нові файли конфігурації — достатньо вказати існуючий *config* і перевизначити потрібний параметр (наприклад, змінити лише *train_seed* або *llm_name*) прямо в командому рядку.

Архітектура модуля рефлексії (Діаграма 4.6) базується на транзакційній моделі оновлення бази знань інтелектуального агента, ізолюючи логіку взаємодії з великою мовною моделлю від управління станом пам’яті. Центральним компонентом підсистеми є клас `TransactionalKBReflector`, який відповідає за періодичну обробку епізодичної історії, формування контексту та управління життєвим циклом записів бази знань. Формалізована схема десеріалізації відповідей LLM реалізована через клас `ModelOutput`, який інкапсулює ланцюжок міркувань, виявлені помилки агента та масив операцій модифікації бази знань. Ці операції успадковуються від `TransactionBase`, а їхня семантика визначається переліком `TransactionType` і включає створення,

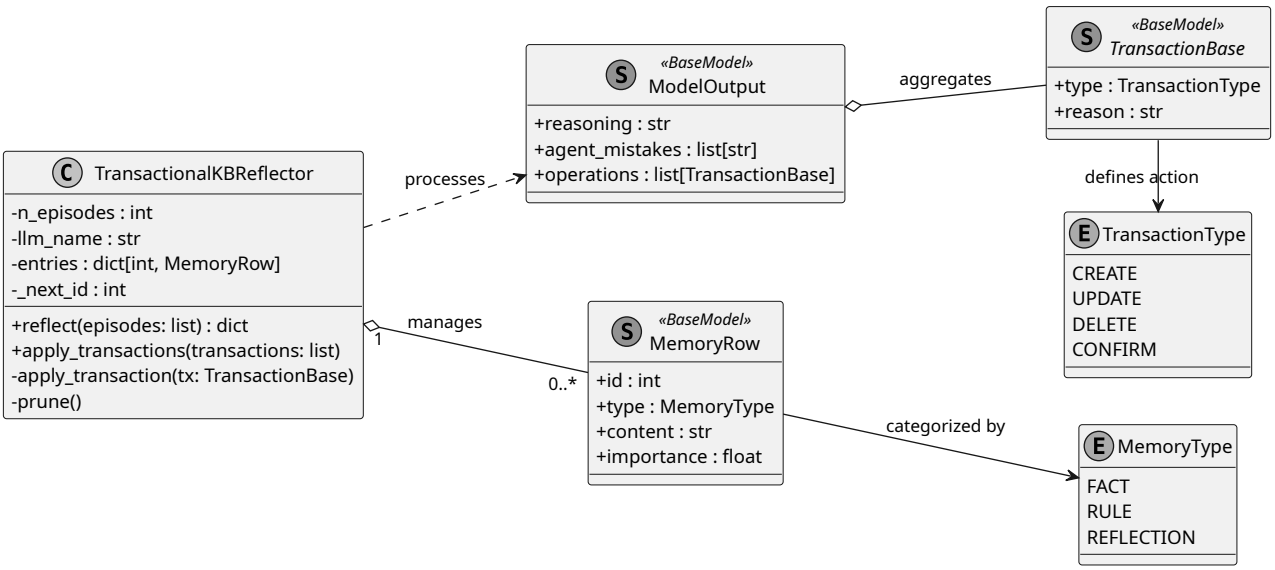



Рисунок 4.6: Діаграма класів компонента Reflection Memory.

оновлення, зменшення важливості для подальшого видалення та підтвердження важливості запису пам’яті. Атомарним елементом стану бази знань виступає клас `MemoryRow`, який містить категорію пам’яті, текстовий вміст запису пам’яті і скалярний параметр важливості. Кожному запису пам’яті відповідає унікальний ключ, який має цілочисельний тип.

Процес рефлексії епізодичної історії ініціюється викликом методу рефлексії, який трансформує історію в текст для LLM. LLM повертає послідовність транзакцій, після чого отримані транзакції послідовно застосовуються до існуючих записів. Модифікація параметра важливості спирається на механізм експоненційного згладжування, який ітеративно коригує вагу запису залежно від типу застосованої транзакції та цільового значення. Завершальним етапом рефлексії є процедура проріджування, яка остаточно вилучає з набору записів пам’яті всі елементи з рівнем важливості нижчим за фіксований поріг.

UML-діаграма 4.7 деталізує ієрархію поліморфних типів транзакцій, які застосовуються для модифікації бази знань LLM-агента. Базовою абстракцією виступає клас `TransactionBase`, який визначає спільні атрибути для всіх операцій, а саме, тип транзакції та текстове обґрунтування її генерації мовною моделлю. Від базового класу `TransactionBase` успадковуються чотири конкретні реалізації, кожна з яких визначає унікальну сигнатуру параметрів відповідно до семантики дії. Клас `TransactionCreate` забезпечує додавання нових знань та вимагає текстового контенту і категорії пам’яті через перелік `MemoryType`. Клас `TransactionUpdate` визначає зміни параметра

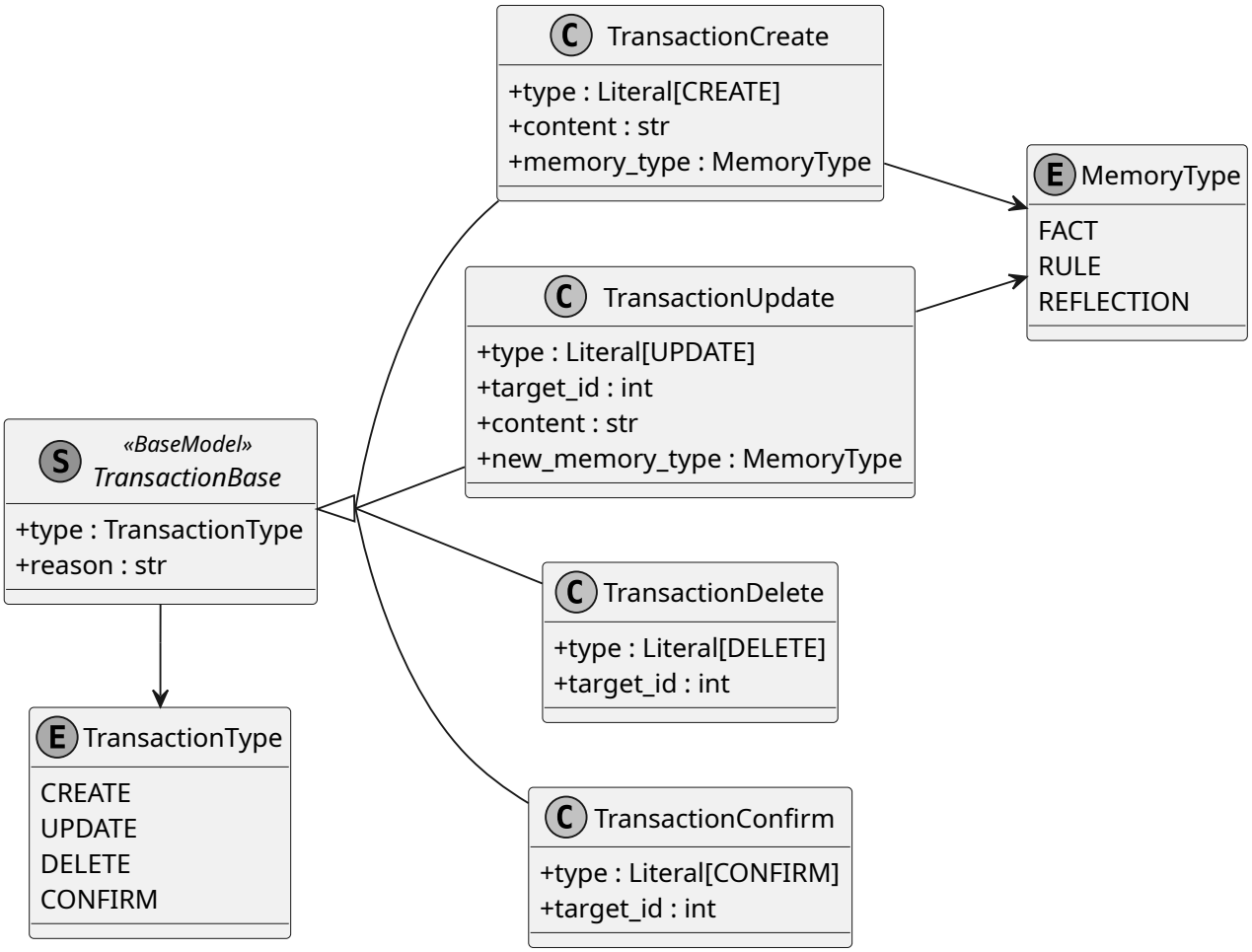



Рисунок 4.7: Діаграма класів транзакцій пам’яті.

існуючого запису, вимагаючи ідентифікатор цільового запису, оновлений контент та нову категорію пам’яті, якщо така зміна необхідна. Операції видалення та підтвердження, представлені класами `TransactionDelete` та `TransactionConfirm` відповідно, потребують виключно цілочисельного ідентифікатора цільового запису. Загальна типізація дій забезпечується переліком `TransactionType`, що гарантує синтаксичний контроль при десеріалізації виводу великої мовної моделі та унеможливлює виконання непередбачених операцій над станом бази знань. У поточній реалізації перелік типів записів пам’яті `MemoryType` не використовується LLM при генерації узагальнень. Повноцінне використання типів записів розглядається як перспективний напрямок подальших досліджень. У такому разі, `FACT` може бути використано для збереження фактів, які залишаються актуальними між епізодами, `RULE` — для збереження поведінкових інструкцій агента, а `REFLECTION` — для міркувань модуля рефлексивної пам’яті, що дозволить переносити проміжні логічні висновки на наступні ітерації рефлексії.

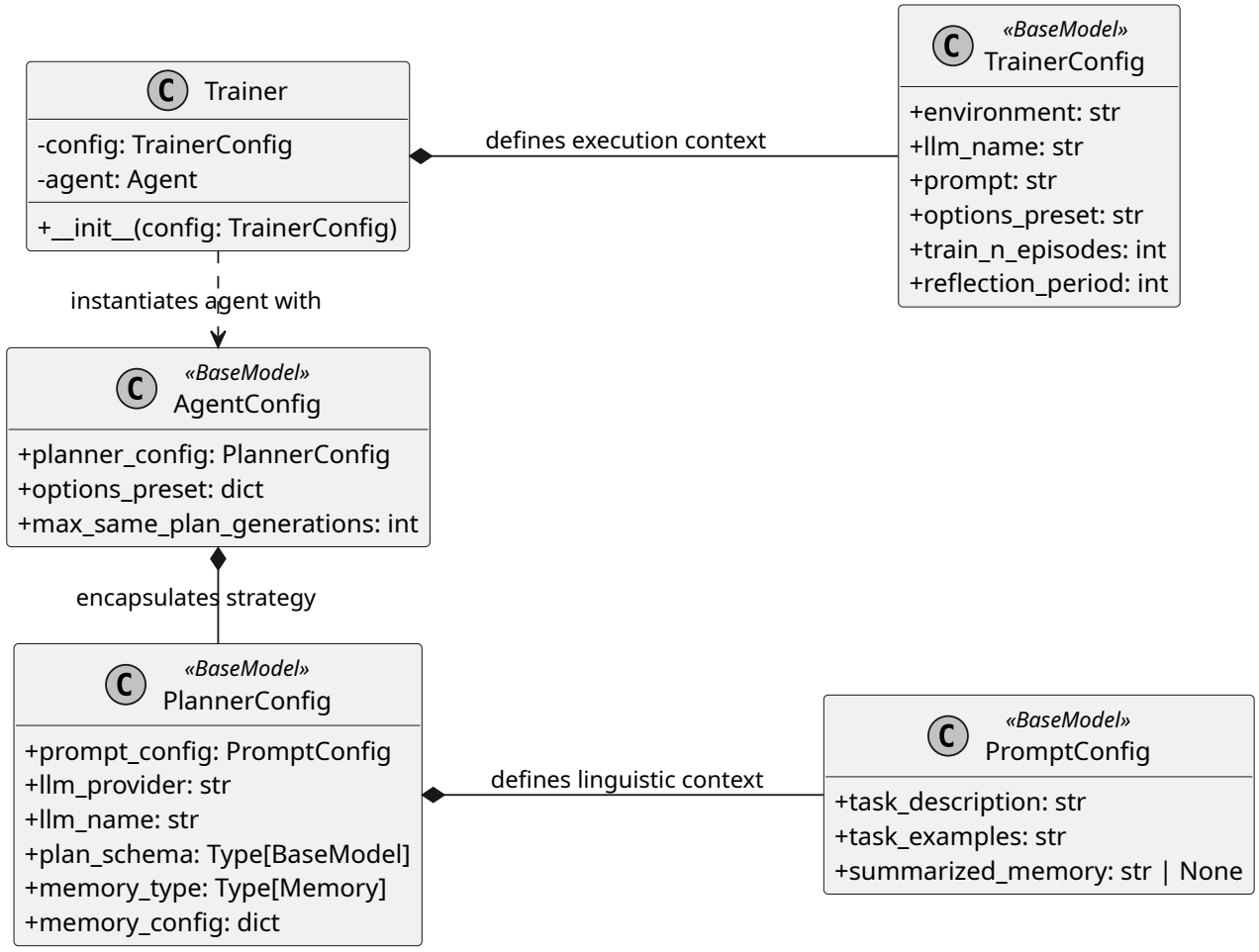



Рисунок 4.8: Діаграма класів конфігурації тренування агента.

Діаграма класів 4.8 ілюструє ієрархічну архітектуру конфігураційних модулів, розроблену для оркестрації процесу ініціалізації та навчання LLM-агента. Базовим компонентом виступає клас `Trainer`, який містить екземпляр агента та об'єкт конфігурації `TrainerConfig`, керуючи життєвим циклом експерименту. Параметри виконання визначаються класом `TrainerConfig`, який наслідує клас `BaseModel` з бібліотеки `pydantic` для суворої типізації конфігураційних параметрів, включаючи кількість епізодів навчання та ідентифікатор середовища. Клас конфігурації визначає делегування відповідальності, де параметри агента визначені в `AgentConfig`, який визначає обмеження на кількість однотипних планів та пресети опцій. Клас `TrainerConfig` агрегує `PlannerConfig`, який визначає параметри взаємодії Планувальника з великими мовними моделями, фіксуючи цільову схему планування, параметри підсистеми пам'яті та ідентифікатор LLM. На найнижчому рівні ієрархії функціонує клас `PromptConfig`, який визначає контекст для LLM — опис задачі, few-shot приклади та стан пам'яті. Така

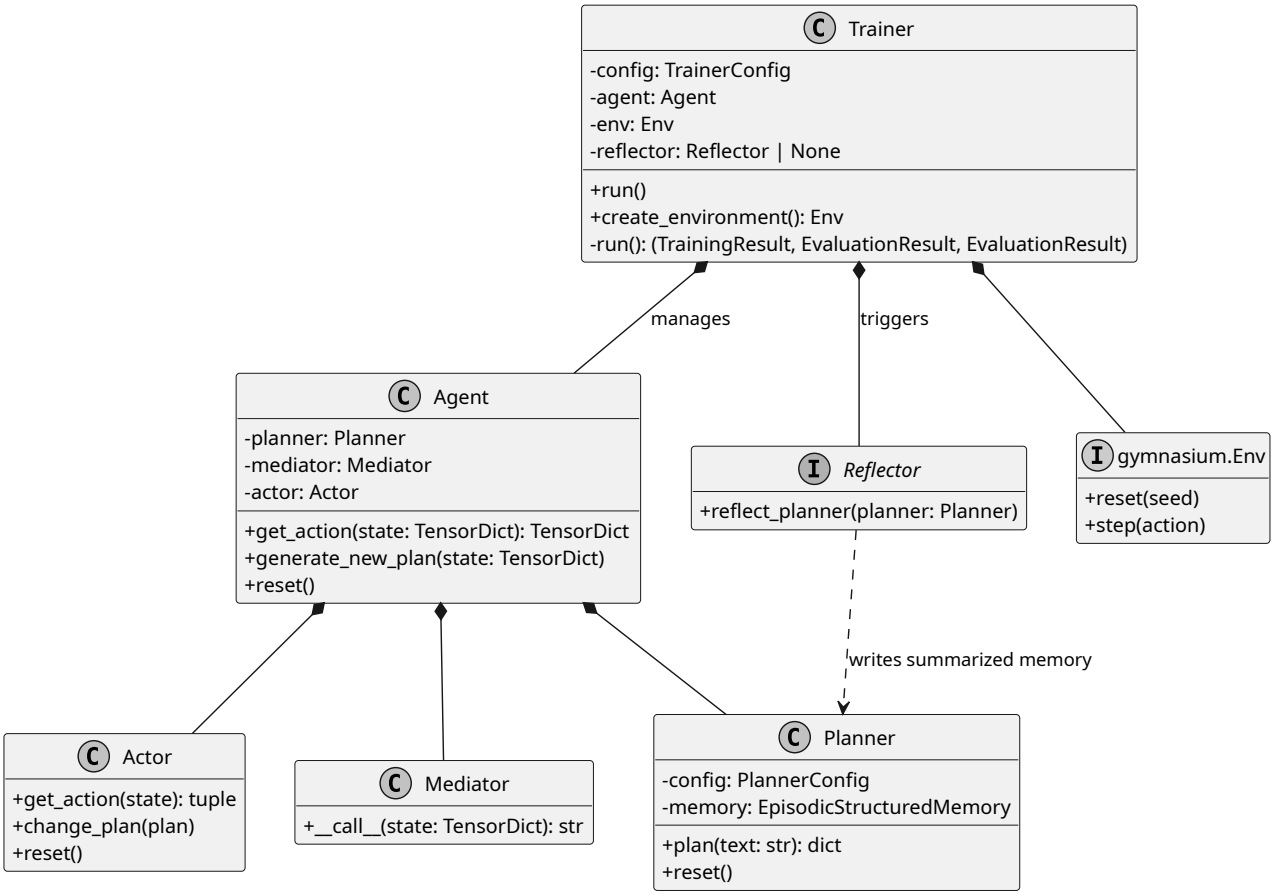



Рисунок 4.9: Діаграма класів тренування агента.

структура зв'язків конфігураційних класів мінімізує зчеплення модулів та дозволяє ізольовано змінювати конфігурацію і гіперпараметри без необхідності рефакторингу суміжних компонентів системи.

Діаграма класів 4.9 формалізує архітектурну декомпозицію системи навчання та виконання LLM-агента. Головним координаційним вузлом є клас `Trainer`, який включає середовище `Env` через композицію, об'єкт `Agent` та модуль рефлексії `Reflector`, який є опціональним. Клас `Agent` імплементує парадигму ієрархічного планування та виконання дій через структурну композицію трьох компонентів: `Mediator`, `Planner` та `Actor`. Модуль `Mediator` виконує відображення тензора стану у текстове подання. Клас `Planner` генерує абстрактний план на основі лінгвістичного контексту та епізодичної пам'яті. Безпосереднє виконання згенерованого плану делегується модулю `Actor`, який перетворює план у послідовність опцій та, в подальшому, у примітивні дії в середовищі. Рефлексивне покращення стратегії генерування планів формалізовано через інтерфейс `Reflector`, який ініціюється класом `Trainer` для рекурсивної модифікації внутрішнього стану модуля `Planner`.

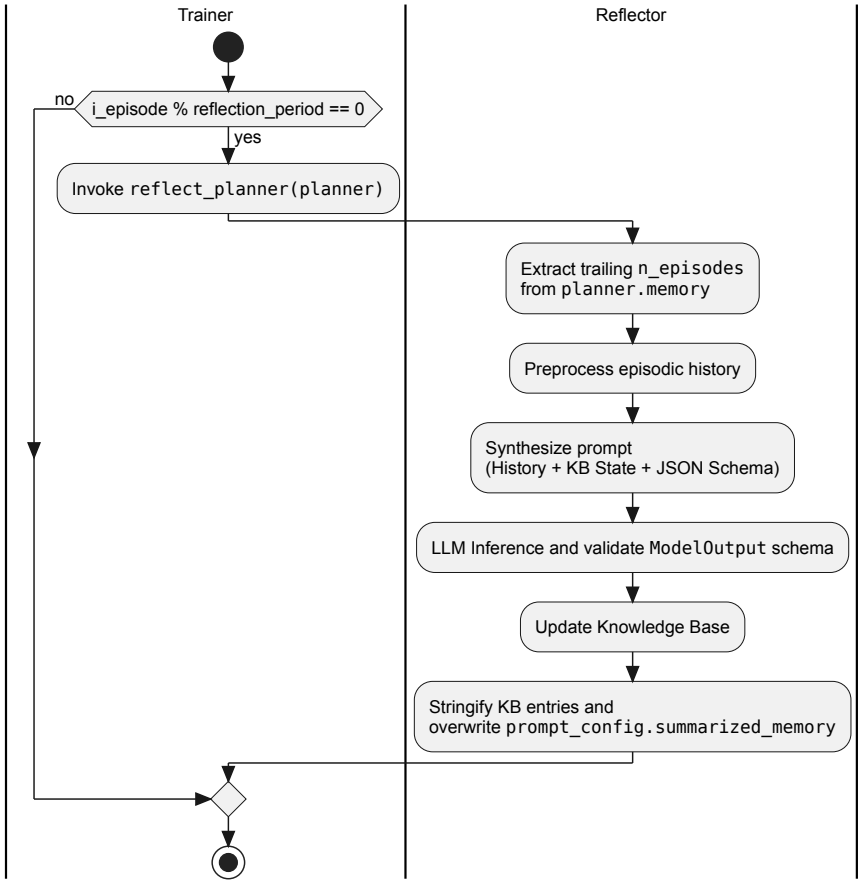



Рисунок 4.10: Діаграма активності навчання рефлексивної пам’яті.

Запропонована структура забезпечує ізоляцію функцій LLM-агента та створюює основу для незалежного масштабування підсистем.

На діаграмі 4.10 показано процес рефлексії. Процес рефлексії інтегровано в цикл навчання агента та керується класом `Trainer`. Після завершення кожного епізоду алгоритм перевіряє умову ініціалізації рефлексії, рефлексія відбувається кожні `reflection_period` навчальних епізодів. Ця умова реалізована як перевірка залишку від ділення яка має дорівнювати нулю. Процес продовжується викликом методу `reflect_planner`, до якого передається екземпляр планувальника `planner`.

Після передачі управління до компонента `Reflector`, алгоритм бере задану кількість останніх епізодів `n_episodes` безпосередньо з планувальника `planner.memory`, який накопичує історію взаємодії агента з середовищем. Отримані дані проходять етап попередньої обробки для структурування історії. Далі відбувається створення запиту до мовної моделі, який об’єднує історію епізодів, поточний стан бази знань агента та чітку JSON-схему для структурування вихідних даних.

Сформований контекст передається до LLM для генерації змін до бази знань. Згенерований LLM результат проходить валідацію на відповідність схемі `ModelOutput`. На основі отриманих транзакцій виконується оновлення бази знань. У цьому процесі записи можуть створюватися, оновлюватися, підтверджуватися або видалятися залежно від їхньої релевантності.

На фінальному етапі записи бази знань, які не були видалені, трансформуються у єдиний текстовий рядок. Цей текст перезаписує значення поля `summarized_memory` у конфігурації `prompt_config` планувальника, що дозволяє агенту використовувати оновлені знання у наступних епізодах. В подальшому процес рефлексії завершується, і управління повертається до головного циклу компонента `Trainer`.

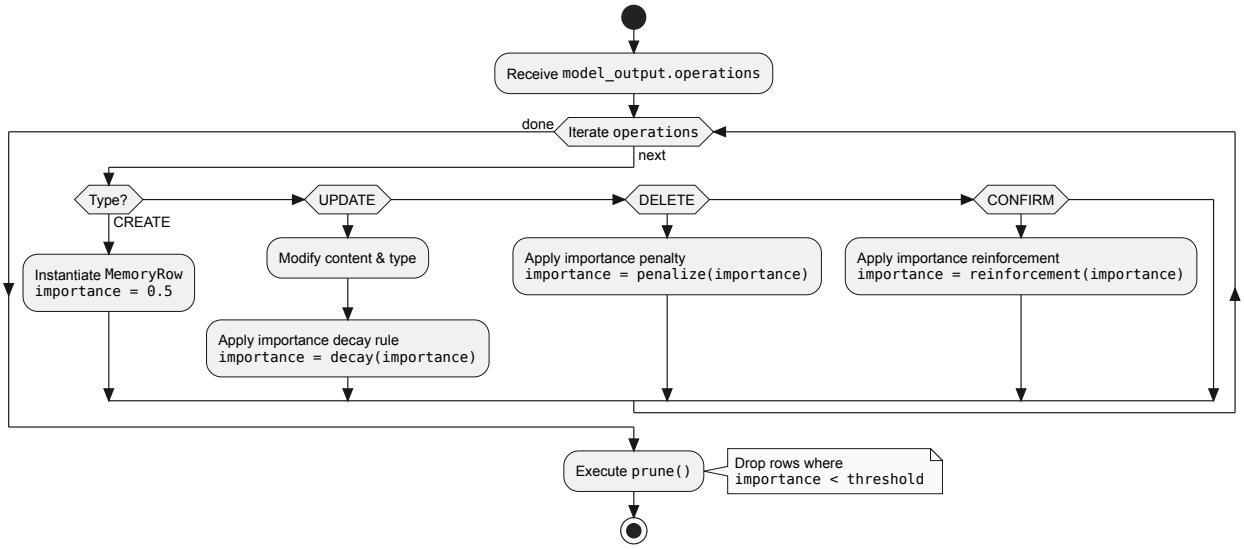



Рисунок 4.11: Діаграма активності оновлення стану пам’яті.

Діаграма 4.11 зображує процес оновлення бази знань, який починається з отримання списку транзакцій `model_output.operations`. Далі алгоритм послідовно проходить через кожну операцію в цьому списку, визначаючи її тип для виконання відповідної логіки. Якщо ідентифіковано тип `CREATE`, система створює та додає до пам’яті новий об’єкт `MemoryRow` та встановлює його початкове значення важливості `importance` на рівні 0.5. У випадку операції `UPDATE` відбувається зміна вмісту та типу існуючого запису, після чого застосовується правило поступового згасання, яке зменшує параметр важливості за допомогою правила `decay`. Для транзакції `DELETE` застосовується функція `penalize`, яка накладає штраф на параметр `importance`, суттєво знижуючи його значення замість миттєвого видалення запису. Якщо ж

надходить операція `CONFIRM`, алгоритм застосовує правило `reinforcement`, яке збільшує значення параметра `importance`, тим самим захищаючи правило від видалення.

Після того, як усі операції з набору було застосовано, цикл завершується і система ініціює виклик методу `prune()`. Під час цього фінального кроку алгоритм перевіряє всю базу знань та безповоротно видаляє ті записи, значення параметра `importance` яких опустилося нижче заданої величини `threshold`. На цьому виконання процесу оновлення пам’яті завершується.

## 4.5 Експериментальне дослідження та порівняльний аналіз ефективності різних конфігурацій агента

У дослідженні було використано середовище ColoredDoorKey, яке є двовимірним середовищем типу Minigrid з дискретними станами та діями. Середовище містить задачу, яка вимагає від агента відкрити замкнені двері за допомогою ключа відповідного кольору. На початку епізоду об’єкти в середовищі приховані, агент спочатку має дослідити середовище. Середовище містить два ключі різного кольору, один з яких є необхідним для відкриття дверей, а другий є відволікаючим об’єктом. Через обмежену видимість та обмежену інформацію агент змушений досліджувати середовище, щоб отримати необхідну інформацію про колір дверей, а також планувати дії, щоб знайти ключ відповідного кольору, підняти ключ, перейти до дверей та відкрити їх.

У цьому дослідженні як інструменти генерації та обробки природної мови було використано дві LLM сімейства Qwen3 — Qwen3-8b та Qwen3-30b, представлені у травні 2025 року [100]. Вибір цих моделей обґрунтований їхньою високою продуктивністю у задачах логічного міркування (reasoning), а також можливістю вибору кількості параметрів (як показано в таблиці 4.1), що дозволяє обирати моделі відповідно до обмежень пам’яті та обчислень.

Моделі Qwen3 пройшли попереднє навчання на корпусі обсягом 36 трильйонів токенів, який охоплює 119 мов. Моделі підтримують два режими роботи: стандартний та режим «міркувань» (Thinking Mode), в якому LLM генерує ланцюжок думок (Chain-of-Thought).

Сімейство моделей Qwen3 включає моделі різного розміру та архітектури.

Таблиця 4.1: Значення гіперпараметрів архітектури щільних моделей Qwen3.

| Models | Layers | Heads (Q / KV) | Tie Embedding | Context Length |
|---|---|---|---|---|
| Qwen3-0.6B | 28 | 16 / 8 | Yes | 32K |
| Qwen3-1.7B | 28 | 16 / 8 | Yes | 32K |
| Qwen3-4B | 36 | 32 / 8 | Yes | 128K |
| Qwen3-8B | 36 | 32 / 8 | No | 128K |
| Qwen3-14B | 40 | 40 / 8 | No | 128K |
| Qwen3-32B | 64 | 64 / 8 | No | 128K |

Таблиця 4.2: Значення гіперпараметрів архітектури MoE моделей Qwen3.

| Models | Layers | Heads (Q / KV) | # Experts (Total / Activated) | Context Length |
|---|---|---|---|---|
| Qwen3-30B-A3B | 48 | 32 / 4 | 128 / 8 | 128K |
| Qwen3-235B-A22B | 94 | 64 / 4 | 128 / 8 | 128K |

Моделі з щільною архітектурою (таблиця 4.1), в яких при обчисленнях активуються всі параметри нейромережі, мають число параметрів від 0.6 млрд. до 32 млрд. За результатами експериментів, виконаних в розділі 3, серед щільних моделей було обрано модель Qwen3-8B, а серед MoE моделей — модель Qwen3-30B-A3B. У дослідженні були використані моделі з квантуванням Q4_K_M.

Модель Qwen3-8B є представником щільних трансформерів з сімейства Qwen3 (таблиця 4.1). Qwen3-8B — це модель середнього розміру, яка призначена для розгортання на споживчому обладнанні. Модель включає 36 Transformer-шарів, використовує 32 голови для запитів та 8 голів для ключів-значень. На відміну від менших версій (0.6B/1.7B), у цій моделі не використовується техніка Tie Embedding, що дозволяє підвищити точність представлення токенів. Модель підтримує розмір контекстного вікна до 128 тисяч токенів. Модель Qwen3-8B з квантуванням Q4_K_M при завантаженні в пам’ять графічного прискорювача використовує 5.2 GB відеопам’яті, без врахування пам’яті для завантаження контексту.

Модель Qwen3-30B-A3B (таблиця 4.2) побудована на архітектурі Mixture of Experts (MoE). Для завантаження моделі з квантування Q4_K_M необхідно від 19GB пам’яті, без врахування пам’яті для завантаження контексту моделі. Модель містить 128 експертів, з яких для обробки кожного токена маршрутизатор (router) обирає 8 найбільш релевантних експертів. У цій архітектурі відсутні «спільні експерти» (shared experts), а спеціалізація

досягається через функцію втрат балансування навантаження (global-batch load balancing loss). Для кожного токена активується близько 3 млрд. параметрів, що підвищує швидкість обчислень та робить її близькою до моделей з розміром від 3 до 4 млрд. параметрів при збереженні якості генерації, характерної для моделей з розміром 30 млрд. параметрів. Qwen3-30B-A3B має 48 шарів, 32 голови для запитів та 8 голів для ключів-значень. Модель підтримує довжину контексту до 128 тисяч токенів. Як і інші моделі з сімейства Qwen3, модель Qwen3-30B-A3B підтримує міркування, вимкнення міркувань, визначення інтенсивності міркувань.

Більша з моделей — Qwen3-30b, була використана для узагальнення історії епізодів та управління пам’яттю. Менша з моделей — Qwen3-8b, була використана в модулі планування, вона використовувала узагальнення отримані від більшої моделі для планування в просторі опцій. Застосування моделей різного розміру є доцільним, оскільки планування має бути швидким, оскільки виконується декілька разів за епізод. Натомість, узагальнення історії відбувається один раз за декілька епізодів, тому до цього етапу застосовуються слабші вимоги що до тривалості.

Для забезпечення можливості інференсу моделей Qwen3-30B та Qwen3-8B в умовах обмеженої відеопам’яті (24 ГБ) були використані LLM, квантовані методом пост-тренувального квантування (Post-Training Quantization, PTQ) до формату GGUF з використанням схеми Q4_K_M.

Цей метод належить до сімейства «k-quants», яке використовує блокове квантування для мінімізації втрат перплексії порівняно з вихідними вагами FP16. Схема Q4_K_M (4-bit quantization, K-series, Medium mix) є гібридним підходом, який розподіляє бітову точність залежно від важливості конкретних тензорів. Ваги моделі розбиваються на суперблоки (super-blocks), кожен з яких містить менші блоки (зазвичай 32 ваги). Для кожного блоку зберігається масштабний коефіцієнт (scale) та мінімальне значення (minimum), що дозволяє ефективніше упаковувати дані порівняно з квантуванням Q4. Критично важливі тензори, такі як ваги механізму уваги `output.weight` та ваги feed-forward шарів `ffn_down` (часто позначаються як w2), квантуються з вищою точністю — 6 біт (Q6_K). Це дозволяє зберегти точність моделі у найбільш чутливих до змін параметрів шарах. Інші тензори, зокрема, `attn_q`, `attn_k`, `attn_v`, `ffn_up`, `ffn_gate`, квантуються до 4 біт (Q4_K).

З використанням квантування розмір моделі Qwen3-8B зменшується з 16 ГБ (FP16) до 5.5 ГБ. Для моделі Qwen3-30B-A3B необхідний обсяг відеопам'яті зменшується з 30 ГБ (FP16) до 19-20 ГБ, що дозволяє виконати повне завантаження моделі у пам'ять NVIDIA RTX 3090 без необхідності завантаження частини шарів у повільну системну RAM (offloading). Хоча модель Qwen3-30B-A3B має 30 мільярдів параметрів, активними є лише 3 мільярди. Однак у пам'яті GPU повинні зберігатися всі 30 мільярдів параметрів (усі експерти), щоб маршрутизатор (router) міг швидко до них звертатися. Без квантування Q4_K_M локальний запуск моделі Qwen3-30B-A3B на використаному для цього дослідження обладнанні виявився б неможливим.

Щоб перевірити вклад запропонованого механізму пам'яті були проведені експерименти з різними підходами до пам'яті агента. Найпростіший варіант — це агент без пам'яті, у якому агент отримує безпосередньо спостереження середовища, але не має доступу ні до історії епізоду, ні до історії в цілому. Цей найпростіший підхід показує граничний нижній рівень точності, який може досягти агент. Наступний за складністю підхід — це епізодична пам'ять, в якій агент отримує на вхід історію взаємодії з поточним епізодом. Такий підхід дозволяє агенту виправляти зроблені в поточному епізоді помилки, але не дозволяє переносити досвід між епізодами.

Взаємодія агента з середовищем була розділена на три етапи: Train, Eval, Test. Для кожного етапу був обраний окремий, фіксований набір значень ініціалізації генератора псевдовипадкових чисел для відтворюваності експериментів. На етапі Train агент збирав історію взаємодії з середовищем, ця історія була використана або для формування offline узагальнень, або в online оновленні пам'яті. Етап Eval чергувався з Train, кожні N епізодів, та був необхідний для оцінювання динаміки якості роботи агента. Етап Test був завершаючим етапом, на якому агент тестувався на великому числі епізодів, для обчислення остаточної оцінки якості роботи агента.

Метрики для оцінки ефективності агента обчислювалися на трьох рівнях: покроково, для окремого епізоду та агреговані для декількох епізодів. Наступні метрики обчислювалися покроково: кількість перепланувань на поточному кроці, кількість викликів LLM на поточному кроці, тривалість процесу планування на поточному кроці (в секундах), винагорода на поточному кроці. По завершенню кожного епізоду обчислювалися такі метрики: успішність

виконання завдання (бінарний показник), кількість кроків виконаних агентом, сумарна винагорода за епізод, кількість перепланувань за епізод, кількість викликів LLM за епізод, тривалість планування за весь епізод, середня кількість викликів LLM за крок. Агреговані метрики, які розраховуються для послідовності епізодів: середня частка успішності вирішення завдання, середня кількість кроків до завершення епізоду, середня сумарна винагорода за епізод, середня кількість перепланувань на епізод.

Успішність вирішення завдання в середовищі Minigrid визначається бінарним значенням успіх/невдача. Для $N$ епізодів ми визначаємо середню частку успішності виконання завдання наступним чином:

$$SR = \frac{1}{N} \sum_{i=1}^{N} s_i, \tag{4.24}$$

де

$$s_i = \begin{cases} 1, & \text{якщо епізод завершено та } R_i > 0; \\ 0, & \text{в іншому випадку.} \end{cases} \tag{4.25}$$

Для обчислення сумарної винагороди за $i$-тий епізод, ми додаємо винагороди $r_t$ за кожен крок $t$ без знецінювання:

$$G_i = \sum_{t=0}^{T} r_t, \tag{4.26}$$

де $t$ — індекс часового кроку, $T$ — фактична кількість кроків, виконана агентом.

У середовищі Minigrid винагороди розріджені, тобто винагорода $r_t$ може бути відмінною від 0 лише в кінці епізоду, на кроці $T$:

$$r_T = 1 - 0.9 \cdot \left( \frac{T}{T_{max}} \right). \tag{4.27}$$

де $T_{max}$ — максимальна дозволена довжина епізоду.

Якщо агент не досяг цілі, то $r_t = 0$ для всіх $t$, тому сумарна винагорода може бути обчислена як винагорода за останній крок:

$$G_i = \begin{cases} 1 - 0.9 \cdot \left( \frac{T}{T_{max}} \right), & \text{якщо ціль досягнута;} \\ 0, & \text{в іншому випадку.} \end{cases} \tag{4.28}$$

Щоб розрхувати винагороду для $N$ епізодів, ми обчислюємо середнє

значення сумарної винагороди:

$$\overline{G} = \frac{1}{N} \sum_{i=1}^{N} G_i. \tag{4.29}$$

Кількість перепланувань розраховується на трьох рівнях: перепланування за крок, за епізод та середня кількість перепланувань за декілька епізодів. На кожному кроці модуль Планувальник може виконати 0 або більше перепланувань, $n_t \in \{0, 1, ...\}$, де $n_t$ — це кількість викликів Планувальника на часовому кроці $t$. На рівні одного епізоду кількість перепланувань обчислюється як сума перепланувань за кожен крок в епізоді. Якщо $T$ — це загальна кількість кроків в епізоді, то сумарна кількість перепланувань $M_i$ для $i$-го епізоду дорівнює:

$$M_i = \sum_{t=0}^{T} n_t. \tag{4.30}$$

У випадку, коли потрібно обчислити кількість перепланувань для декількох епізодів $N$, ми обчислюємо середнє значення по кількості перепланувань для всіх епізодів:

$$\overline{M} = \frac{1}{N} \sum_{i=1}^{N} M_i = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=0}^{T_i} n_{i,t} \right), \tag{4.31}$$

де $N$ — загальна кількість епізодів, $T_i$ — кількість кроків в $i$-му епізоді, $n_{i,t}$ — кількість перепланувань на кроці $t$ в епізоді $i$.

Також ми обчислювали метрику кількості викликів LLM на одне перепланування, мовна модель може викликатися додатково у випадку виникнення синтаксичних помилок:

$$\overline{N}_{\text{LLM calls}} = \frac{\sum_{t=0}^{T} n_{\text{LLM calls},t}}{M_{\text{replan}}}. \tag{4.32}$$

Щоб визначити усереднену тривалість одного перепланування протягом епізоду, ми обчислювали відношення суми часу до першого успішно згенерованого плану до кількості перепланувань:

$$\overline{T}_{\text{planning}} = \frac{\sum_{t=0}^{T} \text{planning\_duration}_t}{M}. \tag{4.33}$$

Ми провели обчислювальні експерименти в середовищі Minigrid ColoredDoorKey. Всі експерименти проводилися на комп'ютерній системі з графічним прискорювачем NVIDIA RTX 3090 (24 GB VRAM). Для виконання LLM була використана утиліта Ollama 0.13.5. Були використані LLM квантовані у форматі Q4_K_M. Гіперпараметр «температура» був встановлений на значення 0.1. Максимальне число згенерованих токенів було обмежено до 4096 одиниць.

Ми порівняли декілька конфігурацій пам'яті агента:

1. **w/o Memory:** У агента була відсутня пам'ять та інструкції.
2. **Baseline E-1:** Агент пам'ятав лише поточний епізод.
3. **Offline Reflection:** Агент пам'ятав поточний епізод і отримав інструкції від LLM Qwen3-30b, згенеровані завчасно на заздалегідь зібраних даних.
4. **Text Reflection E-2:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 2 епізоди на останніх двох епізодах.
5. **Text Reflection E-5:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 5 епізодів на останніх п'яти епізодах.
6. **Text Reflection E-2 CP-1:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 2 епізоди на останній парі, яка складалася з вдалого та не вдалого епізодів.
7. **Text Reflection E-2 CP-3:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 2 епізоди на останніх трьох парах, які складалися з вдалих та не вдалих епізодів.
8. **Text Reflection E-5 CP-1:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 5 епізодів на останній парі, яка складалася з вдалого та не вдалого епізодів.
9. **Text Reflection E-5 CP-3:** Агент мав пам'ять поточного епізоду та виконував не структуровану рефлексію кожні 5 епізодів на останніх трьох парах, які складалися з вдалих та не вдалих епізодів.

10. **Reflection E-2:** Агент мав пам’ять поточного епізоду та виконував структуровану рефлексію з інструментами управління кожні 2 епізоди.
11. **Reflection E-5:** Агент мав пам’ять поточного епізоду та виконував структуровану рефлексію з інструментами управління кожні 5 епізодів.
12. **Manual Instructions:** Агент пам’ятав поточний епізод і отримав інструкції сформульовані людиною.

Процес планування модуля Планувальник контролюється шляхом визначення промпту, який описує роль, завдання та структуру плану. Розділення управління пам’яттю та планування високрівневих дій агента дозволило ізолювати вплив алгоритмів пам’яті від здатності мовної моделі до загального логічного висновку.

**Промпт 4.2: Промпт агента без пам’яті**

**Role**

```
You are an agent in the minigrid environment.
```

**Tasks**

```
Your task is to build a plan of actions to reach the main objective: open
the door. Format the plan as a compact JSON object.
```

**Промпт 4.3: Промпт з вказівками від людини-експерта**

**Role**

```
You are an agent in the minigrid environment.
```

**Tasks**

```
Your task is to build a plan of actions to reach the main objective: open
the door. Format the plan as a compact JSON object.
```

**Summary**

```
1. There are two keys with different color. 2. You need to find and use one
which has the same color as the door. 3. To interact with any object you
need to go close to it. For example, go to key to pick up it. 4. Door is
locked, you need a key with the same color to open it. 5. If you picked up
a key with wrong color, you may drop it. 6. The final goal is to open the
door.
```

У конфігурації *w/o Memory* модуль Планувальник агента отримував мінімалістичний промпт (Лістинг 4.2), який містив лише опис ролі та фінальної

цілі (завдання — відкрити двері), змушуючи агента покладатися виключно на власні параметричні знання.

**Промпт 4.4: Промпт з offline узагальненнями від моделі Qwen3-30b**

**Role**

```
You are an agent in the minigrid environment.
```

**Tasks**

```
Your task is to build a plan of actions to reach the main objective: open
the door. Format the plan as a compact JSON object.
```

**Summary**

```
1. Explore for objects (key and door) 2. If key is found but not accessible,
move to key's location 3. Pick up the key 4. Explore for the door 5. Move to
the door's location 6. Open the door using the key
```

Для конфігурації *Manual Instructions* був розроблений промпт (Лістинг 4.3), який включав експертні знання про структуру середовища, зокрема правила взаємодії з ключами та дверима. Ця конфігурація слугувала верхньою межею (upper bound) ефективності для оцінки алгоритмів автоматичного узагальнення досвіду.


Рисунок 4.12: Залежність значення успішності вирішення завдання від епізоду під час навчання, обчислене на валідаційному наборі епізодів та усереднене для набору початкових станів. Показано конфігурації *Reflection E-2*, *Reflection E-5*, *Text Reflection E-2*, *Text Reflection E-5*.

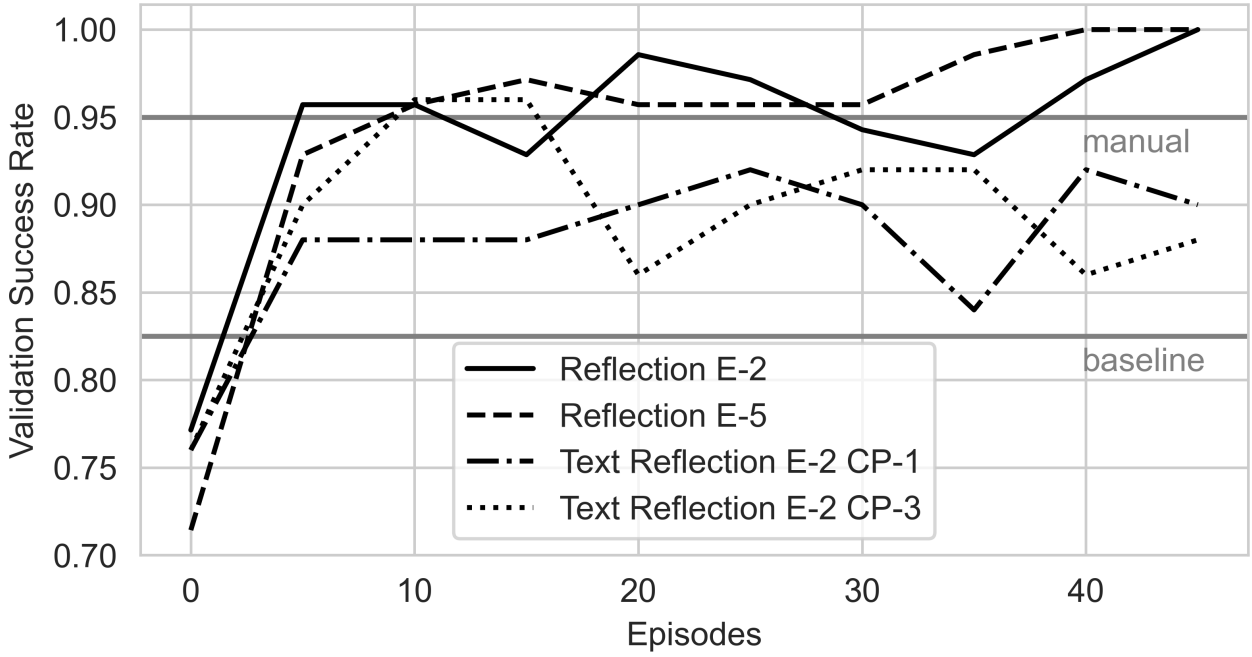



Рисунок 4.13: Залежність значення успішності вирішення завдання від епізоду під час навчання, обчислене на валідаційному наборі епізодів та усереднене для набору початкових станів. Показано конфігурації *Reflection E-2*, *Reflection E-5*, *Text Reflection E-2 CP-1*, *Text Reflection E-2 CP-1*.

Конфігурація *Offline Reflection* використовувала промпт (Лістинг 4.4), який містив узагальнений покроковий план, згенерований потужнішою моделлю Qwen3-30b на основі заздалегідь зібраного набору траєкторій. Це дозволило оцінити ефективність статичного узагальнення порівняно з динамічним механізмом самокорекції. Усі конфігурації вимагали від моделі форматування абстрактного плану дій у вигляді компактного JSON-об’єкта для подальшого виконання модулем Актор.

Рис. 4.12 показує динаміку середньої точності вирішення завдання на валідаційному наборі епізодів. На перших 20 епізодах точність зростає майже для всіх конфігурацій, але після 20 епізодів для *Text Reflection E-2* спостерігається нестабільність і значне падіння. До кінця навчання конфігурації *Reflection E-2* та *Reflection E-5* збергають зростання точності і досягають максимального результату (1.0).

Рис. 4.13 показує динаміку середньої точності вирішення завдання на валідаційному наборі епізодів. Конфігурації *Text Reflection E-2 CP-1* та *Text Reflection E-2 CP-1* спочатку показують швидке зростання, але швидко досягають плато точності і в кінці навчання (епізод 50) не досягають точності

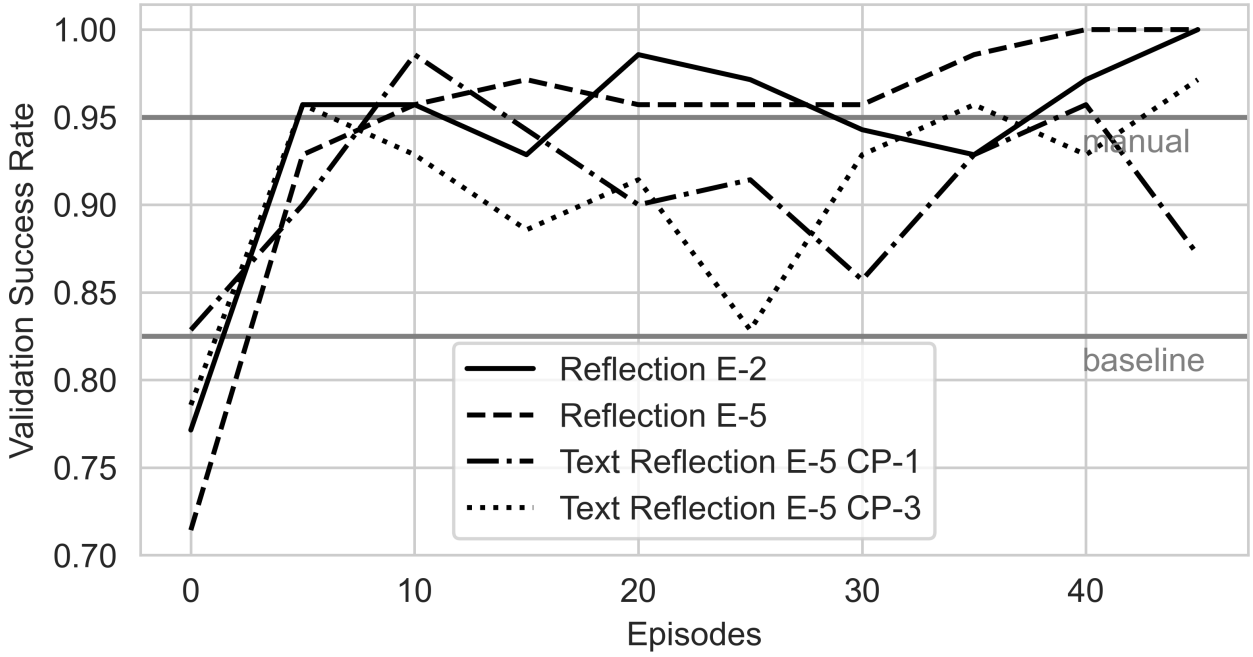



Рисунок 4.14: Залежність значення успішності вирішення завдання від епізоду під час навчання, обчислене на валідаційному наборі епізодів та усереднене для набору початкових станів. Показано конфігурації *Reflection E-2*, *Reflection E-5*, *Text Reflection E-5 CP-1*, *Text Reflection E-5 CP-3*.

*Manual Instructions*. На тому ж епізоді конфігурації *Reflection E-2* та *Reflection E-5* досягають максимальної точності, що ілюструє перевагу структурованої рефлексії.

На рис. 4.14 можна побачити динаміку середньої точності вирішення завдання на валідаційному наборі епізодів для структурованої рефлексії (конфігурації *Reflection E-2*, *Reflection E-5*) та не структурованої рефлексї з контрастивними прикладами (конфігурації *Text Reflection E-5 CP-1* та *Text Reflection E-5 CP-3*). Конфігруації з неструктурованою рефлексією досягають рівня *Manual Instructions*, але поступаються структурованій рефлексії.

Рис. 4.15 показує динаміку сумарної винагороди на валідаційному наборі епізодів. Можна побачити, що сумарна винагорода для всіх конфігурацій пам’яті спочатку швидко зростає та перевершує показники для *Baseline E-1* та *Manual Instructions*, обчислених на валідаційному наборі епізодів. Після 20 епізодів показники конфігурацій *Text Reflection E-2* та *Text Reflection E-5* погіршуються. Натомість, значення сумарної винагороди для конфігурацій *Reflection E-2* та *Reflection E-5* продовжують зростати та досягають максимальних значень. Ці результати демонструють перевагу


Рисунок 4.15: Залежність значення сумарної винагороди від епізоду під час навчання, обчислене на валідаційному наборі епізодів та усереднене для набору початкових станів. Показано конфігурації *Reflection E-2*, *Reflection E-5*, *Text Reflection E-2*, *Text Reflection E-5*.

структурованої рефлексії під час навчання, адже сумарна винагорода для конфігурацій *Reflection E-2* та *Reflection E-5* зростала протягом навчальних епізодів.

Рис. 4.16 показує динаміку сумарної винагороди на валідаційному наборі епізодів. Можна побачити, що сумарна винагорода для всіх конфігурацій пам'яті спочатку швидко зростає та перевершує показники для *Baseline E-1* та *Manual Instructions*, після 10 епізодів відбувається зниження, після чого показники зростають. В результаті конфігурації *Text Reflection E-5 CP-1*, *Text Reflection E-5 CP-1* наближаються, але не досягають показників конфігурацій *Reflection E-2*, *Reflection E-5*, що вказує на перевагу структурованої рефлексії.

У підсумку, криві динаміки навчання, які містять середню точність вирішення завдання та сумарну винагороду показують перевагу структурованої рефлексії над неструктурованою. Конфігурації агента з структурованою рефлексією зазнають помітно менших коливань та падінь, досягають вищих фінальних результатів. Наприклад, конфігурації *Reflection E-2* та *Reflection E-5* досягли максимальної точності на валідаційному наборі епізодів в кінці навчання агента.

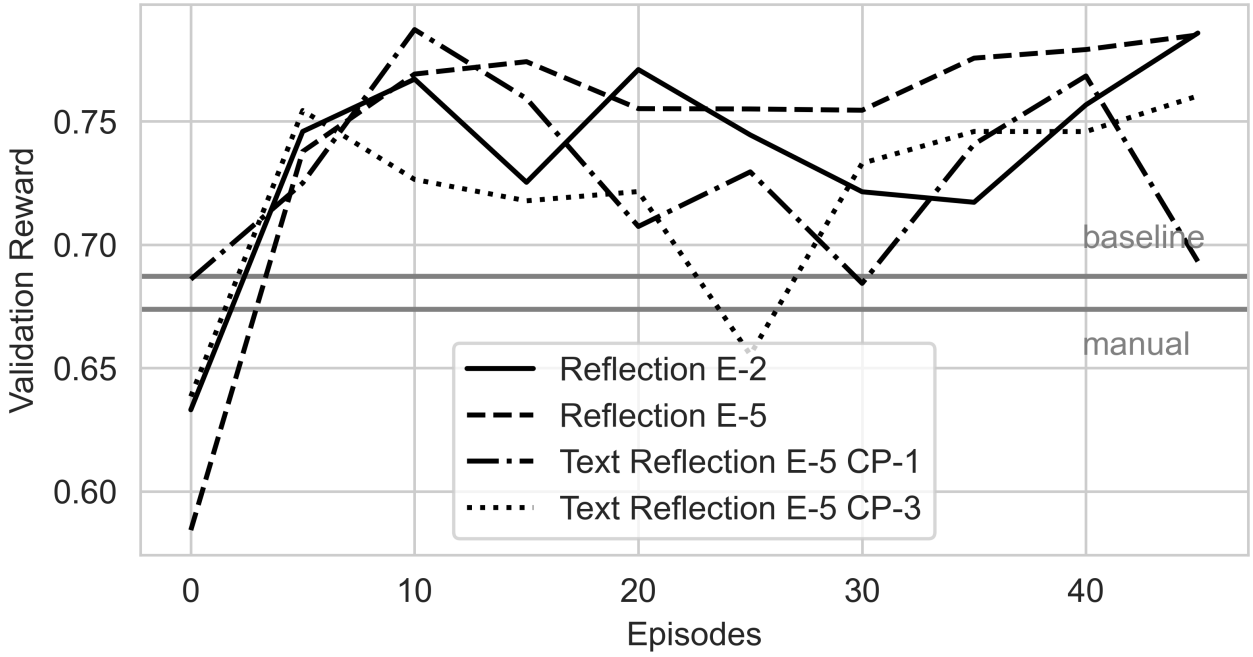



Рисунок 4.16: Залежність значення сумарної винагороди від епізоду під час навчання, обчислене на валідаційному наборі епізодів та усереднене для набору початкових станів. Показано конфігурації *Reflection E-2*, *Reflection E-5*, *Text Reflection E-5 CP-1*, *Text Reflection E-5 CP-1*.

Приклад фінальної версії набору правил для методу Reflection:

**Вихід моделі 4.1: Фінальні правила для механізму рефлексії**

```
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.
```

Аналіз результатів експериментів в таблиці 4.3 свідчить про статистично значиму різницю між результатами агентів з різною конфігурацією пам’яті: у той час як базові конфігурації *w/o Memory* та *Baseline E-1* демонструють низьку результативність через відсутність механізмів навчання на помилках, використання структурованої онлайн-рефлексії в *Reflection E-5* дозволяє

Таблиця 4.3: Порівняння успішності виконання завдання для різних конфігурацій пам'яті агента, розраховані по вибірці значень для різних початкових станів. Наведено середнє значення, середнє квадратичне відхилення, нижня та верхня межі 95% довірчого інтервалу, мінімальне та максимальне значення у вибірці.

| Configuration | Mean | STD | CI low | CI high | Min | Max |
|---|---|---|---|---|---|---|
| w/o Memory | 0.61 | 0.03 | 0.59 | 0.63 | 0.56 | 0.64 |
| Baseline E-1 | 0.63 | 0.04 | 0.61 | 0.66 | 0.59 | 0.70 |
| Text Reflection E-2 CP-1 | 0.72 | 0.13 | 0.65 | 0.79 | 0.51 | 0.98 |
| Offline Reflection | 0.72 | 0.02 | 0.71 | 0.74 | 0.71 | 0.75 |
| Text Reflection E-5 CP-1 | 0.75 | 0.15 | 0.68 | 0.83 | 0.46 | 1.00 |
| Text Reflection E-2 | 0.80 | 0.07 | 0.76 | 0.85 | 0.73 | 0.95 |
| Text Reflection E-5 CP-3 | 0.84 | 0.15 | 0.76 | 0.92 | 0.57 | 0.99 |
| Text Reflection E-2 CP-3 | 0.86 | 0.06 | 0.82 | 0.89 | 0.78 | 0.96 |
| Text Reflection E-5 | 0.88 | 0.10 | 0.82 | 0.92 | 0.60 | 0.99 |
| Reflection E-2 | 0.92 | 0.11 | 0.86 | 0.97 | 0.63 | 1.00 |
| Reflection E-5 | 0.93 | 0.08 | 0.89 | 0.97 | 0.78 | 1.00 |
| Manual Instruction | 0.97 | 0.02 | 0.96 | 0.98 | 0.95 | 0.99 |

досягти показника 0.93, що близько до верхньої межі, заданої *Manual Instruction* (0.97). Виявлено, що структурований вихід є ефективнішим за використання єдиного текстового рядка для пам'яті, оскільки *Reflection E-2* (0.92) суттєво перевершує *Text Reflection E-2* (0.80), а розширення вікна контексту до п'яти епізодів призводить до знаходження кращої стратегії порівняно з двома епізодами. Окремої уваги заслуговує дестабілізуючий ефект контрастивних пар у конфігураціях *Text Reflection E-5 CP-1* та *Text Reflection E-2 CP-1*, де високий STD (0.13-0.15) та низькі мінімальні значення (0.46-0.51) вказують на ризик перенавчання на випадкових шумах одиничних невдалих епізодів, що нівелюється лише при збільшенні кількості пар до трьох (*Text Reflection E-2 CP-3* (0.86)). Зрештою, перевага динамічної адаптації над статичною очевидна: навіть за умови використання моделі відносно великого розміру — Qwen3-30B-A3B, *Offline Reflection* (0.72) значно поступається онлайн-методам.

У таблиці 4.4 наведено дані інтерквартильного середнього (IQM) для тих же даних, на яких обчислена таблиця 4.3. Результати підтверджують високу ефективність структурованого методу рефлексії, при цьому *Reflection E-2* та *Reflection E-5* демонструють ідентичний показник 0.95, що свідчить

Таблиця 4.4: Порівняння успішності виконання завдання для різних конфігурацій пам'яті агента. Наведено інтерквартильне середнє значення, нижня та верхня межі 95% довірчого інтервалу для інтерквартильного середнього значення.

| Configuration | IQM | IQM CI low | IQM CI high |
|---|---|---|---|
| w/o Memory | 0.61 | 0.58 | 0.63 |
| Baseline E-1 | 0.63 | 0.60 | 0.67 |
| Offline Reflection | 0.71 | 0.71 | 0.75 |
| Text Reflection E-2 CP-1 | 0.71 | 0.64 | 0.79 |
| Text Reflection E-5 CP-1 | 0.77 | 0.68 | 0.84 |
| Text Reflection E-2 | 0.79 | 0.74 | 0.85 |
| Text Reflection E-2 CP-3 | 0.86 | 0.81 | 0.90 |
| Text Reflection E-5 CP-3 | 0.88 | 0.75 | 0.96 |
| Text Reflection E-5 | 0.90 | 0.84 | 0.94 |
| Reflection E-2 | 0.95 | 0.89 | 0.98 |
| Reflection E-5 | 0.95 | 0.89 | 0.98 |
| Manual Instruction | 0.97 | 0.95 | 0.98 |

про можливість наближення до рівня *Manual Instruction* (0.97) за умови виключення екстремальних невдач. Інтерквартильні середні показують покращення конфігурацій з довгим контекстом та контрастивними парами: *Text Reflection E-5 CP-3* зросла до 0.88, а *Text Reflection E-5* досягла 0.90, що вказує на те, що низькі середні значення в попередній таблиці були зумовлені поодинокими критичними помилками, а не загальною слабкістю конфігурації. Водночас *Offline Reflection* та *Text Reflection E-2 CP-1* зрівнялися на позначці 0.71, демонструючи обмеженість короткого вікна рефлексії, яке за стабільністю лише на 10% перевершує *w/o Memory* (0.61). Звуження довірчих інтервалів для *Reflection E-5* порівняно з текстовими конфігураціями демонструє, що структурована рефлексія не тільки підвищує якість відповідей, але й забезпечує більш передбачувану поведінку агента.

Як продемонстровано в таблиці 4.5, результати експериментів вказують, що за значенням середньої винагороди архітектури зі структурованою рефлексією, зокрема *Reflection E-5* (0.69) та *Reflection E-2* (0.68), демонструють найвищу ефективність серед наведених автоматизованих методів, наближаючись до рівня *Manual Instruction* (0.72). Порівняльний аналіз свідчить, що підхід зі структурованою пам'яттю у вигляді списків правил перевершує підхід із

Таблиця 4.5: Порівняння середньої винагороди для різних конфігурацій пам'яті агента. Наведено середнє значення, середнє квадратичне відхилення, нижня та верхня межі 95% довірчого інтервалу, мінімальне та максимальне значення у вибірці.

| Configuration | Mean | STD | CI low | CI high | Min | Max |
|---|---|---|---|---|---|---|
| w/o Memory | 0.44 | 0.02 | 0.42 | 0.45 | 0.41 | 0.46 |
| Baseline E-1 | 0.46 | 0.03 | 0.45 | 0.48 | 0.43 | 0.51 |
| Text Reflection E-2 CP-1 | 0.53 | 0.09 | 0.48 | 0.59 | 0.38 | 0.73 |
| Offline Reflection | 0.54 | 0.02 | 0.53 | 0.55 | 0.52 | 0.56 |
| Text Reflection E-5 CP-1 | 0.56 | 0.11 | 0.50 | 0.61 | 0.34 | 0.73 |
| Text Reflection E-2 | 0.59 | 0.06 | 0.56 | 0.63 | 0.54 | 0.71 |
| Text Reflection E-5 CP-3 | 0.62 | 0.11 | 0.56 | 0.67 | 0.43 | 0.72 |
| Text Reflection E-2 CP-3 | 0.63 | 0.05 | 0.61 | 0.67 | 0.56 | 0.72 |
| Text Reflection E-5 | 0.65 | 0.07 | 0.61 | 0.68 | 0.45 | 0.73 |
| Reflection E-2 | 0.68 | 0.08 | 0.64 | 0.72 | 0.46 | 0.76 |
| Reflection E-5 | 0.69 | 0.06 | 0.66 | 0.72 | 0.58 | 0.75 |
| Manual Instruction | 0.72 | 0.01 | 0.71 | 0.73 | 0.70 | 0.74 |

текстовим поданням, про що свідчить різниця 0.09 між *Reflection E-2* та *Text Reflection E-2*, що може вказувати на роль структурованої рефлексії як регуляризатора, який запобігає деградації стратегії. Встановлено, що збільшення обсягу історичних даних для виконання рефлексії (E-5 проти E-2) покращує показники очікуваної винагороди, тоді як впровадження контрастивних пар у конфігураціях *Text Reflection E-5 CP-3* та *Text Reflection E-2 CP-3* забезпечує вищу стабільність порівняно з варіантами *CP-1*, де високе середнє квадратичне відхилення (0.09) сигналізує про чутливість до конкретних помилок. Низька продуктивність *Offline Reflection* (0.54) відносно онлайн методів підкреслює критичну важливість динамічної адаптації до поточного розподілу даних у середовищі, а мінімальний розрив між *Reflection E-5* та *Manual Instruction* (0.03) підтверджує можливість використання LLM для автономної генерації ефективних стратегій.

Аналіз інтерквартильних статистик у таблиці 4.6 підтверджує статистичну значущість переваги структурованої рефлексії: конфігурації *Reflection E-2* (0.71) та *Reflection E-5* (0.70) наближаються до рівня *Manual Instruction* (0.72) із перекриттям довірчих інтервалів, що свідчить про спроможність системи до автономної генерації субоптимальних стратегій. Використання

Таблиця 4.6: Порівняння середньої винагороди для різних конфігурацій пам’яті агента. Наведено інтерквартильне середнє значення, нижня та верхня межі 95% довірчого інтервалу для інтеркваритльного середнього значення.

| Configuration | IQM | IQM CI low | IQM CI high |
|---|---|---|---|
| w/o Memory | 0.44 | 0.42 | 0.46 |
| Baseline E-1 | 0.46 | 0.44 | 0.49 |
| Text Reflection E-2 CP-1 | 0.53 | 0.48 | 0.58 |
| Offline Reflection | 0.53 | 0.52 | 0.56 |
| Text Reflection E-5 CP-1 | 0.57 | 0.50 | 0.62 |
| Text Reflection E-2 | 0.59 | 0.55 | 0.63 |
| Text Reflection E-2 CP-3 | 0.64 | 0.60 | 0.67 |
| Text Reflection E-5 CP-3 | 0.65 | 0.56 | 0.70 |
| Text Reflection E-5 | 0.66 | 0.62 | 0.69 |
| Reflection E-5 | 0.70 | 0.66 | 0.73 |
| Reflection E-2 | 0.71 | 0.65 | 0.73 |
| Manual Instruction | 0.72 | 0.71 | 0.73 |

IQM виявило, що при структуруванні пам’яті вища частота застосування рефлексії в конфігурації *Reflection E-2* забезпечує навіть дещо вищу стабільність, ніж *Reflection E-5*, нівелюючи потребу у великому вікні контексту за рахунок частішої корекції помилок, що може світчити про здатність структурованої рефлексії ефективно зберігати знання між ітераціями. Розрив між структурованими підходами та *Text Reflection E-5* (0.66) підкреслює, що формалізація знань є критичнішим фактором, ніж обсяг сирих даних у контексті. Крім того, перехід до *CP-3* у конфігураціях *Text Reflection E-5 CP-3* та *Text Reflection E-2 CP-3* суттєво підвищує нижню межу CI, підтверджуючи, що декілька контрастивних пар ефективно фільтрують шум і стабілізують процес навчання агента.

В таблиці 4.7 наведені статистики розраховані на наборах середніх значень кількості кроків, які виконав агент до завершення завдання в середовищі. Кращі результати відповідають меншій кількості кроків до завершення завдання. Найкращий результат (найменшу середню кількість кроків) отримано для конфігураці *Manual Instruction* (30.69), а конфігурація *Reflection E-5* (33.68) має найменшу середню кількість кроків серед автономних методів. Порівняння значень для *Reflection E-2* (34.19) та *Text Reflection E-2* (42.79) вказує на те, що при рівній частоті виконання рефлексії, структурована рефлексія

Таблиця 4.7: Порівняння середньої кількості кроків до завершення завдання для різних конфігурацій пам’яті агента. Наведено середнє значення, середнє квадратичне відхилення, нижня та верхня межі 95% довірчого інтервалу, мінімальне та максимальне значення у вибірці.

| Configuration | Mean | STD | CI low | CI high | Min | Max |
|---|---|---|---|---|---|---|
| Manual Instruction | 30.69 | 1.22 | 29.87 | 31.53 | 29.03 | 32.53 |
| Reflection E-5 | 33.68 | 5.36 | 31.10 | 36.49 | 27.44 | 43.82 |
| Reflection E-2 | 34.19 | 8.25 | 30.50 | 38.62 | 26.56 | 55.76 |
| Text Reflection E-5 | 37.87 | 6.89 | 34.78 | 41.38 | 30.09 | 56.42 |
| Text Reflection E-2 CP-3 | 38.97 | 4.98 | 36.05 | 41.89 | 31.12 | 46.10 |
| Text Reflection E-5 CP-3 | 40.27 | 10.41 | 35.15 | 45.79 | 30.84 | 58.39 |
| Text Reflection E-2 | 42.79 | 5.52 | 39.40 | 45.87 | 32.00 | 48.50 |
| Text Reflection E-5 CP-1 | 46.29 | 10.32 | 41.32 | 51.68 | 30.09 | 67.01 |
| Offline Reflection | 48.20 | 1.71 | 46.67 | 49.44 | 45.88 | 49.82 |
| Text Reflection E-2 CP-1 | 48.64 | 8.95 | 43.21 | 53.67 | 29.46 | 63.13 |
| Baseline E-1 | 55.48 | 2.74 | 53.59 | 57.36 | 51.46 | 58.40 |
| w/o Memory | 57.81 | 1.93 | 56.52 | 59.29 | 56.21 | 60.82 |

знаходить стратегію, яка виконує завдання за менше число кроків. Високе значення STD для *Reflection E-2* свідчить про нестабільність при рефлексії на невеликій кількості епізодів. У конфігурації *Text Reflection E-2 CP-3* (38.97) використано застосування рефлексії кожні 2 епізоди та 3 пари (вдалий, не вдалий) епізодів, що дозволило отримати найменше число кроків серед конфігурацій з контрастивним навчанням. Значна розбіжність між онлайн-методами та *Offline Reflection* (48.20) підкреслює необхідність ітеративного застосування рефлексії для адаптації до динаміки середовища. Найгірші результати отримано для конфігурацій *Baseline E-1* (55.48) та *w/o Memory* (57.81), які підтверджують нездатність агента ефективно вирішувати поставлене завдання без міжепізодичної пам’яті та здатності узагальнювати досвід.

У таблиці 4.8 наведено дані для порівняння міжквартильних статистик середньої кількості кроків до виконання завдання. Конфігурація *Reflection E-2* (32.11) тепер випереджає *Reflection E-5* (32.34), що свідчить про те, що вища дисперсія середнього значення в попередніх тестах була зумовлена поодинокими невдалими епізодами, тоді як типова траєкторія агента при частішій структурованій рефлексії є найбільш ефективною. Зменшення

Таблиця 4.8: Порівняння середньої кількості кроків до завершення завдання для різних конфігурацій пам'яті агента. Наведено інтерквартильне середнє значення, нижня та верхня межі 95% довірчого інтервалу для інтеркваритльного середнього значення.

| Configuration | IQM | IQM CI low | IQM CI high |
|---|---|---|---|
| Manual Instruction | 30.65 | 29.71 | 31.64 |
| Reflection E-2 | 32.11 | 29.19 | 36.86 |
| Reflection E-5 | 32.34 | 30.26 | 36.61 |
| Text Reflection E-5 | 36.68 | 33.88 | 40.35 |
| Text Reflection E-5 CP-3 | 37.68 | 32.46 | 46.33 |
| Text Reflection E-2 CP-3 | 38.92 | 35.41 | 42.73 |
| Text Reflection E-2 | 43.62 | 39.09 | 47.03 |
| Text Reflection E-5 CP-1 | 45.41 | 40.08 | 51.80 |
| Offline Reflection | 48.56 | 45.88 | 49.82 |
| Text Reflection E-2 CP-1 | 49.10 | 43.74 | 53.90 |
| Baseline E-1 | 55.70 | 53.10 | 57.91 |
| w/o Memory | 57.46 | 56.38 | 59.59 |

розбіжності між рефлексією та *Manual Instruction* (30.65) за метрикою інтерквартильного середнього вказує на успішне знаходження субоптимальної стратегії, тоді як суттєва перевага *Text Reflection E-5 CP-3* (37.68) над базовою *Text Reflection E-2* (43.62) може вказувате на те, що рідше виконання рефлексії на більшій кількості даних може призвести до знаходження кращої стратегії. Нарешті, збереження низької продуктивності в *Offline Reflection* (48.56) та *Baseline E-1* (55.70) навіть при використанні робастних оцінок IQM, вказує на перевагу методів онлайн-рефлексії.

Дані в таблиці 4.9 демонструють середнє значення кількості перепланувань за один епізод в середовищі Minigrid ColoredDoorKey. Дані показують, що конфігурація *Manual Instruction* (4.72) встановлює верхню межу ефективності агента, а конфігурації структурованої рефлексії *Reflection E-5* (6.21) та *Reflection E-2* (6.22) показують результати найближчі до агента з правилами сформульованими людиною. Аналіз підтверджує перевагу структурованої рефлексії над неструктурованою *Text Reflection E-5* (7.05) та критичну роль об'єму контрастивного контексту: перехід від однієї пари прикладів у *Text Reflection E-2 CP-1* (9.33) до трьох у *Text Reflection E-2 CP-3* (7.27), що супроводжується збільшенням кількості епізодів які обробляються під час

Таблиця 4.9: Порівняння середньої кількості планувань в епізоді для різних конфігурацій пам'яті агента. Наведено середнє значення, середнє квадратичне відхилення, нижня та верхня межі 95% довірчого інтервалу, мінімальне та максимальне значення у вибірці.

| Configuration | Mean | STD | CI low | CI high | Min | Max |
|---|---|---|---|---|---|---|
| Manual Instruction | 4.72 | 0.36 | 4.53 | 4.99 | 4.39 | 5.48 |
| Reflection E-5 | 6.21 | 2.23 | 5.22 | 7.47 | 4.32 | 11.76 |
| Reflection E-2 | 6.22 | 2.26 | 5.17 | 7.44 | 3.70 | 11.41 |
| Text Reflection E-5 | 7.05 | 1.62 | 6.31 | 7.88 | 5.38 | 10.82 |
| Text Reflection E-2 CP-3 | 7.27 | 1.45 | 6.37 | 8.07 | 4.59 | 8.95 |
| Text Reflection E-5 CP-3 | 7.41 | 2.22 | 6.34 | 8.57 | 4.77 | 11.56 |
| Offline Reflection | 8.42 | 1.09 | 7.39 | 9.28 | 6.94 | 9.56 |
| Text Reflection E-5 CP-1 | 9.00 | 3.49 | 7.41 | 10.92 | 5.22 | 17.61 |
| Text Reflection E-2 | 9.11 | 3.31 | 7.35 | 11.26 | 5.35 | 16.48 |
| Text Reflection E-2 CP-1 | 9.33 | 3.32 | 7.48 | 11.40 | 5.25 | 16.33 |
| Baseline E-1 | 9.60 | 1.57 | 8.63 | 10.77 | 8.07 | 12.62 |
| w/o Memory | 9.90 | 1.28 | 8.98 | 10.83 | 8.10 | 11.63 |

рефлексії, призводить до зменшення кількості планувань та зниження дисперсії результатів. Низька ефективність *w/o Memory* (9.90) та *Baseline E-1* (9.60) у порівнянні з *Offline Reflection* (8.42) і методами онлайн-рефлексії підкреслює, що накопичення між-епізодичного досвіду та ітеративна адаптація до помилок є ключовими факторами мінімізації кількості планувань.

Результати в таблиці 4.10 демонструють інтерквартильне середнє (IQM) та верхні та нижні межі 95% довірчого інтервалу середньої кількості планувань на епзіод. Застосуваня інтерквартильного середнього призводить до покращення результатів для конфігурацій *Reflection E-2* (5.58) та *Reflection E-5* (5.61), що вказує на нявність в даних невеликої кількості точок з порівняно високим значенням кількості перепланувань. Конфігурація з інструкціями сформованими людиною *Manual Instruction* (4.63), зберігає значний відрив від найкращих автоматизованих методів. У текстових конфігураціях критичним залишається обсяг контрастивного контексту, де *Text Reflection E-5 CP-3* (7.15) суттєво переважає *Text Reflection E-5 CP-1* (8.13). Порівняно високі значення кількості перепланувань для конфігурацій *Offline Reflection* (8.59) та *Baseline E-1* (9.30), які не застосовують ітеративну рефлексію, підтверджують необхідність використання динамічної між-епізодичної пам'яті

Таблиця 4.10: Порівняння середньої кількості планувань в епізоді для різних конфігурацій пам'яті агента. Наведено інтерквартильне середнє значення, нижня та верхня межі 95% довірчого інтервалу для інтеркваритльного середнього значення.

| Configuration | IQM | IQM CI low | IQM CI high |
|---|---|---|---|
| Manual Instruction | 4.63 | 4.51 | 5.00 |
| Reflection E-2 | 5.58 | 5.00 | 7.05 |
| Reflection E-5 | 5.61 | 5.00 | 7.00 |
| Text Reflection E-5 | 6.75 | 6.00 | 7.76 |
| Text Reflection E-5 CP-3 | 7.15 | 5.74 | 8.77 |
| Text Reflection E-2 CP-3 | 7.45 | 6.27 | 8.33 |
| Text Reflection E-5 CP-1 | 8.13 | 6.92 | 10.37 |
| Offline Reflection | 8.59 | 6.94 | 9.56 |
| Text Reflection E-2 | 8.68 | 6.77 | 11.03 |
| Text Reflection E-2 CP-1 | 9.12 | 6.94 | 11.22 |
| Baseline E-1 | 9.30 | 8.47 | 10.84 |
| w/o Memory | 9.93 | 8.85 | 11.08 |

для покращення результатів.

Результати в таблиці 4.10 демонструють інтерквартильне середнє (IQM) та інші пов'язані статистики середньої кількості планувань на епізод. Застосуваня інтерквартильного середнього призводить до покращення результатів для конфігурацій *Reflection E-2* (5.58) та *Reflection E-5* (5.61), що вказує на нявність в даних невеликої кількості точок з порівняно високим значенням кількості перепланувань. Конфігурація з інструкціями сформованими людиною *Manual Instruction* (4.63), зберігає значний відрив від найкращих автоматизованих методів. У текстових конфігураціях критичним залишається обсяг контрастивного контексту, де *Text Reflection E-5 CP-3* (7.15) суттєво переважає *Text Reflection E-5 CP-1* (8.13). Порівняно високі значення кількості перепланувань для конфігурацій *Offline Reflection* (8.59) та *Baseline E-1* (9.30), які не застосовують ітеративну рефлексію, підтверджують необхідність використання динамічної між-епізодичної пам'яті для покращення результатів.

На рисунку 4.17 зображено діаграму розмаху для середньої успішності вирішення завдання на тестовому наборі зі 100 епізодів, однакових для всіх запусків та конфігурацій агента. Центральні лінії позначають медіану, пунктирна лінія позначає середнє значення, а точки відображають результати

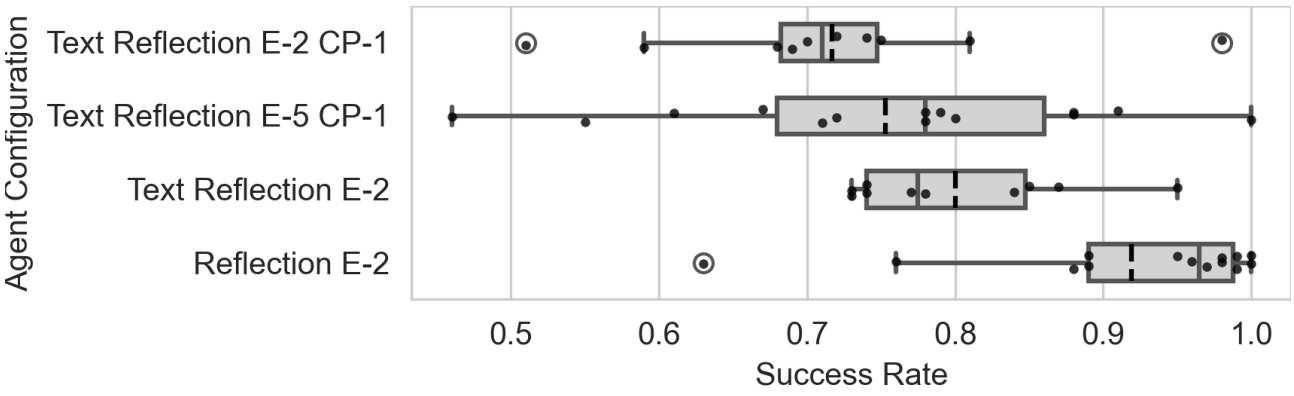



Рисунок 4.17: Залежність квартильних значень середньої успішності вирішення завдання на тестовому наборі середовищ від конфігурації агента. Показані лише конфігурації Text Reflection E-2 CP-1, Text Relfection E-5 CP-1, Text Reflection E-2, Reflection E-2.

окремих запусків. На діаграмі зображено конфігурації агентів, які на кожній ітерації рефлексії отримували на вхід історію з двох епізодів. З діаграми можна побачити, що конфігурація *Reflection E-2* показує найкращі результати, хоча і включає декілька викидів з низькими значеннями. Конфігурація *Text Reflection E-5 CP-1* призвела до значної дисперсії результатів, що може бути спричиненим низькою кількістю ітерацій рефлексії та обмеженою кількістю даних для рефлексії.

На рисунку 4.18 зображено діаграму розмаху для середньої успішності вирішення завдання на тестовому наборі зі 100 епізодів, однакових для всіх запусків та конфігурацій агента. Центральні лінії позначають медіану, пунктирна лінія позначає середнє значення, а точки відображають результати окремих запусків. На діаграмі зображено усі досліджені конфігурації агента. З діаграми можна побачити, що дві конфігурації з рефлексивною пам'яттю на основі набору правил та операцій — *Reflection E-2* та *Reflection E-5* показують значення медіани успішності вирішення завдання близьке до рівня *Manual Instruction* — агента з інструкціями, які створено людиною. Але *Reflection E-2* та *Reflection E-5* мають значну дисперсію результатів. Конфігурації в яких пам'ять зберігалася у вигляді текстових рядків — *Text Reflection E-2* та *Text Reflection E-5* показують гірші результати, що може бути пояснено гіршою здатністю до збереження знань в процесі навчання. Базові конфігурації *w/o Memory*, *Baseline E-1* показують очікувано низьку успішність, оскільки ці агенти не невчалися.

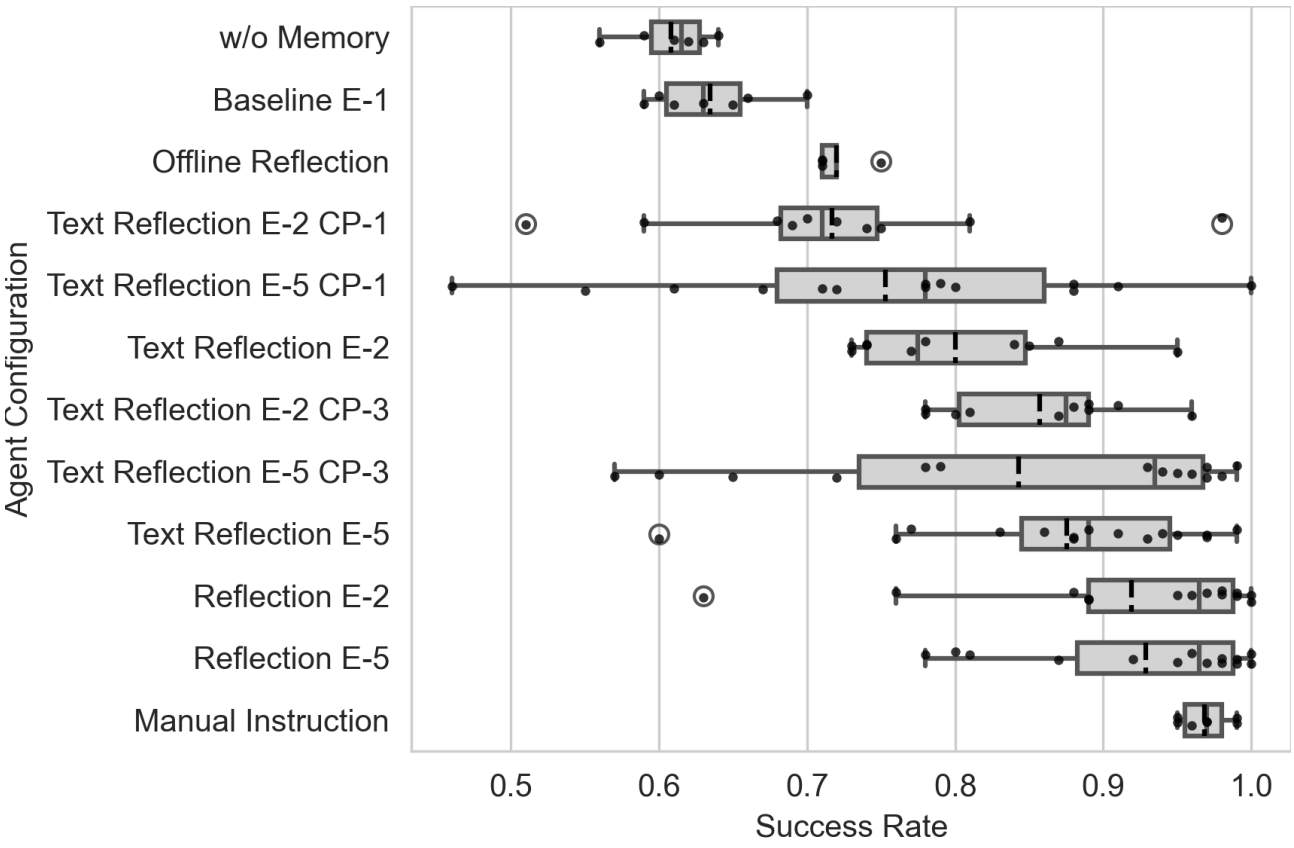



Рисунок 4.18: Залежність квартильних значень середньої успішності вирішення завдання на тестовому наборі середовищ від конфігурації агента.

## Висновки до розділу 4

У четвертому розділі було розроблено та досліджено архітектуру LLM-агента з дворівневою моделлю пам’яті, в якій епізодичні дані трансформуються у вербальні узагальнення для подальшого використання в стратегії агента. Перша складова пам’яті представлена епізодичною пам’яттю, яка накопичує повну історію взаємодії агента з середовищем. Як другу складову запропоновано транзакційну модель структурованої рефлексивної пам’яті, яка виконує ітеративне узагальнення епізодичного досвіду в компактний набір поведінкових правил за допомогою LLM.

Виконано модифікацію простору процедурних опцій, у якому агент здійснює планування, шляхом інтеграції вербального зворотного зв’язку про статус завершення дій та виявлені помилки. Запропоновано виконувати навчання агента у віртуальних середовищах (на прикладі середовища Minigrid ColoredDoorKey) методом навчання в контексті (In-Context Learning). Динамічне оновлення бази знань агента реалізовано шляхом автономного вибору з використанням LLM операцій створення, оновлення, підкріплення та

видалення правил, що забезпечило здатність системи автономно коригувати помилки планування між епізодами без обчислювально складного донавчання параметрів LLM.

Результати емпіричного дослідження показали, що використання в LLM-агенті методу транзакційної структурованої рефлексії суттєво покращує результативність агента в задачі просторової навігації та планування. Зокрема, застосування структурованої пам’яті забезпечило досягнення інтерквартильного середнього значення (IQM) успішності на рівні 95%. Це відповідає зростанню на 32 відсоткові пункти порівняно з базовим підходом, який використовує лише неузагальнену епізодичну пам’ять (63%) та на 34 пункти порівняно з агентом без пам’яті (61%). Отриманий результат свідчить про здатність запропонованої архітектури до автономного навчання, оскільки її ефективність лише на два відсоткові пункти поступається показникам агента, який діє за інструкціями людини-експерта (97%).

Аналіз статистичних даних демонструє успішне покращення стратегії поведінки агента. Застосування структурованої рефлексії призвело до зменшення середньої кількості кроків, необхідних для виконання завдання, з 55.48 (базовий підхід) до 33.68, а також забезпечило збільшення середньої сумарної винагороди за епізод на 50% (з 0.46 до 0.69). Експериментально продемонстровано перевагу структурованої рефлексії порівняно з текстовою рефлексією. Архітектура зі структурованою рефлексією досягає показника успішності 95% проти 79% у агента з неструктурованою текстовою рефлексією (відносне зростання на 20%).

Аналіз статистичного розподілу результатів виявив звуження довірчих інтервалів та зниження дисперсії середньої успішності виконання завдання при застосуванні структурованої пам’яті. Зменшення дисперсії вказує на те, що агент уникає екстремальних невдач та демонструє більш стабільну поведінку незалежно від початкових умов. Отже, структурована рефлексія дозволяє ефективно відфільтрувати шумову інформацію з окремих невдалих епізодів. У результаті процес навчання стабілізується, що запобігає деградації накопичених знань агента під час довготривалої взаємодії з середовищем.

Основні наукові результати розділу опубліковані в роботі [110].

# ВИСНОВКИ

У дисертаційній роботі вирішено актуальну науково-прикладну задачу вдосконалення методів та моделей керування ієрархічними інтелектуальними агентами, які використовують великі мовні моделі (LLM) як модуль стратегічного планування. Мету дослідження досягнуто в повному обсязі, а поставлені теоретичні та практичні завдання виконано, що підтверджується основними результатами, отриманими в дисертаційній роботі:

1. **Розроблено вперше метод адаптивної рефлексивної пам'яті** для ієрархічних агентів, який реалізує динамічну транзакційну модель автономного оновлення бази знань агента через оператори модифікації та механізм важливості правил. Застосування цього методу забезпечило зростання показника інтерквартильного середнього значення (IQM) успішності вирішення завдання в середовищі до 95%, що на 32 відсоткові пункти перевищило результати агентів з епізодичною пам'яттю та на 50% збільшило середню сумарну винагороду (з 0.46 до 0.69).

2. **Запропоновано вперше архітектуру ієрархічного інтелектуального агента**, у якій LLM функціонує як стратегічний планувальник у просторі параметризованих абстрактних дій, інтегрований з модулем запропонованої структурованої рефлексивної пам'яті. Доведено, що використання структурованої рефлексивної пам'яті порівняно з неструктурованою текстовою формою пам'яті підвищує успішність на 20% (з 79% до 95%) та скорочує середню кількість кроків за епізод на 39% (з 55.48 до 33.68).

3. **Удосконалено метод планування** в ієрархічних агентах шляхом впровадження декодування з обмеженням граматики (GCD), що гарантує 100% синтаксичну валідність згенерованих JSON-планів відповідно до заданої схеми. Це дозволило усунути понад 50% критичних помилок генерації для моделей малого розміру (від 1.5 млрд. до 4 млрд. параметрів) та скоротити час планування від 6 до 25 разів для окремих моделей за рахунок обмеження надлишкових ланцюжків міркувань.

4. **Набули подальшого розвитку методи застосування великих мовних**

**моделей у задачах планування та навігації** через поєднання ієрархічних структур керування з генерацією багатокрокових планів у просторі параметризованих абстрактних дій. Встановлено, що багатокрокове планування на 10.5% ефективніше за покрокове, а інтеграція плану за замовчуванням у стратегію агента забезпечує додатковий приріст середньої винагороди на 1.63%.

5. **Набули подальшого розвитку методи створення контексту для LLM** як інструменти керування поведінкою агента, зокрема через використання промпт-інжинірингу та навчання в контексті (In-Context Learning). Доведено, що усунення граматичних помилок в інструкціях та використання few-shot прикладів підвищує середню винагороду на 7.34%, а оптимізація комунікаційної стратегії дозволяє знизити кількість звернень до LLM до 3 викликів на епізод.

Практичне значення отриманих результатів полягає у розроблених рішеннях — зокрема методі планування високорівневих дій агента з декодуванням з обмеженням граматики (GCD) та архітектурі динамічної транзакційної рефлексивної пам’яті — які можуть бути впроваджені у системи автономного управління. Створені алгоритми та програмні модулі можуть бути адаптовані та використані для керування автономними роботизованими системами, програмними інформаційними агентами та системами автоматизації багатокрокових бізнес-процесів, при функціонуванні агентів в умовах часткової невизначеності та наявності вимоги гарантованої коректності згенерованих планів та контролю за тривалістю планування.

Перспективи подальших досліджень полягають у розширенні функціональних можливостей розроблених архітектур шляхом їх перенесення та адаптації до роботи у складних тривимірних симуляційних середовищах з високою комбінаторною складністю та розширеним горизонтом планування. Доцільним напрямком є дослідження масштабованості підходу при використанні LLM більшого розміру, а також застосування мовних моделей, адаптованих до конкретних типів завдань шляхом цільового донавчання параметрів. Крім того, перспективним напрямком розвитку є дослідження парадигм мультиагентної взаємодії, де кілька автономних ієрархічних агентів, об’єднаних розподіленою рефлексивною пам’яттю, будуть здатні

координувати власні стратегії для колективного розв’язання комплексних задач.

# ПОДЯКИ

Насамперед, щиро дякую моїм науковим керівникам: кандидату технічних наук Володимиру Михайловичу Струкову — за цінні поради та супровід під час виконання досліджень та підготовки дисертації; кандидатці технічних наук, доцентці Подоляці Оксані Олександрівні — за віру та підтримку у найбільш непередбачувані часи; доктору фізико-математичних наук, професору Кукліну Володимиру Михайловичу — за надану можливість розпочати цей науковий шлях.

Окрему вдячність висловлюю моєму вчителю та провіднику в галузь штучного інтелекту Гущину Івану Валерійовичу за допомогу у формулюванні наукової задачі.

Окрема подяка О.О. Борисенку та М.І. Братченку за запрошення до наукової співпрації та можливість розширити наукові горизонти Європи. Дякую представникам EURIZON FELLOWSHIP PROGRAMME та європейським платникам податків за підтримку української науки в період, коли підтримка була найбільш важливою.

Символічно і цілком заслужено хочу подякувати об’єктам мого наукового інтересу — великим мовним моделям (зокрема Gemini, Claude та ChatGPT). Вони стали джерелом натхнення та орієнтиром у моїй роботі.

Дякую співробітникам навчально-наукового інституту комп’ютерних наук та штучного інтелекту, колективу кафедри математичного моделювання та аналізу даних, працівникам відділу аспірантури та докторантури Харківського національного університету імені В.Н. Каразіна за забезпечення навчального та наукового процесу.

Найбільша та найтепліша подяка моїй сім’ї. Ваша щоденна присутність і підтримка зробили можливим те, що здавалося неможливим.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Huang W., Abbeel P., Pathak D., et al. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *Proceedings of the 39th International Conference on Machine Learning*. PMLR. 2022, P. 9118–9147.

[2] Sutton R., Precup D., Singh S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*. 1999. Vol. 112. P. 181–211.

[3] Barto A., Mahadevan S. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*. 2003. Vol. 13. P. 41–77.

[4] Konidaris G., Kaelbling L., Lozano-Perez T. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*. 2018. Vol. 61. P. 215–289.

[5] Hu B., Zhao C., Zhang P., et al. Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach. *Reinforcement Learning Journal*. 2024. Vol. 1.

[6] Schulhoff S., Ilie M., Balepur N., et al. *The Prompt Report: A Systematic Survey of Prompt Engineering Techniques*. 2025. arXiv: 2406.06608. URL: https://arxiv.org/abs/2406.06608.

[7] Schepanowski C., Ling C. *On the Limits of Innate Planning in Large Language Models*. 2025. arXiv: 2511.21591. URL: https://arxiv.org/abs/2511.21591.

[8] Bunel R., Hausknecht M., Devlin J., et al. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=H1Xw62kRZ.

[9] Brown T., Mann B., Ryder N., et al. Language models are few-shot learners. *Advances in neural information processing systems*. 2020. Vol. 33. P. 1877–1901.

[10] Dong Q., Li L., Dai D., et al. A Survey on In-context Learning. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Y. Al-Onaizan, M. Bansal, Y. Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, P. 1107–1128. DOI: 10.18653/v1/2024.emnlp-main.64. URL: https://aclanthology.org/2024.emnlp-main.64/.

[11] Park J., O'Brien J., Cai C., et al. Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th annual acm symposium on user interface software and technology*. 2023, P. 1–22.

[12] Shinn N., Cassano F., Gopinath A., et al. Reflexion: language agents with verbal reinforcement learning. *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=vAElhFcKW6.

[13] Zhang W., Tang K., Wu H., et al. Agent-Pro: Learning to Evolve via Policy-Level Reflection and Optimization. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L. Ku, A. Martins, V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, P. 5348–5375. DOI: 10.18653/v1/2024.acl-long.292. URL: https://aclanthology.org/2024.acl-long.292/.

[14] Fikes R., Nilsson N. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*. 1971. Vol. 2. P. 189–208.

[15] Ghallab M., Nau D., Traverso P. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[16] McDermott D., Ghallab M., Howe A., et al. *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR98003/DCS TR1165. Yale Center for Computational Vision and Control, 1998.

[17] Sutton R., Barto A., et al. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge, 1998.

[18] Kaelbling L., Littman M., Cassandra A. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*. 1998. Vol. 101. P. 99–134.

[19] Vaswani A., Shazeer N., Parmar N., et al. Attention is all you need. *Advances in neural information processing systems*. 2017. Vol. 30.

[20] Wang L., Ma C., Feng X., et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*. 2024. Vol. 18. P. 186345.

[21] Kojima T., Gu S., Reid M., et al. Large language models are zero-shot reasoners. *Advances in neural information processing systems*. 2022. Vol. 35. P. 22199–22213.

[22] Wei J., Wang X., Schuurmans D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*. 2022. Vol. 35. P. 24824–24837.

[23] Zhou D., Schärli N., Hou L., et al. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WZH7099tgfM.

[24] Khot T., Trivedi H., Finlayson M., et al. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=_nGgzQjzaRy.

[25] Madaan A., Tandon N., Gupta P., et al. Self-Refine: Iterative Refinement with Self-Feedback. *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, et al. Vol. 36. Curran Associates, Inc., 2023, P. 46534–46594. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

[26] Yao S., Rao R., Hausknecht M., et al. Keep CALM and Explore: Language Models for Action Generation in Text-based Games. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by B. Webber, T. Cohn, Y. He, et al. Online: Association for Computational Linguistics, Nov. 2020, P. 8736–8754. DOI: 10.18653/v1/2020.emnlp-main.704. URL: https://aclanthology.org/2020.emnlp-main.704/.

[27] Zhang F., Li J., Li Y., et al. Improving Sample Efficiency of Reinforcement Learning with Background Knowledge from Large Language Models. *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, et al. Vol. 37. Curran Associates, Inc., 2024, P. 38740–38762.

[28] Di Palo N., Hasenclever L., Humplik J., et al. Diffusion Augmented Agents: A Framework for Efficient Exploration and Transfer Learning. *Proceedings of the 3rd Conference on Lifelong Learning Agents (CoLLAs)*. 2024.

[29] Andreas J., Klein D., Levine S. Learning with Latent Language. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018, P. 2166–2179. DOI: 10.18653/v1/N18-1197.

[30] Paischer F., Adler T., Hofmarcher M., et al. Semantic HELM: A Human-Readable Memory for Reinforcement Learning. *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=ebMPmx5mr7.

[31] Belta C., Bicchi A., Egerstedt M., et al. Symbolic Planning and Control of Robot Motion. *IEEE Robotics & Automation Magazine*. 2007. Vol. 14. P. 61–70.

[32] Hu S., Clune J. Thought Cloning: Learning to Think while Acting by Imitating Human Thinking. *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://github.com/ShengranHu/Thought-Cloning.

[33] Lyu D., Yang F., Liu B., et al. SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, P. 2970–2977.

[34] Kimura D., Ono M., Chaudhury S., et al. Neuro-Symbolic Reinforcement Learning with First-Order Logic. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by M. Moens, X. Huang, L. Specia, et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, P. 3505–3511. DOI: 10.18653/v1/2021.emnlp-main.283. URL: https://aclanthology.org/2021.emnlp-main.283/.

[35] Fu J., Kumar A., Nachum O., et al. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*. 2021. arXiv: 2004.07219. URL: https://arxiv.org/abs/2004.07219.

[36] Levine S., Kumar A., Tucker G., et al. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *Foundations and Trends® in Machine Learning*. 2020. Vol. 14. P. 1–191.

[37] Shah D., Toshev A., Levine S., et al. Value Function Spaces: Skill-Centric State Abstractions for Long-Horizon Reasoning. *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=vgqS1vkkCbE.

[38] Shridhar M., Yuan X., Côté M., et al. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. *International Conference on Learning Representations (ICLR)*. 2021.

[39] Ahn M., Brohan A., Brown N., et al. Do As I Can, Not As I Say: Grounding Language Models in Robotic Affordances. *Conference on Robot Learning (CoRL)*. 2022.

[40] Khetarpal G., Ahmed Z., Gheisarnejad H., et al. What can I do here? A Theory of Affordances in Reinforcement Learning. *International Conference on Machine Learning (ICML)*. PMLR. 2020, P. 5243–5253.

[41] Costales R., Iqbal S., Sha F. Possibility Before Utility: Learning and Using Hierarchical Affordances. *International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=7b4zxUnrO2N.

[42] Wang Z., Cai S., Chen G., et al. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[43] Xu D., Nair S., Zhu Y., et al. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, P. 3795–3802. DOI: 10.1109/ICRA.2018.8460689.

[44] Jiang Y., Gu S., Murphy K., et al. Language as an Abstraction for Hierarchical Deep Reinforcement Learning. *Advances in Neural Information Processing Systems (NeurIPS 2019)*. 2019.

[45] Andrychowicz M., Wolski F., Ray A., et al. Hindsight Experience Replay. *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Luxburg, S. Bengio, et al. Vol. 30. Curran Associates, Inc., 2017. URL: https : / / proceedings . neurips . cc / paper _ files / paper / 2017 / file / 453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

[46] Colas C., Karch T., Lair N., et al. Language as a Cognitive Tool to Imagine Goals in Curiosity-Driven Exploration. *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. Dec. 2020.

[47] Akakzia A., Colas C., Oudeyer P., et al. Grounding Language to Autonomously-Acquired Skills via Goal Generation. *International Conference on Learning Representations*. 2021. URL: https : / / api . semanticscholar.org/CorpusID:227183037.

[48] Nair A., Pong V., Dalal M., et al. Visual Reinforcement Learning with Imagined Goals. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*. Montréal, Canada, 2018.

[49] Nair A., Pong V., Bahl S., et al. Contextual Imagined Goals for Self-Supervised Robotic Learning. *Conference on Robot Learning (CoRL)*. Osaka, Japan, 2019.

[50] Mirchandani S., Karamcheti S., Sadigh D. ELLA: exploration through learned language abstraction. *Proceedings of the 35th International Conference on Neural Information Processing Systems*. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., 2021. ISBN: 9781713845393.

[51] Jansen P. Visually-Grounded Planning without Vision: Language Models Infer Detailed Plans from High-level Instructions. *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by T. Cohn, Y. He, Y. Liu. Online: Association for Computational Linguistics, Nov. 2020, P. 4412–4417. DOI: 10.18653/v1/2020.findings-emnlp.395. URL: https://aclanthology.org/2020.findings-emnlp.395/.

[52] Sharma P., Torralba A., Andreas J. Skill Induction and Planning with Latent Language. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by S. Muresan, P. Nakov, A. Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, P. 1713–1726. DOI: 10.18653/v1/2022.acl-long.120. URL: https://aclanthology.org/2022.acl-long.120/.

[53] Goldberg Y. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[54] Anderson P., Wu Q., Teney D., et al. Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[55] Das A., Gkioxari G., Lee S., et al. Neural Modular Control for Embodied Question Answering. *Proceedings of the 2nd Conference on Robot Learning (CoRL)*. Zürich, Switzerland, 2018.

[56] Xiao T., Chan H., Sermanet P., et al. Robotic Skill Acquisition via Instruction Augmentation with Vision-Language Models. *Robotics: Science and Systems*. 2023. URL: https://doi.org/10.15607/RSS.2023.XIX.029.

[57] Bohdan N., Tvoroshenko I., Gorokhovatskyi V., et al. Development of a Hybrid Method to Enhance Context Memory for a Chatbot Application Based on Large Language Models. *International Journal of Academic Information Systems Research (IJAISR)*. 2025. Vol. 9. P. 7–18.

[58] Slyusar V. “Distributed Multi-agent Systems Based on the Mixture of Experts Architecture in the Context of 6G Wireless Technologies”. *Applied Innovations in Information and Communication Technology*. Apr. 2025, P. 81–110. DOI: 10.1007/978-3-031-89296-7_6.

[59] Murphy K. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press, 2012. ISBN: 9780262018029. URL: https : / / books . google . com . ua / books ? id = NZP6AQAAQBAJ.

[60] Russell S., Norvig P. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2022. ISBN: 978-1-292-40117-1.

[61] Svoboda I. Automating Cybersecurity Decision-Making with AI and the Analytic Hierarchy Process. *Theoretical and Applied Cybersecurity*. 2025. Vol. 7. P. 91–97.

[62] Kozlovska M., Piskozub A. Hybridizing Large Language Models and Markov Processes: A New Paradigm for Autonomous Penetration Testing. *Advances in Cyber-Physical Systems*. 2025. Vol. 10. P. 146–150.

[63] Yao S., Zhao J., Yu D., et al. ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR)*. 2023. URL: https://openreview.net/forum?id=WEtQo0w5DBA.

[64] Wang L., Xu W., Lan Y., et al. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by A. Rogers, J. Boyd-Graber, N. Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, P. 2609–2634. DOI: 10.18653/v1/2023.acl-long.147. URL: https://aclanthology.org/2023.acl-long.147/.

[65] Lee S., Kim G. Recursion of Thought: A Divide-and-Conquer Approach to Multi-Context Reasoning with Language Models. *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by A. Rogers, J. Boyd-Graber, N. Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, P. 623–658. DOI: 10.18653/v1/2023.findings-acl.40. URL: https://aclanthology.org/2023.findings-acl.40/.

[66] Dua D., Gupta S., Singh S., et al. Successive Prompting for Decomposing Complex Questions. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Y. Goldberg, Z. Kozareva, Y. Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, P. 1251–1265. DOI: 10.18653/v1/2022.emnlp-main.81. URL: https://aclanthology.org/2022.emnlp-main.81/.

[67] Karpas E., Abend O., Belinkov Y., et al. *MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning*. 2022. arXiv: 2205.00445. URL: https://arxiv.org/abs/2205.00445.

[68] Zeng A., Attarian M., ichter brian, et al. Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=G2Q2Mh3avow.

[69] Sel B., Al-Tawaha A., Khattar V., et al. Algorithm of thoughts: enhancing exploration of ideas in large language models. *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria, 2024.

[70] Yao S., Yu D., Zhao J., et al. Tree of thoughts: deliberate problem solving with large language models. *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[71] Valmeekam K., Olmo A., Sreedharan S., et al. Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change). *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022. URL: https://openreview.net/forum?id=wUU-7XTL5XO.

[72] Wu Z., Zeng Q., Zhang Z., et al. Large Language Models Can Self-Correct with Key Condition Verification. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Y. Al-Onaizan, M. Bansal, Y. Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, P. 12846–12867. DOI: 10.18653/v1/2024.emnlp-main.714. URL: https://aclanthology.org/2024.emnlp-main.714/.

[73] Haller P., Ibrahim M., Kirichenko P., et al. *LLM Knowledge is Brittle: Truthfulness Representations Rely on Superficial Resemblance*. 2025. arXiv: 2510.11905. URL: https://arxiv.org/abs/2510.11905.

[74] Zhao A., Huang D., Xu Q., et al. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, P. 19632–19642.

[75] Zhong W., Guo L., Gao Q., et al. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, P. 19724–19731.

[76] Packer C., Wooders S., Lin K., et al. *MemGPT: Towards LLMs as Operating Systems*. 2024. arXiv: 2310.08560. URL: https://arxiv.org/abs/2310.08560.

[77] Xu W., Liang Z., Mei K., et al. A-Mem: Agentic Memory for LLM Agents. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=FiM0M8gcct.

[78] Acharya K., Raza W., Dourado C., et al. Neurosymbolic Reinforcement Learning and Planning: A Survey. *IEEE Transactions on Artificial Intelligence*. 2024. Vol. 5. P. 1939–1953.

[79] Xiong W., Hoang T., Wang W. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Ed. by M. Palmer, R. Hwa, S. Riedel. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, P. 564–573. DOI: 10.18653/v1/D17-1060. URL: https://aclanthology.org/D17-1060/.

[80] Das R., Dhuliawala S., Zaheer M., et al. *Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning*. 2018. arXiv: 1711.05851. URL: https://arxiv.org/abs/1711.05851.

[81] Silver T., Chitnis R., Kumar N., et al. Predicate Invention for Bilevel Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2023. Vol. 37. P. 12120–12129.

[82] Dasgupta I., Kaeser-Chen C., Marino K., et al. Collaborating with language models for embodied reasoning. *Second Workshop on Language and Reinforcement Learning*. 2022. URL: https://openreview.net/forum?id=YoS-abmWjJc.

[83] Huang W., Xia F., Xiao T., et al. Inner Monologue: Embodied Reasoning through Planning with Language Models. *Proceedings of The 6th Conference on Robot Learning*. Ed. by K. Liu, D. Kulic, J. Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 14–18 Dec 2023, P. 1769–1782. URL: https://proceedings.mlr.press/v205/huang23c.html.

[84] Schulman J., Wolski F., Dhariwal P., et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347. URL: https://arxiv.org/abs/1707.06347.

[85] Xiao T., Zhu J. *Foundations of Large Language Models*. 2025. arXiv: 2501.09223. URL: https://arxiv.org/abs/2501.09223.

[86] ZJLAB-AMMI. *LLM4RL: Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach Repository*. https://github.com/ZJLAB-AMMI/LLM4RL. Accessed: 2026-01-14. 2024.

[87] Chevalier-Boisvert M., Dai B., Towers M., et al. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2023. URL: https://openreview.net/forum?id=PFfmfspm28.

[88] Towers M., Kwiatkowski A., Balis J., et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2025. URL: https://openreview.net/forum?id=qPMLvJxtPK.

[89] Brockman G., Cheung V., Pettersson L., et al. *OpenAI Gym*. 2016. arXiv: 1606.01540. URL: https://arxiv.org/abs/1606.01540.

[90] Omelchenko I., Strukov V. On the impact of prompts on agent performance in a virtual environment. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 65. P. 83–91.

[91] Murphy K. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[92] Beurer-Kellner L., Fischer M., Vechev M. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. *Proceedings of the 41st International Conference on Machine Learning*. Ed. by R. Salakhutdinov, Z. Kolter, K. Heller, et al. Vol. 235. Proceedings of Machine Learning Research. PMLR, 21–27 Jul 2024, P. 3658–3673. URL: https://proceedings.mlr.press/v235/beurer-kellner24a.html.

[93] Willard B., Louf R. *Efficient Guided Generation for Large Language Models*. 2023. arXiv: 2307.09702. URL: https://arxiv.org/abs/2307.09702.

[94] Sparrenberg L., Deußer T., Berger A., et al. Small and Fast LLMs on Commodity Hardware: Post-Training Quantization in llama. cpp. *2025 IEEE 12th International Conference on Data Science and Advanced Analytics (DSAA)*. 2025, P. 1–10. DOI: 10.1109/DSAA65442.2025.11247985.

[95] Shi T., Ding Y. *Systematic Characterization of LLM Quantization: A Performance, Energy, and Quality Perspective*. 2025. arXiv: 2508.16712. URL: https://arxiv.org/abs/2508.16712.

[96] Bray T. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. RFC. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259.

[97] Wright A., Andrews H., Hutton B., et al. *JSON Schema: A Media Type for Describing JSON Documents*. Internet-Draft draft-bhutton-json-schema-01. Work in Progress. Internet Engineering Task Force, Dec. 2020. URL: https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-01.

[98] Bray T., Paoli J., Sperberg-McQueen C., et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. Available at https://www.w3.org/TR/2008/REC-xml-20081126. World Wide Web Consortium (W3C), Nov. 2008. URL: https://www.w3.org/TR/xml.

[99] Google LLC. *Protocol Buffers: Google's Data Interchange Format*. 2024. URL: https://protobuf.dev (visited on 01/19/2026).

[100] Yang A., Li A., Yang B., et al. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388. URL: https://arxiv.org/abs/2505.09388.

[101] Team G., Kamath A., Ferret J., et al. *Gemma 3 Technical Report*. 2025. arXiv: 2503.19786. URL: https://arxiv.org/abs/2503.19786.

[102] Grattafiori A., Dubey A., Jauhri A., et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783. URL: https://arxiv.org/abs/2407.21783.

[103] Guo D., Yang D., Zhang H., et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*. 2025. Vol. 645. P. 633–638.

[104] Omelchenko I., Strukov V. Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 67. P. 101–112.

[105] 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. 1–47.

[106] 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. URL: https://arxiv.org/abs/2305.17144.

[107] Robbins H., Monro S. A stochastic approximation method. *The annals of mathematical statistics*. 1951. P. 400–407.

[108] Kiefer J., Wolfowitz J. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*. 1952. P. 462–466.

[109] Kushner H., Yin G. *Stochastic approximation and recursive algorithms and applications*. Springer, 2003.

[110] Omelchenko I., Strukov V. Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 68. P. 62–69.

# ДОДАТОК А

## СПИСОК НАУКОВИХ ПУБЛІКАЦІЙ ЗДОБУВАЧА

*Публікації, в яких опубліковано основні наукові результати дисертації*

*Статті у наукових виданнях, що входять до міжнародних наукометричних баз:*

1. Borysenko O., Bratchenko M., Lukin I., Luhanko M., **Omelchenko I.**, Sotnikov A., Lomi A. Application of Langevin dynamics to advance the Quantum Natural Gradient optimization algorithm. *Physica A: Statistical Mechanics and its Applications*. 2026. Vol. 682. P. 131158. DOI: https://doi.org/10.1016/j.physa.2025.131158 (Scopus, Web of Science, Q2).

*Статті у наукових фахових виданнях України:*

2. **Omelchenko I.**, Strukov V. On the impact of prompts on agent performance in a virtual environment. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 65. P. 83-91. DOI: https://doi.org/10.26565/2304-6201-2025-65-07
   *Особистий внесок здобувача:* проведення аналізу існуючих моделей агентів на основі LLM та методів керування такими агентами, розробка програмного середовища для тестування агента в середовищі Minigrid, розробка мовних інструкцій для керування агентом, виконання обчислювальних експериментів, аналіз та порівняння отриманих результатів, підготовка тексту роботи.
   *Особистий внесок Володимира Струкова:* перевірка отриманих результатів, перевірка тексту роботи.

3. **Omelchenko I.**, Strukov V. Impact of decoding methods in LLMs on the correctness of agent action planning in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 67. P. 101-112. DOI: https://doi.org/10.26565/2304-6201-2025-67-10
   *Особистий внесок здобувача:* проведення аналізу методів декодування з обмеженням граматики послідовностей згенерованих LLM, розробка програмного середовища з інтеграцією методу декодування, підготовка

датасету на основі середовища Minigrid для тестування модуля планування агента, виконання обчислювальних експериментів, аналіз та порівняння отриманих результатів, підготовка тексту роботи.
*Особистий внесок Володимира Струкова:* обговорення ідей та методів дослідження, перевірка отриманих результатів, перевірка тексту роботи.

4. **Omelchenko I.**, Strukov V. Reflective memory architecture for adaptive planning in hierarchical LLM agents in virtual environments. *Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»*. 2025. Vol. 68. P. 62-69. DOI: https://doi.org/10.26565/2304-6201-2025-68-06
*Особистий внесок здобувача:* проведення аналізу існуючих моделей пам'яті та методів управління пам'яттю автономних LLM агентів, теоретична розробка моделі рефлексивної пам'яті, розробка програмного середовища для дослідження LLM-агента з рефлексивною пам'яттю, проведення обчислювальних експериментів в середовищі Minigrid, аналіз отриманих результатів, підготовка тексту роботи. *Особистий внесок Володимира Струкова:* перевірка отриманих результатів, перевірка тексту роботи.

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

1. **Омельченко І.**, Гущин І., Струков В. Аналіз впливу текстових підказок на ефективність навчання агента. *Матеріали X Міжнародної науково-практичної конференції «Комп'ютерне моделювання у наукоємних технологіях» (КМНТ-2024)*. ХНУ імені В. Н. Каразіна. Україна, 2024, С. 160-163.

2. **Омельченко І.**, Струков В. Вплив методів декодування на коректність планування дій LLM-агентів у віртуальних середовищах. *Матеріали Міжнародної науково-технічної конференції «Інтелектуальні технології у міждисциплінарних дослідженнях» (ІТМД-2025)*. ХНУ імені В. Н. Каразіна. Україна, 2025, С. 179-182.

3. **Омельченко І.** Рефлексивна пам'ять в задачі адаптивного планування ієрархічних LLM-агентів у віртуальних середовищах. *Матеріали Міжнародної науково-практичної конференції молодих вчених, аспірантів*

*і студентів «Інформаційні технології в сучасному світі: дослідження молодих вчених»*. ХНЕУ імені Семена Кузнеця. Харків, Україна, 2026, с. 216.

Онлайн сервіс створення та перевірки кваліфікованого та удосконаленого електронного підпису

ПРОТОКОЛ
створення та перевірки кваліфікованого та удосконаленого електронного підпису

Дата та час: 11:41:36 12.05.2026

Назва файлу з підписом: Omelchenko_diss.pdf.asice
Розмір файлу з підписом: 4.3 МБ

Перевірені файли:
Назва файлу без підпису: Omelchenko_diss.pdf
Розмір файлу без підпису: 5.1 МБ

Результат перевірки підпису: Підпис створено та перевірено успішно. Цілісність даних підтверджено

Підписувач: Омельченко Ігор Валерійович
П.І.Б.: Омельченко Ігор Валерійович
Країна: Україна
РНОКПП: 3442806392
Час підпису (підтверджено кваліфікованою позначкою часу для підпису від Надавача): 11:41:10 12.05.2026
Сертифікат виданий: "Дія". Кваліфікований надавач електронних довірчих послуг
Серійний номер: 514B5C86A1E5DA1104000000670F1F0062268805
Тип носія особистого ключа: ЗНКІ криптомодуль ІІТ Гряда-301
Алгоритм підпису: ДСТУ 4145
Тип підпису: Кваліфікований
Тип контейнера: Підпис та дані в архіві (розширений) (ASiC-E)
Формат підпису: З повними даними для перевірки (XAdES-B-LT)
Сертифікат: Кваліфікований

Версія від: 2026.04.06 13:00