

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему: Можливості та методи адаптації AI агентів всередині симуляції з
умовами що змінюються.

Виконав: студент 4 курсу, групи МФ-41
спеціальність 122 «Комп'ютерні науки»
освітньо-професійна програма
«Інформатика»

Чайка О. Ю.

(прізвище та ініціали)

Керівник Волков І. В.

(прізвище та ініціали)

Рецензент Руккас К. М.

(прізвище та ініціали)

Харків – 2024 року

ЗМІСТ

1. Вступ
 - 1.1. Мета та задача дослідження
 - 1.2. Актуальність теми
 - 1.3. Теоретичне та практичне значення результатів, можливі галузі використання
2. Основна частина
 - 2.1. Постановка задачі
 - 2.2. Компоненти
 - 2.2.1. Бібліотека для роботи з API OpenAI
 - 2.2.2. Проміжний API
 - 2.2.3. World Engine
 - 2.2.4. Habitat
 - 2.3 Архітектура агента
 - 2.3.1. Пам'ять
 - 2.3.2. Perceive
 - 2.3.3. Retrieve
 - 2.3.4. Reflect
 - 2.3.5. Plan
 - 2.3.6 Execute
 - 2.4. API
 - 2.5. Результати
3. Висновки
4. Список використаних джерел
5. Додатки

1. Вступ

1.1 Мета та задача дослідження

Метою даного дослідження є розробка та адаптація архітектури ШІ-агента, що здатний самостійно виконувати послідовні завдання без участі людини, зокрема забезпечити його здатність планувати, аналізувати інформацію та виконувати дії в середовищі симуляції. Для досягнення цієї мети були поставлені такі завдання:

- 1) вивчити основні проблеми сучасних великих мовних моделей (LLM) та їх обмеження щодо довгострокового планування і виконання послідовних завдань;
- 2) розробити архітектуру ШІ-агента, здатного працювати з мінімальним контекстним вікном при збереженні когерентності;
- 3) впровадити та протестувати модульну структуру системи, яка включатиме супервізор, агенти та API;
- 4) оцінити ефективність запропонованої системи шляхом порівняння з оригіналом щодо використання токенів та вартості.

1.2 Актуальність теми

Поява великих мовних моделей (LLM), заснованих на архітектурі трансформерів, стала революційним кроком на шляху розвитку штучного інтелекту. З кожним новим кроком у цій галузі можливості моделей неухильно зростають, що відкриває нові горизонти для їх застосування. Сучасні LLM тепер здатні не лише складати вірші та складати університетські іспити на високі оцінки, а й вирішувати завдання рівня олімпіадних змагань з математики [1].

Однак, незважаючи на всі ці вражаючі досягнення, LLM все ще стикається з серйозною проблемою: вони не можуть самостійно вирішувати завдання, які

вимагають виконання великої кількості послідовних кроків без втручання людини. Для вирішення складних і комплексних завдань все ще потрібна участь людини, яка допомагатиме моделі зберігати фокус на вирішенні проблеми, підказувати правильний напрямок та коригувати її дії. Це відбувається через обмежене вікно уваги та особливості сприйняття поточних архітектур LLM. Коли це вікно повністю заповнюється, генерація тексту моделями стає менш когерентною і починає відхилятися від початкового завдання. Навіть у межах свого контекстного вікна, при його повному використанні, моделі починають відчувати проблеми зі сприйняттям переданої їм інформації [2].

Ця проблема призводить до необхідності пошуку способів скорочення необхідного контекстного вікна, а також стиснення інформації для ефективної обробки мовною моделлю.

1.3 Теоретичне та практичне значення результатів, можливі галузі використання

Для вирішення цієї задачі в рамках цієї роботи було адаптовано архітектуру ІІІ-агента [3], здатного самостійно виконувати послідовні завдання без участі людини. Такий агент може планувати та аналізувати інформацію самостійно. Демонстрація концепції була проведена у формі симуляції повсякденного життя трьох агентів, які самостійно аналізують оточення та події, що відбуваються, планують свої дії, взаємодіють з об'єктами та один з одним.

Важливим досягненням цієї роботи стало зниження витрат на симуляцію, покращення продуктивності системи та впровадження модульної структури програмного коду. Також вдалося значно зменшити використання токенів: в оригінальній системі використовувалося близько 1.7 мільйона токенів, тоді як у представлений у цій роботі системі — приблизно 40 тисяч токенів, за аналогічних параметрів (чотири кроки симуляції та три агенти).

2. Основна частина

2.1 Постановка задачі

У процесі розробки та адаптації сучасних мовних моделей для виконання безлічі різноманітних завдань розробники часто стикаються з проблемою управління цими моделями. Основна складність полягає в тому, що мовна модель, створюючи текст, може втрачати початковий контекст, відволікатися від основного завдання і, зрештою, надавати результати, які не відповідають очікуванням.

Для розв'язання цієї проблеми в цій роботі розроблена [7] архітектура штучного агента, яка дозволяє обмежувати простір допустимих запитів і відповідей мовної моделі. Крім цього, архітектура включає в себе компоненти, що забезпечують створення і підтримання пам'яті про минулі події, а також гнучкий механізм аналізу і обробки отриманої інформації.

Ключові структурні елементи програми

1. Супервайзер (World Engine, WE)

Супервайзер має функцію управління поведінкою агентів у симуляції. WE контролює створення симуляції та наповнення її агентами, а також перевіряє і застосовує всі зміни, внесені агентами в оточення. Крім того, супервайзер відповідає за інкапсуляцію помилок, що виникають у процесі симуляції.

2. Оточення

Оточення — це середовище, в рамках якого агенти здійснюють свою діяльність. Оточення надає агентам можливості для сприйняття та взаємодії. У поточній реалізації оточення представлено однією квартирою і зовнішнім світом, який фактично є ще однією кімнатою, де можна виконувати умовні дії, недоступні для виконання в квартирі.

3. Агенти

Агенти представляють собою основних дійових осіб симуляції. Агенти можуть виконувати як послідовні, так і конкурентні дії. У даній реалізації агенти здійснюють свої дії послідовно для спрощення програмної бази і синхронізації дій.

4. Зв'язуюча ланка (API)

Зв'язуюча ланка є мостом між мовною моделлю і агентами. API дозволяє отримати відповіді від моделі і передавати їх агентам для подальших дій.

Порядок виконання симуляції

Симуляції виконуються покроково: кожен крок дорівнює 12 хвилинам віртуального часу. Агентам змодельована повсякденна діяльність, під час якої вони виконують різноманітні дії залежно від часу доби: вночі вони сплять, вранці снідають, вдень працюють, а ввечері проводять свій вільний час.

На даний момент оточення обмежене однією квартирою і зовнішнім світом, який, по суті, є ще однією кімнатою, призначеною для виконання дій, неможливих усередині квартири. У цій конфігурації симуляції агенти можуть працювати як послідовно, так і конкурентно, однак поточна програмна база передбачає виконання послідовних дій для спрощення.

Важливо зазначити, що запропонована архітектура не обмежується симуляцією, схожою на текстову версію гри «Сімс» [6]. Ця структура надає можливість задавати агентам як довільну поведінку, так і конкретні завдання. Наприклад, взаємодія з предметами в симуляції може слугувати інтерфейсом до зовнішніх API, таких як корпоративна пошта, пошукові системи, хмарні редактори документів і таблиць. Таким чином, агенти, взаємодіючи з об'єктами оточення, через проксі можуть реалізовувати взаємодію із зовнішнім світом і виконувати робочі завдання.

Іншим варіантом використання цієї архітектури може стати створення неігрових персонажів (non-player character, NPC) у відеоіграх. Використання такої системи дозволить створювати персонажів, які не будуть обмежені заздалегідь визначеними діалоговими опціями або діями, що, у свою чергу, може значно підвищити рівень занурення в ігровий світ і взаємодію гравців із віртуальними персонажами.

2.2 Компоненти

Перш ніж переходити до детального аналізу кожного з компонентів, необхідно сформулювати загальне розуміння структури розглядуваної програми. У даній роботі основний код складається з чотирьох основних і одного допоміжного компонента, які виконують різні, але важливі ролі в загальній архітектурі системи. Ці компоненти включають: бібліотеку для відправлення запитів до API OpenAI (допоміжний компонент), агент, оточення, WE і проміжний API для мовної моделі.

1. Бібліотека для відправлення запитів до API OpenAI

Цей допоміжний компонент слугує інтерфейсом для взаємодії з зовнішнім API, наданим OpenAI. Його основним завданням є спрощення процесів формування запитів, відправлення їх до сервісу OpenAI і отримання відповідей. Важливо зазначити, що хоча цей компонент є допоміжним, він критично важливий для забезпечення коректної роботи всієї системи, оскільки від його ефективності залежить загальна продуктивність і надійність взаємодії з API.

2. Агент

Перший ключовий компонент – це агент. Агент являє собою виконавчу одиницю, яка обробляє отриману інформацію і виконує визначені їй дії. Залежно від безлічі різних параметрів і умов, агент може приймати рішення і керувати подальшим потоком виконання програми. Його завдання полягає в

обробці отриманих даних і видачі відповідних відповідей або рішень на основі закладеної логіки.

3. Оточення

Оточення – другий за важливістю компонент нашої системи. Оточення можна розглядати як контекст, у якому функціонує агент. Воно включає в себе всі зовнішні фактори і дані, з якими взаємодіє агент у процесі своєї роботи. Оточення визначає правила і умови, за яких агент здійснює ту чи іншу дію. Це можуть бути різні змінні, середовища даних або поточні стани системи.

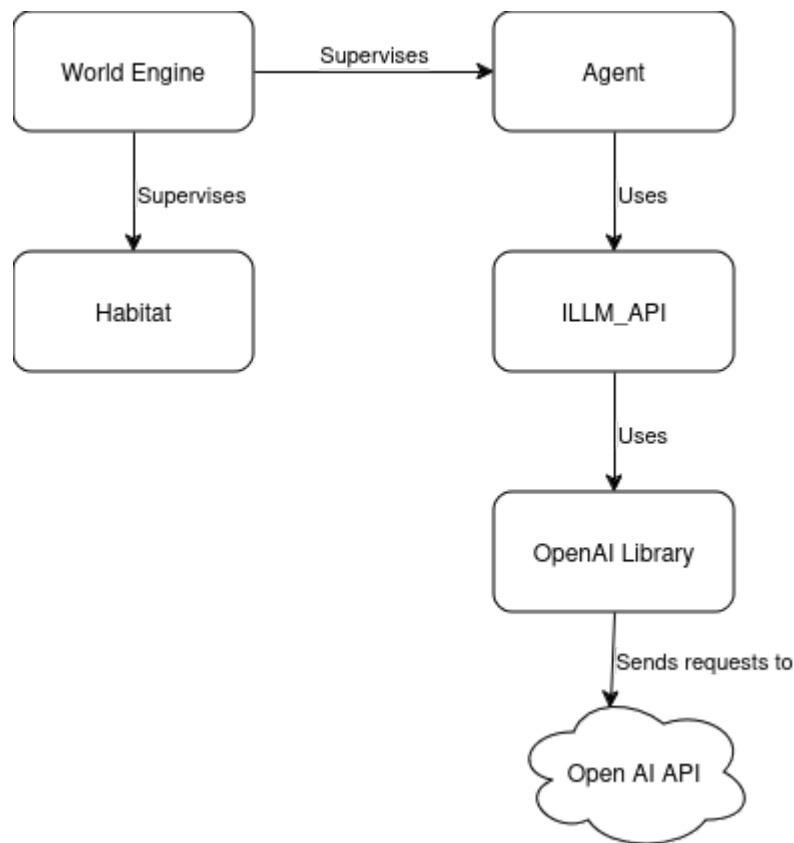
4. World Engine

Супервайзер виконує функцію управління поведінкою агентів у симуляції. Він контролює створення симуляції та наповнення її агентами, а також перевіряє та застосовує всі зміни, внесені агентами в оточення. Крім того, супервайзер відповідає за обробку та інкапсуляцію помилок, що виникають під час симуляції.

5. Проміжний API

Цей компонент відіграє роль посередника між бібліотекою для відправлення запитів і агентом, забезпечуючи синхронізацію і передачу даних. Проміжний API спрощує взаємодію між різними частинами системи, виступаючи як стандартизований механізм обміну інформацією. Він обробляє запити, відправляє їх мовній моделі та повертає результати назад для подальшого використання.

Таким чином, у сукупності ці п'ять компонентів формують складну і взаємопов'язану систему, кожна частина якої виконує свою унікальну роль в забезпеченні її ефективної роботи.



2.2.1 Бібліотека для роботи с API OpenAI

Розробка власної бібліотеки для роботи з API OpenAI була зумовлена відсутністю офіційних інструментів для мови програмування C# [5]. На момент початку проекту не існувало офіційної бібліотеки, наданої OpenAI для інтеграції з їхніми сервісами на платформі .NET. Існуючі рішення, створені користувачами, були перевантажені функціоналом, що ускладнювало їх використання.

Щоб усунути ці недоліки та отримати інструмент, що задовольняє конкретні потреби проекту, була створена нова бібліотека. Вона є легким рішенням, яке включає лише необхідне підмножину всіх можливих запитів до API OpenAI, виключаючи надлишкові функції, що не мають відношення до задач проекту.

Основна функціональність цієї бібліотеки зосереджується на двох ключових аспектах:

1. Створення embedding-векторів: Робота з embedding-векторами є необхідним кроком для підготовки текстових даних перед їх використанням у моделі. Нова бібліотека дозволяє ефективно надсилати запити та отримувати embedding-вектори від API OpenAI, що забезпечує зручність та швидкість обробки даних.

2. Генерація тексту: Генерація текстів є однією з головних можливостей API OpenAI. Розроблена бібліотека спрощує процес надсилання запитів на генерацію текстів, отримуючи відповіді від моделі та обробляючи їх відповідно до вимог симуляції. Це дозволяє інтегрувати інтелектуальні генеративні можливості OpenAI безпосередньо в проєкт, забезпечуючи високу якість автоматичної генерації контенту.

2.2.2 Проміжний API

Проміжний API представляє собою інтерфейс, який включає в себе всі необхідні методи для виконання ключових операцій. Клас, що реалізує цей інтерфейс, використовує спеціалізовану бібліотеку, призначену для відправлення запитів до мовної моделі, з метою отримання необхідних даних. Такий підхід дозволяє забезпечити модульність і гнучкість системи, розділяючи реалізацію мовної моделі та поведінку, необхідну для роботи конкретного додатка.

Інтерфейс виконує роль контракту, якому має відповідати будь-яка реалізація, використовуючи чи хмарну зовнішню мовну модель, чи розгортану локально. Це забезпечує можливість легкої заміни однієї моделі на іншу без необхідності внесення змін у основний код додатку. В результаті, якщо виникає потреба у використанні іншої зовнішньої мовної моделі, чи локального розгортання, можна просто реалізувати інтерфейс відповідним чином та інтегрувати необхідну реалізацію при запуску симуляції в робочому середовищі (WE).

Крім того, повернені системою embedding-вектори обмежені 256 значеннями. Це обмеження введено з метою економії пам'яті. Обмеження розміру векторів

допомагає зменшити обсяг оброблюваних даних, що призводить до прискорення обробки запитів і підвищення загальної ефективності роботи додатка. Таким чином, проміжний API не тільки забезпечує гнучкість і модульність системи, але й сприяє оптимізації її продуктивності.

2.2.3 World Engine

World Engine (WE)—це центральне ядро системи, що забезпечує основне функціонування та взаємодію всіх компонентів симуляції. Під час ініціалізації в його конструктор передається проміжний API, який відіграє роль мосту між різними елементами симуляції. На цій фазі також генеруються агенти та оточення, а також вказується початкове розташування агентів у конкретних стартових локаціях.

Після завершення етапу ініціалізації запускається основний цикл, який працює протягом заздалегідь заданої кількості кроків. На кожному кроці циклу послідовно викликається метод `Invoke` кожного з агентів. Цей метод повертає об'єкт типу `Action`, який є структурою даних, що описує всі деталі передбачуваної дії агента.

Структура `Action` проходить процес верифікації. Якщо дана дія коректна (тобто агент не намагається використати об'єкт, який взагалі не існує, або який знаходиться в іншій локації, або не намагається переміститися в неіснуючу локацію тощо), то відповідна дія реєструється в оточенні і виконується. У випадку, якщо дія передбачає переміщення, агент фізично переміщується в іншу локацію. Якщо ж структура дії виявляється некоректною, реєструється подія, що докладно описує помилку. Ця подія буде оброблена агентом на наступному кроці, надаючи йому інформацію, необхідну для виправлення помилки.

Ключовим елементом на цьому рівні є структура `Event`. `Event` відображає або успішну, або неуспішну дію агента. Приклади подій могли б бути такими: "Джон використовує мікрохвильовку протягом 12 хвилин" або "Еліза намагалася переміститися на кухню, використовуючи двері до спальні! Вона

повинна змінити дію або використати правильні двері". Важливо зазначити, що з точки зору агентів усі події однакові—немає різниці між подією помилки і подією дії.

Мета Event — це передавати певну текстову інформацію всім агентам у локації. Події мають тривалість, яка визначає часовий інтервал, протягом якого вони можуть бути сприйняті. Наприклад, якщо помилка має тривалість 1, то вона буде сприйнята на наступному кроці. Якщо ж ми говоримо про більш тривалу дію, наприклад використання ліжка протягом 8 годин, то тривалість події буде дорівнювати 40 крокам ($60 \text{ хвилин} / 12 \text{ хвилин} * 8 \text{ годин}$).

Усі події зберігаються в межах локації та оновлюються на початку кожного кроку симуляції. Якщо тривалість події спливла, то вона видаляється з локації, що означає, наприклад, якщо Джон припинив використовувати мікрохвильовку, то новоприбула Еліза не знатиме про те, що він її використовував.

Таким чином, World Engine забезпечує цілісність і взаємодію між агентами, подіями та оточенням, реалізуючи складні сценарії в рамках симуляції.

2.3 Архітектура агента

Агент має двокомпонентну структуру пам'яті, що включає просторовий та загальний компоненти. В основі пам'яті агента лежить безперервний потік спогадів, які формуються і використовуються в процесі симуляції. На кожному етапі симуляції (timestep) агенту необхідно виконати чотири ключові завдання: сприймати події, що відбуваються навколо, обробляти отриману інформацію, складати як довгострокові, так і короткострокові плани, а потім вирішувати, які дії вжити у відповідь на виявлені події.

Для реалізації цих завдань були розроблені п'ять фундаментальних функцій: Perceive, Retrieve, Plan, Reflect і Execute.

1. Perceive

Ця функція відповідає за сприйняття подій, що відбуваються в безпосередньому оточенні агента. Вона обробляє всі вхідні події, оновлюючи поточний стан сприйняття агента.

2. Retrieve

Функція Retrieve займається вилученням релевантних спогадів з пам'яті агента. Ці спогади допомагають у формуванні контексту для поточних дій.

3. Plan

На основі отриманих даних і вилучених спогадів, функція Plan складає стратегії і плани. Ці плани дробляться на довгострокові цілі і короткострокові задачі, пристосовуючись до змінної ситуації.

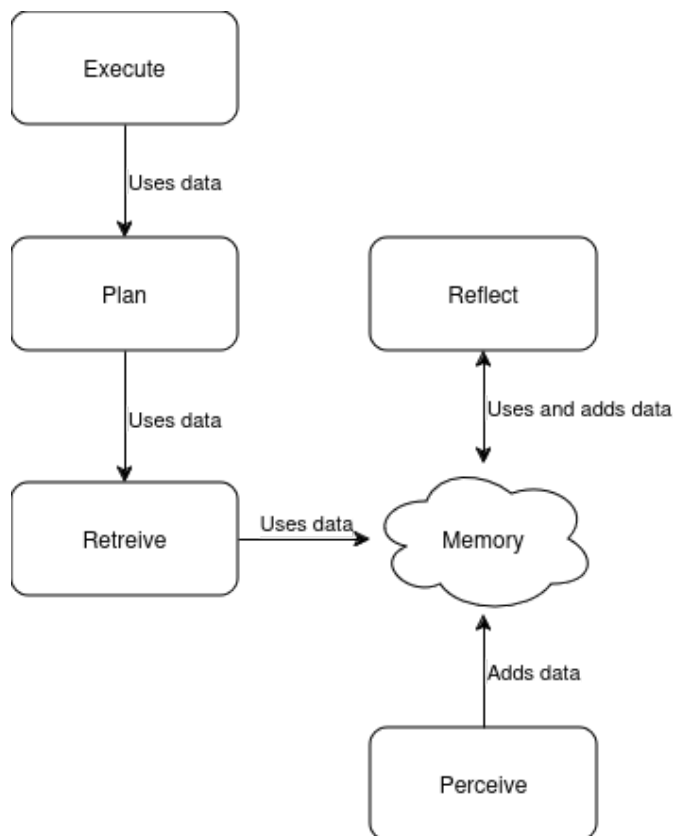
4. Reflect

Дана функція виконує обробку і аналіз інформації, узагальнює отримані дані та висновки, які агент може використовувати в майбутньому для покращення своїх рішень і дій.

5. Execute

Функція Execute відповідає за безпосереднє виконання дій, які обрав агент. Вона слідує раніше розробленим планам і адаптує їх до поточних обставин.

Усі ці функції об'єднані оболонкою у вигляді функції Invoke, яка викликається на кожному кроці симуляції. У цю функцію передаються всі події в локації, де знаходиться агент, об'єкт спогадів (remembrance-object) цієї локації і поточний крок симуляції (timestep). Таким чином, Invoke слугує центральним елементом, що зв'язує всі процеси сприйняття, обробки, планування, рефлексії і виконання дій, що дозволяє агенту ефективно реагувати на змінні умови і досягати поставлених цілей.



2.3.1. Пам'ять

Пам'ять агента в нашій системі поділяється на два основних типи: просторову пам'ять і загальну, також відому як генеральна пам'ять. Проста пам'ять реалізована за допомогою об'єкта `HabitatRemembrance (HR)`, тоді як генеральна пам'ять керується класом `AgentMemory (AM)`. Якщо не зазначено інше, під терміном "пам'ять" надалі буде розумітися саме `AM`.

Основним завданням пам'яті агента є збереження спогадів і надання зручного інтерфейсу для визначення таких характеристик, як недавність, важливість та релевантність спогадів.

Кодова структура пам'яті:

```

1. Stack<MemoryObject> _unreflectedMemories;
2. uint _unreflectedMemoriesImportance;
3. Dictionary<Guid, MemoryObject> _memoryDictionary;
4. Queue<string> _longTermPlanning;
5. Queue<string> _shortTermPlanning;

```

Пам'ять агента складається з кількох ключових компонентів:

1. Словник спогадів

Це структура даних, у якій зберігаються всі спогади агента. Вона дозволяє швидко знаходити конкретний спогад за його унікальним ідентифікатором.

2. Черги планування

У пам'яті агента існують дві черги для планування – одна призначена для довготривалих цілей (`_longTermPlanning`), інша для короткотривалих (`_shortTermPlanning`).

3. Стек непереглянутих спогадів

Це колекція спогадів, для яких агент ще не провів процедуру рефлексії. Кожен новий спогад додається в цей стек.

4. Об'єкти спогадів

Кожен спогад представлений об'єктом класу `MemoryObject`, який включає в себе опис спогаду, час його додавання, значення важливості від 0 до 10 і `embedding`-вектор. Цей клас має наступні параметри:

1. `Guid Id`;
2. `string Description`;
3. `uint Timestep`;
4. `uint Importance`;
5. `float[] Embedding`;

Існує також похідний клас `EventMemoryObject`, який розширює `MemoryObject`, додаючи властивість "тривалість" і змінюючи функцію обчислення недавності спогадів.

При додаванні нового об'єкта в пам'ять, на основі його опису обчислюються його важливість і `embedding`-вектор. Після цього об'єкт додається в стек `_unreflectedMemories`, а його важливість збільшує загальний лічильник важливості непереглянутих спогадів.

Недавність спогаду відносно поточного моменту часу визначається за допомогою оберненої експоненціальної функції, де ($t = \text{currentTimestamp} - \text{Timestamp}$):

$$\tau = e^{-0.995 * t}$$

Для об'єктів типу `EventMemoryObject` функція включає в себе як час збереження спогаду, так і тривалість події, тобто t дорівнює $\text{currentTimestamp} - \text{Timestamp} + \text{Duration}$.

Релевантність спогаду обчислюється як косинус подібності між embedding-векторами двох спогадів, результатом чого є значення в діапазоні від 0 до 1:

$$\cos(\theta) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Ця архітектура дозволяє ефективно і гнучко керувати спогадами агента, їх асоціаціями та важливістю для різних завдань в рамках роботи агента.

2.3.2 Perceive

Метод "Perceive" є важливим елементом функціональності системи, відповідальним за сприйняття агентом навколишнього середовища та різних подій. Цей процес включає оновлення просторової та основної пам'яті агента, щоб він міг успішно аналізувати та реагувати на зміни в навколишньому середовищі.

На першому етапі роботи методу агент отримує оновлені дані про поточну локацію через спеціальний об'єкт "remembrance", який передається з модуля World Environment (WE) у метод "Invoke", а далі в "Perceive". Цей об'єкт містить

всі необхідні деталі про поточний стан оточення, включаючи інформацію про позиції об'єктів, їхні властивості, а також про події, що відбулися.

Далі процес оновлення основної пам'яті агента включає такі кроки:

1. Очищення даних

Масив івентів, що надійшов з модуля WE, спершу очищується від дублікатів та тих івентів, які вже присутні в пам'яті агента. Це дозволяє уникнути зайвого повторення інформації та зменшити навантаження на пам'ять.

2. Збереження унікальних подій

Після очищення залишаються тільки унікальні події, які ще не були збережені. Ці події інтегруються в пам'ять агента як об'єкти типу "EventMemoryObject". Кожному такому об'єкту присвоюється ідентифікатор (ID), що збігається з ID івенту. Це робиться для того, щоб уникнути дублювання інформації та полегшити подальший доступ до конкретних подій.

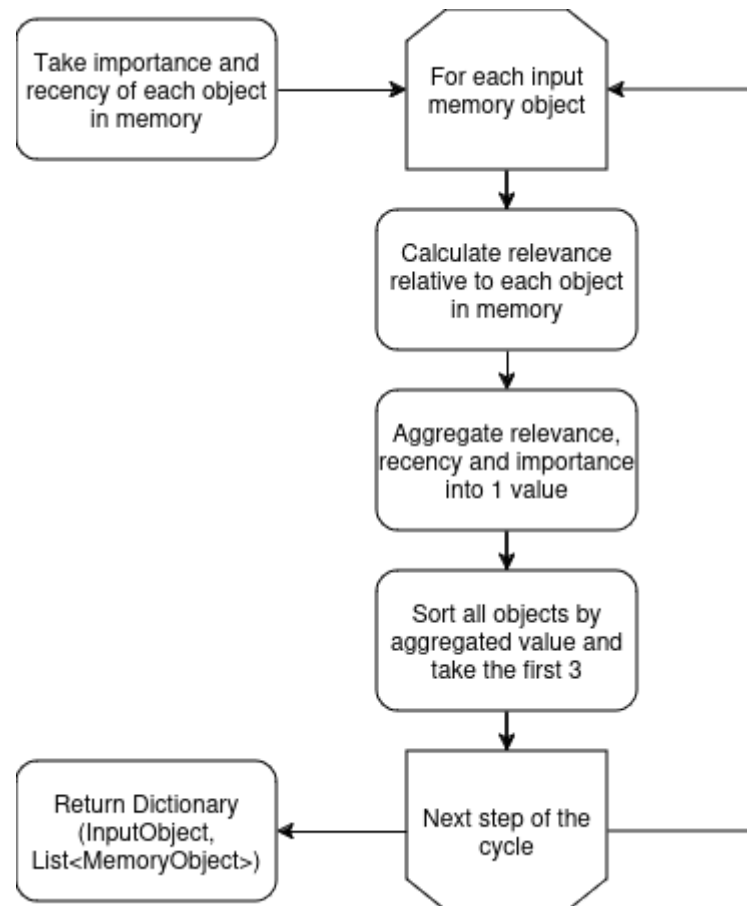
3. Обчислення embedding-вектора та важливості

Для кожної нової події обчислюється embedding-вектор, що являє собою числову модель опису івенту. Разом із цим обчислюється важливість події. Ці значення є ключовими для подальшого аналізу і вибору найбільш значущих подій.

Після виконання всіх цих кроків метод повертає перелік усіх збережених об'єктів пам'яті. Цей список дозволяє агенту мати повне та актуальне уявлення про всі минулі та поточні події. Таким чином, метод "Perceive" відіграє критичну роль у підтриманні актуальності та точності даних про середовище, що, у свою чергу, дозволяє агенту приймати більш обґрунтовані та ефективні рішення.

2.3.3 Retrieve

Метод Retrieve являє собою інструмент, що використовується для вилучення найбільш релевантного набору спогадів відносно до якогось заданого спогаду. Цей метод відіграє ключову роль у системі пам'яті, забезпечуючи можливість пошуку і сортування спогадів на основі їх значимості, свіжості та інших метрик.



Опис роботи методу

На вхід цьому методу подається список спогадів, для яких потрібно отримати релевантний набір спогадів. В результаті роботи методу створюється словник, де ключами є спогади з переданого списку, а значеннями — колекції найбільш релевантних спогадів для кожного ключового спогаду.

Визначення релевантності

Релевантність між двома спогадами розраховується за наступною формулою:

$$\rho = a + 2b + 3c$$

де:

- a — значимість спогаду. Ця метрика враховує важливість спогаду в контексті загальної системи пам'яті.
- b — свіжість спогаду, яка обчислюється за формулою, описаною в розділі 2.3.1. Свіжість вказує на відносний час, що минув з моменту створення або оновлення спогаду.
- c — косинусна відстань між embedding-векторами двох спогадів. Косинусна відстань вимірює їх подібність на основі векторів, які представляють спогади в багатовимірному просторі.

Порядок виконання

Для всіх спогадів, що знаходяться в пам'яті, враховується їх релевантність з кожним із переданих на вхід спогадів. Потім ці спогади сортуються за спаданням значень релевантності. Після сортування з списку беруться три найбільш релевантних спогади, які включаються в підсумковий словник.

Алгоритм

1. Ініціалізація

Створення пустого словника для зберігання результатів.

2. Обчислення релевантності

Для кожного спогаду зі списку на вході проводиться підрахунок релевантності між цим спогадом і всіма іншими спогадами в пам'яті.

3. Сортування

Отримані результати сортуються за спаданням значення релевантності.

4. Формування підсумкового набору

Вибираються топ-3 найбільш релевантні спогади для кожного вихідного спогаду і заносяться в словник.

5. Повторення

Процес повторюється для кожного об'єкта пам'яті з переданого списку.

Приклад

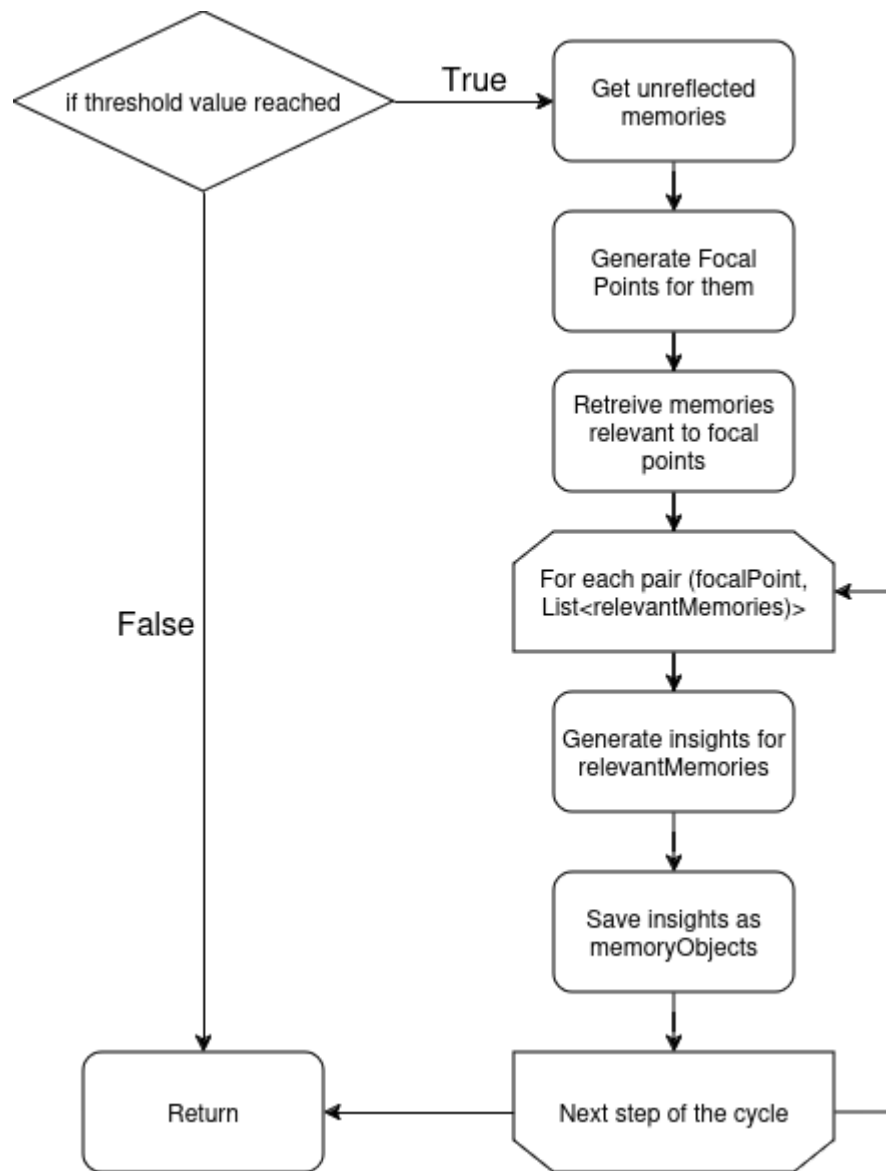
Припустимо, у агента є спогад "А", для якого потрібно знайти релевантні спогади. Метод Retrieve візьме цей спогад і проаналізує всі доступні спогади в пам'яті. Потім він обчислить релевантність кожного з них до "А", відсортує їх за спаданням, і вибере три найбільш релевантних, повертаючи їх у вигляді колекції, пов'язаної з ключем "А".

Таким чином, метод Retrieve забезпечує структурований та ефективний механізм для пошуку і агрегування найбільш релевантних спогадів, що сприяє поліпшенню роботи системи пам'яті.

2.3.4 Reflect

Рефлексія — це процес генерації нових об'єктів пам'яті на основі вже існуючих. Значна частина цього процесу полягає в аналізі подій, що відбулися, та генерації нових думок і висновків щодо них. У процесі рефлексії агент систематично переробляє інформацію, що дозволяє йому осмислювати події на більш глибокому рівні.

Процес рефлексії ініціюється лише у випадку, якщо сумарна значущість спогадів з моменту останньої рефлексії перевищує певне порогове значення. Це означає, що якщо значення всіх спогадів, накопичених з моменту останньої рефлексії, перевищують порогове значення, тоді запускається процес рефлексії.



Перший крок у процесі рефлексії — генерація фокальних точок для всіх не відрефлексованих спогадів. Для цього нейромережі передається список спогадів з метою виділення найбільш значущих питань. Запит до мовної моделі формулюється таким чином:

Given only the information below, what are 3 most salient high-level questions we can answer about the subjects in the statements?

Відповідь, отримана від мовної моделі, перетворюється на три об'єкти пам'яті, які, однак, на цьому етапі не зберігаються. Це важливо для того, щоб запустити функцію retrieve, використовуючи ці фокальні точки як орієнтир. Суть функції

retrieve полягає в тому, щоб виділити найбільш релевантні записи пам'яті на основі заданих фокальних точок.

Після цього ми отримуємо словник, у якому ключами є три найбільш значущі високорівневі питання, пов'язані з не відрефлексованими спогадами, а значеннями — списки спогадів, найбільш релевантних кожному з питань.

Наступний крок — запит до мовної моделі для отримання високорівневих інсайтів. Запит формулюється таким чином:

Given the following information, what 3 high-level insights can you infer from the below statements?

Для кожного з фокальних питань передається відповідний список спогадів, отриманий за допомогою методу retrieve. Високорівневі інсайти, що генеруються на основі цих даних, зберігаються в пам'ять як нові об'єкти пам'яті.

Коли нові об'єкти пам'яті додаються до колекції, вони стають частиною не відрефлексованих спогадів. Це означає, що при наступному запуску процесу рефлексії ці об'єкти пам'яті також будуть враховані. Подібний підхід дозволяє з часом вибудовувати та генерувати все більш складні і високорівневі висновки з простих спостережень, надаючи системі гнучкість і адаптивність у реакції на зміни навколишнього середовища та накопичення нового досвіду.

Таким чином, процес рефлексії є невід'ємною частиною інтелектуальних систем, забезпечуючи їх здатність до самовдосконалення та більш глибокого розуміння накопиченого досвіду.

2.3.5 Plan

Планування можна розділити на два основні види: довгострокове і короткострокове. Кожен з цих видів планування слідує за власною часовою шкалою та виконує свою окрему функцію в процесі симуляції. Довгострокові плани розробляються для більш тривалих часових інтервалів, а саме на кожні

дві години. Короткострокові ж, навпаки, орієнтовані на менші часові проміжки – на кожен крок симуляції, який складає 12 хвилин. Відповідно, кожен довгостроковий план використовується для генерації 10 короткострокових планів, що дозволяє забезпечити безперервне та гнучке реагування на зміни середовища.

Довгострокове планування

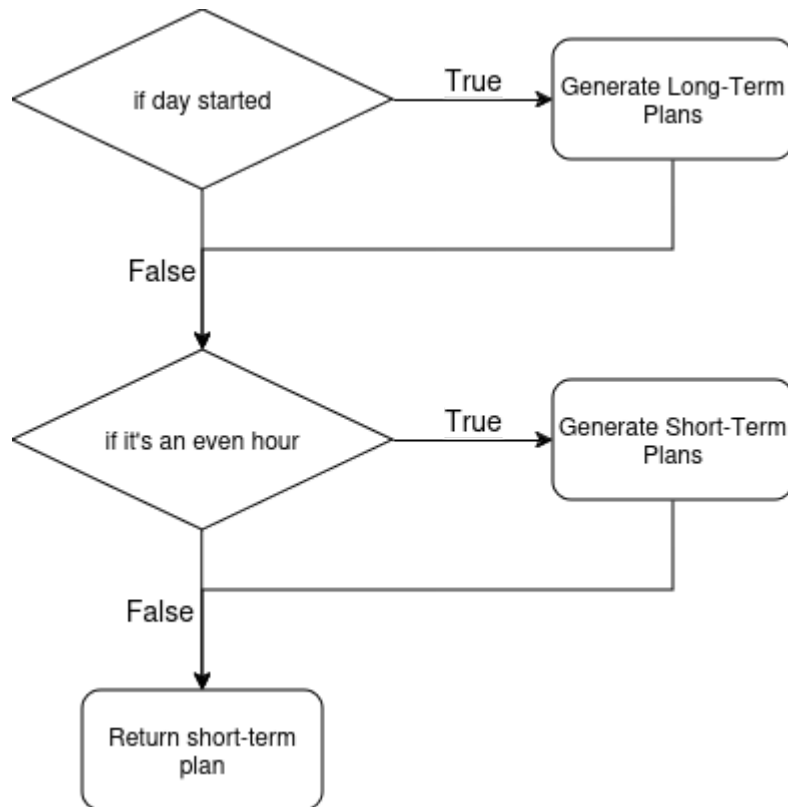
Для розробки довгострокових планів використовуються кілька джерел даних. Це включає список подій, які відбулися протягом попереднього дня, оскільки довгострокове планування здійснюється один раз на добу. На додаток до цього використовується особиста інформація агента та текстове представлення його просторової пам'яті. На основі цих даних формується запит до мовної моделі. Результат роботи моделі перетворюється на набір окремих рядків, що представляють собою плани. Ці плани потім зберігаються в колекцію в пам'яті, причому використовується структура даних черга, щоб перший план, доданий у пам'ять, був першим, який витягується з неї за потреби.

Короткострокове планування

Короткострокові плани генеруються на основі поточного довгострокового плану. Щоб сформувати короткостроковий план, з пам'яті витягується наступний довгостроковий план разом із просторовою пам'яттю та актуальною інформацією про агента. Додатково використовується результат функції Retrieve, яка обробляє події, додані на цьому кроці в пам'ять із запуском функції Perceive. Це дозволяє агенту враховувати найсвіжішу та релевантну інформацію про навколишні його події, що важливо для точного та своєчасного реагування на зміни в середовищі. Отримані 10 короткострокових планів додаються до відповідної черги.

Логіка генерації планів

Завдання генерації планів здійснюється за розкладом:



Довгострокові плани генеруються кожні 120 кроків симуляції (що еквівалентно 24 годинам реального часу), а короткострокові кожні 10 кроків (або кожні 120 хвилин). Таким чином, функція завжди повертає короткостроковий план, який використовується для прийняття подальших рішень та визначення поточних дій агента.

2.3.6 Execute

Функція Execute представляє собою обгортку навколо методу ParsePlan API і відіграє важливу роль у періодичному плануванні та координації дій агентів. ParsePlan використовує короткостроковий план на наступні 12 хвилин та просторову пам'ять для генерації структури даних з наступними параметрами:

1. Guid AgentId,
2. string Description,
3. string Area,

- 4. string TargetItem,
- 5. uint Duration

Цей метод істотно розширює можливості планування, комбінуючи короткостроковий часовий горизонт з геопросторовою пам'яттю для створення детальних і скоординованих планів дій для агентів.

Функція починається з отримання короткострокового плану, який деталізує дії, заплановані для середнього та короткого проміжку часу. Використовуючи надані дані, метод мовної моделі генерує JSON-об'єкт. Цей JSON-об'єкт містить всю необхідну інформацію для створення нового екземпляра структури Activity. Ось кроки, які виконує функція:

1. Отримання JSON-об'єкта

Метод ParsePlan API генерує JSON-об'єкт за допомогою LLM на основі опису дії, області виконання, цільового предмета і тривалості.

2. Десеріалізація JSON-об'єкта

Генерований JSON-об'єкт перетворюється в екземпляр структури Activity з використанням процесу десеріалізації. Цей процес аналізує JSON-об'єкт і зводить його до структури Activity з необхідними параметрами.

3. Верифікація та застосування

Отримана структура Activity повертається в робоче середовище (WE), яке відповідає за перевірку коректності та цілісності даних. WE виконує верифікацію даних, перевіряє відповідність отриманої активності встановленим правилам і вимогам.

4. Виконання та повторення

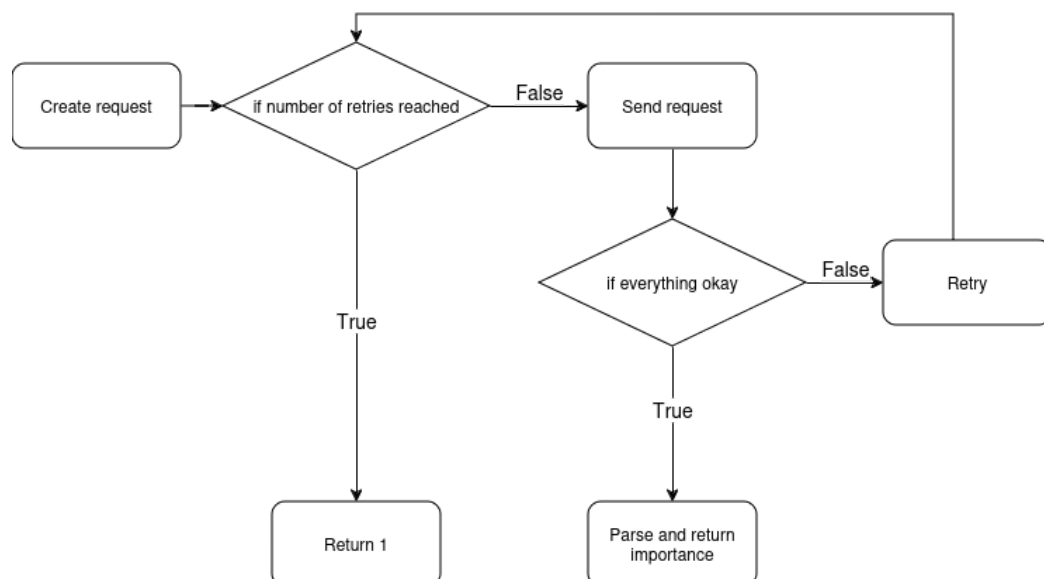
Якщо дані структури успішно верифіковані, WE застосовує зміни, визначені новою активністю, до відповідного агента. Після успішного застосування змін процес повторюється для всіх інших агентів, забезпечуючи скоординовані дії в рамках заданого планування.

2.4 API

API представляє собою важливий компонент у ланцюжку взаємодії різних програмних модулів, однак на перший погляд його функціональність може здатися скромною. Основне призначення API полягає у створенні коректних запитів, перевірці відповідей від мовної моделі та подальшій передачі результатів клієнту. Проте, враховуючи сучасні реалії, коли мовні моделі, незважаючи на їх високий рівень розвитку, іноді можуть повертати результати, які не відповідають очікуванням користувачів з різних причин, в API передбачені механізми, що сприяють підвищенню надійності виконання запитів.

Так, кожна функція, за винятком генерації embedding-векторів, має вбудовану систему повторних запитів, що допомагає гарантувати отримання коректної відповіді від нейромережі. У разі некоректного результату або при виникненні помилки у процесі відповіді, запит перенаправляється знову до n разів (у контексті цього обговорення значення n дорівнює 2). Це дозволяє збільшити ймовірність отримання коректної відповіді, зменшивши можливість помилок, зумовлених випадковими факторами або тимчасовими збоями в роботі нейромережі.

Розглянемо схематичну структуру методів API:



Якщо у процесі виконання виникає помилка, функція повторно відправляє запит до нейромережі. При досягненні максимальної кількості повторних спроб (у нашому випадку - двох), функція логує помилку і повертає наперед визначене значення. Такий підхід дозволяє гнучко контролювати процес виконання та підвищувати стабільність роботи системи, незважаючи на можливі проблеми у роботі нейромережі.

2.5 Результати

У процесі симуляції агенти переміщувались по оточенню, взаємодіючи як один з одним, так і з різними предметами. Ми запускали симуляцію на 20 кроків, але, на жаль, досягли ліміту в API до її завершення. Тим не менш, ми змогли успішно провести 14 кроків без жодних помилок. Це доводить, що всі компоненти системи функціонували коректно. Система включала генерацію як короткострокових, так і довгострокових планів, всі функції API працювали, а рідкісні помилки, що виникали під час симуляції, ефективно і без затримок виправлялися агентами на подальших кроках завдяки інкапсуляції помилок.

Фінальна версія нашої симуляції використовувала три моделі: gpt-3.5, gpt-4-turbo і embedding-small. З цих моделей, gpt-3.5 виконувала більшу частину запитів. Проте, при використанні gpt-3.5 ми зіткнулися з проблемами, пов'язаними з просторовим мисленням, що потребували використання gpt-4-turbo на етапі парсингу планів у json-об'єкти. Ці проблеми проявлялися в нездатності агентів правильно орієнтуватися в просторах і переходити з однієї локації в іншу. Спроби використовувати новітню модель gpt-4-omni замість gpt-4-turbo також не увінчалися успіхом. Дана модель генерувала занадто велику кількість планів на день або некоректно створювала json-об'єкти, порушуючи прописані правила.

За винятком завдання перетворення активності (парсинг планів у json), з рештою завдань gpt-3.5 справлялася відмінно. Це дозволило значно знизити

загальну вартість симуляції, оскільки використання gpt-3.5 є більш економічно вигідним.

Підведемо підсумок витрат на 14 кроків симуляції в даній конфігурації. Загальні витрати склали приблизно 90 000 токенів для gpt-4-turbo і близько 50 000 токенів для gpt-3.5. У сумі це становить 140 000 токенів або приблизно 10 000 токенів на один крок симуляції. Це приблизно в 40 разів менше, ніж витрати на аналогічну симуляцію в оригінальному проекті [4].

Таким чином ми змогли показати, що наша система функціонує стабільно при економії використання токенів у 40 разів у порівнянні з оригіналом.

3. Висновки

По-перше, значущість даного дослідження очевидна в контексті широкого спектра можливостей, які відкриваються перед застосуванням великих мовних моделей у різних галузях. Сучасні LLM здатні розв'язувати складні завдання, сочиняти тексти та створювати багато інших корисних застосувань, однак, існує ключова проблема — підтримання когерентності при виконанні довгострокових і послідовних завдань. У рамках цієї роботи була реалізована архітектура, яка вирішує дану проблему, знижуючи залежність від довжини контекстного вікна та покращуючи здатність моделі справлятися з багатоступеневим плануванням без участі людини.

Зниження витрат на симуляцію, а також підвищення продуктивності системи є одними з ключових досягнень даної роботи. Порівняльний аналіз показує, що оригінальна система споживала близько 1,7 мільйона токенів на виконання симуляції [4], тоді як представлена тут система споживає всього близько 40 тисяч токенів, що становить приблизно 4 кроки симуляції на трьох агентів. Це представляє значне покращення в області ефективності та робить розроблену систему значно більш економічною з точки зору використання ресурсів.

Розроблена система включає ряд компонентів — супервізор (World Engine, WE), оточення, агенти і API. Внаслідок впровадження модульної структури коду стало можливим не тільки спрощення симуляції, але й її подальша адаптація та розширення під різні завдання. Наприклад, взаємодія із зовнішніми API, створення NPC персонажів для відеоігор та інші можливі сценарії використання демонструють універсальність запропонованого рішення.

Аналіз архітектури агента показав, що використання двокомпонентної структури пам'яті для просторових і загальних даних допомагає ефективно зберігати та витягати спогади, оптимізуючи процес прийняття рішень. Функції Perceive, Retrieve, Plan, Reflect та Execute дозволяють агенту ефективно сприймати, аналізувати та планувати свої дії, що робить його поведінку більш

осмисленою та цілеспрямованою. Алгоритми оновлення пам'яті та планування дозволяють агенту адаптуватися до змінних умов середовища, що зміцнює його здатність до розв'язання складних багаторівневих завдань.

Важливим досягненням також стала стійкість симуляції до помилок. Відповідні помилки, які виникали під час симуляції, ефективно оброблялися супервізором, а агенти швидко адаптувалися до нових умов, що підтверджує високу надійність запропонованої архітектури (див. додаток).

За результатами симуляцій вдалося досягти значного скорочення використання токенів при збереженні когерентності дій агента. Це доводить, що запропонована архітектура та методи є дієвим засобом для покращення роботи великих мовних моделей у реальних застосуваннях. Наукове значення та практична цінність таких розробок відкривають нові горизонти для подальших досліджень та широкого використання LLM у різних навчальних та професійних сферах.

Майбутні дослідження можуть поглибити розробку архітектури ШІ-агентів, включивши в неї більш складні алгоритми планування та взаємодії із зовнішнім оточенням, інтеграції з новими API та методами машинного навчання. Також можна розглянути можливість використання розроблених методів для розв'язання реальних завдань у бізнесі, медицині, освіті та інших галузях.

4. Список використаних джерел

[1] – A Comprehensive Overview of Large Language Models

<https://arxiv.org/abs/2307.06435>

[2] – Long-context LLMs Struggle with Long In-context Learning

<https://arxiv.org/html/2404.02060v1>

[3] – Generative Agents: Interactive Simulacra of Human Behavior

<https://arxiv.org/abs/2304.03442>

[4] – Price and mitigations

https://github.com/joonspk-research/generative_agents/issues/115

[5] – Available OpenAI libraries

<https://platform.openai.com/docs/libraries/python-library>

[6] – The Sims game series https://en.wikipedia.org/wiki/The_Sims

[7] – Our work: <https://github.com/Lesaje/Reflexity>

5. Додатки

Консольний лог:

```
[14:02:26 INF] Starting Reflexity
[14:02:26 INF] The global logger has been configured
[14:02:26 INF] WE was initialized successfully!
[14:02:26 INF] Agent Paul Braumer is invoked at time step 0
[14:02:38 INF] Event Paul Braumer is moving to: Kitchen was created in Bedroom
[14:02:38 INF] Paul Braumer moved to Kitchen
[14:02:38 INF] Agent Laura Braumer is invoked at time step 0
[14:02:49 INF] Something went wrong with activity Laura Braumer at timestep 0
[14:02:49 INF] Event Laura Braumer is trying to make following action: Move from
Bedroom to Kitchen with the wrong item! Because the item is already in use he
must either change item to another one, or wait until item will be no longer in
use! The item in question isDoor to Kitchen The item is currently used by: Paul
Braumer and will be in use for another: 1 time steps. was created in Bedroom
[14:02:49 INF] Agent Luisa Braumer is invoked at time step 0
[14:03:01 INF] Something went wrong with activity Luisa Braumer at timestep 0
[14:03:01 INF] Event Luisa Braumer is trying to make following action: Move from
Bedroom to Kitchen with the wrong item! Because the item is already in use he
must either change item to another one, or wait until item will be no longer in
use! The item in question isDoor to Kitchen The item is currently used by: Paul
Braumer and will be in use for another: 1 time steps. was created in Bedroom
[14:03:01 INF] Agent Paul Braumer is invoked at time step 1
[14:03:03 INF] Event Paul Braumer is moving to: Bedroom was created in Kitchen
[14:03:03 INF] Paul Braumer moved to Bedroom
[14:03:03 INF] Agent Laura Braumer is invoked at time step 1
[14:03:07 INF] Event Laura Braumer is moving to: Bathroom was created in Bedroom
[14:03:07 INF] Laura Braumer moved to Bathroom
[14:03:07 INF] Agent Luisa Braumer is invoked at time step 1
[14:03:09 INF] Event Luisa Braumer is moving to: Kitchen was created in Bedroom
[14:03:09 INF] Luisa Braumer moved to Kitchen
[14:03:09 INF] Agent Paul Braumer is invoked at time step 2
[14:03:13 INF] Event Paul Braumer is moving to: Bathroom was created in Bedroom
[14:03:13 INF] Paul Braumer moved to Bathroom
[14:03:13 INF] Agent Laura Braumer is invoked at time step 2
```


[14:03:15 INF] Event Laura Braumer is moving to: Child Room was created in Bathroom

[14:03:15 INF] Laura Braumer moved to Child Room

[14:03:15 INF] Agent Luisa Braumer is invoked at time step 2

[14:03:17 INF] Event Luisa Braumer is making following action: Clean up dishes and kitchen after breakfast was created in Kitchen

[14:03:17 INF] Luisa Braumer is using Sink for 1 time steps

[14:03:17 INF] Agent Paul Braumer is invoked at time step 3

[14:03:19 INF] Event Paul Braumer is moving to: Child Room was created in Bathroom

[14:03:19 INF] Paul Braumer moved to Child Room

[14:03:19 INF] Agent Laura Braumer is invoked at time step 3

[14:03:21 INF] Event Laura Braumer is moving to: Bathroom was created in Child Room

[14:03:21 INF] Laura Braumer moved to Bathroom

[14:03:21 INF] Agent Luisa Braumer is invoked at time step 3

[14:03:23 INF] Event Luisa Braumer is moving to: Bedroom was created in Kitchen

[14:03:23 INF] Luisa Braumer moved to Bedroom

[14:03:23 INF] Agent Paul Braumer is invoked at time step 4

[14:03:25 INF] Event Paul Braumer is moving to: Bathroom was created in Child Room

[14:03:25 INF] Paul Braumer moved to Bathroom

[14:03:25 INF] Agent Laura Braumer is invoked at time step 4

[14:03:28 INF] Event Laura Braumer is moving to: Bedroom was created in Bathroom

[14:03:28 INF] Laura Braumer moved to Bedroom

[14:03:28 INF] Agent Luisa Braumer is invoked at time step 4

[14:03:30 INF] Event Luisa Braumer is making following action: Relax on the bed and read a book or listen to music was created in Bedroom

[14:03:30 INF] Luisa Braumer is using Bed for 1 time steps

[14:03:30 INF] Agent Paul Braumer is invoked at time step 5

[14:03:32 INF] Event Paul Braumer is making following action: Take a relaxing shower in the Bathroom was created in Bathroom

[14:03:32 INF] Paul Braumer is using Shower for 1 time steps

[14:03:32 INF] Agent Laura Braumer is invoked at time step 5

[14:03:34 INF] Event Laura Braumer is moving to: Kitchen was created in Bedroom

[14:03:34 INF] Laura Braumer moved to Kitchen

[14:03:34 INF] Agent Luisa Braumer is invoked at time step 5

[14:03:36 INF] Event Luisa Braumer is making following action: Organize wardrobe and plan outfits was created in Bedroom

[14:03:36 INF] Luisa Braumer is using Wardrobe for 1 time steps

[14:03:36 INF] Agent Paul Braumer is invoked at time step 6

[14:03:39 INF] Event Paul Braumer is moving to: Bedroom was created in Bathroom

[14:03:39 INF] Paul Braumer moved to Bedroom

[14:03:39 INF] Agent Laura Braumer is invoked at time step 6

[14:03:40 INF] Event Laura Braumer is moving to: Bedroom was created in Kitchen

[14:03:40 INF] Laura Braumer moved to Bedroom

[14:03:40 INF] Agent Luisa Braumer is invoked at time step 6

[14:03:42 INF] Event Luisa Braumer is moving to: Bathroom was created in Bedroom

[14:03:42 INF] Luisa Braumer moved to Bathroom

[14:03:42 INF] Agent Paul Braumer is invoked at time step 7

[14:03:45 INF] Event Paul Braumer is moving to: Kitchen was created in Bedroom

[14:03:45 INF] Paul Braumer moved to Kitchen

[14:03:45 INF] Agent Laura Braumer is invoked at time step 7

[14:03:47 INF] Event Laura Braumer is moving to: Bathroom was created in Bedroom

[14:03:47 INF] Laura Braumer moved to Bathroom

[14:03:47 INF] Agent Luisa Braumer is invoked at time step 7

[14:03:49 INF] Event Luisa Braumer is making following action: Take a refreshing shower and get ready for the day was created in Bathroom

[14:03:49 INF] Luisa Braumer is using Shower for 1 time steps

[14:03:49 INF] Agent Paul Braumer is invoked at time step 8

[14:03:52 INF] Event Paul Braumer is making following action: Have a family snack time was created in Kitchen

[14:03:52 INF] Paul Braumer is using Coffee Maker for 1 time steps

[14:03:52 INF] Agent Laura Braumer is invoked at time step 8

[14:03:54 INF] Event Laura Braumer is moving to: Home Office was created in Bathroom

[14:03:54 INF] Laura Braumer moved to Home Office

[14:03:54 INF] Agent Luisa Braumer is invoked at time step 8

[14:03:56 INF] Something went wrong with activity Luisa Braumer at timestep 8

[14:03:56 INF] Event Luisa Braumer is trying to make following action: Move from Bathroom to Home Office with the wrong item! Because the item is already in use he must either change item to another one, or wait until item will be no longer in use! The item in question isDoor to Home Office The item is currently used

by: Laura Braumer and will be in use for another: 1 time steps. was created in Bathroom

[14:03:56 INF] Agent Paul Braumer is invoked at time step 9

[14:03:58 INF] Event Paul Braumer is making following action: Clean up the Kitchen was created in Kitchen

[14:03:58 INF] Paul Braumer is using Sink for 1 time steps

[14:03:58 INF] Agent Laura Braumer is invoked at time step 9

[14:04:00 INF] Event Laura Braumer is moving to: Bathroom was created in Home Office

[14:04:00 INF] Laura Braumer moved to Bathroom

[14:04:00 INF] Agent Luisa Braumer is invoked at time step 9

[14:04:02 INF] Event Luisa Braumer is moving to: Home Office was created in Bathroom

[14:04:02 INF] Luisa Braumer moved to Home Office

[14:04:02 INF] Agent Paul Braumer is invoked at time step 10

[14:04:09 INF] Event Paul Braumer is making following action: Take a break and have a light snack was created in Kitchen

[14:04:09 INF] Paul Braumer is using Refrigerator for 1 time steps

[14:04:09 INF] Agent Laura Braumer is invoked at time step 10

[14:04:16 INF] Event Laura Braumer is moving to: Child Room was created in Bathroom

[14:04:16 INF] Laura Braumer moved to Child Room

[14:04:16 INF] Agent Luisa Braumer is invoked at time step 10

[14:04:22 INF] Event Luisa Braumer is moving to: Bathroom was created in Home Office

[14:04:22 INF] Luisa Braumer moved to Bathroom

[14:04:22 INF] Agent Paul Braumer is invoked at time step 11

[14:04:24 INF] Event Paul Braumer is making following action: Arrange the kitchen cabinets for better organization was created in Kitchen

[14:04:24 INF] Paul Braumer is using Cabinet for 1 time steps

[14:04:24 INF] Agent Laura Braumer is invoked at time step 11

[14:04:26 INF] Event Laura Braumer is moving to: Bathroom was created in Child Room

[14:04:26 INF] Laura Braumer moved to Bathroom

[14:04:26 INF] Agent Luisa Braumer is invoked at time step 11

[14:04:30 INF] Event Luisa Braumer is moving to: Home Office was created in Bathroom

[14:04:30 INF] Luisa Braumer moved to Home Office

[14:04:30 INF] Agent Paul Braumer is invoked at time step 12

[14:04:32 INF] Event Paul Braumer is making following action: Prepare a healthy lunch using the stove or another kitchen appliance was created in Kitchen

[14:04:32 INF] Paul Braumer is using Stove for 1 time steps

[14:04:32 INF] Agent Laura Braumer is invoked at time step 12

[14:04:34 INF] Event Laura Braumer is moving to: Child Room was created in Bathroom

[14:04:34 INF] Laura Braumer moved to Child Room

[14:04:34 INF] Agent Luisa Braumer is invoked at time step 12

[14:04:36 INF] Event Luisa Braumer is making following action: Work on a creative project was created in Home Office

[14:04:36 INF] Luisa Braumer is using Desk for 1 time steps

[14:04:36 INF] Agent Paul Braumer is invoked at time step 13

[14:04:38 INF] Event Paul Braumer is moving to: Bedroom was created in Kitchen

[14:04:38 INF] Paul Braumer moved to Bedroom

[14:04:38 INF] Agent Laura Braumer is invoked at time step 13

[14:04:41 INF] Event Laura Braumer is moving to: Bathroom was created in Child Room

[14:04:41 INF] Laura Braumer moved to Bathroom

[14:04:41 INF] Agent Luisa Braumer is invoked at time step 13

[14:04:43 INF] Event Luisa Braumer is moving to: Bathroom was created in Home Office

[14:04:43 INF] Luisa Braumer moved to Bathroom

[14:04:43 INF] Agent Paul Braumer is invoked at time step 14

[14:04:45 INF] Event Paul Braumer is moving to: Bathroom was created in Bedroom

[14:04:45 INF] Paul Braumer moved to Bathroom

[14:04:45 INF] Agent Laura Braumer is invoked at time step 14

[14:04:47 INF] Event Laura Braumer is moving to: Bedroom was created in Bathroom

[14:04:47 INF] Laura Braumer moved to Bedroom

[14:04:47 INF] Agent Luisa Braumer is invoked at time step 14

[14:04:48 ERR] Parse Plan failed, with error message:
System.NullReferenceException: Object reference not set to an instance of an object.