https://orcid.org/0009-0001-4959-0940
https://orcid.org/0000-0002-1208-3427
https://orcid.org/0000-0001-6268-3204
https://orcid.org/0000-0002-4840-0236
https://orcid.org/0009-0002-8400-6401

УДК 004.7

В.Ф. ГОЛОВСЬКИЙ[*], І.М. ОКСАНИЧ[*], В.Ф. ГРЕЧАНІНОВ[*], А.В. ЛОПУШАНСЬКИЙ[*], В.Ф. ГРЕЧАНІНОВ[*] (мол.)

МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ПОБУДОВИ ГРАФА ЗНАНЬ

[*]Інститут проблем математичних машин і систем НАН України, м. Київ, Україна

Анотація. *На теперішній час використання графів знань є потужним способом подання інформації у багатьох областях господарства, як-то картографія, планування будівництва, логістика, біологія, медицина та ін. Але побудова графа знань із неструктурованого тексту являє собою доволі непросте завдання. Воно потребує ідентифікації сутностей та їх взаємозв'язків, написання правил вилучення вручну чи використання спеціалізованих моделей машинного навчання. Водночас великі мовні моделі (LLM), які наразі дуже швидко розвиваються, можуть бути успішно використані для вирішення цієї задачі. Застосування LLM, підсилене одночасним використанням мультиагентних систем, має дати у підсумку суттєве підвищення стійкості й адаптивності графів знань. У роботі було досліджено та наведено приклади трансформації моделі реляційних відносин між таблицями у структурованій базі даних до моделі зв'язків між вузлами графа та приклади роботи і взаємодії агентів у мультиагентній архітектурі. Наведено перелік і особливості основних фреймворків для агентних LLM у 2025 році. Досліджено та створено модель мультиагентної системи побудови графа знань із використанням LLM. Описано роботу агентів структурованих і неструктурованих даних моделі. У моделі використовується технологія генерації, доповненої пошуком RAG (Retrieval-Augmented Generation). Агент GrafRAG відповідає за роботу з неструктурованим текстом.Показано створення планів побудови графа знань із структурованого і вилучення знань із неструктурованого текстів. Наведено приклади функцій завантаження вузлів і зв'язків між ними з CSV-файлів на мові Cyphery у базу даних Neo4j. Побудована діаграма вилучення сутностей і відношень із тексту при побудові графа знань. Робота може бути корисною при побудові мультиагентних систем графів знань, які використовують LLM,оскільки їх взаємна робота приводить до зменшення помилок і галюцинацій при відповідях на запити користувача.*

Ключові слова: *графи знань, LLM, мультиагентні системи, GrafRAG.*

Abstract. *Currently, the use of knowledge graphs is a powerful way to represent information in many areas of the economy, such as cartography, construction planning, logistics, biology, medicine, etc. However, building a knowledge graph from unstructured text is rather difficult. It requires identifying entities and their relationships, writing manual extraction rules, or using specialized machine learning models. At the same time, large language models (LLM), which are currently developing very rapidly, can be successfully used to solve this problem. The use of LLM, when reinforced by the simultaneous use of multi-agent systems, should result in a significant increase in the stability and adaptability of knowledge graphs. The paper investigates and provides examples of the transformation of the model of relational relationships between tables in a structured database to a model of connections between graph nodes and examples of the work and interaction of agents in a multi-agent architecture. The list and features of the main frameworks for agent-based LLMs in 2025 are presented. A model of a multi-agent knowledge graph*

ISSN 1028-9763. Математичні машини і системи. 2026. № 2

construction system using LLM is studied and created. The work of agents of structured and unstructured model data is described. The model uses the technology of generation supplemented by search — RAG (Retrieval-Augmented Generation). The GrafRAG agent is responsible for working with unstructured text. The creation of plans for building a knowledge graph from structured and extracting knowledge from unstructured texts is shown. Examples of functions for loading nodes and connections between them from CSV files in the Cypher language into the Neo4j database are given. A diagram of extracting entities and relations from text when building a knowledge graph is constructed. The work can be useful for building multi-agent knowledge graph systems that use LLM, since their mutual work leads to a reduction in errors and hallucinations when responding to user requests.
Keywords: *knowledge graphs, LLM, multi-agent systems, GrafRAG.*

DOI: 10.34121/1028-9763-2026-2-79-91

1. Вступ

На сучасному етапі розвитку інформаційних технологій створення та використання графів знань і онтологій відіграє ключову роль в організації та осмисленні складних наборів даних, які створюються щодня. Через обмежену можливість машинної інтерпретації неструктурованих даних цінні висновки часто виявляються недоступними для автоматичного аналізу. У цьому випадку графи знань є потужним способом подання інформації, зокрема неструктурованого тексту.

Традиційно побудова графа знань із необробленого тексту — непросте завдання. Воно потребує ідентифікації сутностей та їх взаємозв'язків, написання правил вилучення вручну чи використання спеціалізованих моделей машинного навчання для вирішення таких задач. Великі мовні моделі (LLM) дуже гнучкі та можуть бути використані для цієї мети. LLM можуть зчитувати текст у вільній формі та видавати структуровану інформацію, яку можна використовувати у складі автоматизованого конвеєра для створення графів знань.

LLM є відмінним інструментом для створення графів знань. Вони чудово справляються з розпізнаванням іменованих сутностей і встановлюють зв’язки між ними. Крім того, LLM можуть бути налаштовані на використання певної онтології для вилучення даних, що є дуже корисним при побудові графа знань [1].

Нова концепція розвитку інформаційних систем Software v3.0 [2] передбачає використання LLM як нових операційних систем у контексті їх організації та управління агентними підсистемами при обробці та зберіганні інформації. У середовищі Software v3.0 інформаційні потоки подаються у вигляді інформаційних повідомлень, які надалі трансформуються у набори багатовимірних векторів для ефективного зберігання та подальшої обробки даних. Використовуючи методи математичного та статистичного аналізу, можна знайти значущі ознаки та кореляційні залежності між багатовимірними векторами для подання інформації у зручному, для подальшого аналізу, вигляді (класифікації, кластеризації, оптимального пошуку, спорідненості).

На сьогоднішній день пропонується досить великий вибір мовних моделей як комерційних, так і рішень на основі Open Source, які можуть використовуватися в залежності від класу вирішуваних завдань і наявності доступних ресурсів автоматизації [3]. Наприклад, компанія IBM пропонує сімейство LLM загального призначення з фокусом на безпеку та стійкість [4].

Для зниження галюцинацій та підвищення точності відповідей LLM використовується технологія генерації, доповненої пошуком RAG (Retrieval-Augmented Generation) [5, 6]. RAG надає LLM актуальні зовнішні дані, дозволяючи використовувати закриті або свіжі бази знань, не перенавчаючи модель, що робить ШІ більш точним та надійним. Для отримання інформації, що стосується запиту користувача, зазвичай використовується векторна база даних із RAG. Потім ця одержана інформація надсилається

як контекстна інформація у LLM разом із запитом користувача. LLM використовує цю інформацію для складання графа знань.

Застосування технології мультиагентних систем [7, 8] до LLM відкриває для них нові можливості, дозволяючи безперервно обмінюватися свіжими даними та стратегіями, щоб підтримувати актуальність і ефективність вирішуваних ними задач. У мультиагентних LLM кожний агент виконує унікальну роль та використовує різні інструменти, як-то пошук в інтернеті чи обробка документів. Командна робота агентів дозволяє надавати кращі результати роботи LLM при вирішенні складних задач у реальних умовах.

Не дивлячись на значну кількість досліджень і напрацювань, побудова моделей ШІ з найменшою кількістю неточностей та галюцинацій є найбільш актуальною задачою на сьогодні.

Метою статті є дослідження та побудова моделі мультиагентної системи створення графа знань із використанням великих мовних моделей.

2. Граф знань

Граф знань (Knowledge Graph) — це семантична мережа, яка зберігає інформацію про різні сутності (наприклад, про людину, предмет, концепцію) та взаємозв'язки між ними. Він забезпечує об'єднання та структурування даних із різних джерел, що дає можливість зрозуміти зміст інформації. Сутності є вузлами графа, а зв'язки — це ребра, які відображають відносини між ними. Приклад графа знань наведений на рис. 1 [9].

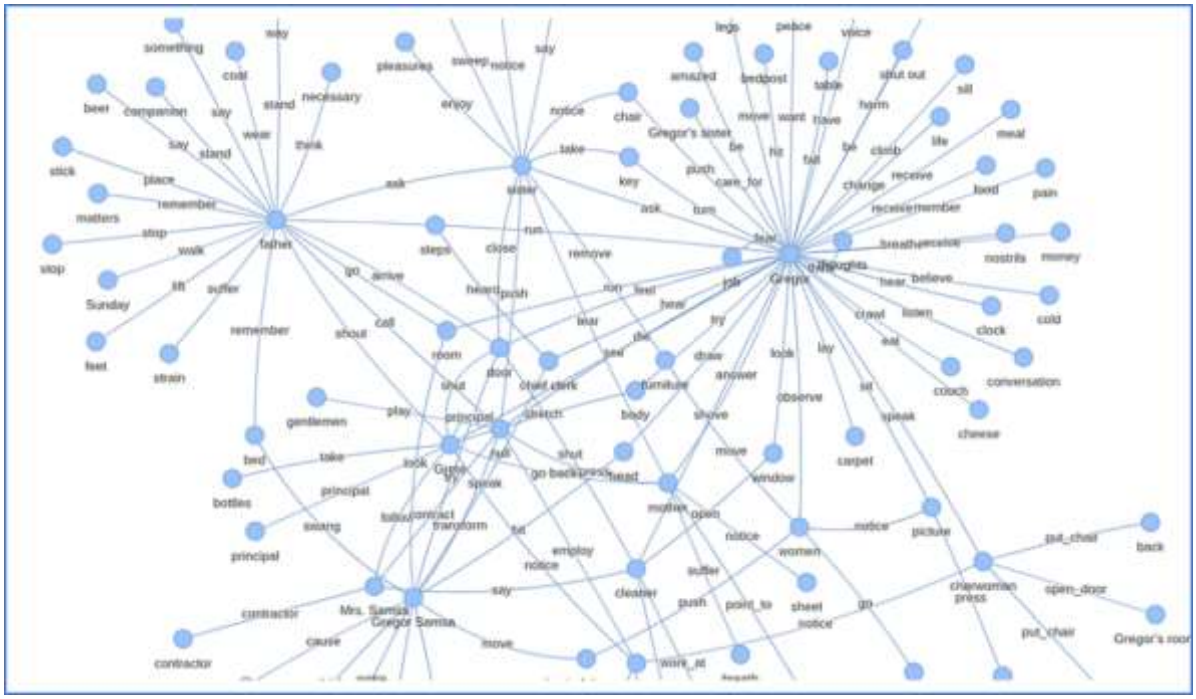

Рисунок 1 — Приклад графа знань

Граф знань може застосовуватись у багатьох сферах людської діяльності для аналізу складних взаємозв'язків, прискорення пошуку інформації та підвищення якості прийняття рішень. Наприклад, у охороні здоров'я, телекомунікаціях, наукових дослідженнях, юриспруденції, торгівлі тощо.

Вузли графа можуть містити ознаки (метрики), які отримують із різнорідних джерел великого обсягу інформації. Ручна побудова графа знань є досить трудомістким процесом. Тому використання мультиагентної підсистеми із застосуванням LLM може автоматизувати та полегшити цей процес.

Для зберігання та обробки даних, структурованих у вигляді графів, де дані представлені у вигляді вузлів, ребер та властивостей, використовуються графові бази даних. Прикладом такої бази є база даних Neo4j. Neo4j — це провідна високопродуктивна NoSQL СУБД, яка використовує графову модель даних (вузли та зв'язки) для зберігання інформації замість таблиць. Вона оптимізована для роботи зі складними взаємозв'язками та використовує декларативну мову запитів Cypher. База даних Neo4 ідеально підходить для аналізу графів, соціальних мереж тощо [10].

Для побудови графа знань використовуються як структуровані, так і неструктуровані дані з різних джерел інформації. Щоб перетворити структуровані та неструктуровані дані на граф знань, спочатку необхідно визначити схему графа, тобто визначити, які типи сутностей чи вузлів можна отримати з даних і який зв’язок між ними існує. Визначивши схему, можна побудувати сам граф і зберегти його у базі даних графів [11, 12].

Нижче наведено приклад трансформації моделі реляційних відносин між таблицями у структурованій базі даних до моделі зв'язків між вузлами мережевого графа (рис. 2, 3) [13].

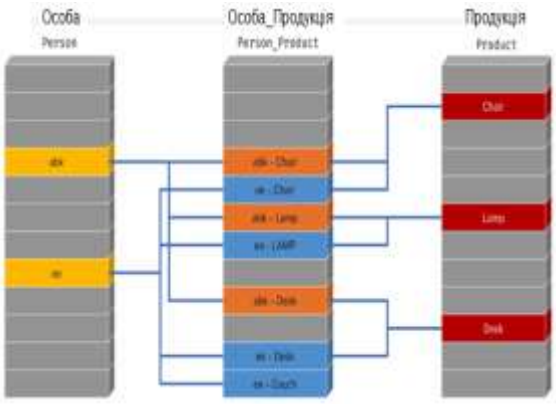

Рисунок 2 — Модель реляційних відносин між таблицями

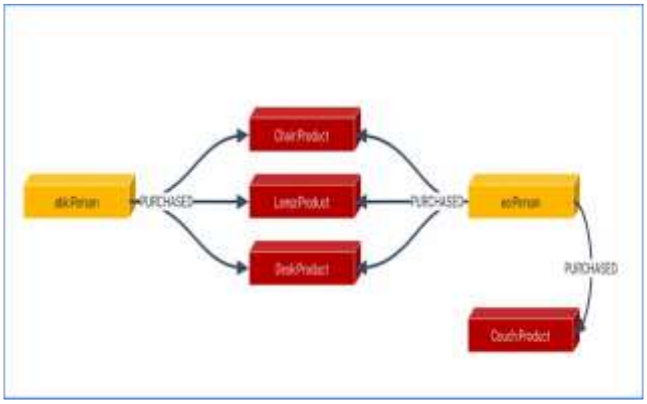

Рисунок 3 — Модель зв’язків між вузлами мережевого графа

Вузли в мережевому графі зберігають лише значущу інформацію з реляційних таблиць. Відносини між реляційними таблицями замінені ребрами спрямованих зв’язків кожного вузла мережевого графа. Отже, і у вузлів, і у зв’язків є властивості «ключ-значення», у них є кілька міток на вузлах. Зв’язки мають напрям і один тип.

Використання моделі зв'язків між вузлами мережевого графа дозволяє спростити процес пошуку та отримання інформації за шаблоном, наприклад, для рекомендаційних систем.

Ще одна цікава особливість графів знань полягає в тому, що вони дуже зручні для об’єднання неструктурованих даних із структурованими даними.

Вузли графа можуть бути доповнені додатковими ознаками даних, що витягуються з неструктурованих даних. Із використанням LLM з’являється можливість вилучення з неструктурованих даних сутностей та логічних зв’язків між ними для побудови чи зміни структури графа.

У складних системах граф знань предметної області може бути пов’язаний зі структурованими даними. З іншого боку, неструктуровані джерела даних будуть представлені лексичним графом, який є вихідними текстовими даними, а також структурою навколо цих текстових даних. Щоб з'єднати ці дві частини (неструктуровані та структуровані дані), може використовуватися додатковий підграф суб’єктів-об’єктів.

3. Модель мультиагентної системи побудови графа знань

3.1. Архітектура мультиагентної системи

Мультиагентні системи складаються з групи взаємодіючих агентів, які спільно вирішують прикладні завдання (визначені цілі) з можливістю розподілу рольових функцій та функціонального призначення.

Агенти — це програмні модулі, які можуть розглядатися як кінцеві автомати або спеціалізовані оператори потоку управління. У середині агента є цикл управління з можливістю зберігання проміжних даних (елемент пам’яті). У середині циклу відбувається виклик до LLM, який визначає подальші кроки дій у залежності від отриманих даних і передає потік управління оператору switch (перемикач), який виконує потенційні дії, використовуючи доступний для агента інструментарій. Інтелектуальна поведінка агента визначається викликами до LLM, а дії — через виклики доступного інструментарію. На рис. 4 і 5 зображено принцип роботи простого агента та взаємодії декількох агентів [13].

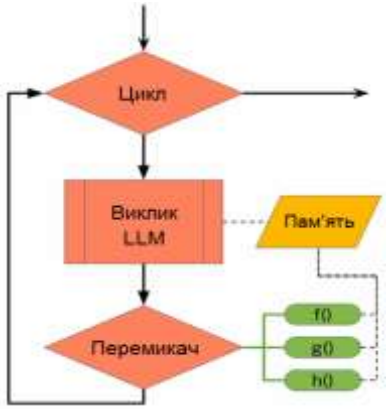

Рисунок 4 — Приклад роботи агента

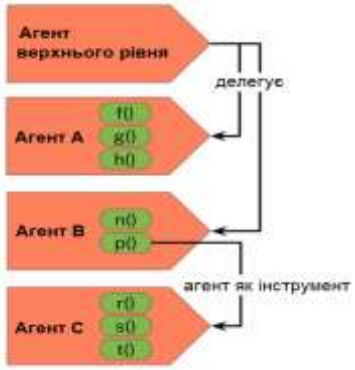

Рисунок 5 — Приклад взаємодії агентів

Перевага цієї архітектури в тому, що можливості LLM імітувати мислення та звертатися до інструментів, котрі можуть працювати з кодом, додають потужності агентам. Така архітектура адаптивна, тому що LLM може отримувати інформацію з пам'яті або з історії діалогу, чи зі збережених у пам'яті даних, які є важливими для прийняття на їх основі рішень у майбутньому.

Опис функціонального призначення агента, завдання для виконання роботи агентом можуть бути описані на природній мові у вигляді підказок (prompts).

У кожного агента повинні бути інструкції, призначені для того, щоб сам агент розумів, що він має робити (свою мету) і які інструменти у нього для цього є, і коли він їх може використовувати. Фактично, ми повинні сформувати середовище виконання для агентів.

Команда з кількох агентів, які можуть працювати разом, здатні делегувати завдання один одному, і кожен з цих агентів також має інструменти, які він може використовувати. Останній важливий компонент для всіх агентських систем — це пам’ять. Пам’ять — це внутрішній стан, який зберігається протягом конкретного сеансу роботи користувача.

Агенти зазвичай організовані у свого роду ієрархію, де є агент верхнього рівня, що управляє загальним процесом, та необмежена кількість агентів нижнього рівня, які відповідають за різні етапи роботи або виконують конкретні завдання.

Агенти взаємодіють кількома способами. Є основний потік розмови, який фактично відбувається з користувачем, де користувач може мати якесь повідомлення, з якого він починає роботу. А потім кожен з агентів повинен вирішити, чи може він виконати цю

роботу, чи це повинен зробити інший агент. Агенти можуть спеціалізуватися на виконанні завдань за допомогою доступних інструментів. Існують два способи взаємодії агентів: або агенти делегують завдання один одному, або агент, що діє як інструмент, викликається іншим агентом [14, 15].

На сьогоднішній день розроблено велику кількість різних програмних бібліотек і цілих фреймворків для проєктування, розробки та впровадження спеціалізованих агентів і агентних підсистем в об'єкти автоматизації. Найбільш перспективні та зрілі програмні бібліотеки та фреймворки для побудови агентських і мультиагентних підсистем, що взаємодіють із LLM у 2025 році, наведені у табл. 1 [16, 17].

Таблиця 1 — Основні фреймворки для агентних систем із LLM

Фреймворк	Особливості	Найкраще застосування
LangChain	Модульний, багато інтеграцій, гарна пам'ять та інструменти	Інтелектуальні чат-боти, складні LLM-додатки
LangGraph	Графова логіка виконання, керування станом	Складні багатокрокові та циклічні агент-системи
AutoGen	Інтерактивні мультиагенти за участю людини	Кооперація та складні робочі процеси
CrewAI	Командні агенти з розподілом ролей	Промислові мультиагентні системи
LlamaIndex	Робота із зовнішніми джерелами даних	Програми із зовнішніми базами знань
AutoGPT	Тривала пам'ять та контекст	Автономні агенти для багатозадачності
Semantic Kernel	Інтеграція LLM із сервісами	Корпоративні інтелектуальні агенти
Rasa	Розмовні агенти, розширення через LLM	Чат-боти та інтерактивні помічники

3.2. Модель високорівневого представлення мультиагентної системи графа знань (Knowledge Graph Agent)

На діаграмі рис. 6 показаний один із прикладів моделі високорівневого представлення мультиагентної системи побудови графа знань [13].

Агент графа знань

На верхньому рівні моделі (рис. 6) знаходиться Агент графа знань, який керує загальним уявленням користувача про можливості та допомагає користувачеві зрозуміти можливий робочий процес, починаючи з побудови графа знань та закінчуючи його вилученням. Це фактично діалоговий агент, який сам по собі не виконуватиме жодної роботи, але вестиме користувача через різні етапи роботи, які відбуватимуться.

Рівень агентів, розташований нижче, фактично керуватиме окремими робочими процесами.

Існують три основні робочі процеси, якими керують відповідні агенти:

1. Агент структурованих даних (Structured Data Agent).
2. Агент неструктурованих даних (Untructured Data Agent).
3. Агент GraphRAG відповідає за допомогу користувачеві в побудові графа знань.

Кожен з агентів структурованих даних і неструктурованих даних є агентами робочих процесів. Вони делегують завдання субагентам, що спеціалізуються на кожному етапі робочого процесу.

Рисунок 6 — Приклад моделі високорівневого представлення мультиагентної системи побудови графа знань

Агент структурованих даних (Structured Data Agent) — перший робочий процес, що може делегувати такі завдання субагентам:

1. Агент намірів користувача (User Intent Agent).
2. Агент пропозицій джерела (Source Suggestion Agent).
3. Агент пропозицій схеми (Schema Proposal Agent).

Агент намірів користувача (User Intent Agent) використовується для діалогової взаємодії з користувачем при формуванні цілей завдань і подальшого делегування виконання завдання. Він фіксує мету користувача.

Агент намірів користувача — агент розмовного зв'язку, який співпрацює з користувачем для визначення цільового імпорту даних зі структурованих джерел.

Для агента User Intent Agent визначається набір інструментів [tools]:

– set_perceived_user_goal — встановлює передбачувану мету користувача, включаючи вигляд графа та його опис;

– approve_perceived_user_goal — записує мету як схвалену після узгодження з користувачем.

Використання інструментів визначення мети користувача допомагає агенту зосередитися на вимогах. Замість відкритого тексту мета користувача визначається конкретними аргументами, що передаються інструменту. На підставі мети користувача наступний агент Sources Suggestion Agent переглядає доступні джерела структурованих даних.

Агент пропозицій джерела (Source Suggestion Agent) — це агент використання інструментів, який пропонує джерело для імпорту даних на основі затвердженої мети користувача з попереднього кроку. Вихідним результатом його роботи є список запропонованих джерел даних, схвалених користувачем. Потім ці дані передаються наступному агенту Schema Proposal Agent. Як джерела даних можуть розглядатися CSV-файли.

Для агента Source Suggestion Agent визначається набір інструментів [tools]:

– list_import_files — перелічує файли, доступні для побудови графа знань (усі файли з каталогу імпорту);

– sample_file — обирає файл, зчитуючи його вміст як текст;

– set_suggested_files — встановлює запропоновані файли для імпорту даних;

– approv_suggested_files — позначає запропоновані файли в статусі для подальшої обробки як схвалені.

Агент пропозицій схеми (Schema Proposal Agent) — один із пари агентів, сформованих за шаблоном критика, який відповідає за висування пропозиції про те, як може виглядати схема. Наступний із пари виступає в ролі критика. Ідея полягає в тому, що ця пара агентів внутрішньо перебиратиме можливі варіанти, проводитиме самокритику, і результатом їх роботи має бути хороша схема графа, яка використовує структуровані дані, схвалені попереднім агентом, а також відповідає меті користувача, встановленій першим агентом, створюючи схему графа, що може відповідати на запитання, корисні для користувача.

Декілька агентів можуть спільно пропонувати правила побудови графа знань. Процедура schema_refinement_loop на вищому рівні координуватиме роботу трьох агентів, що працюють у циклі:

1. Schema_proposal_agent пропонує схему та план побудови графа знань, використовуючи ітеративний метод уточнення схеми графа.

2. Schema_critic_agent критикує запропоновану схему та план побудови графа знань.

3. CheckStatusAndEscalate перевіряє зворотний зв’язок агента-критика.

Агент верхнього рівня керуватиме співпрацею з користувачем та координуватиме роботу субагентів, викликаючи їх як інструмент.

Результат цієї роботи агентів називається планом побудови графа (Grapth Construction Plan). Це не сам граф, а опис того, як його побудувати.

План побудови графа (Grapth Construction Plan) містить:

– результат робочого процесу зі структурованими даними;

– затверджені правила побудови для перетворення вихідних даних на граф.

Агент неструктурованих даних (Untructured Data Agent) — другий робочий процес, перші два кроки якого повністю збігаються з робочим процесом для структурованих даних (Structured Data Agent). Але третій крок робочого процесу відрізняється — замість розробки схеми на основі структурованих джерел потрібно визначити, які типи сутностей і фактів мають бути витягнутими з неструктурованих даних (тексту).

При роботі з неструктурованими даними треба розуміти, навіщо користувачеві потрібні неструктуровані дані, а саме файли розмітки. Після цього можна визначити, які файли йому підходять. Таку процедуру робить агент підказки файлів File Suggestion Agent аналогічно до роботи зі структурованими даними.

Агент пропозиції типу сутності і факту (Entity & Fact Type Agent) може делегувати такі завдання субагентам:

– агент використання інструменту;

– агент співпраці з користувачем для визначення типу сутностей та відповідних фактів, які можна витягти з файлів розмітки (markdown).

Мета агента Entity & Fact Type Agent полягає в тому, щоб з'ясувати, які типи фактів можна витягти і їх описати. Цей опис називається планом вилучення знань (Knowledge Extraction Plan).

План вилучення знань (Knowledge Extraction Plan) містить:

– вихід із робочого процесу неструктурованих даних;

– затверджені типи сутностей;

– затверджені факти про сутності.

Агент GraphRAG

На основі неструктурованого тексту будується граф. Агент GraphRAG переробляє неструктурований текст у структуровану базу даних, витягуючи сутності (вузли) та відносини між ними (ребра) за допомогою LLM. Він поєднує їх у ієрархічні спільноти, створюючи семантичну мережу, яка дозволяє знаходити взаємозв'язки, зберігати контекст та зменшувати галюцинації (генерування LLM хибної, вигаданої або безглуздої інформації, яка виглядає достовірною).

До інструментів побудови графа знань у даному випадку можна віднести базу даних Neo4j, велику мовну модель LLM та конструктори типу Knowledge Graph Builder.

Побудова графа знань (Knowledge Graph)

Після того як всі плани складено, можна побудувати граф знань (Knowledge Graph). Для побудови графа агент не потрібен. План побудови містить всю інформацію, необхідну для здійснення імпорту з урахуванням правил.

Все, що необхідно буде зробити, це обробити всі правила побудови вузлів і всі правила побудови відносин. Основна увага приділяється розумінню загальної картини перетворення структурованих даних у структуру графа з урахуванням плану побудови.

Запити Cypher дозволяють виконати побудову графа предметної області із CSV-файлів. Cypher — це декларативна, візуально-інтуїтивна мова запитів, спеціально створена для ефективної роботи з графовими структурами даних, де відносини між сутностями мають ключове значення. Cypher використовує оператори MATCH для пошуку шаблонів у графі замість SELECT та JOIN, як у SQL.

Нижче наведена спеціально розроблена функція load_nodes_from_csv, яка виконує пакетне завантаження вузлів з CSV-файлу в базу даних Neo4j. Вона використовує команду LOAD CSV із операцією MERGE для створення вузлів, уникаючи дублікатів на основі унікального стовпця. Запит Cypher обробляє дані пакетами по 1000 рядків для кращої продуктивності.

```
def load_nodes_from_csv(
    source_file: str,
    label: str,
    unique_column_name: str,
    properties: list[str],
) -> Dict[str, Any]:
    """Batch loading of nodes from a CSV file"""
    # load nodes from CSV file by merging on the unique_column_name value
    query = f"""LOAD CSV WITH HEADERS FROM "file:///" + $source_file AS row
    CALL (row) {{
        MERGE (n:$($label) {{ {unique_column_name} : row[$unique_column_name] }})
        FOREACH (k IN $properties | SET n[k] = row[k])
    }} IN TRANSACTIONS OF 1000 ROWS
    """
    results = graphdb.send_query(query, {
        "source_file": source_file,
        "label": label,
        "unique_column_name": unique_column_name,
        "properties": properties
    })
    return results.
```

Інша функція **i**mport_relationships імпортує зв’язки між вузлами з CSV-файлу. Вона використовує запит Cypher, який знаходить і зіставляє пари існуючих вузлів та створює зв’язки з заданими властивостями між ними.

```
def import_relationships(relationship_construction: dict) -> Dict[str, Any]:
   """Import relationships as defined by a relationship construction rule."""
   # load nodes from CSV file by merging on the unique_column_name value
   from_node_column = relationship_construction["from_node_column"]
   to_node_column = relationship_construction["to_node_column"]
   query = f"""LOAD CSV WITH HEADERS FROM "file:///" + $source_file AS row
   CALL (row) {{
      MATCH (from_node:$($from_node_label) {{ {from_node_column} : row[$from_node_column]
}}),
          (to_node:$($to_node_label) {{ {to_node_column} : row[$to_node_column] }} )
      MERGE (from_node)-[r:$($relationship_type)]->(to_node)
      FOREACH (k IN $properties | SET r[k] = row[k])
   }} IN TRANSACTIONS OF 1000 ROWS
   """
  results = graphdb.send_query(query, {
      "source_file": relationship_construction["source_file"],
      "from_node_label": relationship_construction["from_node_label"],
      "from_node_column": relationship_construction["from_node_column"],
      "to_node_label": relationship_construction["to_node_label"],
      "to_node_column": relationship_construction["to_node_column"],
      "relationship_type": relationship_construction["relationship_type"],
      "properties": relationship_construction["properties"]
   })
   return results.
```

Інструмент Knowledge Graph Construction Tool (конструктор графа знань) може сам пройти цикл по всіх правилах побудови, щоб створити граф домену.

Knowledge Graph Construction Tool:

– виконує побудову графа та вилучення знань;

– перебирає правила побудови для створення графа домену;

– перебирає файли markdown для зчитування, а потім вилучає сутності та факти;

– з’єднує вилучені сутності з визначеними сутностями домену.

Рисунок 7 — Діаграма вилучення сутностей і відношень

Конструктор графа знань витягує сутності та зв’язки з текстових фрагментів і зберігає їх у базі даних Neo4j у вигляді графа. Для обробки фрагментів та вилучення

сутностей зі зв’язками потрібно створити кілька допоміжних функцій для завантаження, фрагментації даних та вилучення сутностей [18, 19].

Наприклад, у бібліотеці Neo4j GraphRAG є зручний інструмент SimpleKGPipeline, який можна використовувати для обробки фрагментів та вилучення сутностей зі зв’язками. На рис. 7 наведено приклад діаграми вилучення сутностей і відношень.

4. Приклад побудови графа знань

Нижче наведено приклад функції **make_kg_builder**, яка створює налаштований конвеєр для певного файлу, витягуючи контекст файлу та створюючи контекстуалізований запит (prompt) на вилучення. Вона дозволяє отримувати сутності та взаємозв’язки з текстових фрагментів і зберігати їх у базі даних Neo4j у вигляді графа сутностей або об’єктів.

```
def make_kg_builder(file_path:str) -> SimpleKGPipeline:
    """Builds a KG builder for a given file, which is used to contextualize the chunking and entity
extraction."""
    context = file_context(file_path)
    contextualized_prompt = contextualize_er_extraction_prompt(context)

    return SimpleKGPipeline(
        llm=llm_for_neo4j, # the LLM to use for Entity and Relation extraction
        driver=neo4j_driver,  # a neo4j driver to write results to graph
        embedder=embedder,  # an Embedder for chunks
        from_pdf=True,   # sortof True because you will use a custom loader
        pdf_loader=MarkdownDataLoader(), # the custom loader for Markdown
        text_splitter=RegexTextSplitter("---"), # the splitter you defined above
        schema=entity_schema, # that you just defined above
        prompt_template=contextualized_prompt,
    )
from helper import get_neo4j_import_dir

neo4j_import_dir = get_neo4j_import_dir() or "."

for file_name in approved_files:
    file_path = os.path.join(neo4j_import_dir, file_name)
    print(f"Processing file: {file_name}")
    kg_builder = make_kg_builder(file_path)
    results = await kg_builder.run_async(file_path=str(file_path))
    print("\tResults:", results.result)
print("All files processed.")
```

5. Висновки

Сьогодні використання графів знань є потужним способом подання інформації у багатьох областях. За допомогою графів знань можна створити процеси автоматичного вилучення даних із безлічі джерел та збору інформації про об’єкти, що цікавлять користувача в будь-якій області (про людей, товари, місця. події тощо), і встановити зв’язки між ними. Прикладами областей використання графів знань є картографія, планування будівництва, логістика, біологія, медицина тощо.

Розвиток LLM спонукав розширення використання і розвиток графів знань.За допомогою LLM будують графи знань, а за допомогою графів знань покращують роботу LLM. Графи знань дозволяють LLM мислити, надаючи точну інформацію. Вони організують дані як мережу, імітуючи людське уявлення про зв'язки між ними, що

допомагає LLM краще розуміти поставлену задачу. Графи знань допомагають подолати галюцинації LLM, забезпечуючи структуру прозорості та масштабованості.

Використання технології GraphRAG додає гнучкості графам знань, допомагаючи використовувати семантичний пошук, пошук за ключовими словами або текстом, геопросторові фільтри тощо з різних баз неструктурованих даних (PDF-файли, текст, вебсторінки тощо), що покращує точність відповідей на запити користувача.

Використання мультиагентних систем у поєднанні з графами знань забезпечує суттєве підвищення їх стійкості і адаптивності при їх створенні і роботі в умовах зростаючої складності бізнес-процесів.

Кожен з агентів мультиагентної системи відповідає за окреме завдання та узгоджено діє з іншими агентами у процесах обміну даними, розподіленні завдань, перевірки та оптимізації рішень один одного. У міру розвитку цих систем та поєднання їх роботи з великими мовними моделями вони продовжуватимуть підвищувати ефективність роботи графів знань, розширювати їх можливості та надавати семантичний шар, який робить дані більш доступними та придатними для практичного застосування.

Майбутнє графів знань, імовірно, буде зосереджено на їх інтеграції з більш широкими екосистемами даних, поліпшенні масштабованості та вдосконаленні інструментів для додатків реального часу. Можна сказати, що графи знань лежать в основі майбутнього даних.

СПИСОК ДЖЕРЕЛ

1. SAP. Що таке велика мовна модель (LLM) — SAP. URL: https://www.sap.com/ukraine/resources/what-is-large-language-model.
2. Software in the Age of AI. URL: https://www.latent.space/p/s3.
3. The platform where the machine learning community collaborates on models, datasets, and applications. URL: https://huggingface.co/models.
4. IBM’s Granite foundation models are targeted for business. URL: https://www.ibm.com/think/news/granite-foundation-models.
5. What is knowledge graph — IBM. URL: https://www.ibm.com/think/topics/knowledge-graph.
6. What is knowledge graph — neo4j. URL: https://neo4j.com/blog/knowledge-graph/what-is-knowledge-graph/.
7. Що таке мультиагентні системи — SAP. URL: https://www.sap.com/ukraine/resources/what-are-multi-agent-systems.
8. Мультиагентні системи — MAS — InterNetri. URL: https://internetri.net/qntm/2025/09/23/multyagentni-systemy-mas/.
9. How to Build Knowledge Graphs using LLMs on Local Machine. URL: https://medium.com/the-muse-junction/how-to-build-knowledge-graphs-using-llms-on-local-machines-5926261c04eb.
10. Neo4j Graph Database. URL: https://neo4j.com/docs.
11. Building a Knowledge Graph From Scratch Using LLMs. URL: https://towardsdatascience.com/building-a-knowledge-graph-from-scratch-using-llms-f6f677a17f07/.
12. Building Knowledge Graphs Using Large Language Models. URL: https://medium.com/@shuchawl/building-knowledge-graphs-using-large-language-models-07da1935b21a.
13. neo4j-contrib/agentic-kg. URL: https://github.com/neo4j-contrib/agentic-kg.
14.Multi-agent LLMs in 2025 [+frameworks]. URL: https://www.superannotate.com/blog/multi-agent-llms.
15. Choosing the Right LLM Agent Framework in 2025. URL: https://botpress.com/blog/llm-agent-framework.
16. Top 5 multi-agent frameworks for Building Automation Systems. URL: https://www.zams.com/blog/multi-agent-frameworks.
17. Top 9 AI Agent Frameworks as of September 2025. URL: https://www.shakudo.io/blog/top-9-ai-agent-frameworks.

18. Optimizing the Interface Between Knowledge Graphs and LLMs for Complex Reasoning. URL: https://arxiv.org/html/2505.24478v1.
19. Automate building knowledge graphs with LLMs and Neo4J. URL: https://blog.gopenai.com/automate-building-knowledge-graphs-from-scientific-abstracts-with-llms-and-neo4j-2-ways-b9d5152a12da.

Стаття надійшла до редакції 30.01.2026 / прийнята до друку 28.04.2026