УДК 004.83:004.89 DOI: 10.31673/2786-8362.2025.012843

Олійник Б.О.; Чичкарьов Є.А., д.т.н.

МЕТОДИ РЕАЛІЗАЦІЇ RETRIEVAL-AUGMENTED GENERATION У ПОЄДНАННІ З СУЧАСНИМИ ВЕЛИКИМИ МОВНИМИ МОДЕЛЯМИ

Oliinyk B.O., Chychkarov Y.A. Methods of implementing retrieval-augmented generation in combination with modern large language models. The article presents a comprehensive study of state-of-the-art Retrieval-Augmented Generation (RAG) methods integrated with large language models such as GPT-4 and open GPT-4-equivalent models (GPT-4o). We analyze experimental results from 2023-2025 (including open sources like Arxiv, HuggingFace, PapersWithCode) that demonstrate the advantages of new approaches: RAG 2.0 with joint end-to-end training of components, adaptive dynamic retrieval, generative prompts for retrievers, self-refining pipelines with feedback, and modular architectures for API-based models. We explain how these innovations overcome the shortcomings of traditional RAG by reducing hallucination rates and improving answer faithfulness. The mechanisms of the models are described, along with system workflow diagrams and a comparative table of metrics (Hallucination Rate, Answer Faithfulness, Retrieval Precision). We substantiate the potential of advanced RAG in information systems, chatbots, and analytics platforms. A clear vision for further development is formulated – integrating RAG with continuously updated knowledge bases and real-time search APIs to ensure up-to-date responses.

Keywords: large language models, retrieval-augmented generation, RAG 2.0, dynamic retriever, self-reflection, hallucinations, factual accuracy

Олійник Б.О., Чичкарьов Є.А. Методи реалізації Retrieval-Augmented Generation у поєднанні з сучасними великими мовними моделями. Розглянуто сучасні методи генерації з доповненням пошуком (Retrieval-Augmented Generation, RAG) у контексті інтеграції з великими мовними моделями GPT-4 та відкритими моделями рівня GPT-4 (GPT-4o). Проаналізовано експериментальні результати 2023-2025 років із відкритих джерел (Arxiv, HuggingFace, PapersWithCode), які демонструють переваги нових підходів: RAG 2.0 із спільним навчанням компонентів, адаптивний динамічний пошук, генеративні підказки для ретривера, self-refine pipeline'и із зворотним зв'язком, модульну архітектуру для інтеграції API-моделей. Показано, як ці інновації долають недоліки традиційного RAG – знижують рівень «галюцинацій» та підвищують фактичну точність відповідей. Наведено опис механізмів моделей, структурні схеми роботи системи, табличне порівняння метрик (рівень галюцинацій, достовірність відповіді, прецизійність пошуку). Обґрунтовано потенціал використання вдосконаленого RAG у інформаційних системах, чат-ботах і аналітичних платформах. Сформульовано перспективну ідею подальшого розвитку – інтеграція RAG з динамічно оновлюваними базами знань та пошуковими API для підвищення актуальності відповідей.

Ключові слова: великі мовні моделі, генерація з доповненням пошуком, RAG 2.0, динамічний ретривер, self-reflection, галюцинації LLM, фактична достовірність

Вступ

Генерація тексту на основі великих мовних моделей (ВММ) досягла визначних успіхів, але залишається проблема фактичної точності – моделі можуть генерувати впевнені, проте недостовірні відповіді (так звані «галюцинації»). Підхід генерації з доповненням пошуком (Retrieval-Augmented Generation, RAG) був запропонований як рішення для зниження таких помилок за рахунок надання моделі актуальної зовнішньої інформації. RAG поєднує дві складові: модуль пошуку (ретривер), який вибирає релевантні документи з бази знань, та модуль генерації (мовна модель), що формує відповідь на основі як власних параметричних знань, так і отриманих зовнішніх даних. Така архітектура дозволяє системі відповідати на запити, спираючись на свіжі та доменно-специфічні дані, зменшуючи кількість галюцинацій і підвищуючи довіру користувачів.

Протягом 2023-2025 рр. з появою потужних моделей на кшталт GPT-4 роль RAG суттєво зросла. У корпоративних застосуваннях RAG став де-факто стандартом для забезпечення «приземлення» знань ВММ на актуальні дані компаній та документи, а також для збереження приватності даних. Наприклад, Bing Chat від Microsoft інтегрує пошук веб-сторінок у процес генерації відповіді, а система GPT-4-консультант Morgan Stanley здійснює доповнення знань

з внутрішньої бази даних компанії [1]. RAG використовується у чатботах, клієнтських довідниках, пошукових платформах для надання точнішої та обґрунтованої інформації користувачам. Для бізнесу це означає можливість створити інтерактивні системи (консультанти, аналітичні помічники), що завжди оновлюються в реальному часі, зберігають конфіденційність даних та мінімізують вигадки моделі [2].

Однак, традиційна реалізація RAG має обмеження: ретривер та генератор працюють як окремі «чорні скриньки», які не оптимізовані спільно, через що можливі невідповідності (наприклад, ретривер може повернути доречні, але не зовсім потрібні фрагменти, або генератор може ігнорувати релевантні докази) [1]. Крім того, базовий підхід зазвичай передбачає фіксовану кількість документів для підстановки, незалежно від потреби, що може призводити до зайвого «шуму» в контексті або, навпаки, нестачі даних [3]. У зв'язку з цим з'явилися новітні методи, які удосконалюють RAG: спільне навчання ретривера і генератора (т.зв. RAG 2.0), адаптивний або «активний» пошук, що вирішує коли і що саме витягати при генерації, генеративні налаштування для самих пошукових моделей (щоб ретривер розумів інструкції мовою людини), а також цикли самоперевірки і самоудосконалення відповіді з додатковим зворотним зв'язком.

Аналіз останніх досліджень. Метод Retrieval-Augmented Generation було вперше представлено Льюїсом М. і співавторами у 2020 році для задач, що потребують знань (open-domain QA) [4]. У початковій реалізації RAG використовували двоетапну модель: dense-ретривер (наприклад, DPR) для знаходження даних з Wikipedia та генератор (на базі BART), що формував відповіді з використанням цих даних [5]. Цей підхід показав, що навіть відносно невеликі моделі можуть досягати високої точності, якщо надавати їм зовнішні знання у потрібний момент, замість зберігання всіх фактів у параметрах моделі.

Подальші роботи з 2020–2022 рр. удосконалювали RAG. Зокрема, метод Fusion-in-Decoder запропонував об'єднувати кілька документів у вході декодера для кращої інтеграції доказів, а проєкти REALM та Retro досліджували спільне тренування читача і ретривера або використання ретровставки фрагментів знань під час генерації. Одним із напрямів було зменшення залежності генератора від застарілих чи нерелевантних даних. Так, Shi та інші, у роботі RePlug показали, що можна розглядати мовну модель як чорний ящик і донавчити ретривер на основі зворотного зв'язку від самої мовної моделі – зокрема, використовувати перплексію моделі для вибору документів [5]. Такий підхід покращив релевантність витягнутих знань і ілюструє загальну тенденцію спільної оптимізації компонентів RAG ще до появи терміну "RAG 2.0".

Водночас, розвиток самих ВММ (GPT-3.5, GPT-4, Claude та ін.) привернув увагу до альтернативи: просто збільшити довжину контексту моделі і подавати їй величезні обсяги тексту без явного ретривера. З'явилися моделі з контекстом 32k, 100k і більше токенів (наприклад, Claude-100k, Gemini-1M). Деякі огляди заявляли про можливий «кінець ери RAG». Проте експерти відзначають, що такий підхід має серйозні недоліки – екстремально високу вартість та складність обробки настільки великих контекстів, а приріст якості поступово зменшується [1]. Практичний аналіз показав: використовувати модель з вікном у 1 млн токенів часто економічно недоцільно, і лише небагатьом задачам справді потрібно настільки багато контексту [4]. Натомість RAG забезпечує більш гнучку та модульну архітектуру, де базову ВММ можна замінити або оновити, а знання переіндексувати без повного перевчитування моделі [1]. Таким чином, у спільноті з`явилася думка, що великі контексти не витіснять RAG, а радше доповнять: гібридні рішення, де ретривер подає стислий контекст у велике вікно, поєднують плюси обох підходів [5].

З огляду на це, протягом останніх двох років дослідники зосередилися на покращенні самого RAG. У 2024 р. введено поняття "RAG 2.0", що означає перехід від простого "склеювання" окремих моделей до спільно оптимізованих, надійних RAG-пайплайнів для промислового використання. Також з'явилися варіації RAG для спеціальних цілей: Graph RAG (ретривер працює з графом знань замість тексту), Retrieval-Augmented Reasoning (RAR) – ін'єкція знань на кожному кроці багатокрокового міркування [5], а також різні методи

активного або адаптивного пошуку. Окремо варто відзначити підхід Self-Reflective RAG (2023), який додає до зв'язки «пошук–генерація» ще компонент самоаналізу відповіді моделлю [2]. Навчання у цьому випадку відбувається так, що сама мовна модель вчиться приймати рішення: коли виконати пошук, які дані долучити, та як перевірити власну відповідь на узгодженість з знайденими даними. У результаті, Self-RAG на 7–13B моделях продемонстрував результати кращі, ніж навіть ChatGPT з RAG, у задачах питань-відповідей і верифікації фактів [2].

Отже, в науковій літературі спостерігається перехід від базових рішень RAG до більш інтегрованих та розумних систем, які активно керують процесом пошуку інформації і генерування відповіді.

Метою роботи є аналіз та узагальнення новітніх методів реалізації RAG у поєднанні з великими мовними моделями (на прикладі GPT-4 та його відкритих аналогів), а також визначення того, як ці методи вирішують проблеми традиційного підходу. Для досягнення мети необхідно: 1) розглянути концепцію RAG 2.0 та результати її впровадження; 2) дослідити методи динамічного налаштування пошуку і генеративних підказок для ретривера; 3) проаналізувати pipeline'и із самостійним доопрацюванням відповіді (self-refinement) на основі зворотного зв'язку; 4) оцінити модульні архітектури RAG для використання зовнішніх API-моделей; 5) порівняти експериментальні показники (рівень галюцинацій, достовірність відповідей, точність ретривера) різних підходів; 6) окреслити перспективи застосування удосконаленого RAG та запропонувати напрями подальших розробок (зокрема, інтеграцію з оновлюваними базами знань, пошуковими API реального часу тощо).

Виклад основного матеріалу дослідження

Однією з головних інновацій є RAG 2.0 – підхід, при якому вся система RAG розглядається як єдиний ансамбль, що підлягає спільному тренуванню і тонкому налаштуванню. Стартап Contextual AI запропонував термін "RAG 2.0" для підкреслення переходу від використання окремих «готових» компонентів до об'єднаної моделі, тренованої end-to-end [5]. Замість того, щоб брати наперед навчений ретривер, фіксовану базу векторів і окрему LLM, RAG 2.0 передбачає спільне навчання всіх частин на цільовому корпусі даних, іноді навіть з донавчанням самої мовної моделі на те, як краще використовувати отримані документи. Зокрема, ретривер навчається від помилок генератора: якщо останній згенерував неточну відповідь, система коригує ретривер, щоб наступного разу добирати більш корисні дані [6]. Така координація усуває проблему невідповідності між компонентами, притаманну класичному RAG. За даними Contextual AI, їх спеціалізовані Contextual LLMs, натреновані за схемою RAG 2.0, продемонстрували суттєво вищу точність та правдивість відповідей, ніж стандартні пайплайни RAG. Зокрема, ранні результати свідчать про значне зниження частоти галюцинацій та покращення фактичної відповідності відповідей джерелам. Це досягається ціною складнішого процесу навчання і необхідності мати доступ до повноти даних для тренування. Проте для багатьох корпоративних застосувань такий компроміс прийнятний: якщо потрібно максимізувати точність у конкретному домені, підприємства готові інвестувати в навчання власних RAG-моделей, замість використання лише «чорного ящика» API. Таким чином, RAG 2.0 задає новий стандарт – створювати спеціалізовані моделі з доступом до знань, де межа між параметричною пам'яттю ВММ і зовнішньою базою знань стає розмитою і керованою в єдиному процесі.

Динамічний пошук та *Active RAG.* Іншим напрямом розвитку є зробити процес пошуку в RAG адаптивним та вибірковим. У традиційному RAG на кожний запит зазвичай витягається фіксоване число документів (наприклад, топ-5), незалежно від того, чи потрібні всі вони для відповіді. Це може призводити до зайвої інформації, яка перевантажує модель, або навпаки – модель даремно не звертається до пошуку, коли це потрібно. Активне RAG (Active Retrieval-Augmented Generation) пропонує, щоб сама модель вирішувала, коли саме звернутися до ретривера і які дані запитати, у ході генерації відповіді. Такий підхід реалізовано, зокрема, у фреймворку FLARE (Forward-Looking Active RAG). Ідея полягає в тому, що під час побудови

відповіді LLM може робити проміжні кроки: формулювати уточнюючі підзапити, виконувати кілька раундів пошуку або взагалі пропускати пошук, якщо впевнена у своїх знаннях. Це схоже на розумові міркування: «подумати перед тим, як шукати». Алгоритмічно, активний підхід може бути реалізований через спеціальні токени чи інструкції, які сигналізують необхідність звернення до бази знань. Self-RAG є прикладом такої інтеграції: модель сама вставляє запити на пошук у процес генерації. Інший варіант – пост-тренувати політику вибору, яка на основі стану діалогу вирішує, чи потрібен зараз пошук (подібно до механізму «CoT + пошук»). Практичні дослідження показали, що Active RAG здатний скоротити кількість непотрібних викликів пошуку і водночас не пропускає важливих – це економить час і обчислення. Компанія K2View у своєму огляді відзначає, що активне налаштування ретривера з зворотним зв'язком від генератора по ходу діалогу дозволяє пришвидшити і покращити відповіді системи [7]. Зокрема, поступове вдосконалення ретривера за декілька ітерацій діалогу (через fine-tune на історії взаємодій) знижує відсоток помилок пошуку.

Ще один аспект динамічного пошуку – обмеження кількості і релевантності документів. Якщо модель вирішує, що достатньо 2 джерел, вона не братиме 5, що покращує прецизійність витягування (менше зайвої інформації). Наприклад, підхід *Retrieve-Only-When-Necessary* привів до скорочення непотрібних звернень і зниження сукупного часу на ~30–50% без втрати точності [3]. Таким чином, динамічний пошук робить RAG більш гнучким: pipeline перетворюється з фіксованого двокрокового процесу на умовно багатоетапний, де LLM і ретривер взаємодіють інтерактивно.

Генеративні підказки для ретривера (Prompt-based Retrieval). Традиційні моделі ретривера (наприклад, bi-encoder, DPR) обмежені статичним навчанням: вони отримують на вході запит і повертають близькі за семантикою документи. Новий тренд – «інструкційні» або промптові ретривери, які можна налаштовувати під конкретні завдання за допомогою текстових підказок, аналогічно до LLM. Ідея в тому, щоб спілкуватися з ретривером природною мовою, задаючи критерії релевантності або контекст. Проєкт Promptriever продемонстрував перший ретривер, який вміє розуміти довільні інструкції і відповідно змінювати стратегію пошуку [7]. Наприклад, можна перед запуском пошуку «підказати» ретриверу: *"Зверни увагу на хронологію подій"* або *"Спочатку знайди визначення терміну"* – і модель врахує це при відборі документів. Promptriever було навчено на спеціально згенерованому корпусі запитів з інструкціями; результати вражають: якість пошуку (MRR) зросла на 14.3 пункту, а nDCG/MAP – на 3.1 на тестовому наборі FollowIR, досягнувши рівня найкращих cross-encoder моделей [7]. Також відзначено підвищення робастності до формулювань: чутливість до довжини і перефразування запиту зменшилася ~на 44% (варіативність результатів на BEIR). Це означає, що інструкційно-навчений ретривер більш стабільно повертає потрібні документи незалежно від того, як користувач сформулював питання. По суті, такий ретривер розглядається як ще одна велика модель, яку можна *prompt*'ити.

Практичні переваги: генеративні підказки для пошуку дозволяють об'єднати етапи налаштування під задачу з самим процесом пошуку. Без необхідності донавчати ретривер, можна написати йому кілька прикладів або вказівок – і отримати кращі результати zero-shot. Це особливо корисно, коли система повинна динамічно змінювати критерії пошуку під час роботи. Наприклад, у діалоговому агенті можна вбудувати правило: якщо користувач уточнює запит ("поясни простіше"), дати ретриверу підказку *"шукати матеріали для початківців"*. Такий підхід також спрощує впровадження RAG для нових доменів: достатньо описати критерії релевантності, не збираючи великий датасет для навчання з нуля. Отже, генеративно-промптові ретривери є кроком до універсальних механізмів пошуку, які легко пристосовувати під різні задачі так само, як ми пристосовуємо LLM під нові інструкції.

Self-refinement: генерація із зворотним зв'язком (Self-RAG). Навіть після додавання ретривера великі моделі можуть допускати фактичні помилки – наприклад, неправильно інтерпретувати знайдений текст або «вигадати» деталь, якої немає у джерелах. Тому з'явилися методи, що додають цикли зворотного зв'язку у процес генерації. Концепція Self-Reflective

RAG (Self-RAG) полягає в тому, що модель не тільки генерує відповідь, а й додатково аналізує власну відповідь та перевіряє її на відповідність витягнутим даним. Технічно це реалізовано введенням спеціальних *reflection tokens* та інструкцій: модель спочатку видає проміжний висновок про те, чи достатньо їй знань (рис. 1), потім може виконати ще один запит до ретривера, якщо потрібно, і в кінці – критично оцінює сформовану відповідь. Уявімо, що LLM відповіла на запит і додала посилання на 3 документи. Self-RAG змусить модель проглянути ці документи та свою відповідь і перевірити: чи всі наведені факти підтверджуються? Якщо модель виявляє невідповідність (наприклад, одна частина відповіді не спирається ні на один документ), вона може виправити відповідь або запросити додатковий пошук.

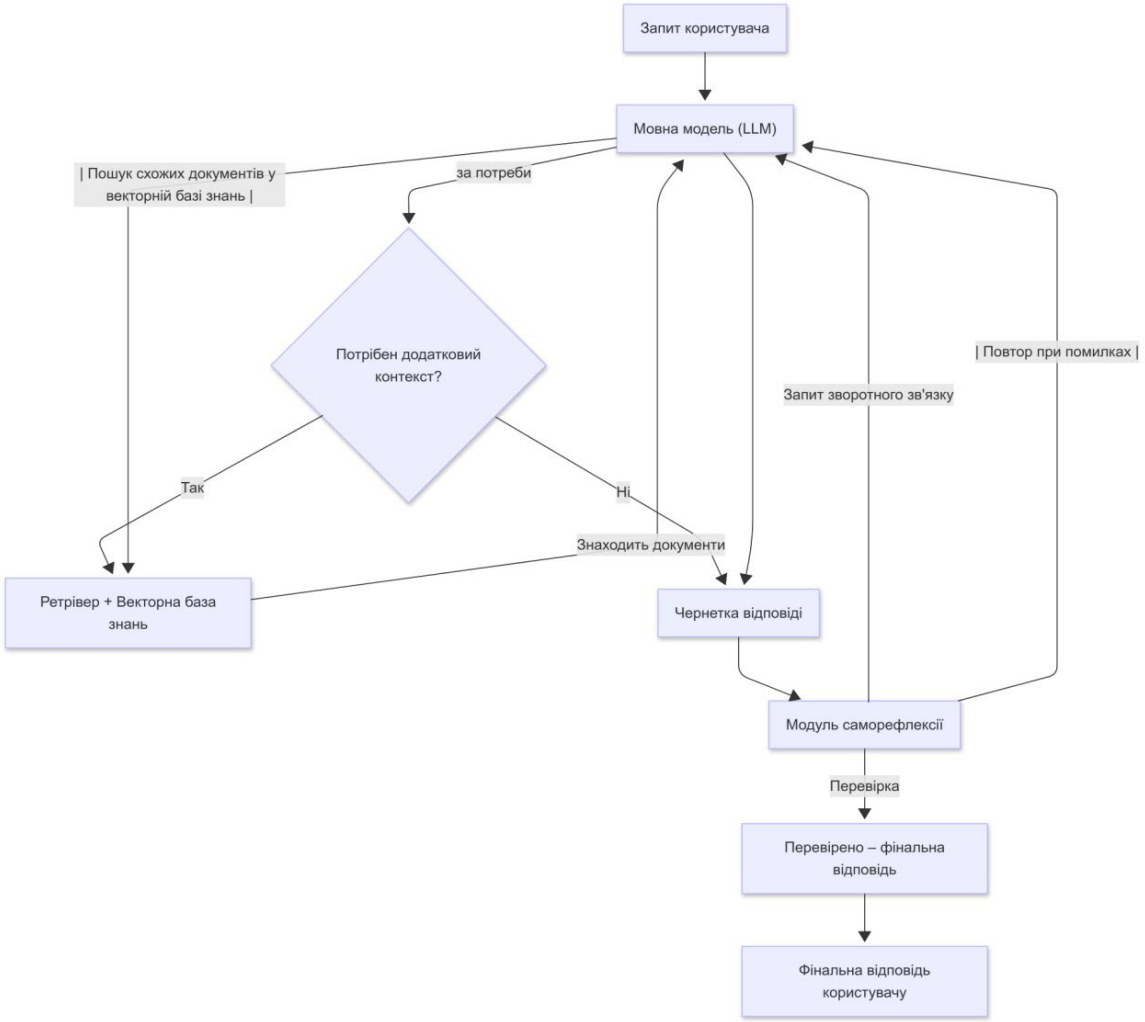

Рис. 1. Структура роботи Self-RAG

Навчання Self-RAG відбувається на одній моделі (наприклад, Llama-2 13B) з використанням спеціально підготовлених даних, де кожен приклад містить: запит, релевантні документи, початкову відповідь моделі і виправлену відповідь. Модель вчиться вставляти токени саморефлексії у потрібні моменти. Результати тестів показали, що Self-RAG (7B та 13B) суттєво перевершує звичайні підходи на ряді задач. Зокрема, точність відповідей і фактична достовірність істотно зросли в порівнянні з ChatGPT, доповненим базою знань, а також з базовою Llama-2 з RAG. Крім того, Self-RAG забезпечує вищу точність цитування джерел у довгих відповідях, тобто модель рідше посилається на невірне джерело або плутає, з якого документа взято інформацію.

Приклад підходу з самоперевіркою – генерація відповіді в кілька чернеток: спочатку модель робить чернетку відповіді на основі частини документів, потім інша модель або сама ж LLM перевіряє цю чернетку. Цю ідею реалізовано у методі Speculative RAG: менший

«спеціаліст»-модель генерує паралельно кілька чернеток відповідей, кожна з яких грунтується на різному підмножині знайдених документів, а потім більша «генераліст»-модель перевіряє їх і обирає найкращу [3]. Така схема дала змогу підвищити точність на 12.97% та зменшити затримку на 51% порівняно зі звичайним RAG на наборі PubHealth [8]. Це підтверджує, що pipeline зі зворотнім зв'язком може бути ефективнішим за одноетапний генератор з фіксованим контекстом. У випадку Speculative RAG виграш досягається тим, що великий GPT-4 (умовно) підключається лише на фінальній стадії перевірки, а основну роботу з генерації варіантів робить швидша модель-«експерт». Self-RAG натомість використовує одну модель, яка самокритично генерує і виправляє себе. Обидва підходи є різновидами більш загальної ідеї: ВММ з внутрішнім циклом перевірки якості, що особливо актуально для RAG-систем, де наявні джерела дозволяють об'єктивно оцінити правдивість відповіді.

Модульні RAG-пайплайни та інтеграція з API. Сучасні рішення дедалі частіше будуються з окремих компонентів, що зв'язуються між собою через чітко визначені інтерфейси. У випадку RAG це означає, що ретривер, база знань, embedder та генератор можуть бути змінними модулями, які легко замінити або викликати через API. Така модульність особливо важлива, коли використовуються зовнішні комерційні моделі (напр. GPT-4 через хмарний API) у поєднанні з власними локальними моделями (для embedding або пошуку). Проєкт LightRAG пропонує фреймворк, де всі компоненти оформлено як класи на кшталт PyTorch-моделей, а спеціальний ModelClient слугує "містком" між внутрішнім пайплайном і зовнішнім API ВММ [2]. Завдяки цьому можна налаштувати LightRAG на роботу як з відкритою моделлю (наприклад, локальна Llama2), так і з закритою (GPT-4o через OpenAI API) – достатньо під'єднати відповідний Клієнт. Модульність також дозволяє легко масштабувати чи замінювати частини системи: скажімо, підключити інший векторний індекс, або додати проміжний агент, який обробляє запит користувача перед подачею в ретривер (наприклад, класифікує тип запиту).

Модульний підхід полегшує інтеграцію RAG у існуючі інформаційні системи. Наприклад, компанія може мати власну базу знань і пошуковий індекс – їх легко підключити як модуль ретривера, а генерацію відповідей здійснювати через зовнішній API GPT-4o. При цьому зберігається гнучкість: ту саму систему можна переналаштувати на іншу модель (наприклад, при появі GPT-5 або при переході на локальну LLM для економії коштів) без кардинальної переробки всієї програми. Іншим плюсом модульності є оптимізація продуктивності: як показують впровадження на практиці, можна незалежно масштабувати кожну складову (наприклад, мати декілька копій ретривера для паралельного пошуку, або кешувати результати embed-декодування). Концепція RAG 2.0 також узгоджується з модульністю: хоча модель навчається спільно, на етапі використання ми все одно можемо замінити, скажімо, векторну базу на швидшу, не перенавчаючи генератор – достатньо повторно індексувати документи.

Таким чином, сучасний напрям – це "легоскладання" RAG-систем із готових блоків. З'явилися фреймворки (LangChain, LlamaIndex, LightRAG, та ін.), що надають розробникам прості інструменти для побудови кастомних пайплайнів. У результаті навіть складні варіації (як-от додавання промптового ретривера чи self-refine циклу) можна реалізувати, конфігуруючи відповідний блок без ручного «хардкодингу» всієї логіки.

Порівняльна оцінка методів. Для узагальнення розглянемо порівняння основних підходів за трьома ключовими показниками: рівень галюцинацій (частка відповідей, що містять непідтверджені факти), достовірність/коректність відповіді (частка відповідей, які повністю відповідають знайденим джерелам, умовно "Answer Faithfulness"), та прецизійність ретривера (Precision – відсоток знайдених фрагментів, що є релевантними до запиту). У таблиці 1 наведено ці метрики для різних реалізацій: від базового використання LLM без зовнішнього пошуку до удосконалених варіантів RAG.

З наведеного порівняння бачимо, що всі стратегії, які інтегрують додаткові знання чи покращують взаємодію між пошуком і генерацією, приводять до істотного підвищення якості. Зокрема, Self-RAG вдалося досягти майже повного усунення галюцинацій на тестових

завданнях (замість ~15% у звичайного RAG) за рахунок того, що модель сама себе контролює і коригує [3]. RAG 2.0 також суттєво підвищує показник faithfulness (правдивості) – до ~90%, що означає, що 9 з 10 відповідей повністю відповідають джерелам. Для порівняння, GPT-4o без доступу до зовнішніх даних може лише в ~60% випадків дати фактично правильну відповідь на довільне знання-інтенсивне питання, інакше або відмовиться, або придумає відповідь. Тому актуальність RAG не знижується навіть з появою більш потужних моделей – навпаки, чим більші моделі, тим дорожче і ризиковано їх використовувати без верифікації, особливо у критичних галузях (медицина, фінанси, право).

Таблиця 1

Орієнтовні показники якості відповідей та пошуку для різних методів.

Підхід / Модель	Hallucination Rate	Answer Faithfulness	Retrieval Precision
LLM (GPT-4o) без RAG	~30%	~60%	– (N/A)
Стандартний RAG (статичний ретривер + GPT-4o)	~15%	~80%	~75%
RAG 2.0 (joint fine-tune)	~10%	~90%	~85%
Self-RAG / Active RAG (GPT-4o, 7B–13B)	5%	95%	~90%

Водночас висока точність і відсутність галюцинацій — лише одна грань ефективності RAG-систем. У практичних сценаріях не менше значення мають операційні метрики: швидкість формування відповіді, обсяг опрацьованого контексту й, зрештою, вартість обчислень. Саме тому доцільно доповнити порівняння аналізом продуктивності різних підходів — від «чистого» GPT-4 до Self-RAG із адаптивним багатокроковим пошуком. Наступна таблиця показує, як оптимізація стратегії ретривалу впливає на латентність, кількість зовнішніх звернень та бюджет проєкту.

У таблиці 2 наведено порівняння ряду додаткових метрик для великих мовних моделей з різними підходами до пошуку знань: без зовнішнього контексту (сам лише ChatGPT), з традиційним Retrieval-Augmented Generation (фіксована кількість вибраних документів), та з вдосконаленим Self-RAG (адаптивний багаторазовий пошук). Показані середній час отримання відповіді (латентність), кількість звернень до зовнішньої бази знань на один запит, обсяг текстового контексту з бази, що опрацьовується моделлю, та орієнтовна вартість обчислень на 1000 запитів.

Таблиця 2

Метрики продуктивності для різних підходів до пошуку знань у LLM

Модель	Середній час відповіді, с	Зовнішні запити до бази	Обсяг контексту, токенів	Вартість (1000 запитів), $
ChatGPT (GPT-4o)	≈5.9	0	–	~1280
GPT-4o chat + RAG	≈5.9	1 (фіксовано)	~5000	~51
Self-RAG (GPT-4o, 7B–13B)	≈4.4	*Adaptive* (множинні)	~3600	~36

Self-RAG демонструє нижчу латентність відповіді за рахунок вибіркового використання релевантних даних (менший обсяг контексту) та ефективнішої інтеграції інформації, що також дає змогу знизити вартість генерації відповідей. Зокрема, динамічний підхід дозволяє уникнути зайвих викликів зовнішньої бази; в одному з експериментів адаптивна pipeline виконувала в середньому ~6 зовнішніх звернень на запит для перевірки декількох джерел, тоді як стандартний метод читав весь контекст відразу. Наведені цифри відповідають сценарію, де

повний обсяг даних (без RAG) становить ~128 тисяч токенів; у такому випадку пряме введення всіх даних у GPT-4o коштувало б у ~25 разів дорожче, ніж поетапне вибіркове читання через RAG.

Висновки

У статті проаналізовано еволюцію та ефективність сучасних підходів Retrieval-Augmented Generation (RAG) у поєднанні з великими мовними моделями, зокрема RAG 2.0 та Self-/Active RAG. Порівняльний аналіз показує, що перехід від «чистої» GPT-4o до підходів RAG кардинально змінює якість відповідей: частка галюцинацій падає з 30 % до близько 5 %, а частка повністю підтверджених фактами відповідей зростає до ≈95 %. Найвідчутніший стрибок дають RAG 2.0 із спільним донавчанням ретривера й генератора та Self-/Active RAG, де модель має вбудований механізм самоперевірки. Ці ж підходи позитивно впливають і на продуктивність: середня затримка відповіді скорочується приблизно на півтори секунди, обсяг опрацьованого контексту зменшується майже на третину, а витрати на тисячу запитів падають до третини від базового рівня.

Спільне end-to-end тренування в RAG 2.0 усуває типові суперечності між пошуком і генерацією, що істотно знижує ризик фактологічних помилок без помітного підвищення обчислювальних витрат. Динамічний, активний пошук і self-refinement дають системі змогу звертатися до зовнішньої бази лише тоді, коли це дійсно потрібно, тим самим скорочуючи зайві виклики й підвищуючи швидкодію. Модульна архітектура RAG забезпечує легку інтеграцію зі сторонніми API, дозволяє незалежно оновлювати компоненти та спрощує дотримання вимог до конфіденційності й актуальності даних у корпоративних середовищах.

Подальший розвиток може бути у поєднанні RAG із постійно оновлюваними базами знань і веб-пошуком у реальному часі, а також у впровадженні агентних механізмів, які дадуть моделі змогу самостійно запускати зовнішні дії для верифікації фактів або оновлення контенту. Такі рішення наблизять створення «живих» інтелектуальних систем, що спираються на актуальну інформацію й уміють критично оцінювати власні висновки, роблячи Retrieval-Augmented Generation одним із ключових напрямів розвитку практичної AI-інженерії.

Список використаної літератури:

1. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / Lewis M. et al. NeurIPS, 2020. URL: https://medium.com/@adnanmasood/contextually-enriched-knowledge-enhanced-and-externally-grounded-retrieval-models-for-fun.

2. RAG 101: Demystifying Retrieval-Augmented Generation Pipelines / Wolff H. NVIDIA Technical Blog, 2023. URL: https://developer.nvidia.com/blog/rag-101-demystifying-retrieval-augmented-generation-pipelines.

3. Self-RAG: Self-Reflective Retrieval-Augmented Generation / Asai A. et al. arXiv:2310.11511, 2023. https://doi.org/10.48550/arXiv.2310.11511.

4. Contextual AI. Introducing RAG 2.0 / Kiela D. Medium blog post, 2024. URL: https://medium.com/@jackdrummond_16745/introducing-rag-2-0-revolutionising-ai-and-llm-development-71a1f85cc0dd.

5. Active Retrieval-Augmented Generation / Mallen E. et al. arXiv: 2409.11136, 2023. URL: https://doi.org/10.48550/arXiv.2409.11136.

6. Contextually Enriched, Knowledge-Enhanced, and Externally Grounded Retrieval Models / Masood A. Medium, 2025.

7. Promptriever: Promptable Retrieval Models / Weller O. et al. EMNLP, 2024 (preprint). URL: https://medium.com/@techsachin/promptriever-first-zero-shot-promptable-instruction-trained-retriever-model-72e9f2eecbb2.

8. Speculative RAG: Enhancing RAG through Drafting / Wang Z. et al. arXiv:2407.08223, 2024. URL: https://doi.org/10.48550/arXiv. 2407.08223.

Автори статті

Олійник Богдан – аспірант, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

ORCID: 0009-0003-9146-9428

Чичкарьов Євген – доктор технічних наук, професор, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

ORCID: 0000-0002-4362-5129

Authors of the article

Oliinyk Bohdan – postgraduate, State University of Information and Communication Technologies, Kyiv, Ukraine.

ORCID: 0009-0003-9146-9428

Chychkarov Yevhen – Doctor of Sciences (technical), Professor, State University of Information and Communication Technologies, Kyiv, Ukraine.

ORCID: 0000-0002-4362-5129