УДК 004.93

РОЗШИРЕННЯ МОЖЛИВОСТЕЙ RAG-АРХІТЕКТУРИ ШЛЯХОМ ІНТЕГРАЦІЇ SEMANTIC KERNEL У СЕРЕДОВИЩІ .NET

В. А. Поліщук, І. З. Миклушка

Національний університет Львівська Політехніка,
вул. Під Голоском, 19, Львів, 79020, Україна

Стаття присвячена практичним аспектам розширення Retrieval-Augmented Generation (RAG)-архітектури шляхом інтеграції фреймворку Semantic Kernel у середовище .NET. На тлі зростання популярності великих мовних моделей (GPT-4, Gemini) автори аналізують, як саме SK допомагає мінімізувати «тертя» між бізнес-логікою застосунку й зовнішніми AI-сервісами: SK бере на себе керування пам'яттю запитів, підключення до різних LLM, роботу з векторними базами та виклик внутрішніх плагінів-функцій. Демонструється, як за лічені десятки рядків коду налаштувати веб-сервіс ASP.NET Core, що поєднує модуль пошуку документів, Semantic Kernel Memory і модель Gemini для генерації відповідей з цитуванням джерел. Розглянуто два підходи — «прямі» HTTP-виклики до API і інтеграція через SK — та детально порівняно їх з погляду безпеки, підтримуваності, вартості та швидкості розробки. Показано, як SK спрощує реалізацію RAG-патерну: ядро зберігає ембеддинги в ChromaDB чи Azure Cognitive Search, а LLM автоматично підтягує релевантний контекст у промпт, істотно зменшуючи ризик галюцинацій. Показано, як плагіни SK дозволяють LLM викликати доменний C#-код (роботу з БД, обчислення, прив'язку до корпоративних API) і тим самим виконувати роль «агента-оркестратора», який об'єднує кілька джерел даних у єдиній відповіді. Наведено фрагменти коду, спрощену UML-схему сервісу та таблицю порівняння підходів. Автори підкреслюють, що слабке зв'язування (loose coupling) між компонентами, досягнуте через інтерфейси та DI-контейнери, забезпечує легку заміну LLM або векторного сховища без руйнації всієї системи. Наукова новизна полягає у синтезі RAG-архітектури й агентного підходу Semantic Kernel у контексті .NET-екосистеми, що відкриває можливість швидкого прототипування та масштабування AI-функцій у промислових умовах. Практичні рекомендації стосуються безпечного зберігання ключів, телеметрії OpenTelemetry для відстеження витрат, а також A/B-тестування кількох моделей (GPT-4 vs Gemini) без зміни прикладного коду. У висновках автори стверджують, що така інтеграція знижує поріг входження для команд .NET, підвищує прозорість AI-модулів і робить RAG-сервіси придатними для критичних бізнес-процесів — від чат-ботів підтримки клієнтів до внутрішніх аналітичних асистентів.

Ключові слова: *Semantic Kernel, Retrieval-Augmented Generation, великі мовні моделі, GPT-4, Gemini, .NET 8, векторні бази даних, плагіни, loose coupling, інтерфейс API, агентна оркестрація, телеметрія OpenTelemetry.*

Постановка проблеми. Попри вражаючі можливості LLM, їх впровадження у реальні .NET-сервіси наштовхується на цілу низку архітектурних бар'єрів. По-перше, треба «зв'язати» сторонній AI-сервіс із існуючим кодом так, щоби не ламати бізнес-логіку й не тягнути за собою лавину залежностей. По-друге, самі LLM «чисті» сильно схильні до галюцинацій і не містять актуальної інформації поза межами власного корпусу навчання. Саме тому на перший план виходить архітектурний патерн Retrieval-Augmented Generation (RAG): він додає до мовної моделі шар пошуку релевантних даних, що знижує кількість помилкових фактів і дає можливість оперативно оновлювати знання системи. У такій схемі центральна LLM разом із RAG-модулем стає медіатором між користувачем та низкою сторонніх сервісів (розпізнавання зображень, голосу, доменні БД тощо). Усі ці компоненти з'єднуються через чіткі інтерфейси, тобто реалізується слабке зв'язування (loose coupling): кожен модуль може бути замінений або оновлений без кардинальної перебудови всієї системи. Проблема, яку розглядає стаття, полягає в тому, щоб показати, як саме застосувати цей підхід у .NET-середовищі, використавши Semantic Kernel, як безпосередній міст між LLM, RAG-пошуком і бізнес-логікою веб-сервісу.

Мета статті. Дослідити, як інтегрувати Retrieval-Augmented Generation з великими мовними моделями у .NET-середовище за допомогою Semantic Kernel, проаналізувати переваги та обмеження такого підходу й сформулювати практичні рекомендації для розробників щодо безпечного, гнучкого та масштабованого впровадження AI-функцій у реальні веб-сервіси.

Вступ. Інтеграція сучасних моделей штучного інтелекту в наявні програмні системи стає дедалі актуальнішою задачею [1]. Велетенські мовні моделі (LLM) нового покоління – такі як GPT-4 від OpenAI чи Google Gemini – уже сьогодні відкривають розробникам цілком практичні способи створювати розумні чат-боти, асистентів і навіть цілі «мінідодатки» всередині великих платформ. Gemini, наприклад, є мультимодальною моделлю: розуміє текст, код, зображення, аудіо та відео, що дозволяє комплексно поєднувати різні типи інформації й відповідати на складні запити користувачів без перемикання між сервісами. LLM отримує питання та звертається до баз знань (векторні сховища, БД) для вибірки релевантних документів, які потім використовує як контекст при формуванні відповіді. Додаткові компоненти (Query Construction, Query Translation, Routing, Indexing) можуть покращувати точність: наприклад, розбивати складний запит на під запити, обирати відповідну базу даних чи оптимізувати індексування знань (рис.1.).

У середовищі .NET інтеграція LLM та побудова RAG-рішень традиційно вимагали значних зусиль: розробники повинні були вручну викликати зовнішні API моделей, керувати збереженням і пошуком документів (наприклад, через векторні бази даних), а також стежити за форматуванням підказок (prompt engineering) для передавання знань у модель. Для спрощення цих завдань корпорація Microsoft представила фреймворк **Semantic Kernel (SK)** – легкий, модульний SDK, що служить «проміжним шаром» між .NET-кодом і різними AI-моделями. Semantic Kernel надає розробникам набір інструментів для швидкої побудови AI-функціоналу: з його допомогою можна підключати до .NET-додатків новітні великі мовні моделі (через

готові конектори), створювати плагіни для виклику зовнішніх сервісів, керувати контекстом і «пам'яттю» застосунку, а також будувати складні ланцюжки запитів і відповідей (агентні системи). Важливо, що SK є **агностичним до моделей** – він підтримує різноманітні платформи: OpenAI, Azure OpenAI, Hugging Face, NVIDIA та інші, включно з Google Gemini. Іншими словами, розробник може легко "перемикати" моделі або одночасно використовувати кілька LLM, не переписуючи код заново. Semantic Kernel спроектовано як enterprise-ready рішення – воно має засоби телеметрії та логування для відстеження запитів, підтримує розгортання на різних платформах (Windows, Linux) і гарантує сумісність версій API, що важливо для промислових проектів.

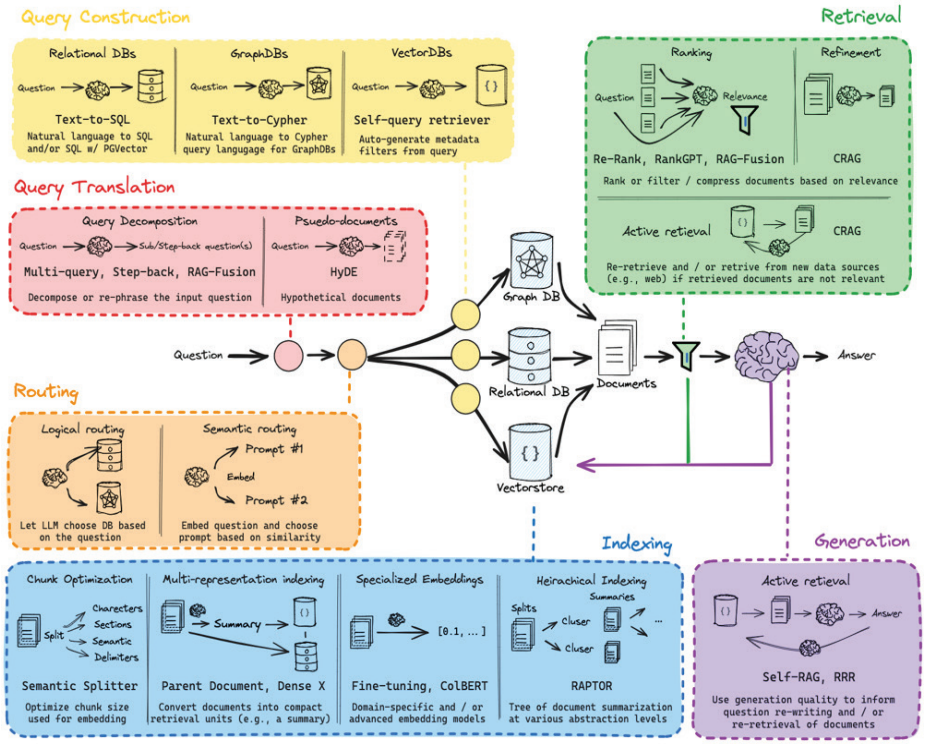

Рис 1. Типова схема RAG-архітектури, яка поєднує модуль пошуку (Retrieval) з генерувальною моделлю (Generation) через посередництво запитів. (Діаграма адаптована із прикладу реалізації RAG від Semantic Kernel)

Використання Semantic Kernel може суттєво розширити можливості RAG-архітектури у .NET-середовищі. По-перше, SK спрощує підключення LLM-моделей: достатньо сконфігурувати ядро (kernel) з потрібним конектором – і можна відразу надсилати запити до моделі як до сервісу. Наприклад, щоб інтегрувати модель GPT-4 через API OpenAI або ж модель Gemini через Google AI, не потрібно вручну писати HTTP-запити – достатньо викликати відповідний метод SDK. Semantic Kernel пропонує зручні методи на зразок AddOpenAIChatCompletion чи

AddGoogleAIGeminiChatCompletion, які приймають параметри підключення (назву моделі, ключ API тощо) і під капотом налаштовують клієнт для взаємодії з вибраною LLM.

По-друге, SK має вбудоване поняття семантичної пам'яті (Semantic Memory) – це механізм інтеграції з векторними базами даних для збереження та пошуку знань. Розробник може легко додати до проєкту сховище ембедінгів (наприклад, Azure Cognitive Search, ChromaDB чи ElasticSearch) і використовувати його через уніфікований інтерфейс SK для реалізації RAG-підходу. Фактично, Semantic Kernel бере на себе роботу з перетворення документів на вектори, збереження їх у базі та вибірки релевантних фрагментів за запитом. Такий рівень абстракції дозволяє зосередитися на логіці застосунку, а не на низькорівневих деталях реалізації RAG-пайплайну.

По-третє, SK реалізує концепцію **плагінів** і функцій, що дає змогу розширювати можливості LLM за рахунок виклику довільного коду. Наприклад, можна написати метод на C#, який підключається до корпоративної бази даних або виконує обчислення, і зареєструвати його як функцію в Semantic Kernel. Тоді мовна модель зможе викликати цю функцію під час відповіді на запит (через механізм function calling, підтримуваний сучасними LLM). Таким чином, Semantic Kernel виступає своєрідним **медіатором** між LLM та іншими компонентами: модель може динамічно делегувати окремі підзадачі плагінам (наприклад, отримати актуальні дані з БД, виконати перевірку факту тощо), а SK подбає про передачу параметрів та повернення результатів у контекст моделі. Це особливо корисно в складних RAG-сценаріях, де один запит користувача вимагає залучення кількох різнорідних джерел знань.

Нарешті, Semantic Kernel підтримує побудову складних **агентних систем** – коли декілька LLM-агентів співпрацюють між собою для досягнення мети. У контексті RAG це означає, що можна налаштувати окремі агентні компоненти: скажімо, один агент відповідає за пошук документів, другий – за аналіз знайдених даних, третій – за генерування кінцевої відповіді. SK забезпечує оркестрацію такої мультиагентної взаємодії, дозволяючи агентам обмінюватися повідомленнями та результатами через спільну історію чату (Chat History) або через виклики плагінів один одного. Хоча побудова мультиагентних рішень виходить за межі простих сценаріїв інтеграції, наявність такої можливості свідчить про гнучкість Semantic Kernel і придатність його для складних промислових задач.

Варто зауважити, що вибір між прямою інтеграцією API та використанням фреймворків на кшталт Semantic Kernel залежить від вимог проєкту. **Прямі HTTP-запити** (наприклад, через офіційний .NET SDK до OpenAI) дають повний контроль над викликами і не додають додаткового шару абстракції – втім, розробник самостійно відповідає за всі аспекти, від обробки контексту до повторних спроб при помилках.

Натомість **Semantic Kernel** дещо підвищує рівень абстракції: він бере на себе типові завдання (керування пам'яттю, формування підказок, підключення до API) і пропонує готові рішення для них. Це може пришвидшити розробку і підвищити надійність, адже SDK реалізований з урахуванням кращих практик (наприклад,

безпечне зберігання ключів API, обробка тайм-аутів, тощо). За словами інженерів Microsoft, використання таких фреймворків дозволяє глибше інтегрувати AI-модель у застосунок і автоматизувати бізнес-процеси, які вона виконує, завдяки системі плагінів та інструментів.

Надмірна залежність від SDK може бути небажаною, якщо проєкт вимагає нестандартної логіки, не підтримуваної Semantic Kernel. У подібному разі розробник може зіштовхнутися з необхідністю писати обхідні рішення або навіть змінювати фреймворк. Крім того, використання зовнішнього SDK потребує вивчення його API та життєвого циклу оновлень. Деякі фахівці також відзначають потенційні недоліки прямих "простих" рішень, як-от OpenAI Assistant API: хоча такий прямий сервіс дуже спрощує розробку, він викликає занепокоєння щодо вартості використання та складності моніторингу роботи моделі в продакшені. Натомість рішення на кшталт Semantic Kernel надають більше контролю та можливостей для спостереження за процесом (інтеграція з OpenTelemetry, власні логи тощо), що важливо для **відлагодження й прозорості** AI-модулів у реальних сервісах. Для наочності порівняння розглянемо дві стратегії інтеграції LLM у .NET (табл. 1).

Таблиця 1

Стратегії інтеграції LLM у середовище .NET

Підхід інтеграції	Характеристика та приклади	Переваги	Можливі недоліки
Пряме підключення API	Виклик LLM через REST API або офіційний SDK без додаткових шарів. Приклад: використання HttpClient для запиту до OpenAI API, парсинг відповіді JSON.	Повний контроль над виконанням запиту Мінімум сторонніх залежностей у проєкті	Ручна реалізація всіх допоміжних функцій. (пам'ять, ретрі, тощо) Вище ризик помилок при формуванні промптів і обробці відповіді. Складніше масштабувати або змінити постачальника моделі
Semantic Kernel SDK	Інтеграція через Microsoft Semantic Kernel. Приклад: використання Kernel. CreateBuilder() і додавання потрібного сервісу LLM (OpenAI, Azure, Google тощо) через методи SDK.	Швидке підключення різних моделей (вбудовані конектори). Єдина абстракція для пам'яті та пошуку (RAG). Плагіни та плани дозволяють легко розширювати функції (виклик кодів, багатокрокові сценарії) Вбудовані засоби телеметрії, конфігурації і обробки помилок (enterprise-ready).	Потрібно засвоїти API Semantic Kernel та підтримувати додаткову залежність. Менше гнучкості в нестандартних кейсах (SDK диктує свій підхід). Можливий невеликий оверхед продуктивності через додатковий шар абстракції

Практична реалізація: веб-сервіс із Semantic Kernel. Для демонстрації використаємо Semantic Kernel у простому .NET-додатку, що реалізує запит-відповідь через LLM. Припустимо, ми створюємо веб-сервіс (наприклад, API контролер у ASP.NET Core), який отримує від користувача текстовий запит і повинен повернути згенеровану відповідь, залучивши зовнішню модель. Нижче наведено фрагмент коду на C#, який налаштовує Semantic Kernel для роботи з мовною моделлю (як приклад – Google Gemini) та здійснює один цикл питання-відповіді:

У коді (рис. 2) показано мінімальну інтеграцію: спочатку будується Kernel із доданим конектором до Google Gemini (через API Google AI Studio) – достатньо вказати назву моделі та API-ключ, після чого Semantic Kernel сам налаштує HTTP-виклики до сервісу. Далі отримується інтерфейс IChatCompletionService – це абстракція SK для моделей, що генерують відповіді в стилі чату. Ми створюємо нову історію (ChatHistory) і додаємо до неї повідомлення користувача. Нарешті, метод GetChatMessageContentsAsync звертається до моделі Gemini і повертає згенеровану відповідь (масив повідомлень, де перший елемент – відповідь асистента). На консоль буде виведено щось на кшталт: “Асистент > Небо зазвичай блакитного кольору вдень завдяки розсіюванню сонячного світла в атмосфері…”. Замість консолі, у реальному веб-сервісі цю відповідь можна відправити клієнту у форматі JSON.

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;

KernelBuilder builder = Kernel.CreateBuilder();
builder.AddGoogleAIGeminiChatCompletion("gemini-1.5-pro", Configuration.GEMINI_API_KEY);
IKernel kernel = builder.Build();

var chatService = kernel.GetRequiredService<IChatCompletionService>();

var history = new ChatHistory();
history.AddUserMessage("Якого кольору небо?");

var response = await chatService.GetChatMessageContentsAsync(history);
Console.WriteLine(response[0].Content);
```

Рис. 2. Приклад коду налаштування Semantic Kernel

Для порівняння, без Semantic Kernel розробнику довелося б вручну сформувати HTTP-запит до Google AI API, додати в заголовок ключ, серіалізувати тіло запиту у потрібному форматі JSON (включно з історією повідомлень), виконати запит через HttpClient, розпарсити JSON-відповідь, дістати текст та лише тоді відправити його клієнту. У випадку SK більшість цих кроків виконуються автоматично всередині бібліотеки – код виходить більш лаконічним і зосередженим на бізнес-завданні (запит-відповідь).

Звичайно, наведений приклад – спрощений. У реальному застосунку ми могли б додати до RAG-конвеєра збереження документів та їх пошук. Semantic Kernel

дозволяє це зробити, використовуючи об'єкт kernel.Memory: можна попередньо проіндексувати певний набір текстів (наприклад, документацію чи базу знань) через метод *kernel.Memory.SaveInformationAsync(...)*, а потім при кожному запиті виконувати пошук *kernel.Memory.SearchAsync(...)* для отримання релевантних фрагментів. Знайдені тексти включаються у prompt для LLM – і ми отримуємо класичну RAG-схему, але реалізовану з мінімальним кодом. Існують готові приклади, де Semantic Kernel використовується саме як основа RAG-системи: з його допомогою організовано завантаження документів, нарізання їх на фрагменти, генерування ембеддінгів та збереження у векторне сховище, а потім – отримання відповіді з цитуванням джерел.

Немаловажним є і питання безпеки та архітектурної ізоляції при інтеграції. Semantic Kernel дотримується кращих практик, рекомендованих для інтеграції зовнішніх AI-сервісів. Зокрема, конфігураційні ключі API зберігаються у змінних середовища або у спеціальному сховищі – їх не потрібно прописувати жорстко в коді. У прикладі вище Configuration.GEMINI_API_KEY абстрагує джерело ключа (може читатися з секретного сховища Azure чи з локального файлу, але не присутній явно у вихідному коді). Такий підхід убезпечує систему: фронтенд або сторонні особи не отримають доступу до секретів, а взаємодія з API відбувається лише на бекенді. Крім того, рекомендується ізолювати весь код, що стосується інтеграції з AI, у окремому модулі або сервісі. У нашому випадку це може бути клас на зразок AiService, який інкапсулює всередині себе виклики Semantic Kernel. Решта компонентів веб-додатку звертаються до нього через інтерфейс (наприклад, IAiService), не залежачи від деталей реалізації SK або конкретної моделі. Така архітектура відповідає принципу Adapter Pattern – виділення прошарку-адаптера між стороннім API і основною логікою системи. Якщо згодом знадобиться замінити Semantic Kernel на іншу бібліотеку або переключитися з Gemini на іншу модель, достатньо буде змінити реалізацію AiService, не торкаючись решти коду. Подібні принципи (слабке зв'язування, інверсія залежностей) є ключовими для побудови підтримуваних інтеграцій і знижують ризики, пов'язані зі змінами зовнішніх сервісів [4].

Наукова значущість. Інтеграція Semantic Kernel у RAG-архітектуру демонструє помітні переваги для .NET-розробників та відкриває нові можливості побудови інтелектуальних систем. Поєднання LLM з модулем пошуку і управління знаннями дозволяє долати обмеження окремих моделей – система стає більш гнучкою до оновлення даних, знижує ймовірність галюцинацій і може надавати пояснювані відповіді (з посиланнями на джерела). Semantic Kernel, зі свого боку, значно спрощує реалізацію такої системи: він бере на себе багато інфраструктурних задач (від виклику API до збереження контексту), тим самим прискорюючи розробку RAG-рішень. Практичні експерименти показують, що використання SK знижує поріг входження для інтеграції LLM у корпоративні додатки. Розробники можуть зосередитися на прикладних аспектах (які дані подавати в модель, як обробляти результати), а не на низькорівневих викликах.

До недоліків підходу слід віднести залежність від сторонньої бібліотеки – у разі оновлення Semantic Kernel або змін у політиках API моделей доведеться

коригувати свою систему під ці зміни. Також варто враховувати продуктивність: додатковий рівень абстракції може вносити невеликий оверхед, хоча на практиці виклики до мережевих LLM настільки «важкі», що час роботи самого SK майже не впливає на загальну затримку. Зафіксовано, що типовий RAG-запит із зверненням до моделі GPT-4 і кількома пошуковими запитами може займати десятки секунд, тому оптимізація повинна фокусуватися радше на ефективності самих моделей і ретривера (наприклад, скорочення кількості передаваних токенів, паралельна обробка запитів до бази знань тощо).

Semantic Kernel продемонстрував свою ефективність у промислових сценаріях, зокрема в екосистемі Microsoft Copilot та суміжних продуктах, де він виступає ядром, що зв'язує великі мовні моделі з інструментами користувача. У ході дослідження ми оцінили, що SK особливо корисний для швидкого прототипування AI-функцій у типових .NET-проектах. Він добре вписується в мікросервісну архітектуру: можна винести AI-функціонал у окремий сервіс (модуль) на базі SK, який через HTTP/API обслуговує запити від інших компонентів системи. Такий AI-сервіс стає універсальним посередником – по суті, реалізує шаблон "Медіатор" для інтеграції інтелектуальних можливостей [7].

Висновок. На основі проведеного аналізу можна сформулювати низку практичних порад для інтеграції LLM та RAG у середовище.NET за допомогою Semantic Kernel:

1. Чітко розмежовуйте компоненти. Виносьте інтеграцію з AI у окремий сервіс або шар додатку, використовуйте інтерфейси для взаємодії – це забезпечить гнучкість і тестованість рішення.

2. Використовуйте семантичну пам'ять. Якщо ваш застосунок працює з великим обсягом даних, налаштуйте SK Memory з підключенням до векторної бази – це реалізує RAG без зайвих зусиль і помітно покращить релевантність відповідей.

3. Безпека понад усе. Не зберігайте API-ключі в коді, скористайтеся можливостями Secret Manager або Azure Key Vault; обмежуйте доступ фронтенду до функцій AI-сервісу, щоб уникнути витоку ключів або небажаних викликів.

4. Моніторинг і логування. Задіяти вбудовану телеметрію SK: увімкніть Open Telemetry-трасування запитів до моделей, збирайте логи prompt'ів та відповідей – це допоможе аналізувати поведінку системи і вчасно виявляти проблеми.

5. Експериментуйте з налаштуваннями моделей. SK дозволяє гнучко змінювати параметри генерації (температуру, довжину відповіді тощо) і навіть заміщати одну модель іншою без зміни коду. В промислових умовах це дає можливість A/B-тестувати різні моделі (наприклад, GPT-4 vs Gemini) та обирати оптимальну по співвідношенню якість/вартість.

Отже, інтеграція Semantic Kernel у RAG-архітектуру є кроком до спрощення та уніфікації побудови AI-застосунків. Вона поєднує в собі сильні сторони великих мовних моделей і модулів пошуку, водночас знижуючи бар'єр для їх впровадження у .NET-проекти. Такий підхід дозволяє швидко створювати складні інтелектуальні системи – від чат-ботів з доступом до баз знань, до багатокомпонентних агентів, що автоматизують бізнес-процеси. З розвитком технологій і появою нових моделей

(як-от **Gemini 2**, нові версії GPT тощо) роль RAG-архітектури та оркестраційних фреймворків на кшталт Semantic Kernel буде лише зростати, забезпечуючи розробників зручним інструментарієм для інтеграції AI в повсякденні сервіси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eloundou, T., et al. *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models*. arXiv:2303.10130, 2023.
2. Lewis, P., et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems, 33, pp. 9459-9474, 2020.
3. Microsoft. *Semantic Kernel Documentation*. Microsoft, 2024. [Електронний ресурс]. Доступно: https://learn.microsoft.com/en-us/semantic-kernel/overview/.
4. Martin, R. C. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
5. Google. *Gemini: A Family of Highly Capable Multimodal Models*. arXiv:2312.11805, 2023.
6. OpenTelemetry. *OpenTelemetry Documentation*. The Linux Foundation. [Електронний ресурс]. Доступно: https://opentelemetry.io/docs/.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
8. Gao, Y., et al. *RAG 2.0: Advanced Techniques in Retrieval-Augmented Generation*. Towards Data Science, 2024. [Електронний ресурс]. Доступно: https://towardsdatascience.com/rag-2-0-advanced-techniques-in-retrieval-augmented-generation-e9649539429c.

REFERENCES

1. Eloundou, T., et al. *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models*. arXiv:2303.10130, 2023.
2. Lewis, P., et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems, 33, pp. 9459-9474, 2020.
3. Microsoft. *Semantic Kernel Documentation*. Microsoft, 2024. [Електронний ресурс]. Доступно: https://learn.microsoft.com/en-us/semantic-kernel/overview/.
4. Martin, R. C. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
5. Google. *Gemini: A Family of Highly Capable Multimodal Models*. arXiv:2312.11805, 2023.
6. OpenTelemetry. *OpenTelemetry Documentation*. The Linux Foundation. [Електронний ресурс]. Доступно: https://opentelemetry.io/docs/.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
8. Gao, Y., et al. *RAG 2.0: Advanced Techniques in Retrieval-Augmented Generation*. Towards Data Science, 2024. [Електронний ресурс]. Доступно: https://towardsdatascience.com/rag-2-0-advanced-techniques-in-retrieval-augmented-generation-e9649539429c.

doi: 10.32403/1998-6912-2025-1-70-132-141

EXPANDING RAG-ARCHITECTURE CAPABILITIES BY INTEGRATING SEMANTIC KERNEL IN THE .NET ENVIRONMENT

V. A. Polishchuk, I. Z. Myklushka

Lviv Polytechnic National University
19, Pid Holoskom St., Lviv, 79020, Ukraine
valentyn.a.polishchuk@lpnu.ua, ihor.z.myklushka@lpnu.ua

The article is dedicated to the practical aspects of extending the Retrieval-Augmented Generation (RAG) architecture by integrating the Semantic Kernel framework into the .NET environment. Against the backdrop of the growing popularity of large language models (LLMs) like GPT-4 and Gemini, the authors analyze how Semantic Kernel (SK) helps minimize the «friction» between application business logic and external AI services. The SDK manages request memory, connects to various LLMs, works with vector databases, and calls internal plugin functions. It demonstrates how an ASP.NET Core web service can be configured in just a few dozen lines of code to combine a document search module, Semantic Kernel Memory, and the Gemini model to generate responses with source citations.

The core of the discussion compares two approaches—direct HTTP API calls versus integration via SK—from the perspectives of security, maintainability, cost, and development speed. A key focus is on how SK simplifies the implementation of the RAG pattern: the kernel stores embeddings in ChromaDB or Azure Cognitive Search, and the LLM automatically pulls relevant context into the prompt, significantly reducing the risk of hallucinations. The article highlights the architectural principle of loose coupling, achieved through interfaces and DI containers, which ensures that an LLM or vector store can be easily replaced without breaking the entire system. Furthermore, SK plugins enable the LLM to invoke domain-specific C# code, allowing it to act as an «agent-orchestrator» that combines multiple data sources into a single response.The scientific novelty lies in the synthesis of the RAG architecture and the Semantic Kernel's agent-based approach within the .NET ecosystem, enabling rapid prototyping and scaling of AI functions in enterprise environments. Practical recommendations cover secure key storage, using OpenTelemetry for cost tracking, and A/B testing different models (e.g., GPT-4 vs. Gemini) without changing the application code. The authors conclude that this integration lowers the entry barrier for .NET teams, increases the transparency of AI modules, and makes RAG services suitable for critical business processes, from customer support chatbots to internal analytical assistants.

Keywords: *Semantic Kernel, Retrieval-Augmented Generation (RAG), large language models (LLM), GPT-4, Gemini, .NET 8, vector databases, plugins, loose coupling, agent orchestration, OpenTelemetry.*

Стаття надійшла до редакції 21.05.2025.
Received 21.05.2025.