DOI 10.34132/mspc2025.01.14.08

Олександр Єлісєєв,
Гліб Горбань

АВТОМАТИЧНА ГЕНЕРАЦІЯ ПРОЄКТІВ ЗА ДОПОМОГОЮ LLM-АГЕНТІВ

Тези досліджують використання LLM-агентів для автоматичної генерації програмних проєктів, аналізуючи їхні можливості, переваги та ризики порівняно з традиційними алгоритмами та простими LLM-запитами. Розглядається архітектура агентів, їх інтеграція з сучасними технологіями (Django, React, OpenAI API), а також сценарії застосування в бізнесі, освіті та фінансових технологіях. Розглядається система автоматичної генерації проєктів із залученням LLM-агентів. Наголошується на важливості обґрунтованого вибору між складними агентами та простими рішеннями залежно від вимог проєкту для забезпечення ефективності, безпеки та мінімізації технічного боргу.

Ключові слова: *LLM-агенти, великі мовні моделі, автоматична генерація коду, штучний інтелект, розробка програмного забезпечення, Django, React, OpenAI API.*

У прагненні до використання штучного інтелекту існує поширене переконання, що найсучасніше рішення – зазвичай це AI-агент – завжди є найкращим вибором. Однак у більшості випадків простіші підходи, такі як виклики великих мовних моделей (LLM) або традиційні алгоритми, не лише є достатніми, але й значно практичнішими та ефективнішими. Багато проєктів автоматично використовують AI-агенти, хоча простий виклик LLM або традиційний алгоритм можуть дати необхідний результат із меншими витратами ресурсів.

Останнім часом великі мовні моделі (LLM) стали неймовірно популярними. На сьогоднішній день вже спостерігається активний розвиток цієї галузі, що підтверджується численними дослідженнями, публікаціями, фреймворками та проєктами провідних компаній. Прикладом може бути GitHub CoPilot.

Розглянемо, що саме мається на увазі під поняттям LLM-агенти.

Агенти – це системи, які взаємодіють із динамічним середовищем, сприймають його та діють відповідно до закладених цілей або завдань. У цьому контексті LLM (великі мовні моделі) – це потужні моделі, здатні передбачати ймовірний наступний токен на основі вхідного тексту (де токен – це слово або його частина) [1].

Агенти, які працюють виключно з текстовою інформацією, можуть ефективно виконувати широкий спектр складних завдань, отримуючи дані про стан середовища у текстовому вигляді. Водночас їхні можливості істотно зростають, якщо вони здатні обробляти інші модальності, такі як аудіо чи зображення. Для цього важливо не лише перетворити кожну з таких модальностей у вектор (або послідовність векторів), але й забезпечити, щоб усі ці вектори належали до єдиного прихованого простору.

Автоматична генерація проєктів із використанням LLM-агентів докорінно змінює процес створення програмного забезпечення. Сучасні LLM-агенти, зокрема побудовані на GPT-4, Claude 3, Gemini та інших передових архітектурах, дозволяють переходити від трудомісткого, ручного написання коду до сценаріїв, де вся основна робота виконується штучним інтелектом. На практиці це виглядає так: замовнику або розробнику достатньо описати функціональність і вимоги зрозумілою мовою, після чого агент аналізує промпт, формує технічне завдання, створює репозиторій, ініціалізує структуру проєкту, генерує код для бекенду (наприклад, на Django чи Flask), одразу прописує тестові сценарії з pytest або unittest, формує docker-контейнери, налаштовує CI/CD на GitHub Actions або GitLab CI, дописує документацію в docstring та створює README, що пояснює використання, а потім автоматично розгортає базову версію проєкту на Heroku, Vercel чи іншому хмарному провайдері.

LLM-агенти не обмежуються звичайною генерацією коду. Вони здатні самостійно аналізувати помилки компіляції, запускати юніт-тести, знаходити й виправляти баги, отримувати фідбек від користувача через інтерактивний чат і далі адаптувати код або інтерфейс. Якщо, наприклад, агент отримує повідомлення про невідповідність API-ендпоінтів чи помилкову логіку автентифікації, він самостійно вносить зміни, фіксує їх у git та повторно тестує результати. Тобто це не просто генератор коду, а багатофункціональний автоматизований виконавець із можливістю ітеративного вдосконалення проєкту без постійної участі людини.

Яскравим прикладом може слугувати генерація повноцінного MVP SaaS-сервісу. Припустимо, що користувач хоче отримати вебзастосунок для бронювання готелів із системою авторизації, панеллю адміністратора, інтеграцією з поштою, автогенерацією інвойсів і мобільною адаптацією. LLM-агент самостійно розбиває задачу на етапи: генерує backend на Django (створює моделі для готелів, бронювань, користувачів, реалізує REST API через DRF), підключає простий frontend на Node.js або генерує React-компоненти, прописує Swagger/OpenAPI документацію, додає автоматичні тести, створює docker-compose

для локального запуску, налаштовує email-інтеграцію через SMTP або сторонній сервіс, а потім моніторить логи й оптимізує код за результатами тестування. Весь цей процес може відбуватись у межах одного інтерактивного діалогу, де агент уточнює деталі (наприклад, зовнішній вигляд інтерфейсу чи специфіку платежів) і одразу втілює зміни.

Щоб зрозуміти, чим саме є LLM-агенти, варто розкласти їхню архітектуру на складові. Це багатоетапна система, що поєднує в собі LLM (яка відповідає за генерацію і аналіз текстів та коду), інтерфейс зв'язку з користувачем, спеціалізовані інструменти для роботи з файлами, системами контролю версій, інфраструктурою, а також модулі планування і координації завдань. Наприклад, популярний агент Open Interpreter здатен не тільки генерувати Python-скрипти для обробки CSV-файлів, а й автоматично завантажувати файли, виконувати скрипти на сервері, зчитувати отримані результати, будувати діаграми і генерувати нові інструкції на основі проміжних даних.

Конкретні типові сценарії вже застосовуються у сучасному бізнесі, наприклад у сфері фінансових технологій, де LLM-агенти генерують індивідуальні робочі кабінети для кожного клієнта автоматично після підписання договору – від створення бази даних до налаштування інтеграції з банківськими API. Також у онлайн торгівлі агенти пишуть промо-лендінги, налаштовують канали комунікації через Telegram Bot API, генерують сценарії масових email-розсилок, шаблони лендингів та системи персоналізації. В освітніх проєктах агент формує динамічні навчальні платформи з реєстрацією, системою модулів, автоматичними тестами та завантаженням сертифікатів у PDF.

Проте при всій технологічності LLM-агентів їх треба застосовувати обґрунтовано. Задачі, де потрібна максимальна швидкість – наприклад, масовий парсинг даних, конвертація великих обсягів даних у різні формати, перетворення тексту чи структурованих даних – часто ефективніше вирішити через pipeline із класичних скриптів і одноразових LLM-запитів. Якщо бізнес-процес вже усталений, а специфікація чітка, дешевше й надійніше використати генерацію шаблонів через template-інструменти та традиційні build-системи. AI-агенти виправдані для реальних R&D-завдань, стартапів, виробництва прототипів, коли замовник або команда постійно коригують вимоги на ходу [2].

Серйозною перевагою LLM-агентів виступає їх здатність до автоматизації повного циклу розробки. Наприклад, агент може проаналізувати issue у JIRA, сформувати план розробки з оцінкою часу, згенерувати основні класи і функції, налаштувати автоматичне тестування через pytest-cov, пройтись по checklist якості, запустити лінтер, згенерувати pull request і самостійно зібрати звіт про покриття тестами. В окремих екосистемах агенти навіть інтегруються з MLOps-середовищем: після тренування моделі агент створює REST API для її розгортання, прописує dockerization, підключає Prometheus для моніторингу, тестує latency та додає алерти для DevOps-команди. Не менш важливо, що такі агенти можуть запускати бета-тестування, збирати поведінкові дані користувачів, автоматично аналізувати фідбек і пропонувати зміни у функціоналі та UI. Таким чином, до переваг використання анентів можна віднести наступне: 1) немає потреби в розробці складної системи AI; 2) здатні виконувати різноманітні завдання, пов'язані з обробкою текстової інформації; 3) надають швидкі відповіді, що робить їх оптимальними для використання в режимі реального часу.

Істотні ризики пов'язані з контролем якості: згенерований код часто містить дублювання, неефективні структури, пропущені валідації, а вразливості можуть бути внесені разом із шаблонними рішеннями. Є ризик появи технічного боргу, який важко виявити й усунути без залучення досвідчених фахівців. Крім того, LLM-агенти залежать від актуальності моделі й доступності API – збої або втрати зв'язку з платформою можуть зупинити весь процес розробки. Також слід зазначити питання безпеки: за неналежного налаштування агент може внести сторонній код, залишити тестові бекдори, або помилково розкрити чутливі дані у відкритому доступі. До недоліків використання агентів можна віднести: 1) невпевненість або нестійкість під час ведення тривалих діалогів; 2) спроможні генерувати різні варіанти відповідей на один і той самий запит; 3) позбавлені передбачуваності типових алгоритмів, що може ускладнити процес контролю; 4) часте звернення до LLM може бути затратним і призводити до затримок у роботі [3].

Для підвищення якості сучасні команди інтегрують у пайплайн додаткові перевірки: автоматичні статики-аналітики (наприклад, SonarQube), fuzzing-тести, верифікацію ліцензій сторонніх бібліотек, динамічне тестування продуктивності. Часто агенти комбінуються із внутрішніми репозитаріями шаблонів та best practices, що дозволяє формувати більш стандартизований і безпечний код. З боку бізнесу велику роль відіграє інтеграція агента з системами аналітики (BI/ETL), які одразу після релізу дозволяють збирати метрики використання й отримувати інсайти для наступних ітерацій розвитку.

Як приклад роботи LLM-агентів більш детально розглянемо систему автоматичної генерації проєктів із залученням LLM-агентів. Дана система має продуману структуру, чітку взаємодією компонентів і реалістичний підхід до розмежування відповідальностей між класами, агентами та користувачами. В основі архітектури в даному випадку лежить інтеграція Django як бекенду, React – як фронтенду та набору агентів, що реалізують різноманітну бізнес-логіку. Уся система організована пакетами, де кожен пакет представляє певний функціональний домен: користувачі, чати, генерація, налаштування, маршрутизація, компонентна логіка фронтенду тощо. Такий розподіл дозволяє швидко масштабувати та супроводжувати рішення, а також впроваджувати окремі поліпшення, не впливаючи на інші частини проєкту.

У пакеті Accounts визначається клас CustomUser, що містить не лише ідентифікаційні атрибути, а й зберігає абсолютний шлях до згенерованого проєкту користувача та статус активності. Це дозволяє системі ефективно відстежувати сесію, керувати правами доступу та виконувати персоналізацію бізнес-логіки для кожного користувача. Пакет Chats містить клас PlanningChat, який працює з історією взаємодії у JSON-форматі та має пряме відношення до користувача, що створює або володіє чатом. Це дозволяє фіксувати всю

логіку спілкування з LLM-агентом, підтримувати чат-контекст і забезпечувати зручне повернення до історії завдань.

Найсильнішою стороною реалізації даної системи є структурована ієрархія агентів у пакеті Generation. Базовий клас BaseAgent відповідає за загальні функції взаємодії з OpenAI API, опрацювання шляхів до проєктів та виконання типових запитів. На основі базового агента будуються спеціалізовані агенти, кожен з яких має чітко окреслені обов'язки: FunctionalAgent реалізує запис технічних завдань та результатів генерації у файл; NonFunctionalAgent здійснює перевірку чи валідацію плану; DoerAgent запускає механізм безпосередньої генерації та реалізації плану; InspectorAgent фокусується на інспекції згенерованого коду; UpdaterAgent займається оновленням існуючих проєктів та завдань. Такий підхід забезпечує максимальну гнучкість і розширюваність, а також дозволяє швидко додавати нових агентів із унікальними сценаріями використання, не змінюючи фундаментальну логіку системи.

Пакет Django Project містить компоненти для налаштувань (Settings) та маршрутизації (URLs), що забезпечує централізоване управління параметрами застосунку та спрощує масштабування. На фронтенді, у пакеті React Project, виділені ключові компоненти: AuthContext є ядром логіки автентифікації, ChatBox відповідає за організацію чат-сесії з агентом, UpdatingBox дозволяє користувачу оперативно отримувати інформацію про процес оновлення або результатів генерації. Це дозволяє створити зрозумілий, швидкий і гнучкий клієнтський інтерфейс для роботи розробника чи кінцевого користувача.

Загалом вся система спроєктована так, що користувачі взаємодіють із платформою через інтуїтивно зрозумілий вебінтерфейс. Типовий сценарій роботи може бути визначеним наступним чином: 1) користувач авторизується, 2) створює новий проєкт, 3) формулює вимоги, які передаються одним з агентів на бекенді, 4) формується план, 5) генерується код, 6) результати надходять у чат, де користувач може поставити додаткові запитання чи скорегувати завдання. Якщо потрібно оновити вже існуючий проєкт, UpdaterAgent приймає зміни, перевіряє актуальність, вносить корекції та повертає оновлений код на фронтенд. Вся взаємодія супроводжується чіткими статусами, повідомленнями про успішність чи помилки, а також підтримкою журналу подій для адміністрування.

З точки зору бази даних, дана структура побудована під максимальне розширення. Окрім типових таблиць для користувачів (CustomUser), управління сесіями (UserSession) та збереження проєктів (GeneratedProject), ведеться історія чатів (PlanningChat), що дозволяє швидко відновлювати контекст навіть при повторному запуску агента або після втрати з'єднання. Для забезпечення безпеки використовується сучасна система автентифікації на основі JWT, що дозволяє зручно масштабувати рішення з точки зору багатокористувацької роботи та захищеного обміну даними між фронтендом і бекендом.

Базова логіка агентів передбачає обробку помилок на стороні API, повторні спроби, ведення журналу використаних токенів для оптимізації витрат на AI-запити, а також підтримку декількох моделей для порівняння якості результатів генерації. Це дозволяє не лише автоматизовано створювати проєкти, а й оперативно реагувати на збої або обмеження з боку зовнішніх провайдерів.

Всі ключові сценарії використання системи деталізуються у вигляді usecase-описів та діаграм. Так, наприклад, сценарій створення нового проєкту охоплює весь ланцюг дій: від аутентифікації користувача, збору вимог і генерації плану – до перевірки коду й повідомлення про завершення. Оновлення проєкту передбачає як типове редагування, так і альтернативні сценарії з автоматичним виправленням помилок або відхиленням через недоступність зовнішніх сервісів. Система підтримує поділ на різні ролі користувачів: розробник зосереджений на генерації та редагуванні проєкту, адміністратор здійснює контроль, моніторинг і налаштування параметрів, а також аналізує логи взаємодій.

Щодо деталізації життєвих циклів об'єктів через state- і activity-діаграми, то вони чітко описують послідовність подій при генерації, оновленні чи управлінні проєктом. Це дозволяє виявляти вузькі місця, оптимізувати ключові процеси та гарантувати високий рівень якості кінцевого продукту, зменшуючи ризик виникнення помилок ще на етапі проєктування.

Всі взаємозв'язки в системі ілюструються через діаграми класів, компонентів, розгортання та пакетів. Це дає чітке уявлення про залежності між модулями, коректно організовану архітектуру та можливість швидкої заміни технологічних стеків у майбутньому, без втрати цілісності рішень.

Таким чином, розглянута структура системи демонструє сучасний підхід до автоматизації розробки через LLM-агентів. Використання чітких інтерфейсів, обґрунтоване розділення відповідальностей, підтримка розширення та гнучкості у взаємодії між компонентами роблять цю платформу максимально ефективною для задач генерації та супроводу програмних проєктів різної складності. Фокус на автоматизації рутинних та складних процесів, супроводжуваність через систематизовану документацію, чіткі діаграми та структуровану основу дозволяють не лише реалізувати сьогоднішні бізнес-вимоги, а й гарантують легку адаптацію до майбутніх змін технологічного ландшафту.

Підсумовуючи, автоматична генерація проєктів за допомогою LLM-агентів найбільш ефективна там, де потрібна швидка розробка, гнучкість та інтеграція багатьох технологічних компонентів. Для задач із чіткими алгоритмами та фіксованими вимогами краще застосовувати прості підходи – вузькі LLM-запити або класичні скрипти – адже вони споживають менше ресурсів і забезпечують вищу передбачуваність. Вибір між агентом, LLM чи традиційним рішенням має базуватись на детальному аналізі задачі й ресурсів, а не на загальному хайпі. Тільки тоді штучний інтелект стане не просто ще одним модним засобом, а справді продуктивним і безпечним інструментом.

References

1. Liu, J. (2024). Large Language Model-Based Agents for Software Engineering: A Survey. arXiv:2409.02977 [cs.SE].

2. Tan, S. H. (2024). Automatic Programming: Large Language Models and Beyond. arXiv:2405.02213 [cs.SE].
3. Sergii Cherbadzhy. AI-агенти vs. LLM vs. Агоритми. [DOUDay форум] Retrieved from: https://dou.ua/forums/topic/52895/

DOI 10.34132/mspc2025.01.14.08

Yelisieiev Oleksandr
Horban Hlib

AUTOMATIC PROJECT GENERATION WITH THE HELP OF LLM AGENTS

The thesis explores the use of LLM agents for the automatic generation of software projects, analyzing their capabilities, advantages, and risks compared to traditional algorithms and simple LLM queries. The author discusses the architecture of agents, their integration with modern technologies (Django, React, OpenAI API), and application scenarios in business, education, and financial technologies. The system of automatic project generation involving LLM agents is considered. The importance of making an informed choice between complex agents and simple solutions depending on project requirements to ensure efficiency, security, and minimize technical debt is emphasized.

Keywords: *LLM agents, large language models, automatic code generation, artificial intelligence, software development, Django, React, OpenAI API.*

Відомості про автора / Information about the Author

Єлісєєв Олександр, здобувач першого рівня вищої освіти (бакалавр) зі спеціальності 121 «Інженерія програмного забезпечення», Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна. E-mail: alexander0402112@gmail.com.

Yelisieiev Oleksandr, Student, Specialty 121 "Software Engineering", Petro Mohyla Black Sea National University, Mykolaiv, Ukraine. E-mail: alexander0402112@gmail.com.

Горбань Гліб, кандидат технічних наук, доцент, доцент кафедри інженерії програмного забезпечення, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна. E-mail: hlib.horban@chmnu.edu.ua, ORCID: https://orcid.org/0000-0002-6512-3576

Horban Hlib, PhD, Associate Professof of Department of Software Engineering, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine. E-mail: hlib.horban@chmnu.edu.ua, ORCID: https://orcid.org/0000-0002-6512-3576

© О. Єлісєєв, Г. Горбань *Стаття отримана редакцією 19.05.2025.*