**УДК 004.4:004.8**

**К.т.н., ст. викладач Хіцко Я.В., магістрант Черній Г.А.**

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**

# СПОСІБ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ ПРОГРАМІСТА

## Abstract

**Iana Khitsko, senior lecturer, PhD ; Hlib Chernii, student**
***Method and software for automation of programmer's routine tasks***

*This paper presents a method for automating frequent developer tasks directly inside the IDE. The approach combines deep context analysis (language, file structure, edit history, selected fragments) with cost-aware batching of LLM requests. We provide a structured review of existing tools and their limitations, formalize the method with compact equations, and give an analytical summary table of batching effects. External measurements reported in the literature indicate up to five-fold efficiency gains with batch prompting under few-shot in-context learning, while maintaining comparable performance.*

## Вступ

Сьогодні інтеграція штучного інтелекту в щоденні інженерні процеси є стійким трендом. Інструменти автодоповнення, пояснення коду, генерації тестів і рефакторингу вже входять до базового набору розробника. Водночас практичне застосування штучного інтелекту ускладнюється трьома чинниками: (1) відсутність прозорого контролю над формуванням промпта, (2) підписні моделі без тонкого керування вартістю, (3) слабка підтримка масових сценаріїв (коли потрібні десятки пояснень/тестів/рефакторингів за один підхід). Відповідно до наявних досліджень, приріст продуктивності стійко досягається тоді, коли інструмент глибоко враховує контекст коду й органічно інтегрований у IDE-процес, а накладні витрати взаємодії з моделлю знижені до розумних меж [1; 2].

## Постановка задачі

Мета дослідження – запропонувати та описати спосіб автоматизації рутинних дій програміста безпосередньо в IDE, який: виконує глибокий контекстний аналіз (мова, структура файлу, історія правок, виділені фрагменти); забезпечує прозорість промпта та керування параметрами генерації (вибір/заміна моделі, температура, формат і політики якості);

об’єднує низку підзавдань у пакет (batch), щоб зменшити вартість і затримки; узгоджується з реальним робочим процесом у IDE без прив’язки до певної реалізації.

Для досягнення мети необхідно: окреслити терміни, виконати огляд існуючих підходів і їхніх обмежень, формально описати спосіб та подати узагальнені результати у вигляді таблиці.

**Термінологія**

*LLM* (Large Language Model) – велика мовна модель, що генерує текст/код на підставі промпта.

*IDE* (Integrated Development Environment) — інтегроване середовище розроблення (редактор, дебагер, тести тощо).

*Промпт* (prompt) – структурована інструкція/запит до LLM із вимогами до формату та якості відповіді.

*Контекстна генерація коду* – формування промпта з урахуванням мови файлу, структури, історії змін і виділених фрагментів.

*Пакетна обробка* (batching) – групування кількох логічно незалежних підзавдань в один сеанс взаємодії з LLM для спільного використання інструкцій/прикладів і зменшення накладних витрат.

**Аналіз існуючих підходів та інструментів**

Наявні інструменти на базі штучного інтелекту (автодоповнення, пояснення, генерація тестів) підтверджують корисність підходу для типових інженерних задач, однак мають спільні обмеження: закритість промптів/архітектури, підписні моделі без тонкого контролю вартості, відсутність ефективної пакетизації серійних дій. Узагальнений огляд бекуенд практик демонструє, що приріст швидкодії й зниження людських помилок досяжні, якщо інструмент коректно «вплітається» в інженерні процеси та враховує контекст коду [1]. Додатково, інтеграція ШІ з усталеними інженерними інструментами (IDE, дебагери, профайлери) є критичною для масштабування ефекту в реальних процесах [2]. Узагальнюючи, це обґрунтовує потребу в керованому способі, що поєднує врахування контексту разом з економною пакетною взаємодією.

**Пропонований спосіб**

Запропонований спосіб полягає у поєднанні контекстного аналізу з пакетизацією підзавдань, щоби зменшити витрати токенів і при цьому утримати релевантність відповідей. Послідовність обробки підзавдань у межах запропонованого способу – від ініціації в IDE до контролю якості результатів – зображено на рисунку 1.

*Етап 1* – Збір контексту. Визначення languageId, витяг структурних елементів (класи, функції, сигнатури), короткий диф останніх змін, позиція курсора та виділені фрагменти.

*Етап 2* – Нормалізація підзавдань. Формування множини {$Q_1$,…,Q_b} (пояснити фрагмент, згенерувати тести, локальний рефакторинг) із мінімально достатнім локальним контекстом кожного фрагмента.

*Етап 3* – Batch-промпт. Спільний «пролог» (системні інструкції, політики якості, приклади) та індексований список підзавдань Q[i] із локальним контекстом. Відповідно до наявних досліджень, такий прийом — batch prompting – зменшує витрати майже обернено лінійно до розміру пакета й забезпечує до ≈5× виграшу за токенами/часом при b≈6 у few-shot ICL, зберігаючи порівняну якість [3].

*Етап 4* – Постобробка. Парсинг відповідей A[i], зіставлення з джерельними фрагментами, підготовка диф-вставок, чернеток тестів, коментарів.

*Етап 5* – Базовий контроль якості. Лінтери/компіляція/мінімальні тести; повторна пакетизація лише для проблемних Q[i].

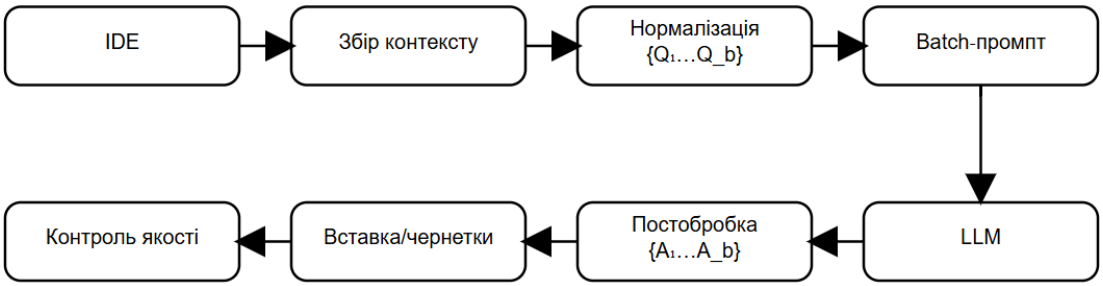


Рис. 1. Послідовність обробки: від IDE до контролю якості з пакетизацією підзавдань.

Вартість одного виклику LLM лінійно залежить від кількості токенів, включаючи як вхідні токени, так і згенеровані токен. Ефективність використання токенів $\eta$ визначається як частка токенів, витрачених на згенеровані токени в одному виклику LLM. Таким чином, чим більша частка токенів витрачається на згенеровані токени, тим економічнішою є загальна вартість:

$$\eta_{\text{звич}} = \frac{1}{K+1}$$
$$\eta_{\text{батч}} = \frac{b}{K+b} \quad (1)$$

де $b$ – це розмір пакета: кількість підзавдань/запитів, які об'єднуєно й відправлено за один виклик LLM; K – обсяг спільної частини промпта, який однаковий для всіх підзавдань і в батчі надсилається один раз.

З формули (1) випливає: чим більший b і чим вагоміший спільна частина K, тим вища відносна ефективність. Практичний ефект

підтверджено зовнішніми вимірюваннями на різних датасетах і моделях (GPT-3/3.5/4): зменшення витрат до ≈5× при b=6 із порівняною точністю [3].

**Результати експериментальних досліджень**

Подамо узагальнену табличну інтерпретацію очікуваного ефекту пакетизації в сценаріях з відносно великим спільним прологом і «легшими» виходами (пояснення, тести, локальні рефакторинги):

Таблиця 1

| Розмір пакета b | Очікуваний виграш за часом/токенами* | Коментар щодо якості |
|---|---|---|
| 1 | 1× | Індивідуальні виклики без економії |
| 3-4 | ~2–4× | Як правило, стабільна якість завдяки спільному прологу |
| 6 | до ~5× | Можливе невелике просідання на «важких» довгих входах |

Орієнтуючись на це, пакетизація дає найбільший виграш у задачах із коротким виходом (пояснення, сигнатури тестів, локальні рефакторинги) та відносно великим спільним прологом $K$. На практиці доцільно починати з b = 3–4, підтримувати стабільні і компактні інструкції, групувати схожі підзавдання в один пакет і чітко маркувати індекси.

За довгих або різнорідних виходів варто застосовувати мікробатчі та контролювати сумарний обсяг контексту, аби не наближатися до межі вікна моделі. Такий режим узгоджується з опублікованими спостереженнями щодо batch prompting та очікувано забезпечує кратне зменшення витрат без помітної втрати якості за помірних b [3].

**Висновки**

Запропонований спосіб та програмний прототип Dev Assist надають можливість значного скорочення часу, що витрачається на рутинні завдання програміста, з одночасною оптимізацією витрат на LLM-запити. Порівняно з вже наявними реалізаціями, підхід надає прозорість промпта, керування параметрами генерації, підтримку економної пакетизації та можливості роботи з безкоштовними та локальними моделями. Даний підхід робить рішення доступним як для індивідуальних користувачів, так і для команд. Подальші дослідження можна спрямувати на інтеграцію семантичного

аналізу коду, розширення підтримки мов та побудову централізованого бекенду для командного використання.

**Література**


1. Marzovanova M., Mihova V. Integration of Artificial Intelligence for Automation and Optimization in Backend Programming. ICAICTSEE-2024, UNWE, Sofia, 2024, pp. 151–157.
2. Zaripova R., Mentsiev A., Kovrizhnykh O. Advancing Parallel Programming Integrating Artificial Intelligence for Enhanced Efficiency and Automation. E3S Web of Conferences 460, 04017, 2023.
3. Cheng Z., Kasai J., Yu T. Batch Prompting: Efficient Inference with Large Language Model APIs, 2023.
4. OpenAI. Batch API documentation. [Електронний ресурс]. Режим доступу: https://platform.openai.com/docs/guides/batch.