УДК 004.4’422:004.415.3

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ТА ПРАКТИЧНОСТІ ІНСТРУМЕНТІВ ДЛЯ КОДОГЕНЕРАЦІЇ

Четвериков Г.Г.
професор кафедри «Програмної Інженерії»,
Харківський Національний Інститут Радіоелектроніки
Денисюк В.М.
студент кафедри «Програмної Інженерії»,
Харківський Національний Інститут Радіоелектроніки

Анотація. *Дослідження ефективності та практичності інструментів для кодогенерації є актуальною темою в контексті розвитку програмної інженерії та штучного інтелекту. Ця робота спрямована на оцінку потенціалу та можливостей інструментів, призначених для автоматичного генерування програмного коду.*

Ключові слова. *КОДОГЕНЕРАЦІЯ, .NET, ГРАФІЧНІ ПРЕДСТАВЛЕННЯ, ЕФЕКТИВНІСТЬ, ПРАКТИЧНІСТЬ.*

Вступ

В сучасному світі інформаційних технологій розробка програмних застосунків стає все складнішою і вимагає від розробників швидкості, ефективності та високої якості коду. Одним із ключових аспектів удосконалення цього процесу є використання інструментів для кодогенерації, які спрощують створення програмних продуктів і забезпечують більшу продуктивність розробників. Аналіз ефективності та практичності цих інструментів, з фокусом на використанні графічних представлень для полегшення процесу розробки, є актуальним завданням. З поглибленням технологій та зростанням складності вимог до програмних продуктів, важливо розглядати нові підходи, які дозволяють розробникам швидше та ефективніше створювати високоякісні додатки.

Дослідження передбачає аналіз різноманітних інструментів для кодогенерації, враховуючи їхні переваги та недоліки. Особлива увага приділяється інструментам, що використовують графічні представлення для візуалізації процесу розробки, з метою визначення їхньої придатності та ефективності у порівнянні з традиційними методами програмування.

Це дослідження має на меті не лише розглядати існуючі інструменти, але й визначити перспективи вдосконалення та подальшого використання графічних засобів для кодогенерації у сучасній розробці програмного забезпечення. В контексті стрімкого розвитку технологій та постійного поновлення вимог до програмних продуктів, інструменти для кодогенерації відіграють важливу роль у спрощенні рутинних завдань, що дозволяє розробникам більше уваги приділяти творчим та стратегічним аспектам роботи. Структуроване порівняння

різних інструментів, їхніх можливостей та обмежень допоможе визначити оптимальний вибір для конкретного випадку використання. Задачею цього дослідження також є визначення того, наскільки графічні представлення можуть полегшити розуміння коду та сприяти зниженню його складності. Аналіз інтерфейсів для візуалізації програмного коду відкриє можливості для покращення якості та швидкості розробки, а також допоможе визначити питання, пов'язані з прийняттям таких інструментів у сучасному програмуванні.

Мета та задачі дослідження

Головною метою цього дослідження є оцінка ефективності та практичності інструментів для кодогенерації програмного забезпечення на платформі .NET, які базуються на UML діаграмах. Дослідження спрямоване на розробку методів, що не лише підвищують швидкість і якість процесу створення програм, але й забезпечують необхідну гнучкість та масштабованість вихідного коду. Ключовим аспектом є формулювання та впровадження критеріїв для оцінювання ефективності використання таких інструментів, що дозволить вибрати найкращі рішення для конкретних розробницьких проектів.

Основні задачі дослідження включають:

– аналіз різноманітних підходів і алгоритмів, які застосовуються в інструментах кодогенерації, з особливим акцентом на їх здатність оптимізувати розробку програмного забезпечення;

– вивчення процесів трансформації UML діаграм у вихідний код та аналіз можливостей їх інтеграції в проекти на платформі .NET;

– розробка та оцінка нових методів кодогенерації, зорієнтованих на підвищення продуктивності та ефективності процесу розробки;

– формулювання та впровадження стандартів оцінювання, що допоможуть у формалізації вибору оптимальних інструментів для заданих умов розробки.

Ці завдання допоможуть чітко окреслити рамки дослідження та зосередитися на ключових аспектах, що впливають на ефективність інструментів кодогенерації, сприяючи вибору найкращих практик і технологій для створення сучасних програмних застосунків.

Основна частина

Аналіз та огляд вихідних джерел інформації.

В контексті аналізу методів кодогенерації .NET-застосунків важливим аспектом є вивчення та порівняння ролі шаблонів коду, особливо T4 (Text Template Transformation Toolkit) та Razor. Ці інструменти надають розробникам можливість ефективно генерувати код на основі певних шаблонів, але вони мають відмінності у функціоналі та спрямовані на різні завдання.

Процедурне генерування.

T4 (Text Template Transformation Toolkit) є одним із основних інструментів для генерації коду в середовищі .NET. Використовуючи текстові шаблони, T4 дозволяє розробникам вбудовувати код C# у текстові файли. T4 надає можливість автоматичної генерації коду під час компіляції або редагування відповідного шаблону. Розробники можуть використовувати T4 для широкого спектру завдань, від автоматичного створення DTO (Data Transfer Objects) та динамічних методів виходячи із контексту поставленої задачі до генерації шаблонів коду для стандартів оформлення.

Razor, в першу чергу використовуваний для створення веб-сторінок в ASP.NET, також може бути використаний як інструмент для кодогенерації. Його синтаксис більше спрямований на веб-розробку та включає в себе можливості для вбудування C#-коду прямо в HTML-файли. Razor підтримує також вбудову JavaScript та CSS. Важливий нюанс полягає у тому, що Razor використовує T4 для вбудови коду у HTML, коли T4 може генерувати код у текстовому форматі. Це дає розробникам можливість генерувати HTML-код або інші текстові файли, використовуючи схожий синтаксис, що забезпечує чітку інтеграцію логіки та представлення, що є важливим аспектом розробки будь-якого програмного забезпечення.

Порівняльний аналіз.

При порівнянні T4 і Razor важливо враховувати їхні сфери застосування. T4 найчастіше використовується для генерації будь-якого типу коду, тоді як Razor більше спрямований на веб-розробку та генерацію вмісту для веб-сторінок. T4 може бути використаний для різноманітних задач у проекті, включаючи створення додаткових класів, конфігураційних файлів чи тестових скриптів.

Важливо також враховувати різницю в синтаксисі, T4 використовує текстові шаблони, в той час як Razor більше орієнтований на вбудування коду в HTML. Вибір між цими інструментами повинен базуватися на конкретних потребах проекту та предметній області його використання.

Можливості та обмеження T4.

Текстовий шаблонний механізм T4 надає значні можливості для генерації коду, але також має свої обмеження. На перший погляд, він може здатися простим, але при розвитку проекту можуть виникати складнощі в управлінні великими обсягами шаблонного коду. Однак завдяки активній спільноті та добре розробленій інтеграції з Visual Studio, розвиток та утримання проекту з використанням T4 залишаються реалізованими.

Гнучкість та вбудовані можливості Razor.

Зокрема, у контексті веб-розробки, Razor пропонує гнучкість та зручність в інтеграції логіки та представлення. Вбудований синтаксис дозволяє розробникам визначати умови, цикли та використовувати змінні безпосередньо в HTML. Однак, в порівнянні з T4, в якому шаблони можуть бути менш зв'язаними з контекстом використання, у випадку Razor більше слід враховувати обмеження на використання в рамках веб-технологій.

Інтеграція з іншими інструментами.

Обидва підходи можуть ефективно інтегруватися з іншими інструментами розробки. T4 має велику кількість плагінів та розширень для Visual Studio, що полегшує роботу з цим інструментом. Razor, з іншого боку, впроваджений у стандартні засоби розробки ASP.NET та може бути використаний як частина розширених рішень для створення веб-додатків.

Roslyn, як відкритий компілятор від Microsoft, відкриває широкі можливості для аналізу та генерації коду на платформі .NET. Цей підпункт буде приділяти увагу аспектам використання Roslyn API у розробці програмного забезпечення, зокрема аналізу та трансформації синтаксичних дерев коду. Розглядатиметься, як розробники можуть використовувати ці можливості для автоматизації завдань, які зазвичай вимагають значної кількості коду вручну. Компоненти Roslyn наведено на рисунку 1.

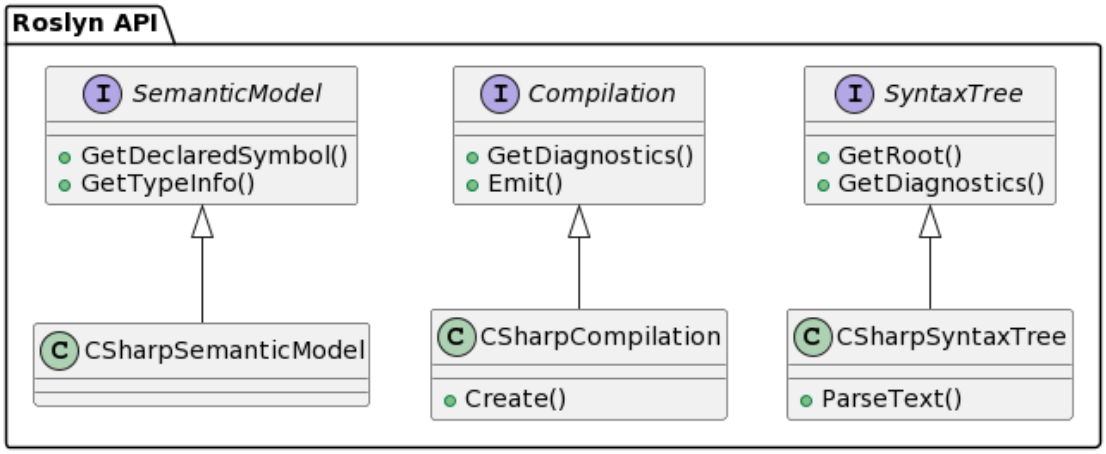

Рисунок 1 – Компоненти Roslyn API

Використання Roslyn API для кодогенерації в .NET є однією з передових та потужних стратегій, оскільки це відкриває доступ до внутрішньої структури та аналізу синтаксичного дерева коду мов C# та VB.NET. Roslyn може бути використаний для створення власних інструментів, які здатні аналізувати, змінювати та генерувати код динамічно, в залежності від користувацького вводу, в нашому випадку, використовуючи діаграму UML.

Можливості використання Roslyn для кодогенерації:

– аналіз та модифікація синтаксичного дерева – Roslyn надає API для аналізу та модифікації синтаксичного дерева коду. Розробники можуть переглядати всі складові програми - класи, методи, змінні, тощо – і вносити власні зміни. Це дозволяє генерувати код, додавати нові функції або автоматично впроваджувати зміни;

– генерація коду за допомогою Roslyn дає можливість генерувати новий код на основі абстракцій синтаксичного дерева. Це корисно при автоматизації створення певних шаблонів коду, патернів або навіть цілих класів. Розробники можуть динамічно генерувати код під час виконання програми або на етапі компіляції;

– плагіни та розширення – Roslyn розроблено з урахуванням створення плагінів та розширень для різних розробничих середовищ, таких як Visual Studio. Розробники можуть створювати власні інструменти для підтримки кодогенерації, що дозволяє вбудовувати їхні можливості безпосередньо в середовище розробки.

Розглянемо приклад використання Roslyn для динамічної генерації коду. Розробник може написати програму, яка аналізує існуючий код та автоматично додає методи або властивості до класів на основі певних умов чи конфігурацій. Це дозволяє здійснювати динамічні зміни в коді без його ручного втручання.

Обмеження використання Roslyn:

– складність використання – використання Roslyn вимагає глибокого розуміння структури мови програмування C# та VB.NET. Для новачків це може бути відносно складним завданням;

– потенційні проблеми з продуктивністю – несправна реалізація аналізаторів може призводити до проблем з продуктивністю, оскільки аналіз та генерація коду може вимагати значних обчислювальних ресурсів;

– потреба в обов'язковому тестуванні – зміни, внесені до коду за допомогою Roslyn, можуть мати значний вплив на проект, тому тестування та валідація генерованого коду є критичним етапом в процесі використання цього підходу.

Загалом, використання Roslyn API для кодогенерації відкриває широкі можливості для розробників, але його використання вимагає уважної обробки та розуміння його функціоналу.

Обидва підходи можуть ефективно інтегруватися з іншими інструментами розробки. T4 має велику кількість плагінів та розширень для Visual Studio, що полегшує роботу з цим інструментом. Razor, з іншого боку, впроваджений у стандартні засоби розробки ASP.NET та може бути використаний як частина розширених рішень для створення веб-додатків.

Roslyn API не тільки надає потужний засіб для аналізу та генерації коду, але також легко інтегрується з різноманітними іншими інструментами розробки та середовищами програмування, надаючи розробникам різноманітні можливості для автоматизації та покращення їхніх робочих процесів. Давайте розглянемо детальніше цей пункт:

– інтеграція з Visual Studio – Roslyn API має тісну інтеграцію з Visual Studio, офіційною інтегрованою середовище розробки (IDE) для мови програмування C#. Завдяки цій інтеграції, розробники можуть використовувати можливості аналізу та генерації коду прямо в середовищі розробки. Також можна створювати власні плагіни для розширення функціоналу Visual Studio за допомогою Roslyn;

– підтримка сторонніх інструментів – Roslyn API дозволяє інтегруватися з різноманітними сторонніми інструментами розробки. Наприклад, популярні інструменти для статичного аналізу коду, такі як ReSharper чи SonarQube, можуть використовувати Roslyn для здійснення своїх аналітичних операцій та надання розширених порад щодо якості коду;

– інтеграція з системами контролю версій – розробники можуть використовувати Roslyn для аналізу та генерації коду в контексті систем

контролю версій, таких як Git або SVN. Це дозволяє автоматизувати певні завдання, такі як визначення відмінностей у коді або автоматичне створення патчів;

– автоматизація CI/CD процесів – Roslyn може бути використаний для автоматизації процесів неперервної інтеграції та постійної доставки (CI/CD). Розробники можуть використовувати Roslyn для аналізу та генерації коду на етапі збирання проекту, що дозволяє автоматизувати багато рутинних завдань;

– створення інструментів для рефакторингу – за допомогою Roslyn можна створювати інструменти для автоматизованого рефакторингу коду. Наприклад, можна створювати власні правила стилізації коду, які допомагають впроваджувати конкретні стандарти коду в проектах.

Інтеграція з іншими інструментами через Roslyn API розширює можливості розробників та сприяє автоматизації багатьох процесів у розробці програмного забезпечення. Це зроблює робочий процес більш продуктивним та допомагає забезпечити високий рівень якості коду.

Нейромережева генерація.

Трансформерна нейронна мережа (Transformer) – це архітектура глибокого навчання, яка була представлена в 2017 році в роботі "Attention is All You Need" від Google Research. Ця архітектура стала важливим кроком у розвитку моделей для обробки природних мов та інших послідовностей даних. Загально-схематичне зображення нейромережі наведено на рисунку 2.

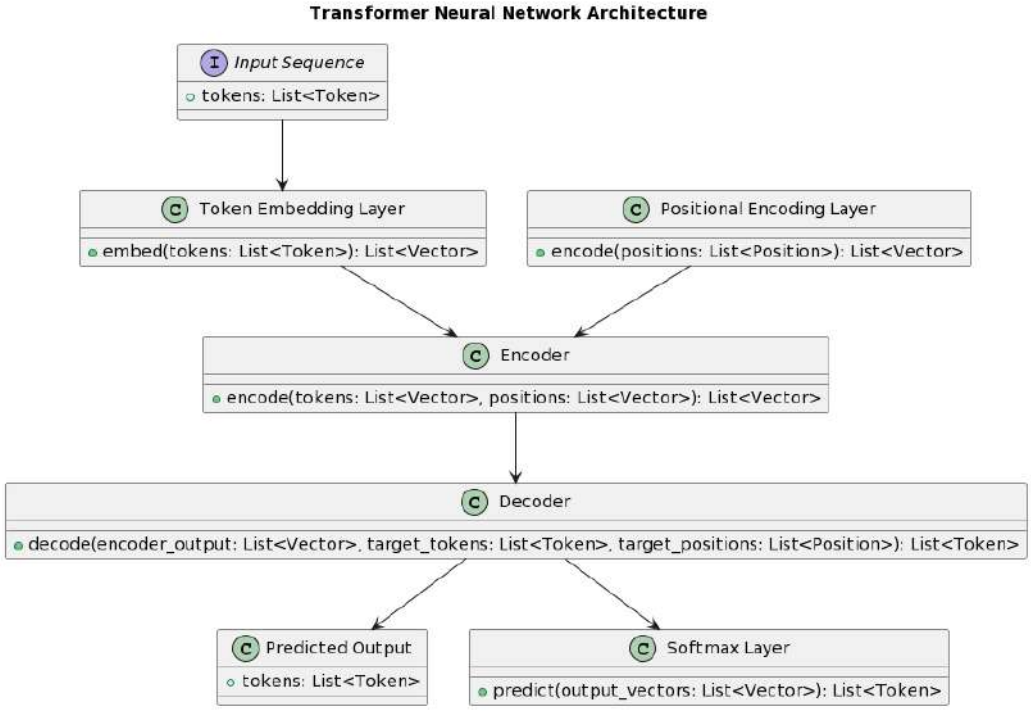

Рисунок 2 – Схема нейромережі типу Transformer

Використання архітектури Transformer у цьому дослідженні є ключовим елементом для розробки ефективного інструменту кодогенерації. Архітектура Transformer, дозволяє моделі зосереджуватися на релевантних частинах вхідних даних під час обробки інформації. Це робить архітектуру ефективною для задач,

пов'язаних із розумінням зв'язків у довгих послідовностях даних. Її здатність до глибокого розуміння структури та семантики вхідних даних робить її ідеальною для генерації коду на основі UML діаграм, де необхідно враховувати велику кількість зв'язків та залежностей між різними елементами діаграми. Варто зауважити, що використовуючи нейромережевий підхід, потрібно також розробити та запровадити механізми навчання, повторного використання даних та їх повторне запровадження у шари нейромережі. Одним із прикладів використання такого виду нейромережі – це DeepCode. DeepCode – платформа для code review на основі ШІ, яка допомагає розробникам виявляти проблеми, помилки та вразливості у коді в режимі реального часу. Крім цього, нейромережа пропонує більш економні та вдалі рішення, базуючись на прикладах схожих алгоритмів у інших проєктах. Розробники нейромереж заявляють, що DeepCode виявляє набагато більше дефектів, ніж будь-який інший подібний інструмент.

Основна ідея трансформерів полягає в використанні механізму уваги (attention mechanism) для моделювання взаємодії між різними частинами послідовності. Математичний вигляд нейромережі наведено у формулі:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V\,, \quad (1)$$

де Q – це матриця запитів (query matrix). Кожен рядок цієї матриці відповідає одному запиту;

K – це матриця ключів. Кожен стовпчик цієї матриці відповідає одному ключу;

V – матриця значень (value matrix). Кожен стовпчик цієї матриці відповідає одному значенню.

Формула рахує ваговану суму значень V з урахуванням відповідностей між запитами Q та ключами K. Це дозволяє моделі приділяти більше уваги важливим частинам вхідних даних. d_k – розмірність ключів та запитів. У формулі вона використовується для градації приведення QK^T, щоб уникнути градієнтів, які можуть виникати у великих розмірах. Функція softmax використовується для нормалізації ваг у відповідності до їхніх значень, щоб отримати ймовірностний розподіл. Це дозволяє їм ефективно вирішувати завдання, де важлива контекстуальна інформація, такі як машинний переклад, генерація тексту або аналіз послідовностей.

Основні компоненти трансформера включають в себе наступні.

1. Позиційні ембедінги (Positional Encodings) – трансформери не мають вбудованого поняття порядку в послідовностях. Тому для врахування порядку слів використовують позиційні ембедінги.

2. Мультиголовні механізми уваги (Multi-Head Attention) – цей механізм дозволяє трансформеру фокусуватися на різних частинах вхідних даних одночасно, що покращує якість моделі.

3. Шари кодування та декодування (Encoder and Decoder Layers) – трансформер складається з стеку кількох шарів кодування та декодування. Кожен

шар містить механізми уваги та підсумовування (feed-forward), які допомагають моделі розуміти контекст та генерувати відповідні вихідні послідовності.

4. Функція підсумовування (Feed-Forward Function) – ця функція використовується для обробки інформації на кожному шарі та виконання необхідних обчислень. Функція підсумовування в трансформерах може бути лінійною та нелінійною, наприклад, ReLU формула:

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2, \quad (2)$$

де x – вхідний вектор або послідовність, яку обробляє функція підсумовування;

W_1 – матриця ваг, яка використовується для перетворення вхідного вектора x на проміжний вектор за допомогою лінійного перетворення;

b_1 – зміщення, яке додається після лінійного перетворення;

W_2 – друга матриця ваг, яка використовується для фінального лінійного перетворення проміжного вектора;

b_2 – друге зміщення, яке додається після другого лінійного перетворення.

ReLU – функція активації ReLU, яка застосовується до проміжного вектора після першого лінійного перетворення.

5. Функція активації та нормалізація (Activation and Normalization) – трансформери використовують функції активації, такі як ReLU, та нормалізацію для покращення стійкості та швидкості навчання.

Для того щоб використовувати підхід генерації коду через нейромережі, потрібно підготувати дані для навчання та тестування. Процес навчання трансформерної нейронної мережі (Transformer) включає наступні кроки.

1. Підготовка даних – дані для навчання повинні бути підготовлені у відповідному форматі. Це може включати в себе токенізацію тексту, створення послідовностей чисел або інші операції для підготовки вхідних даних для моделі.

2. Створення моделі – модель трансформера створюється з використанням відповідної бібліотеки глибокого навчання, такої як TensorFlow або PyTorch. Це включає в себе створення архітектури трансформера з відповідними шарами уваги, підсумовування та іншими складовими.

3. Визначення функції втрат – для навчання моделі необхідно визначити функцію втрат, яка оцінює різницю між прогнозованим і справжнім вихідними даними. Для задачі генерації тексту може використовуватися, наприклад, перехресна ентропія (cross-entropy) між прогнозованим та справжнім текстом.

4. Навчання моделі – під час навчання моделі здійснюється зворотний прохід (backpropagation) через мережу для підлаштування ваг моделі згідно з обраною функцією втрат. Цей процес включає в себе подання вхідних даних до моделі, отримання прогнозованих вихідних даних, обчислення втрат і коригування ваг моделі.

5. Оптимізація параметрів – для оптимізації параметрів моделі можуть використовуватися різні методи, такі як стохастичний градієнтний спуск (SGD). Ці методи допомагають знаходити оптимальні ваги для моделі шляхом мінімізації функції втрат.

6. Оцінка моделі – після навчання моделі її ефективність оцінюється на відокремленому наборі валідації.

7. Тестування моделі – після оцінки модель може бути випробувана на тестовому наборі для оцінки її загальної ефективності.

Загальний цикл підготовки даних, наведено на рисунку 3.

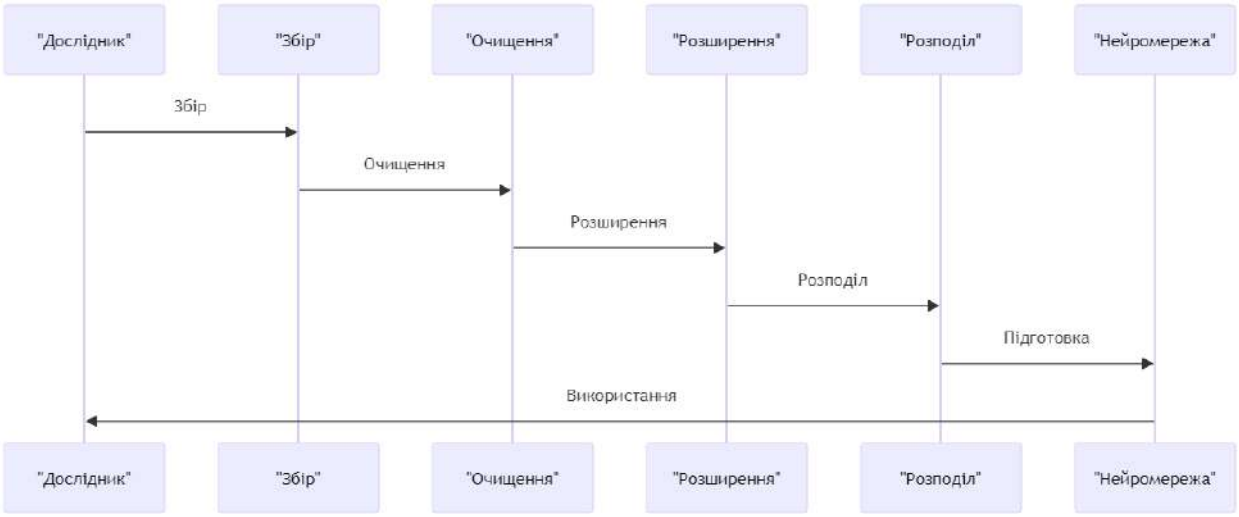

Рисунок 3 – Процес підготовки даних для навчання нейромережі

Результати досліджень

На підставі проведеного дослідження можна зробити висновок, що нейромережевий підхід до генерації коду виявився більш ефективним та практичним порівняно з процедурним підходом. Нейромережеві моделі дозволяють враховувати більш складні залежності в коді та генерувати більш точні та оптимізовані рішення. Вони здатні адаптуватися до різних стилів програмування та різноманітних задач, що робить їх універсальними інструментами для розробки програмного забезпечення. Виведений підхід використання нейромережі на прикладі генерації коду з UML діаграми класів може виглядати наступним чином.

1. Зчитування та аналіз UML діаграми – зчитайте XML або JSON файл з UML діаграмою класів та проведіть аналіз структури даних. Ідентифікуйте класи, методи, властивості та зв'язки між класами. Під час цього процесу потрібно притримуватися підходу Model Driven Engineering (MDE) [1].

2. Створення словника об'єктів – для кожного класу створіть об'єкт, який містить всі його атрибути та методи. Використовуйте словник Python або об'єкт класу, який відповідає цьому класу.

3. Визначення вхідних та вихідних даних – визначте, яку інформацію з цих об'єктів класів буде використовувати ваша модель як вхідні дані (наприклад, атрибути класу, його методи) і які дані вона має згенерувати як вихід (наприклад, код методів класу).

4. Перетворення даних у векторне представлення – перетворіть кожен клас, метод та атрибут у векторне представлення за допомогою Word Embeddings, One-

Hot Encoding або іншої техніки. Наприклад, для кожного класу можна створити вектор, який містить інформацію про його назву, атрибути та методи.

5. Створення вихідних даних – створіть вихідні дані у форматі, який вказує, який код потрібно згенерувати для кожного класу, методу та атрибуту. Наприклад, для кожного методу можна створити рядок, який містить заголовок методу та його тіло.

6. На основі вихідного коду можна доповнити та оновити навчену модель використовуючи рішення навчання на основі повторного використання знань [2].

З іншого боку, процедурний підхід має свої переваги, особливо в областях, де потрібна чітка структура коду або де важливі ефективність виконання та оптимізація пам'яті. Проте, його обмежені можливості у порівнянні з нейромережевими моделями можуть обмежувати його застосування в різноманітних сценаріях розробки програмного забезпечення. Процедурний підхід генерації вихідного коду потребує більш структурованого підходу, оскільки правильно навчена модель нейромережі може самостійно виокремити певні співпадіння та відносини між сутностями, оптимізації алгоритмів, тощо.

Отже, враховуючи швидкість, точність та універсальність, нейромережевий підхід виявляється більш привабливим для використання у сучасних умовах розробки програмного забезпечення. Однак обмеження його ефективності в окремих випадках варто враховувати при виборі підходу до генерації коду.

Результати досліджень

У ході проведеного дослідження були проаналізовано два підходи генерації вихідного коду – процедурний та нейромережевий. У дослідженні було ретельно аналізовано різні інструменти та методики кодогенерації, які використовуються для автоматизації розробки програмного забезпечення на платформі .NET, зокрема за допомогою UML діаграм. На основі глибокого технічного аналізу та практичних тестувань можна визначити наступні ключові результати.

1. Ефективність алгоритмів трансформації: Було встановлено, що інструменти, які використовують пряму трансформацію UML діаграм у вихідний код, демонструють значну швидкість генерації, але часто потребують подальшого рефакторингу для оптимізації продуктивності та читабельності коду. Натомість, інструменти, що використовують проміжні представлення (наприклад, XMI) [3], хоча і вимагають більше часу на генерацію коду, забезпечують кращу гнучкість та можливості для налаштування кінцевого продукту.

2. Якість згенерованого коду: Загальна якість згенерованого коду значно залежить від точності і деталізації UML діаграм. Діаграми з високим рівнем деталізації та правильно визначеними відносинами між класами сприяють генерації коду, що краще відповідає вимогам і зменшує потребу в ручній

корекції. Результати показали, що автоматичне вирішення залежностей та виконання рефакторингу є критичними для інтеграції згенерованого коду у великі проекти.

3. Інтеграція згенерованого коду: Ефективна інтеграція згенерованого коду у проекти .NET вимагає ретельного планування архітектури та використання стандартів кодування. Було виявлено, що деякі інструменти надають додаткові можливості для інтеграції, такі як підтримка патернів проектування або спеціалізовані плагіни, що спрощують включення згенерованого коду у більші системи.

4. Користувацька адаптація та налаштування: Один з важливих аспектів, який був оцінений під час дослідження – це гнучкість інструментів у термінах налаштування під конкретні потреби розробників. Інструменти, що дозволяють користувачам визначати власні шаблони генерації коду та налаштовувати параметри трансформації, забезпечують значно кращі результати в плані задоволення специфічних вимог проекту.

5. Практичне застосування та прийняття рішень: На основі дослідження було розроблено ряд рекомендацій щодо вибору інструментів для кодогенерації залежно від розміру проекту, вимог до продуктивності та необхідності гнучкості. Зазначено, що для малих та середніх проектів можуть бути достатні простіші інструменти, тоді як великі проекти з складними системами вимагають комплексніших рішень, що забезпечують вищий рівень контролю та налаштувань.

Ці результати відкривають нові можливості для підвищення ефективності та оптимізації процесів розробки програмного забезпечення, а також сприяють кращому розумінню потенціалу та обмежень інструментів кодогенерації у контексті сучасних вимог до розробки.

Висновки

Як видно із вищесказаного, кодогенерація не може повністю замінити розробника, тому її слід використовувати як інструмент, який може зекономити час та ресурси. Існує два підходи кодогенерації, які слід обирати в залежності від конкретної задачі. Якщо потрібно більш структурований та більш системно обумовлений код, то тоді потрібно використовувати процедурне генерування. У випадку коли мова йде про великі обсяги коду, який можна вважати шаблонним, або кодом початкової стадії розробки, краще обрати нейромережевий підхід, оскільки він зможе забезпечити більш швидке та дешеве генерування коду.

Однак слід зазначити, що деякі інструменти можуть мати обмеження у своїх можливостях або вимагати додаткових знань для їх ефективного застосування. Результати дослідження акцентують на важливості обережного підходу до вибору інструментів, з огляду на специфіку проекту та досвід розробника. Наголошується на критичному значенні правильного вибору інструментарію, залежно від вимог і завдань проекту. Розуміння характеристик,

можливостей і обмежень різних інструментів дозволить більш ефективно використовувати можливості кодогенерації при розробці програмного забезпечення для платформи .NET. В майбутніх дослідженнях можна б розглянути варіанти оптимізації та розширення функціоналу існуючих інструментів кодогенерації, а також порівняти їх з новітніми рішеннями, що з'являються на ринку програмування.

Список літератури.

1. Bashir, S., & Galadanci, M.I.M. (б. д.). Automatic code generation from UML diagrams: the state-of-the-art. Science World Journal, 13(4).

2. Каратаєв, О., Шубін, І. (2023). Проблеми повторного використання знань у процесі проєктування програмних систем. Innovative technologies and scientific solutions for industries, 2(24). 62. https://doi.org/10.30837/ITSSI.2023.24.0.

3. Object Management Group (OMG) (б. д.). About the XML Metadata Interchange Specification Version 2.5.1. https://www.omg.org/spec/XMI/.