УДК 004.5

АНАЛИЗ КАЧЕСТВА КОДА НА ЯЗЫКЕ C#, ГЕНЕРИРУЕМОГО НЕЙРОСЕТЕВЫМИ АССИСТЕНТАМИ

♦♦♦♦♦

ANALYSIS OF C# CODE QUALITY GENERATED BY NEURAL NETWORK ASSISTANTS

Крюкова Вероника Олеговна
студентка,
Кубанский государственный технологический университет
nika.kr1412@gmail.com

Бендега Дарья Андреевна
студентка,
Кубанский государственный технологический университет
bendegadara17@gmail.com

Вонарх Юлия Сергеевна
старший преподаватель
кафедры информационных систем и программирования,
Кубанский государственный технологический университет
jul_nosova@maill.ru

Аннотация. Представлены результаты экспериментальной оценки качества кода на языке C#, генерируемого современными нейросетевыми ассистентами – GPT-4o, DeepSeek-Coder и GitHub Copilot. Исследование проводилось на трёх типовых задачах, отражающих ключевые аспекты надёжности программного обеспечения: обработка исключительных ситуаций, многопоточное суммирование массива и управление файловыми ресурсами. Каждый сгенерированный фрагмент кода проходил последовательную проверку на этапах компиляции, статического анализа с использованием Roslyn Analyzers и ручного анализа. Установлено, что все модели успешно справляются с синтаксически корректной генерацией кода, однако задачи, требующие тонкого понимания многопоточности и управления ресурсами, вызывают существенные затруднения.

Ключевые слова: LLM, генерация кода, C#, статический анализ, качество программного обеспечения, многопоточность, управление ресурсами, GPT-4o, DeepSeek-Coder, GitHub Copilot.

Kryukova Veronika Olegovna
Students,
Kuban State Technological University
nika.kr1412@gmail.com

Bendega Darya Andreevna
Students,
Kuban State Technological University
bendegadara17@gmail.com

Vonarkh Yulia Sergeevna
Senior Lecturer of the Department
of Information Systems and Programming,
Kuban State Technological University
jul_nosova@maill.ru

Annotation. The article presents the results of an experimental evaluation of the quality of C# code generated by modern neural network assistants – GPT-4o, DeepSeek-Coder, and GitHub Copilot. The study was conducted on three typical tasks reflecting key aspects of software reliability: exception handling, multithreaded array summation, and file resource management. Each generated code fragment underwent sequential verification at the stages of compilation, static analysis using Roslyn Analyzers, and manual analysis. It was found that all models successfully handle syntactically correct code generation; however, tasks requiring a nuanced understanding of multithreading and resource management pose significant challenges.

Keywords: LLM, code generation, C#, static analysis, software quality, multithreading, resource management, GPT-4o, DeepSeek-Coder, GitHub Copilot.

Генеративные нейросетевые модели, такие как GitHub Copilot, ChatGPT и DeepSeek-Coder, активно внедряются в повседневную практику разработки программного обеспечения [1–3]. Эти инструменты позволяют автоматизировать написание шаблонного кода, предлагать алгоритмические решения и ускорять процесс разработки. Однако широкое распространение LLM-ассистентов ставит вопрос о качестве и надежности генерируемого ими кода [4, 5].

В данной работе проводится экспериментальная оценка качества кода на языке C#, сгенерированного современными LLM, на примере трех типовых задач: обработка исключительных ситуаций, многопоточное суммирование и управление файловыми ресурсами.

Постановка эксперимента

Для оценки качества кода, генерируемого нейросетевыми ассистентами, был спланирован и проведён эксперимент, включающий выбор исследуемых моделей,

формулировку типовых задач разработки на языке C#, а также определение метрик и инструментов анализа. Эксперимент построен таким образом, чтобы охватить ключевые аспекты надежности программного обеспечения: обработку исключительных ситуаций, многопоточность и управление ресурсами.

В эксперименте участвуют три генеративные модели, получившие наибольшее распространение в среде разработчиков:

- GPT-4o (OpenAI) – универсальная мультимодальная большая языковая модель, поддерживающая генерацию кода на различных языках программирования [1]. Обладает широкими знаниями в области программирования, но не специализируется исключительно на ней;
- DeepSeek-Coder (DeepSeek) – модель, специализированная на генерации программного кода [2]. Обучалась на корпусе из 2 триллионах токенов кода, что позволяет ей лучше воспроизводить паттерны системного программирования;
- GitHub Copilot (Microsoft / OpenAI) – коммерческий инструмент на базе архитектуры Codex, интегрированный в большинство современных сред разработки [3]. Является одним из наиболее широко используемым LLM-ассистентом в индустрии.

Для оценки качества кода были выбраны три задачи, отражающие ключевые аспекты надежности программного обеспечения: обработка ошибок, многопоточность и управление ресурсами.

Задача 1. Написать класс SafeCalculator с методами деления и извлечения квадратного корня. При делении на ноль или отрицательном подкоренном выражении выбрасывать исключения.

Задача 2. Написать метод, который принимает массив целых чисел и количество потоков, разбивает массив на части, суммирует каждую часть в отдельном потоке, а затем складывает результаты.

Задача 3. Написать класс FileProcessor, который открывает файл, читает из него строки и корректно закрывает ресурс.

Каждый сгенерированный фрагмент кода проходил следующие этапы проверки:

Компиляция – проверка синтаксической корректности.

Статический анализ – оценка предупреждений Roslyn Analyzers.

Ручной анализ – выявление логических ошибок, гонок данных, обработки исключений, утечек ресурсов.

Для систематизации результатов эксперимента использовались следующие метрики:

- Успешность компиляции.
- Количество предупреждений.
- Наличие критических ошибок.
- Итоговая оценка: «пригоден», «требует доработки», «непригоден».

На первом этапе выполнялась компиляция с использованием. NET 8 SDK. Все три исследуемые модели (табл. 1) продемонстрировали высокий уровень владения синтаксисом языка C# – ни в одном из девяти сгенерированных фрагментов кода не было обнаружено ошибок компиляции [4].

Данный результат свидетельствует о том, что современные LLM успешно усваивают формальные правила языка программирования и способны генерировать синтаксически корректные конструкции, пригодные для непосредственной компиляции.

Второй этап проверки выполнялся с помощью встроенного анализатора Roslyn Analyzers. Наибольшее количество предупреждений (табл. 1) (19) было зафиксировано у DeepSeek-Coder в задаче работы с файлами. Однако, как показал последующий ручной анализ, эти предупреждения носили преимущественно стилистический характер и не влияли на функциональную корректность кода.

Напротив, GitHub Copilot сгенерировал для той же задачи код с четырьмя предупреждениями (табл. 1), но ручная оценка оказалась ниже. Данное наблюдение указывает на то, что количество предупреждений статического анализа не является прямым индикатором качества кода.

Третий этап представлял собой ручной анализ, в ходе которого оценивалась логическая корректность решений, наличие гонок данных, утечек ресурсов, корректность обработки исключительных ситуаций и общее качество реализации.

Задача калькулятора оказалась наименее сложной для всех трёх моделей – все сгенерированные решения были признаны пригодными к использованию (табл. 1).

Таблица 1 – Результаты эксперимента

	Задачи	Компиляция		Статический анализ	Ручной анализ	Итог
		Ошибка	Предупреждение			
GPT-4o	Калькулятор	0	0	4	–	Пригоден
	Параллельная сумма	0	0	3	Гонка данных, Обработка исключений	Требует доработки
	Работа с файлом	0	1	4	Утечка ресурсов, Обработка исключений, Гонка данных	Требует доработки
DeepSeek-Coder	Калькулятор	0	0	4	–	Пригоден
	Параллельная сумма	0	1	5	–	Пригоден
	Работа с файлом	0	1	19	–	Пригоден
GitHub Copilot	Калькулятор	0	0	5	–	Пригоден
	Параллельная сумма	0	0	2	Гонка данных, Обработка исключений	Требует доработки
	Работа с файлом	0	4	3	–	Пригоден

Наиболее показательные результаты (табл. 1) были получены при ручном анализе задачи на параллельное суммирование. DeepSeek-Coder создала корректную реализацию. В то время как GPT-4o и GitHub Copilot допустили в этой задаче идентичную критическую ошибку: некорректный захват переменных в замыканиях при создании потоков в цикле. Данная ошибка приводит к состоянию гонки данных, при котором несколько потоков могут обрабатывать один и тот же диапазон массива, а некоторые диапазоны – оставаться необработанными. Результат вычислений становится непредсказуемым и зависит от времени выполнения потоков.

В задаче работы с файлами GPT-4o сгенерировал код, использующий итератор, что привело к потенциальной утечке ресурсов: если перебор строк прерывался досрочно, файловый дескриптор оставался открытым.

Анализ результатов

Проведённый эксперимент позволяет сделать ряд выводов о качестве кода, генерируемого современными LLM-ассистентами, а также выявить области, требующие дальнейшего совершенствования.

Наиболее сложной для LLM-ассистентов оказалась задача, требующая корректной реализации многопоточности. Ошибка захвата переменных в замыканиях, допущенная двумя из трёх моделей, является классическим примером проблемы, связанной с различием между временем захвата переменной и временем её фактического использования.

Задача работы с файлами выявила различия в подходах к управлению неуправляемыми ресурсами. Модель, сгенерировавшая код с использованием итератор, не учла особенностей ленивых вычислений, что привело к потенциальным утечкам ресурсов. Модели, использовавшие явную реализацию IDisposable и оператор using, по-

казали более высокий уровень понимания принципов управления ресурсами в NET. Данное наблюдение позволяет предположить, что LLM склонны воспроизводить паттерны, часто встречающиеся в обучающей выборке, при этом менее распространённые, но более безопасные паттерны могут игнорироваться [4].

Несмотря на высокое качество генерируемого кода, ни одно из решений не может быть признано полностью автономным без участия разработчика. Для получения полностью готового кода требуется обязательный этап ручной проверки, особенно для задач, связанных с многопоточностью и управлением ресурсами. Данный вывод согласуется с результатами исследований [4, 5], авторы которых отмечают, что LLM-ассистенты следует рассматривать как инструмент повышения продуктивности, но не как замену экспертной оценке.

Заключение

Проведённый эксперимент показывает, что современные LLM-ассистенты способны генерировать синтаксически корректный код на языке C#, успешно справляясь с типовыми задачами обработки исключений и управления ресурсами. Однако задачи, требующие тонкого понимания многопоточности и особенностей выполнения кода, остаются сложными для генеративных моделей.

DeepSeek-Coder, как специализированное решение, демонстрирует преимущества перед универсальными моделями в задачах программной инженерии. Полученные результаты подтверждают необходимость дальнейшего совершенствования LLM-ассистентов в области корректной обработки конкурентного выполнения и управления ресурсами.

Литература / References

1. OpenAI. GPT-4 Technical Report // arXiv:2303.08774.
2. DeepSeek-Coder: When the Large Language Model Meets Programming / D. Guo [et al.] // arXiv:2401.14196. – 2024.
3. The GitHub Copilot Experience: A Study of Productivity and Quality / C. Bird [et al.] // Proceedings of the 2023 ACM/IEEE International Symposium on Empirical Software Engineering (ESEM). – 2023.
4. Barke S. Grounded Copilot: How Programmers Interact with Code-Generating Models / S. Barke, M.B. James, N. Polikarpova // Proceedings of the ACM on Programming Languages (OOPSLA). – 2023.
5. Vaithilingam P. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models / P. Vaithilingam, T. Zhang, E.L. Glassman // Proceedings of the 2022 ACM Conference on Human Factors in Computing Systems (CHI). – 2022.