Программные системы и вычислительные методы
Правильная ссылка на статью:
Ишанхонов А.Ю., Пшиченко Д.В., Можаровский Е.А., Алуев А.С. Роль больших языковых моделей в интегрированных средах разработки нового поколения // Программные системы и вычислительные методы. 2024. № 4. DOI: 10.7256/2454-0714.2024.4.72022 EDN: KMTOBG URL: https://nbpublish.com/library_read_article.php?id=72022

Роль больших языковых моделей в интегрированных средах разработки нового поколения

Ишанхонов Азизхон Юнусхон

ORCID: 0009-0009-8934-6289

магистр; кафедра металловедения цветных металлов; Национальный исследовательский технологический университет "МИСИС"

119049, Россия, г. Москва, Ленинский пр-т, 4, стр. 1

m180119@edu.misis.ru

Пшиченко Дмитрий Викторович

ORCID: 0009-0006-8866-8057

независимый исследователь

119049, Россия, г. Москва, ул. Шаболовка, 26-28

dmitry.pshychenko@rambler.ru

Можаровский Евгений Александрович

ORCID: 0009-0005-9957-1632

независимый исследователь

119991, Россия, г. Москва, ул. Ленинские горы, 1

mozharovsky_ea@rambler.ru

Алуев Андрей Сергеевич

ORCID: 0009-0001-6737-7545

магистр; Уральский Федеральный Университет

620062, Россия, г. Екатеринбург, ул. Мира, 19

aluev_andrei@rambler.ru

Статья из рубрики "Методы, языки и модели человеко-машинного взаимодействия"

DOI:

10.7256/2454-0714.2024.4.72022

EDN:

KMTOBG

–

дата направления статьи в редакцию:

18-10-2024

Аннотация: В статье рассматривается роль больших языковых моделей (Large Language Models, LLM) в интегрированных средах разработки (Integrated Development Environment, IDE) нового поколения. Изучаются инструменты, такие как GitHub Copilot, IntelliCode и Alice Code Assistant, в контексте их использования в программировании. Авторы исследуют, каким образом LLM позволяют автоматизировать ключевые задачи разработки, включая автодополнение кода, выявление ошибок, рефакторинг и генерацию фрагментов программного кода, и как автоматизация приводит к повышению эффективности разработки и улучшению качества конечного продукта. Особое внимание уделяется влиянию использования LLM на когнитивные процессы разработчиков, их способность к решению творческих задач, а также на мотивацию и профессиональные навыки. Так же обсуждаются этические аспекты внедрения LLM. Обзор существующих интегрированных сред разработки, в которых применяются большие языковые модели. Оценивались функциональные возможности LLM для автодополнения кода, генерации фрагментов, выявления и исправления ошибок. Применялись сравнительные методы для оценки эффективности LLM по сравнению с традиционными средствами разработки. Новизна исследования заключается в комплексном анализе применения LLM в современных IDE, а также в выявлении их потенциала для повышения продуктивности разработчиков и улучшения качества программного кода. Сделан вывод о том, что интеграция LLM в IDE позволяет не только ускорить процесс создания кода, но и существенно повысить его качество за счет интеллектуальной поддержки и автоматизации рутинных задач. Однако выявлены и ограничения, связанные, в частности, с когнитивной нагрузкой, этическими вопросами и необходимостью сохранения баланса между автоматизацией и развитием навыков программистов. Авторы отмечают, что для успешной интеграции LLM необходим продуманный и ответственный подход, предполагающий баланс между автоматизацией и сохранением творческого потенциала программистов.

Ключевые слова:

большие языковые модели, интегрированные среды разработки, автоматизация программирования, улучшение кода, искусственный интеллект, автодополнение кода, программные системы, машинное обучение, оптимизация процессов разработки, анализ данных

Введение

Большие языковые модели (Large Language Models, LLM), основанные на архитектуре трансформеров, демонстрируют значительный прогресс в области искусственного интеллекта (artificial intelligence, AI) и обработки естественного языка. Изначально разработанные для решения задач, связанных с текстовой обработкой, таких как машинный перевод, генерация текстов и ответы на вопросы, они быстро нашли широкое применение в различных отраслях, включая программирование и разработку программного обеспечения (ПО). В современных интегрированных средах разработки

(Integrated Developer Environment, IDE) LLM являются важным инструментом, способным существенно автоматизировать процессы, повысить эффективность разработки и улучшить качество кода.

Интеграция LLM в IDE предоставляет возможность автоматизации множества задач, включая автодополнение кода, обнаружение и исправление ошибок, а также генерацию фрагментов ПО. Эти функции позволяют разработчикам сократить время, затрачиваемое на выполнение рутинных операций, снизить количество ошибок и сосредоточиться на решении более сложных и творческих задач. Кроме того, LLM способствуют интеллектуальной поддержке, предоставляя рекомендации по оптимизации и улучшению кода, что может значительно повысить производительность и качество работы.

Однако внедрение LLM в процессы разработки ПО порождает ряд вопросов и вызовов. Одним из ключевых аспектов является влияние таких моделей на когнитивные процессы разработчиков, их способность к обучению и самосовершенствованию, а также на профессиональные компетенции. Возникает необходимость оценки того, как постоянное взаимодействие с LLM сказывается на нагрузке программистов, их мотивации и способности к творческому мышлению. Этические аспекты применения LLM в программировании, включая вопросы безопасности данных, защиты интеллектуальной собственности и возможные риски, связанные с чрезмерной зависимостью от технологий AI, являются особенно актуальными.

Целью данной статьи является анализ роли LLM в IDE нового поколения. Делается акцент на когнитивных и социальных аспектах взаимодействия разработчиков с LLM, а также на оценке их влияния на эффективность программирования, на изменение методов работы и взаимодействие в командах.

Основная часть. Эволюция LLM и их применение в IDE

Первые LLM, созданные в 1990-е гг., основывались на статистических подходах, таких как n-граммы и скрытые марковские модели. **N-граммы** – это последовательности из *n* элементов (слов, символов или других единиц), которые встречаются в тексте или данных [1]. В контексте обработки естественного языка (NLP) n-граммы используются для моделирования текста и предсказания следующих элементов на основе предыдущих. Например, для предложения «Машинное обучение помогает решать задачи» биграммы (последовательности из двух элементов): «машинное обучение», «обучение помогает», «помогает решать», «решать задачи».

Скрытые марковские модели основаны на предположении, что вероятность появления каждого элемента кода или текста зависит только от нескольких предыдущих элементов, а не от всей последовательности [2]. Такие модели вычисляют вероятности слов в тексте или следующего шага при написании кода на основе наблюдений из тренировочных данных, где оцениваются частоты появления элементов (рис.1).

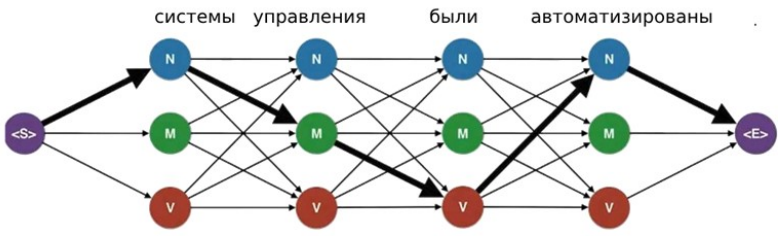

Рисунок 1. Схема марковской модели

Несмотря на свою популярность и широкое применение на ранних этапах разработки интеллектуальных инструментов для IDE, такие технологии, как n-граммы и марковские модели, обладают существенными ограничениями. Эти методы основаны на анализе локального контекста, что не позволяет учитывать долгосрочные зависимости между элементами кода или текста. В IDE это означает, что такие модели могут упускать важные связи между переменными, функциями или блоками кода, особенно в крупных проектах с длинными зависимостями. Игнорирование более глубоких семантических связей ограничивает точность автодополнения, поиска ошибок и анализа кода, так как марковские модели и n-граммы не могут эффективно учитывать глобальную структуру программы.

Существенным этапом в развитии LLM стало появление **Recurrent Neural Network** (RNN) в начале 2000-х гг. [3]. Эти модели способны сохранять контекст предыдущих элементов в коде или тексте на протяжении более длинных последовательностей, что позволяет IDE лучше понимать структуру и логику программ (рис.2).

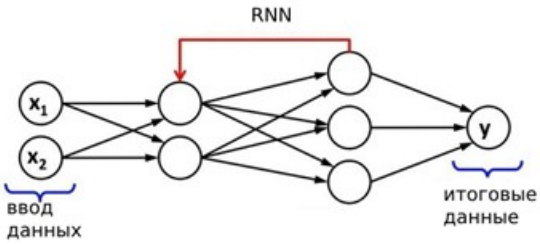

Рисунок 2. Схема RNN

Однако даже более продвинутые варианты RNN, такие как LSTM (The Long Short-Term Memory) и GRU (Gated Recurrent Unit), сталкиваются с проблемой обработки длинных кодов, поскольку в них наблюдается постепенное снижение количества важной информации по мере роста длины последовательностей.

В 2017 году появилась архитектура трансформеров, которая изменила подход к языковому моделированию. **Трансформеры** используют механизм внимания (attention), который позволяет моделям сосредоточиться на значимых частях текста, независимо от того, где они находятся в последовательности (рис.3).

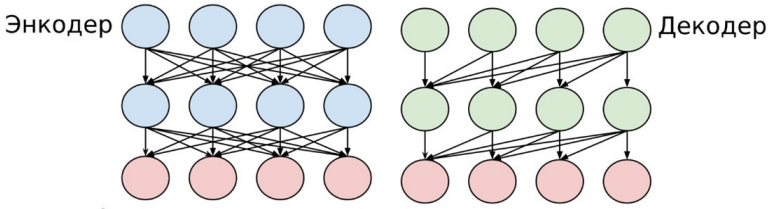

Рисунок 3. Схема архитектуры трансформеров

В IDE блок энкодера играет важную роль в преобразовании исходного кода или текстовых данных в промежуточные представления, которые затем передаются через многослойную архитектуру внимания. Это позволяет IDE анализировать взаимосвязи между различными элементами кода, выявляя ключевые структуры и зависимости, независимо от их местоположения в кодовой базе. Механизм внимания в энкодере обеспечивает взаимодействие между любыми элементами последовательности, будь то переменные, функции или блоки кода, что позволяет эффективно обрабатывать долгосрочные зависимости, такие как связи между определением переменной и ее использованием в других частях программы.

Такой подход значительно улучшает работу IDE по сравнению с предыдущими моделями, такими как RNN и LSTM, которые испытывали трудности с учетом удаленных зависимостей в коде [4]. Это нововведение стало важным шагом в развитии инструментов для автодополнения, анализа ошибок и рефакторинга кода. Блок декодера использует промежуточные представления, созданные энкодером, для генерации новой последовательности кода, например, для автодополнения функции, генерация тестов или предсказания следующего шага разработки.

Модели на основе трансформеров, такие как BERT (Bidirectional Encoder Representations from Transformers) и GPT (Generative Pre-trained Transformer), широко применяются в различных сферах, включая задачи, связанные с программированием. **BERT,** благодаря своей двунаправленной архитектуре, обрабатывает текст, учитывая контекст как слева, так и справа от каждого слова, что делает его особенно полезным в задачах анализа и рефакторинга кода, автодополнения и поиска ошибок [5]. В IDE это позволяет более точно понимать намерения разработчиков и предлагать оптимальные решения для завершения кода или улучшения его структуры. **GPT** с его однонаправленным подходом к генерации, особенно полезен в задачах, связанных с автодополнением и генерацией кода на основе текстовых описаний [6]. В интегрированных средах разработки GPT может использоваться для написания новых фрагментов кода, предсказания следующего шага в программировании, создания шаблонов и автогенерации тестов.

Роль LLM в трансформации современных IDE

Внедрение LLM в современные IDE является значимым этапом эволюции инструментов программирования. LLM обладают способностью анализировать код, предсказывать следующие фрагменты и предлагать исправления, что значительно ускоряет процесс разработки и улучшает его качество. Это не только облегчают рутинные задачи, но и предоставляют разработчикам интеллектуальную поддержку на всех этапах создания ПО (таблица 1).

Таблица 1. Влияние LLM на IDE [7, 8]

Аспект применения	Описание	Преимущества для разработки	Примеры инструментов
Автодополнение кода	Модели прогнозируют следующий фрагмент кода на основе текущего контекста.	Ускорение написания кода, снижение количества синтаксических ошибок.	GitHub Copilot, IntelliCode
Анализ кода в реальном времени	Автоматическая проверка кода на наличие ошибок; предложения по оптимизации во время написания.	Улучшение структуры и качества кода, уменьшение числа ошибок на этапе разработки.	DeepCode, CodeGuru
Обнаружение ошибок и исправление синтаксиса	Модели выявляют логические и синтаксические ошибки с предложением их исправления.	Повышение общей надежности кода, сокращение времени на отладку.	Tabnine, Kite
Прогнозирование и автоматизация рутинных задач	Автоматизация повторяющихся операций, таких как генерация шаблонного кода или создание тестов.	Снижение когнитивной нагрузки разработчиков, ускорение выполнения стандартных задач.	Codex, Replit

Применение LLM в IDE значительно преобразует процесс программирования, предлагая

новые возможности для автоматизации и интеллектуальной поддержки. Эти технологии не только повышают эффективность разработки за счет ускорения рутинных операций, но и влияют на качество создаваемого ПО, снижая вероятность ошибок на ранних этапах [9]. Внедрение LLM способствует более интуитивному взаимодействию с кодом, позволяя разработчикам сосредоточиться на решении творческих и сложных задач, минимизируя при этом время, затрачиваемое на поиск и исправление ошибок.

Успешная интеграция LLM требует учета возможных ограничений, таких как зависимость от качества обучающих данных и необходимость контроля со стороны разработчиков для предотвращения внедрения неэффективных или небезопасных решений. Важно также учитывать этические аспекты использования моделей, особенно в отношении интеллектуальной собственности и конфиденциальности данных.

Когнитивное взаимодействие разработчиков с LLM

Интеграция LLM в процессы разработки ПО оказывает значительное влияние на когнитивное взаимодействие разработчиков с кодом и инструментами разработки. Одним из ключевых аспектов является **снижение когнитивной нагрузки на разработчика**. За счет автоматизации рутинных задач, таких как автодополнение кода и исправление синтаксических ошибок, LLM позволяют разработчикам сосредотачиваться на более сложных и творческих аспектах программирования.

Кроме того, взаимодействие с LLM **изменяет процесс принятия решений**. Модели предоставляют множество возможных вариантов решения задач, что расширяет выбор разработчика и может ускорить процесс нахождения правильного решения. Это помогает сократить количество усилий, связанных с поиском информации, и делает процесс принятия решений более интуитивным и эффективным.

Важным аспектом когнитивного взаимодействия с LLM является **поддержка творческого процесса**. LLM способны предлагать нестандартные решения и помогать разработчикам выйти за рамки привычных шаблонов. Еще один важный аспект – **адаптация LLM к личному стилю программирования разработчика.** По мере использования модели, LLM могут запоминать стиль и предпочтения конкретного сотрудника, предлагая решения, которые соответствуют его привычкам и практикам. Это позволяет сократить время на выполнение задач и сделать взаимодействие с моделью более интуитивным.

Применение LLM в IDE: анализ современных решений в крупных компаниях

Международные компании активно внедряют LLM в IDE, чтобы автоматизировать процессы программирования, улучшить качество кода и повысить производительность разработчиков. Одним из самых известных инструментов является **GitHub Copilot**, разработанный корпорацией Microsoft на базе модели OpenAI Codex[10]. Он внедрен в популярные IDE, такие как Visual Studio Code, и помогает разработчикам за счет автодополнения кода, генерации целых фрагментов программ на основе текстовых описаний и предложений по улучшению кода. GitHub Copilot обучен на миллиардах строк кода из открытых репозиториев, что позволяет ему предоставлять контекстуально точные подсказки для многих языков программирования. Этот инструмент широко используется в американских компаниях для ускорения разработки и снижения нагрузки на программистов. Например, он на 25 % увеличил скорость разработки платформы по изучению языка Duolingo, а среднее время выполнения проверки кода сократилось на 67%.

Еще один продукт от Microsoft, **IntelliCode**, представляет собой интеллектуальную систему автодополнения кода, интегрированную в Visual Studio и Visual Studio Code. IntelliCode использует AI для анализа кода, предлагая рекомендации на основе лучших практик и кода, используемого в реальных проектах [11]. Так, при написании функции на языке Python, IntelliCode может предложить оптимальное использование стандартных библиотек и улучшения для повышения эффективности кода. Рассмотрим следующий пример:

```
def process_data(data):

result = []

for item in data:

result.append(item * 2)

return result

data = [1, 2, 3, 4]

print(process_data(data)), (1)
```

При использовании IntelliCode система может предложить оптимизированный код с использованием встроенных методов Python, таких как map() и lambda, что ускоряет выполнение программы:

```
def process_data(data):

return list(map(lambda x: x * 2, data))

data = [1, 2, 3, 4]

print(process_data(data)), (2)
```

В отличие от GitHub Copilot, который предлагает широкие рекомендации, IntelliCode больше ориентирован на улучшение качества кода за счет следования стандартам и анализу предыдущих действий разработчика. Этот инструмент активно используется американскими и международными компаниями для повышения эффективности разработки ПО.

Компания **«Яндекс»**, один из лидеров российского IT-рынка, разработала собственные решения на основе LLM для поддержки разработчиков. В рамках своей экосистемы LLM были интегрированы в среду разработки через **Alice Code Assistant**, который помогает программистам за счет автодополнения кода, предложения тестов и автоматического исправления ошибок [12]. При написании функции на языке Python для обработки данных, Alice Code Assistant может предложить автодополнение и автоматическую генерацию тестов для функции. Рассмотрим пример:

```
def process_data(data):

processed_data = []

for item in data:

processed_data.append(item.lower())
```

```
return processed_data

data = ["Hello", "World", "Alice"]

print(process_data(data)), (3)
```

Alice Code Assistant может предложить оптимизацию этой функции, используя более эффективные встроенные методы Python, такие как list comprehension, а также предложить сгенерировать тесты для функции:

```
# Оптимизированный код с рекомендацией Alice Code Assistant

def process_data(data):

return [item.lower() for item in data]

# Автоматически сгенерированный тест

def test_process_data():

assert process_data(["Hello", "World", "Alice"]) == ["hello", "world", "alice"]

assert process_data([]) == []

# Запуск теста

test_process_data(), (4)
```

Этот инструмент используется в различных проектах компании Яндекс, в том числе в развитии поисковых и рекламных технологий. Alice Code Assistant обеспечивает поддержку различных языков программирования и активно применяется внутри Яндекса для ускорения разработки новых сервисов и приложений.

Проблемы и вызовы интеграции LLM в интегрированные среды разработки нового поколения

Интеграция LLM в IDE нового поколения приносит множество преимуществ, таких как автоматизация рутинных задач, улучшение качества кода и повышение производительности. Однако этот процесс сопряжен с рядом проблем, включая зависимость от качества данных, недостаточную адаптацию моделей к специфическим проектам, вопросы интеллектуальной собственности и этические риски. Чтобы обеспечить успешное внедрение LLM, необходимо учитывать эти вызовы и разрабатывать эффективные стратегии их решения (таблица 2).

Таблица 2. Проблемы интеграции LLM и способы их решения [13, 14]

Проблема	Описание	Способы решения
Качество и предвзятость данных	Модели могут обучаться на данных с ошибками или предвзятостью, что снижает точность рекомендаций.	Постоянное обновление обучающих данных, использование специализированных данных для конкретных задач.
Отсутствие контекста	LLM не всегда могут учитывать уникальные требования конкретных проектов.	Разработка и внедрение моделей, обученных на данных, связанных с конкретной областью или проектом.
Проблемы с интеллектуальной	Модели могут использовать данные, защищенные	Применение моделей на основе закрытых корпоративных данных и

собственностью	авторскими правами, или нарушать конфиденциальность.	внедрение строгих механизмов управления доступом.
Зависимость от технологий	Чрезмерное использование LLM может привести к снижению навыков разработчиков.	Ограничение автоматизации в ключевых этапах разработки, стимулирование обучения и улучшения навыков разработчиков.
Этические вопросы и безопасность	Использование LLM может создавать риски для безопасности данных.	Разработка и внедрение этических стандартов и протоколов безопасности при использовании LLM.

Анализ проблем внедрения LLM в интегрированные среды разработки показывает, что успешное внедрение этих технологий требует комплексного подхода и внимательного контроля. Хотя LLM могут значительно повысить продуктивность и автоматизировать рутинные задачи, их применение должно сопровождаться адаптацией к специфике проекта и улучшением данных, на которых обучены модели. Это позволит избежать возможных предвзятостей и ошибок в работе моделей. Кроме того, важно обеспечивать баланс между автоматизацией и развитием навыков разработчиков, чтобы не допустить снижения их квалификации.

Этические вопросы также требуют особого внимания. Использование LLM должно осуществляться в рамках строгих стандартов, касающихся безопасности и защиты данных, чтобы предотвратить утечку конфиденциальной информации и нарушение прав на интеллектуальную собственность.

Заключение

Применение LLM в IDE существенно изменяет процессы программирования, способствуя автоматизации рутинных задач и повышению общей производительности. Использование LLM в IDE позволяет разработчикам не только ускорить создание кода, но и улучшить его качество за счет интеллектуальной поддержки и автодополнения. Однако, несмотря на очевидные преимущества, применение таких технологий требует решения ряда этических и технических вопросов. В частности, важно контролировать зависимость от автоматизации, чтобы поддерживать профессиональные навыки разработчиков, а также обеспечивать соблюдение норм безопасности и конфиденциальности данных. Для успешной интеграции LLM необходим продуманный и ответственный подход, предполагающий баланс между автоматизацией и сохранением творческого потенциала программистов.

Библиография

1. Иванов К. Н., Захарова О. И. Обработка естественного языка. Применение языковых моделей // Актуальные проблемы информатики, радиотехники и связи. – 2023. – С. 155-156.
2. Korostin O. Comparative analysis of NLP algorithms for optimizing communications in the maritime industry // Journal of science. Lyon. – 2024. – № 56. – С. 19-22.
3. Qin Z., Yang S., Zhong Y. Hierarchically gated recurrent neural network for sequence modeling // Advances in Neural Information Processing Systems. – 2024. – V. 36.
4. Узких Г. Ю. Применение трансформеров в обработке естественного языка // Вестник науки. – 2024. – Т. 4. – № 8 (77). – С. 186-189.
5. Gweon H., Schonlau M. Automated classification for open-ended questions with BERT // Journal of Survey Statistics and Methodology. – 2024. – V. 12. – № 2. – P. 493-504.
6. Liukko V., Knappe A., Anttila T., Hakala J. ChatGPT as a Full-Stack Web Developer // Generative AI for Effective Software Development. – Cham: Springer Nature Switzerland. –

2024. – P. 197-215.
7. Ponomarev E. Optimizing android application performance: modern methods and practices // Sciences of Europe. – 2024. – № 149. – С. 62-64.
8. Макарьян О. С. Разработка программного обеспечения с использованием искусственного интеллекта // Вестник магистратуры. – 2024. – С. 23.
9. Бобунов А. Ю. Сравнение практик автоматизации тестирования в традиционных банках и финтех-компаниях // Дневник науки. 2024. № 8 [Электронный ресурс]. URL: http://www.dnevniknauki.ru/images/publications/2024/8/technics/Bobunov.pdf
10. Koyanagi K., Wang D., Noguchi K., и т. д. Exploring the effect of multiple natural languages on code suggestion using github copilot // 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR). – 2024. – P. 481-486.
11. Oh S., Lee K., Park S., Kim D. Poisoned chatgpt finds work for idle hands: exploring developers' coding practices with insecure suggestions from poisoned ai models //2024 IEEE Symposium on Security and Privacy (SP). – 2024. – P. 1141-1159.
12. Жикулина К. П., Перфильева Н. В., Мань Л. Цифровой страт парадигмы языка // Вестник Российского университета дружбы народов. Серия: Теория языка. Семиотика. Семантика. – 2024. – Т. 15. – № 2. – С. 364-375.
13. Пекарева, В. В. Семантический анализ дефиниции «информация» в целях систематизации подходов и факторов обеспечения информационной безопасности в условиях цифровизации / В. В. Пекарева, Ю. И. Фроловская // Аграрное и земельное право. – 2024. – № 3(231). – С. 89-92. – DOI: 10.47643/1815-1329_2024_3_89. – EDN PQGQDB
14. Verner D. Integration of artificial intelligence in backend development // Annali d'Italia. – 2024. – № 59. – P. 88-91.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предметом исследования в рецензируемой публикации выступают большие языковые модели (LLM), в работе раскрывается их роль в интегрированных средах разработки (IDE) нового поколения.
Методология исследования базируется на изучении и обобщении научных публикаций по рассматриваемой теме и анализе применения больших языковых моделей в практической работе команд ИТ-разработчиков.
Актуальность работы авторы связывают с тем, что большие языковые модели, основанные на архитектуре трансформеров, демонстрируют значительный прогресс в области искусственного интеллекта и обработки естественного языка, быстро нашли широкое применение в различных отраслях, включая программирование и разработку программного обеспечения, являются важным инструментом, способным существенно автоматизировать процессы, повысить эффективность разработки и улучшить качество кода в современных интегрированных средах разработки.
Научная новизна рецензируемого исследования состоит в представленных результатах анализа роли больших языковых моделей в современных интегрированных средах разработки, в отраженных авторами когнитивных и социальных аспектах взаимодействия разработчиков с большими языковыми моделями, оценках изменения методов работы и взаимодействия в командах, а также повышения эффективности программирования.
В тексте статьи выделены следующие разделы: Введение, Основная часть. Эволюция LLM и их применение в IDE, Роль LLM в трансформации современных IDE, Когнитивное

взаимодействие разработчиков с LLM, Применение LLM в IDE: анализ современных решений в крупных компаниях, Проблемы и вызовы интеграции LLM в интегрированные среды разработки нового поколения, Заключение и Библиография.

В статье проанализировано, как постоянное взаимодействие с большими языковыми моделями сказывается на нагрузке программистов, их мотивации и способности к творческому мышлению, на безопасности данных, защите интеллектуальной собственности и рисках чрезмерной зависимости от технологий искусственного интеллекта. Авторами отмечена важность обеспечения баланса между автоматизацией и развитием навыков разработчиков, чтобы не допустить снижения их квалификации. В публикации освещены проблемы интеграции больших языковых моделей: качество и предвзятость данных, отсутствие контекста, проблемы с интеллектуальной собственностью, зависимость от технологий, этические вопросы и безопасность. Текст статьи иллюстрирован двумя таблицами и тремя рисунками, содержит несколько фрагментов программных кодов.

Библиографический список включает 14 источников – научные публикации на русском и иностранных языках, на которые в тексте приведены адресные отсылки, что подтверждает наличие апелляции к оппонентам.

Рецензируемый материал соответствует направлению журнала «Программные системы и вычислительные методы», отражает результаты проведенной авторами работы, может вызвать интерес у читателей, поскольку содержит интересные сведения о роли больших языковых моделей в интегрированных средах разработки нового поколения, а также потенциальных угрозах и рисках их массового распространения. Статья рекомендуется к опубликованию.