УДК 004

Захаров А.С.
магистрант 2 курса
кафедра бизнес-информатики и менеджмента
Санкт-Петербургский государственный университет
аэрокосмического приборостроения
(г. Санкт-Петербург, Россия)

ДОСТУПНОСТЬ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ПРИМЕНЕНИЕМ ГЕНЕРАТИВНОГО ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Аннотация: *в статье рассматривается влияние генеративного искусственного интеллекта на доступность разработки программного обеспечения. Проанализированы возможности создания небольших программ пользователями без профессиональной подготовки, особенности применения интеллектуальных помощников на различных этапах разработки, а также ограничения, связанные с качеством, безопасностью и сопровождением программных продуктов.*

Ключевые слова: *генеративный искусственный интеллект, разработка программного обеспечения, программирование, генерация кода, автоматизация, доступность технологий.*

Разработка программного обеспечения традиционно относится к видам деятельности, требующим специальных знаний, практического опыта и понимания принципов работы вычислительных систем. Даже создание сравнительно небольшой программы предполагает знакомство с языком программирования, способами обработки данных и правилами выполнения программного кода. Развитие генеративного искусственного интеллекта (ИИ) постепенно меняет условия выполнения подобных задач. Современные интеллектуальные помощники позволяют пользователю описывать желаемую

функциональность на естественном языке и получать готовые фрагменты кода, инструкции по их запуску и рекомендации по исправлению ошибок. В результате некоторые задачи, ранее требовавшие самостоятельного изучения программирования или обращения к специалисту, становятся доступнее широкому кругу пользователей. Однако возможность получить программный код не равнозначна способности создать полноценный, надёжный и безопасный программный продукт. Цель настоящей статьи заключается в определении возможностей и ограничений применения генеративного ИИ для разработки программного обеспечения пользователями с различным уровнем технической подготовки.

До широкого распространения генеративных моделей человек, не обладающий навыками программирования, имел несколько основных способов решения задач, связанных с созданием программных инструментов. Он мог освоить необходимый язык программирования, использовать существующее приложение, воспользоваться визуальным конструктором или обратиться к разработчику. Различные средства автоматизации, включая макросы и платформы разработки без непосредственного написания кода, позволяли создавать простые решения и раньше. Тем не менее такие инструменты обычно требовали изучения определённой среды и её возможностей. Генеративный ИИ дополняет существующие подходы новым способом взаимодействия: вместо непосредственного описания алгоритма средствами языка программирования пользователь может первоначально сформулировать задачу обычными словами.

Для понимания происходящих изменений необходимо различать программирование как написание исходного кода и разработку программного обеспечения как более широкий процесс. Создание программы включает определение потребностей пользователя, формирование требований, выбор технического решения, реализацию, тестирование, внедрение и последующее сопровождение. Генеративные модели могут оказывать помощь практически на каждом из перечисленных этапов, однако степень их самостоятельности и требования к человеческому контролю существенно различаются. Наиболее

очевидные преимущества проявляются при создании ограниченных по функциональности решений, для которых можно достаточно точно сформулировать ожидаемый результат и непосредственно проверить его правильность.

Характерным примером является разработка небольшого скрипта для обработки файлов. Пользователю может потребоваться автоматически переименовать большое количество документов, извлечь определённые сведения из таблицы или преобразовать данные из одного формата в другой. При традиционном подходе такая задача могла потребовать поиска готовой программы, изучения документации или написания собственного кода. При использовании генеративного ИИ человек способен описать последовательность необходимых действий и получить первоначальный вариант программы, например на языке Python. Дополнительно интеллектуальный помощник может объяснить порядок установки необходимых компонентов, способ запуска скрипта и значения используемых параметров. При благоприятных условиях это позволяет решить локальную практическую задачу в пределах одного рабочего сеанса, хотя продолжительность работы зависит от её сложности и опыта пользователя.

Существенное значение имеет возможность последовательного уточнения требований. Пользователю необязательно формулировать окончательное техническое задание с первого обращения. После получения первоначального результата он может попросить изменить формат выходного файла, добавить обработку исключений или предусмотреть выбор исходной папки. Таким образом, процесс создания программы приобретает диалоговый характер. Вместе с тем естественный язык допускает неоднозначность, поэтому модель может неверно интерпретировать отдельные требования. Например, просьба удалить повторяющиеся записи может пониматься по-разному в зависимости от того, какие поля должны использоваться для сравнения. Даже при создании небольших программ необходимо заранее определить, что именно считается правильным результатом.

Доступность генеративных инструментов особенно заметна при разработке решений для индивидуального использования. В отличие от коммерческого программного продукта, небольшая программа для личной задачи обычно имеет ограниченное число пользователей, сравнительно простые требования и известные условия применения. Создатель такого инструмента самостоятельно определяет, достаточно ли его возможностей для решения конкретной проблемы. При этом ограниченный масштаб не отменяет необходимости осторожного обращения с данными. Например, даже простой скрипт для переименования файлов может привести к потере информации, если неправильно реализует перезапись существующих документов. Поэтому проверка работы на копиях исходных данных должна рассматриваться как обязательная практика, доступная в том числе пользователям без специального образования.

Одним из механизмов повышения доступности программирования становится сокращение объёма знаний, необходимых для получения первого работоспособного результата. Пользователю уже не всегда требуется самостоятельно запоминать синтаксис языка, названия стандартных функций или способы подключения распространённых библиотек. Часть соответствующих действий выполняет интеллектуальный помощник. Однако это не означает полного устранения технических требований. Для использования сгенерированной программы по-прежнему может потребоваться понимание файловой системы, разрешений доступа, форматов данных и особенностей программной среды. По мере усложнения задачи возрастает вероятность возникновения проблем, решение которых требует знаний, выходящих за пределы обычного пользовательского опыта.

Применение генеративного ИИ способно изменять и сам подход к поиску программных решений. Ранее пользователю нередко приходилось адаптировать свою задачу к возможностям имеющихся приложений. Теперь появляется возможность сформировать небольшой инструмент под конкретную потребность. Например, стандартная программа может предоставлять широкий

набор функций, из которых человеку необходимы только две или три. Индивидуально созданный скрипт способен выполнять именно требуемую последовательность операций без сложного интерфейса и дополнительных настроек. Это не делает самостоятельную разработку предпочтительной во всех ситуациях, поскольку готовые продукты обычно обладают преимуществами в сопровождении и проверке качества. Однако для отдельных локальных задач генеративный ИИ расширяет набор доступных способов решения.

Исследования подтверждают, что интеллектуальные помощники могут влиять на скорость выполнения задач программирования. В эксперименте С. Пэн и соавторов участники, использовавшие GitHub Copilot, выполнили задание по созданию сервера на JavaScript в среднем на 55,8% быстрее контрольной группы [4]. Полученный результат показывает потенциал технологии для ускорения программирования, но не должен переноситься на все виды разработки. Эксперимент был ограничен конкретным заданием, а создание реальных программных продуктов включает значительно больше операций, чем непосредственное написание кода. Кроме того, повышение скорости работы специалиста не доказывает, что аналогичную задачу сможет самостоятельно выполнить человек без технической подготовки.

Особенности взаимодействия разработчиков с генеративными системами рассмотрены в исследовании Ш. Барке и соавторов. На основе наблюдения за двадцатью участниками авторы выделили два характерных режима применения интеллектуального помощника: ускорение выполнения заранее понятной задачи и исследование возможных способов решения при недостаточной определённости дальнейших действий [1]. Для начинающего пользователя особенно важен второй режим, поскольку ИИ способен предложить первоначальную структуру программы и объяснить возможные подходы к её созданию. Однако исследование также показывает, что взаимодействие с интеллектуальным помощником остаётся процессом принятия решений человеком, а не полностью самостоятельной деятельностью программной системы.

Существенным ограничением доступности разработки становится сложность оценки правильности сгенерированного кода. Человек может получить программу, которая запускается без ошибок и демонстрирует ожидаемый результат на нескольких примерах. Однако отсутствие видимых проблем не гарантирует корректности во всех условиях. Программа может неправильно обрабатывать пустые значения, нестандартные символы, большие объёмы данных или непредусмотренные сочетания входных параметров. Пользователь, не понимающий внутренней логики алгоритма, способен обнаружить только наиболее очевидные недостатки. В результате генеративный ИИ снижает первоначальный барьер создания программы, но не обязательно устраняет барьер её самостоятельной проверки.

Особую значимость данная проблема приобретает для начинающих пользователей. В исследовании С. Нгуен и соавторов, охватившем 120 студентов с начальной подготовкой в программировании, выявлены трудности формулирования запросов и последовательного исправления неудачных результатов генерации [2]. Участникам было непросто точно передавать модели требования даже к заданиям, соответствовавшим их уровню подготовки. Это позволяет сделать вывод, что взаимодействие на естественном языке само по себе не обеспечивает правильного понимания программной задачи. Для успешной работы необходимо уметь определять входные данные, ожидаемые результаты и условия, при которых решение должно считаться корректным.

Дополнительные ограничения установлены П. Вайтхилингамом и соавторами при изучении использования инструментов автоматической генерации кода. Участники исследования положительно оценивали возможность получать первоначальные варианты решений, однако испытывали затруднения при понимании, редактировании и отладке предложенных фрагментов [5]. Данное обстоятельство демонстрирует различие между получением готового кода и владением программой как техническим решением. Если пользователь не способен объяснить принцип работы созданного инструмента, возможности его последующего изменения и устранения неполадок оказываются ограниченными.

При переходе от небольших программ к сложным системам значительно возрастает значение проектирования архитектуры. Полноценный программный продукт может включать пользовательский интерфейс, базу данных, серверную часть, механизмы авторизации и взаимодействие с внешними сервисами. Отдельные компоненты должны согласованно работать между собой, а изменения одного элемента не должны нарушать функционирование остальных. Генеративная модель способна предложить примеры реализации каждого компонента, но качество общей архитектуры зависит от правильности исходных требований и выбранных технических решений. Ошибки проектирования могут оставаться незаметными на ранних этапах и проявляться лишь при расширении функциональности или увеличении количества пользователей.

Не менее существенны требования информационной безопасности. Сгенерированный программный код может содержать уязвимости, связанные с недостаточной проверкой входных данных, некорректным управлением доступом или небезопасным хранением конфиденциальной информации. В исследовании Х. Пирса и соавторов при анализе программных решений, предложенных GitHub Copilot для специально разработанных сценариев, примерно 40% полученных вариантов содержали уязвимости [3]. Данный показатель относится к конкретной экспериментальной выборке и не характеризует безопасность всех современных генеративных моделей. Тем не менее исследование подтверждает необходимость независимой проверки автоматически сформированного кода, особенно при разработке систем, работающих с персональными, финансовыми и другими чувствительными данными.

Для серьёзного программного продукта также важны надёжность и возможность дальнейшего сопровождения. Программа должна сохранять работоспособность при изменении условий эксплуатации, обновлении используемых библиотек и появлении новых требований. Если решение создаётся преимущественно последовательной генерацией отдельных фрагментов, между ними могут возникать несоответствия, усложняющие

последующие изменения. Даже успешно работающий первоначальный вариант не всегда обладает достаточно понятной структурой. Поэтому профессиональная разработка предполагает документирование решений, использование систем контроля версий, проверку совместимости компонентов и систематическое тестирование. Генеративный ИИ способен помогать в выполнении этих операций, но их организационная и техническая необходимость сохраняется.

Отдельной проблемой выступает ответственность за результат. При индивидуальном использовании небольшого скрипта возможные последствия ошибки обычно ограничиваются собственными данными и рабочими процессами пользователя. При создании приложения для других людей требования становятся значительно строже. Разработчик должен учитывать доступность сервиса, защиту информации, обработку ошибочных действий и возможность восстановления после сбоев. Генеративная модель не принимает на себя юридическую или профессиональную ответственность за функционирование программного продукта. Поэтому наличие автоматически созданного кода не устраняет необходимость специалиста, способного оценить риски и принять обоснованные технические решения.

Расширение доступности программирования может иметь и организационные последствия. Сотрудники, не занимающиеся профессиональной разработкой, получают возможность самостоятельно создавать вспомогательные инструменты для обработки информации и автоматизации повторяющихся операций. Это потенциально сокращает потребность в обращении к специализированным подразделениям по поводу простых локальных задач и позволяет быстрее проверять отдельные идеи. Вместе с тем неконтролируемое использование таких решений способно приводить к появлению программ, не соответствующих внутренним требованиям организации. Особенно значимы риски при работе с общими базами данных и информационными системами. Следовательно, расширение самостоятельности пользователей должно сопровождаться понятными правилами применения и проверки созданных инструментов.

С образовательной точки зрения генеративный ИИ создаёт одновременно новые возможности и ограничения. Интеллектуальный помощник может объяснять назначение отдельных команд, предлагать примеры и помогать разбирать сообщения об ошибках. Это позволяет начинающему пользователю осваивать программирование непосредственно в процессе решения практической задачи. Однако постоянное получение готовых решений без самостоятельного анализа способно ограничивать развитие навыков алгоритмического мышления и отладки. Наиболее обоснованным представляется подход, при котором генеративная система используется не только для написания кода, но и для объяснения принципов его работы, а пользователь постепенно приобретает способность самостоятельно проверять и изменять программу.

На основании проведённого анализа можно выделить различие между доступностью первоначального создания программного решения и доступностью полноценной программной инженерии. В первом случае генеративный ИИ действительно способен существенно снизить требования к предварительной подготовке пользователя. Во втором сохраняется необходимость профессиональных компетенций, связанных с архитектурой, качеством, безопасностью, интеграцией и сопровождением. Граница между этими случаями определяется не только количеством строк кода или внешней сложностью интерфейса, но и последствиями возможных ошибок, числом пользователей и требованиями к устойчивости системы.

Таким образом, применение генеративного искусственного интеллекта расширяет возможности самостоятельной разработки небольших программ и способствует снижению первоначального барьера входа в программирование. Пользователь получает инструмент, позволяющий переходить от словесного описания потребности к работоспособному прототипу без предварительного освоения значительного объёма профессиональных знаний. Однако доступность генерации кода не означает автоматического обеспечения его правильности, надёжности и безопасности. При усложнении программного продукта

сохраняется и возрастает значение системного проектирования, тестирования и профессиональной ответственности. Поэтому основным результатом распространения генеративного ИИ следует считать не устранение необходимости в разработчиках, а расширение круга людей, способных самостоятельно решать ограниченные программные задачи, при сохранении ключевой роли технической квалификации в создании сложных и ответственных систем.

СПИСОК ЛИТЕРАТУРЫ:

1. Barke S., James M.B., Polikarpova N. Grounded Copilot: How Programmers Interact with Code-Generating Models // Proceedings of the ACM on Programming Languages. 2023. Vol. 7. OOPSLA1. Article 78. https://doi.org/10.1145/3586030
2. Nguyen S., Babe H.M., Zi Y., Guha A., Anderson C.J., Feldman M.Q. How Beginning Programmers and Code LLMs (Mis)read Each Other // Proceedings of the CHI Conference on Human Factors in Computing Systems. 2024. Article 651. P. 1–26. https://doi.org/10.1145/3613904.3642706
3. Pearce H., Ahmad B., Tan B., Dolan-Gavitt B., Karri R. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions // 2022 IEEE Symposium on Security and Privacy. 2022. P. 754–768. https://doi.org/10.1109/SP46214.2022.9833571
4. Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot // arXiv. 2023. arXiv:2302.06590
5. Vaithilingam P., Zhang T., Glassman E.L. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models // CHI Conference on Human Factors in Computing Systems Extended Abstracts. 2022. P. 1–7. https://doi.org/10.1145/3491101.3519665

Zakharov A.S.
master's degree student of 2nd year
Department of Business Informatics and Management
St. Petersburg State University of Aerospace Instrumentation
(St. Petersburg, Russia)

ACCESSIBILITY OF SOFTWARE DEVELOPMENT USING GENERATIVE ARTIFICIAL INTELLIGENCE

Abstract: *this article examines how generative artificial intelligence affects the accessibility of software development. It analyzes opportunities for users without professional training to create small software applications, the use of AI assistants at various stages of development, and limitations related to software quality, security, and maintenance.*

Keywords: *generative artificial intelligence, software development, programming, code generation, automation, technology accessibility.*