УДК 37.09+004.8
DOI 10.20339/AM.10-24.069

М.В. Сергиевский,
канд. техн. наук, доцент
Кафедра Кибернетики
НИЯУ МИФИ
https://orcid.org/0009- 0005-8528-9177
e-mail: sermax@yandex.ru
А.И. Винокур,
д-р техн. наук, профессор
Кафедра информатики и информационных технологий
Московский политехнический университет
https://orcid.org/0000-0002-6914-2520
e-mail: alex.vinokour@gmail.com

ПЕРСПЕКТИВЫ ИСПОЛЬЗОВАНИЯ СИСТЕМ ИИ В ОБУЧЕНИИ ПРОГРАММИРОВАНИЮ

Можно с уверенностью утверждать, что, каким бы ни было отношение к технологиям искусственного интеллекта (ИИ), в скором времени они будут востребованы в процессе обучения. В данной статье оценивается потенциал применения систем ИИ, в частности нейронных сетей класса GPT, в обучении программированию на алгоритмических языках. Авторы провели исследование, в ходе которого три системы ИИ — ChatGPT-4, Gemini Ultra и GigaChat — использовались для решения ряда модельных задач с целью генерации кода на языках Python и Scala. Результаты работы систем ИИ оценивались по таким критериям, как время работы, эффективность кода, качество комментариев. Проведенное исследование показало, что, несмотря на оптимистическое в целом отношение к использованию ИИ в процессе обучения программированию, необходимо сохранять осторожность и разумный баланс применения как классических методов обучения, так и систем искусственного интеллекта.

Ключевые слова: *программирование, обучение, искусственный интеллект, генерация программного кода, качество кода.*

PERSPECTIVES FOR THE USE OF AI SYSTEMS IN PROGRAMMING TEACHING

Maksim V. Sergievsky, Ph. D., Docent, Assistance Professor of Department Cybernetics, National Research Nuclear University MEPhI, https://orcid.org/0009-0005-8528-9177, e-mail: *sermax@yandex.ru*

Aleksey I. Vinokur, Dr. Sc. (Technic), Professor, Department Informatica and Information, Technology Moscow State Polytechnic University, https://orcid.org/0000-0002-6914-2520, e-mail: *alex.vinokour@gmail.com*

It is safe to say, that whatever the attitude to technology artificial intelli-gence (AI), they will be in demand in the learning process in the near future. This article evaluates the potential of using AI systems, in particular neural networks of GPT-class, in teaching programming in algorithmic languages. The authors conducted a study in which three AI systems — ChatGPT-4, Gemini Ultra and GigaChat — were used to solve a number of model tasks in order to generate code in Python and Scala languages. The results of AI systems were evaluated according to criteria such as running time, code efficiency, and comment quality. The conducted research has shown that, despite the generally op-timistic attitude towards the use of AI in programming teaching, it is necessary to maintain caution and a reasonable balance of using both classical teaching methods and artificial intelligence systems.

Keywords: *programming, teaching, artificial intelligence, programming code generation, code quality*

Введение

В настоящее время системы ИИ всё чаще используются в образовательном процессе как преподавателями, так и студентами [1]. Важно отметить, что на данном этапе развития технологий не существует единого мнения о роли и месте систем ИИ в обучении [2]. Вместе с тем необходимость более эффективной организации учебного процесса подтолкнула авторов к экспериментальному изучению возможностей использования систем ИИ для обучения программированию.

Программы, имитирующие (по крайней мере по результатам) или, точнее, моделирующие работу человеческого мозга, принято называть ИИ. Все они базируются на использовании нейронных сетей (НС) класса GPT [3; 4]. Общие принципы работы НС давно и хорошо известны. Качественный скачок в их работе связан с возможностью вовлечения в оборот огромного объема информации.

А это, в свою очередь, стало реальным только после прорыва в области обработки больших данных. Теперь, как известно, распределенные системы легко справляются с обработкой огромных объемов информации за короткое время. А чем больше сопоставимой информации проанализировано, тем выше вероятность правильного с точки зрения человеческой логики ответа. Построение НС, состоящих из миллиардов узлов-нейронов, вывело технологию НС на принципиально другой уровень. В настоящее время НС легко справляются с переводом текстов с одних языков на другие, генерацией осмысленных текстов практически на любую тему, ведением диалога с человеком и даже с созданием живописных произведений [5; 6]. Скажем больше, результаты в только что перечисленных областях превосходят возможности большинства людей. И это уже сейчас заставляет задуматься о том, как можно использовать достижения в области НС в организации процесса обучения различным дисциплинам.

Речь не идет о выборе правильного ответа из набора вариантов, а о генерации качественно новой соответствующей запросу информации. Здесь уместна такая аналогия: существует ограниченное множество слов (для русского языка примерно 250 тысяч), и стоит задача написать эссе на определенную тему. Вариантов, оперируя имеющимися словами, создать разумный текст, комбинаторное число. Перебор принципиально невозможен. Но используя набор грамматических правил и имеющиеся образцы текстов на различные темы как эталоны, современные НС легко решают подобные задачи.

Кажется, что задача генерации нейросетью компьютерных программ для решения четко сформулированных локальных задач не должна являться проблемой. Но в этом вопросе возникает много подводных камней, которые мы сейчас обсудим.

Первая проблема — результат должен быть представлен не на естественном языке, а на алгоритмическом. А современные НС ориентированы на работу с естественными языками.

Вторая проблема — пооператорная генерация программ (а это именно то, что делает программист) не является инструментарием НС.

Третья проблема — каким критериям качества должен удовлетворять результат? Для начала можно смириться с тем, что программа просто должна правильно работать. А уж затем с помощью дополнительных средств ее можно оптимизировать.

Подход, используемый для генерации кода, базируется исключительно на анализе спецификации (если она есть) или словесной постановки задачи. Далее ищутся адекватные по смыслу спецификации имеющихся в распоряжении НС методов или каких-то других компонентов ПО. Код, следующий за спецификацией, рассматривается как черный ящик, т.е. не анализируется. Другими словами, задача генерации кода сводится к сопоставлению спецификаций, заданных на естественном языке. Правда, НС может протестировать код — проверить, как код работает с конкретными данными.

Насколько такой подход разумен? Большинство программистов в последнее время всё чаще пользуется уже готовыми библиотеками и фреймворками или применяет технологию low code. То есть пооператорное кодирование встречается всё реже и реже. Задача заключается в поиске нужной библиотеки или средства, поддерживающего технологию low code. В нашем случае эта задача просто делегируется НС. И с такой задачей НС скорее всего справится лучше, чем человек — хотя бы потому, что она гораздо быстрее найдет и проверит множество различных вариантов решения.

Проанализировать сложную системную спецификацию и построить программную систему (ПС), состоящую из нескольких связанных компонентов, НС пока не могут. В таком случае задача проектировщика-программиста сведется к разделению задачи на составные более мелкие части-подзадачи. Каждую из них можно поручить выполнить НС, а интеграцией решений этих подзадач как раз и будет заниматься программист. Но последнее не относится к теме данной статьи, а связано, скорее, с применяемыми технологиями разработки кода.

Очевидно, что знание синтаксиса языка необходимо, но явно недостаточно для того, чтобы считаться программистом. Настоящие навыки, которые действительно отличают хороших программистов, не зависят от языка. Любой язык программирования — всего лишь инструмент. Есть вещи более фундаментальные, чем язык, используемый для алгоритмизации. Модульность — одна из них [7]; если ваш код не является модульным, его нельзя назвать качественным. Поэтому важно уметь разделять задачу на отдельные части-подзадачи, что пока труднодостижимо для систем ИИ, а затем каждую подзадачу решать практически автономно. Именно здесь системы ИИ могут оказать человеку неоценимую помощь.

Экспериментальное исследование

Проанализируем возможности НС, связанные с генерацией кода объектно-ориентированных языков. Для примера возьмем языки Python и Scala. Выбор обусловлен тем, что Python — широко востребованный в настоящее время язык, часто используемый для решения задач

в области ИИ. Что касается Scala, то этот язык наиболее выразителен и компактен по сравнению со всеми остальными аналогичными языками, и будет интересно проверить, в какой мере эти его возможности будут использованы при автоматической генерации.

Для исследований были взяты три мощные системы ИИ, использующие технологии НС класса GPT: ChatGPT-4 компании Open AI, Gemini Ultra компании Google и GigaChat компании Сбер. К сожалению, известная система YandexGPT 2 компании Yandex не ориентирована на генерацию программного кода. Хотя последняя версия системы, вышедшая в 2024 г., сделала шаг в этом направлении и может генерировать фрагменты кода, но пока предлагаемые ей ответы (в том числе на языках Python и Scala) содержат многочисленные ошибки.

В работе не использованы интегральные критерии качества [8], поскольку в данном случае именно конкретные характеристики качества будут точнее определять эффективность работы системы. В число таких характеристик включены:

- время работы;
- эффективность (компактность) кода (в строках);
- качество комментариев и пояснений, генерируемых системой ИИ для пояснения кода;
- общие замечания, характеризующие код.

В число критериев не включены: число ошибок или неточностей, поскольку таковые не были обнаружены; и возможность предоставить результаты тестирования (прогона) сгенерированного кода, поскольку все системы ИИ такими возможностями обладают.

Приведем формулировки *восьми задач*, которые были предложены программам ИИ для решения.

1. Сформировать список из произвольного списка, удалив первый и последний элементы.

2. Вывести на печать все элементы одномерного массива, кроме повторяющихся.

3. Подсчитать число пробелов в строке.

4. Задан одномерный массив вещественных чисел. Сформировать новый массив, в котором сначала идут отрицательные элементы, а затем все остальные, сохраняя первоначальный порядок.

5. Реверсировать список. Применить оператор сопоставления с образцом match.

6. Дан одномерный массив целых чисел и целое число. Определить индексы всех тех двух элементов массива, сумма которых равна этому числу.

7. Определить, содержит ли массив произвольной размерности ненулевые элементы.

8. Дополнить функционал целых чисел новым методом — вычислением факториала.

Четыре первые задачи достаточно просты, и реализующий их функционал занимает от одной до трех строк кода. Пятое и шестое задания немного сложнее, подразумевают применение определенных инструментов программирования, но результирующий код также может быть очень коротким. Два последних задания требуют более тонкого понимания сути поставленной задачи, и для их решения в ряде случаев потребуется более значительный по объему код.

С Gemini Ultra общение происходило на английском языке, с остальными системами — на русском. Поскольку перевод текстов, написанных на естественном языке (ЕЯ), выполняется современными системами ИИ на высоком уровне, от языка, на котором формулировались задания, эффективность и качество работы систем практически не зависели.

Результаты испытаний систем ИИ по каждой системе оформлены в виде таблиц.

Gemini Ultra

Критерий Задача	Язык	Время работы менее 5 секунд	Эффективность кода (число строк)	Качество комментариев к коду системы ИИ (низкое, среднее, высокое)	Общая оценка кода (с указанием достоинств и недостатков)
1	Python	+	4	высокое	Положительная: проверяется, содержит ли список не менее двух элементов
	Scala	+	1	высокое	Используются методы tail и init
2	Python	+	6	среднее	Недостаток: сканируются все элементы массива и уникальные записываются в множество
	Scala	+	1	высокое	Положительная: используется метод distinct
3	Python	+	1	высокое	Положительная: используется метод count
	Scala	+	1	высокое	Предлагается три варианта решения, но не дает самый лаконичный вариант
4	Python	+	5	среднее	Используется сортировка по ключу. Не решается поставленная задача, поскольку проводится лишняя сортировка
	Scala	+	2	высокое	Используется метод partition

Gemini Ultra

Продолжение

Критерий Задача	Язык	Время работы менее 5 секунд	Эффективность кода (число строк)	Качество комментариев к коду системы ИИ (низкое, среднее, высокое)	Общая оценка кода (с указанием достоинств и недостатков)
5	Python	+	2	высокое	Не используется согласование с образцом, проверяется, пуст ли список
	Scala	+	3	высокое	Отличное решение, использующее оператор match
6	Python	+	9	среднее	Недостаток: находит только одну пару индексов
	Scala	+	7	среднее	Недостаток: находит только одну пару индексов
7	Python	+	2	среднее	Решается двумерная задача
	Scala	+	2	среднее	Анализируется двумерный массив, используется метод flatten
8	Python	+	9	высокое	Вводится объект factorial, и для него генерируется функция, то есть делается не то, что надо
	Scala	+	9	высокое	Недостаток: просто генерируется функция

GigaChat

Критерий Задача	Язык	Время работы менее 5 секунд	Эффективность кода (число строк)	Качество комментариев к коду системы ИИ (низкое, среднее, высокое)	Общая оценка кода (с указанием достоинств и недостатков)
1	Python	+	2	среднее	Не проверяется, содержит ли список не менее 2 элементов
	Scala	+	1	среднее	Не проверяется, содержит ли список не менее 2 элементов
2	Python	+	4	высокое	Перевод в множество
	Scala	+	1	высокое	Перевод в множество
3	Python	+	1	среднее	Недостаток: оператор match не используется
	Scala	-	4	высокое	Не сразу предлагается короткий вариант
4	Python	+	4	высокое	Недостаток: повторный просмотр массива
	Scala	+	3	среднее	Положительная: используется метод partition
5	Python	-	4	среднее	Не используется метод match
	Scala	+	4	высокое	Положительная: список делится на части
6	Python	+	8	высокое	Положительная: проводится неполный перебор
	Scala	+	10	высокое	Положительная: проводится неполный перебор
7	Python	+	2	среднее	Решается локальная двумерная задача
	Scala	-	8	среднее	Решается локальная двумерная задача, используются вложенные циклы
8	Python	-	8	высокое	Предлагает создать класс, который будет иметь метод factorial
	Scala			низкое	Дается неверная информация, утверждается, что в Scala целые числа не могут быть расширены с помощью новых методов

Chat GPT 4

Критерий Задача	Язык	Время работы менее 5 секунд	Эффективность кода (число строк)	Качество комментариев к коду системы ИИ (низкое, среднее, высокое)	Общая оценка кода (с указанием достоинств и недостатков)
1	Python	-	2	высокое	Положительная. Работает с любыми списками
	Scala	-	1	среднее	Работает только со списками, содержащими больше одного элемента
2	Python	+	4	высокое	Положительная: осуществляется перевод в множество
	Scala	-	2	высокое	Положительная: осуществляется перевод в множество
3	Python	+	2	среднее	Недостаток: используется цикл
	Scala	-	4	среднее	Не сразу предлагается короткий вариант

Chat GPT 4 Продолжение

Критерий Задача	Язык	Время работы менее 5 секунд	Эффективность кода (число строк)	Качество комментариев к коду системы ИИ (низкое, среднее, высокое)	Общая оценка кода (с указанием достоинств и недостатков)
4	Python	-	4	высокое	Недостаток: дважды просматривается массив
	Scala	+	2	высокое	Используется метод partition
5	Python		5	высокое	Положительная
	Scala	+	3	высокое	Положительная
6	Python	+	18	среднее	Нестандартный алгоритм: используется хэш-таблица для хранения пар индекс-значение
	Scala	-	14	высокое	Положительная: не используется полный перебор
7	Python	+	3	среднее	Справляется только с многомерным массивом регулярной структуры. Используется функция any
	Scala	-	3	среднее	Работает только с двумерными массивами
8	Python	-	8	высокое	Создается новый класс с новым методом
	Scala	-	6	высокое	Положительная: используется неявный класс

Проведенная работа преследовала следующие цели.

- Во-первых, показать принципиальную возможность генерации программного кода на языках Python и Scala системами ИИ. Такая воз-можность, без сомнения, есть у всех анализируемых систем и наверняка скоро появится у YandexGPT.
- Во-вторых, проверялось, могут ли системы ИИ объяснять свои действия. Выяснилось, что все системы в состоянии детально объяснять, что и с какой целью они делают. И это относится практически к каждой строке кода. Причем в комментариях систем ИИ сначала часто формули-ровалась общая идея решения, а затем давались подробные комментарии. Например, система ChatGPT выдала такой самодокументируемый код:

```
// Функция для поиска пар индексов массива, чьи элементы в сумме дают заданное число
  def findPairSums(nums: Array[Int], target: Int): Seq[(Int, Int)] = {
    val indices = for {
      i <- nums.indices // для каждого индекса в массиве
      j <- i + 1 until nums.length // и для каждого следующего после i индекса
      if nums(i) + nums(j) == target // если сумма элементов равна за-данной
    } yield (i, j) // возвращаем пару индексов
    indices // возвращаем результат
  }
```

предварив его следующим текстом:

Вот простое решение данной задачи на языке Scala, использующее двойной цикл для перебора всех возможных пар элементов массива.

Код прекрасный, хорошо структурирован и готов к выполнению без каких-либо изменений.

- В-третьих, анализировалось, могут ли системы создавать качественный код, хотя бы с точки зрения лаконичности и производительности. За редчайшим исключением все системы выбирали компактный вариант кода. В качестве отрицательного примера приведем следующий: Gemini Ultra при решении задач № 2 и № 6 выбирает не лучший алгоритм решения относительно простой задачи и, как следствие, создает достаточно громоздкий код.

С производительностью (быстродействием) дела обстояли несколько хуже. Часто массив просматривался дважды, когда можно было ограничиться одним просмотром. Это в равной мере касалось всех систем. Но иногда появлялись такие замечательные преамбулы к коду, как у ChatGPT:

Для решения задачи на языке Python можно эффективно использовать хэш-таблицу для отслеживания индексов чисел, которые уже прошли. Ключ в словаре — это нужное дополнение (разность между целевым числом и текущим элементом), а значение — это индекс текущего элемента. Это позволяет найти решение за линейное время O(n).

Удачные варианты по производительности часто предлагал GigaChat, генерируя несколько версий кода. Так, для задачи № 4 было предложено красивое решение на Scala с использованием метода partition:

```
def partitionArray(arr: Array[Float]): (Array[Float], Array[Float]) = {
  // Используем метод partition для разделения массива на отрицательные и положительные числа
  val (negativeNumbers, positiveNumbers) = arr.partition(_ < 0)

  // Объединяем оба массива обратно вместе, но сначала добавляем отрицательные числа
  negativeNumbers ++ positiveNumbers
}
```

- В-четвертых, мы пытались узнать, как быстро системы ИИ могут генерировать ответы. В качестве порогового значения было взято время, равное 5 секундам. Психологически для человека ожидание, не превышающее 5 секунд, вполне комфортно. Ответ, пришедший менее чем за 5 секунд, расценивается им как мгновенный. С этой точки зрения все анализируемые системы проявили себя очень хорошо. Задержки более 5 секунд чаще всего фиксировались у системы GigaChat. Но, возможно, это связано со скоростью работы интернета.
- В-пятых, одной из целей являлась попытка дать преподавателям информацию о том, какие системы ИИ на данном этапе их развития целесообразно использовать в образовательном процессе.

В целом можно сказать, что с большинством поставленных задач системы ИИ достаточно быстро справлялись. В графе «Общая оценка кода» указаны определенные недочеты, имеющие место при решении некоторых задач, которые преподавателю необходимо учитывать при выборе системы ИИ.

Заключение

Восприятие ИИ в обществе и, в частности, в сфере образования неоднозначно. С одной стороны, широкая доступность мощных нейронных сетей открывает новые возможности практически для любого человека. С другой стороны, специалисты указывают на отсутствие прозрачности «поведения» современных нейросетевых моделей, что вызывает определенную настороженность. Возникает необходимость тщательной проверки и верификации всех решений, принимаемых с помощью систем ИИ.

Ориентируясь на данные, полученные в проведенном исследовании, можно проявить умеренный оптимизм по поводу перспективы использования систем ИИ в программировании. Такая позиция представляется правильной, т.к. быстрое развитие моделей ИИ и появление новых возможностей свидетельствуют о продолжающемся переходном процессе. Но некоторые выводы можно сделать уже сейчас.

Как именно можно использовать системы ИИ в областях, связанных с программированием? В первую очередь, конечно, при обучении программированию. Студенты могут получать от систем ИИ качественный код с комментариями и самостоятельно осуществлять проверку его работоспособности. Важно только уметь грамотно сформулировать условие задачи. Для этого они могут использовать многочисленные онлайн-компиляторы. Так, в данной статье для прогона кода использовались площадки jdoodle и scastie. Практически всегда имеется возможность запросить у систем ИИ дополнительные пояснения, относящиеся к сгенерированному ими коду. Кроме того, можно вводить свой собственный код и просить систему его прокомментировать. Всё это можно делать как на лабораторных занятиях, так и в процессе самостоятельной работы. То есть уже сейчас системы ИИ могут оказать существенную помощь в практическом изучении основ программирования на языках высокого уровня.

Не исключена возможность применять системы ИИ и в качестве консультантов и помощников практикующих программистов [9]. Программист может давать системам ИИ небольшие конкретные задания и корректировать их решения, исходя из требований технологических стандартов компании. Поскольку ответы программист будет получать очень быстро, такое взаимодействие может повысить эффективность его работы. Кроме того, при использовании модели экстремального программирования, которая практикует широкое применение парного программирования, второй вариант решения для задач небольшой сложности можно полностью поручать системам ИИ, что позволит значительно сэкономить человеческие ресурсы. Добавим, что в работе не рассматривались позволяющие дописывать код CASE-системы типа Copilot [10].

Литература

1. *Elsen-Rooney M.* NYC education department blocks ChatGPT on school devices, networks. Chalkbeat. 2023. Jan. 4. URL: https://ny.chalkbeat.org/2023/1/3/23537987/nyc-schools-ban-chatgpt-writing-artificial-intelligence

2. *Сазонов А.П.* Использование ИИ в программировании // Universum: технические науки. 2024. Т. 3 (120) [Электронный ресурс]. DOI: 10.32743/UniTech.2024.120.3.17010, URL: https://7universum.com/ru/tech/archive/item/17010

3. *Маркус Г., Дэвис Э.* Искусственный интеллект: перезагрузка. Как создать машинный разум, которому действительно можно доверять (Библиотека Сбера: Искусственный Интеллект). М.: Изд-во Интеллектуальная литература, 2021. 328 с. URL: https://sberuniversity.ru/research/biblio/10432/ (дата доступа: 18.06.2024).

References

1. *Elsen-Rooney, M.* NYC education department blocks ChatGPT on school devices, networks. *Chalkbeat*. 2023, Jan. 4. URL: https://ny.chalkbeat.org/2023/1/3/23537987/nyc-schools-ban-chatgpt-writing-artificial-intelligence

2. *Sazonov, A.P.* Use of AI in programming. *Universum: Technical Sciences*. 2024. V. 3 (120) [Electronic resource]. DOI: 10.32743/UniTech.2024.120.3.17010, URL: https://7universum.com/ru/tech/archive/item/17010

3. *Marcus, G., Davis, E.* Artificial Intelligence: Rebooting. How to create a machine intelligence that can really be trusted (Biblioteka Sbera: Artificial Intelligence). Moscow: Publ. House Intellectual Literature, 2021. 328 p. URL: https://sberuniversity.ru/research/biblio/10432/ (accessed on: 18.06.2024).

4. *Lapan M.* Deep Reinforcement Learning Hands-On. eBook. Published by Packt, 2018.

5. *Sergievsky G., Sergievsky M.* (2023) Conception and Linguistic Means of Representation and Knowledge Processing at the Semantic Level. *Automatic Documentation and Mathematical Linguistics*. 2023. V. 57. No. 2. P. 127–133.

6. *Зайцев К.С., Сергиевский М.В.* Использование зарубежного опыта при подготовке программ обучения магистрантов в области ИТ // Alma Mater (Вестник высшей школы). 2014. № 5. С. 62–67.

7. *Farley D.* Modern Software Engineering: Doing What Works to Build Better Software Faster. Addison-Wesley, 2021. 256 p. ISBN 978-0137314911 Open Library OL34779880M

8. *Елтаренко Е.А., Сергиевский М.В.* Оценка аппаратных и программных средств по многоуровневой системе критериев // Компьютерпресс. 1998. № 8. С. 268–272.

9. Будущее программирования с помощью ИИ — первые примеры. (2023) [Электронный ресурс]. URL: https://habr.com/ru/companies/timeweb/articles/745442/

10. Writing Code with AI. [Электронный ресурс]. URL: https://docs.superblocks.com/generative-ai/writing-code-with-ai

4. *Lapan, M.* Deep Reinforcement Learning Hands-On. eBook. Published by Packt, 2018.

5. *Sergievsky, G., Sergievsky, M.* Conception and Linguistic Means of Representation and Knowledge Processing at the Semantic Level. *Automatic Documentation and Mathematical Linguistics*. 2023. V. 57. No. 2. P. 127–133.

6. *Zaytsev K.S., Sergievsky, M.V.* Using foreign experience in the preparation of master's degree programs in IT. *Alma Mater (Vestnik vysshey shkoly)*. 2014. No. 5. P. 62–67.

7. *Farley, D.* Modern Software Engineering: Doing What Works to Build Better Software Faster. Addison-Wesley, 2021. 256 p. ISBN 978-0137314911 Open Library OL34779880M

8. *Eltarenko, E.A., Sergievsky, M.V.* Evaluation of hardware and prosoftware by a multilevel system of criteria. *Computer Press*. 1998. No. 8. P. 268–272.

9. The future of AI-assisted programming — first examples. 2023. [Electronic resource]. URL: https://habr.com/ru/companies/timeweb/articles/745442/

10. Writing Code with AI. [Электронный ресурс]. URL: https://docs.superblocks.com/generative-ai/writing-code-with-ai