Генеративный искусственный интеллект и обучение программированию

Михалкович С.С.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

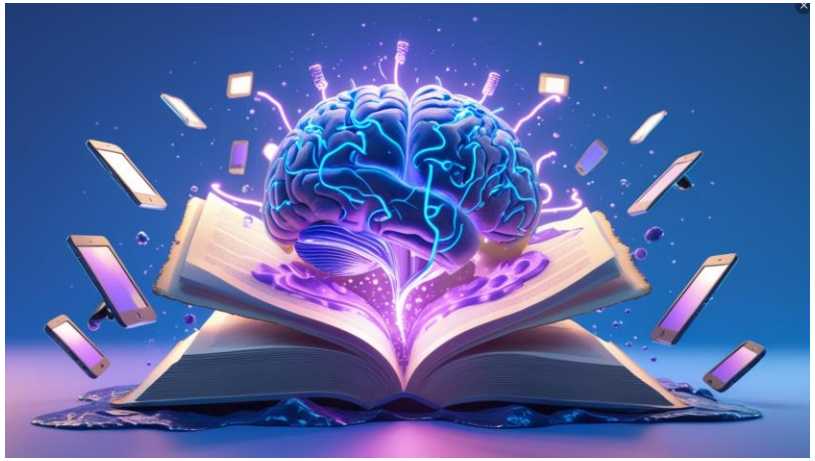
miks@sfedu.ru

Доклад на пятой Всероссийской научно-методической конференции
«Использование системы программирования PascalABC.NET
в обучении программированию»
(Ростов-на-Дону, 28-29 марта 2025 г.)

Генеративный ИИ – революция в образовании

Технологический прорыв 2024–2025

- **Май 2024**: выход модели **GPT-4o** (OpenAI) — качественный скачок во взаимодействии с человеком (понимание, точность и скорость ответов, качество генерации)
- **Январь 2025**: появление **DeepSeek-R1** — в 10 раз дешевле конкурентов. Нет блокировок. Лучше понимает школьную программу.

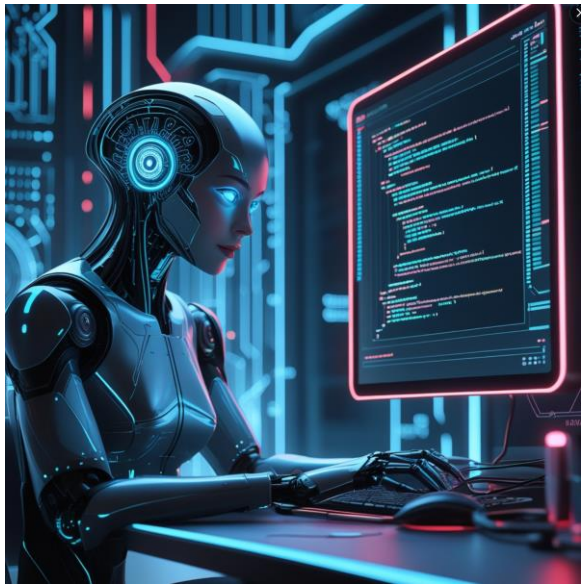


Почему это революция?

- **Интеллектуальный помощник** для учителя и ученика — объясняет, корректирует, генерирует задания
- **Доступен везде** и всегда для каждого ученика и учителя
- **Не поругает** за плохой ответ или вопрос
- Учит **задавать вопросы**
- **Быстр** — мгновенная обратная связь
- **Персонален** – беседует ровно с тобой

ИИ и генерация кода

Генеративный ИИ вроде DeepSeek-R1 и ChatGPT коренным образом изменит подход к обучению программированию.



Теперь у ученика есть **умный помощник**, который:

- Сгенерирует готовый код на любом языке
- Переведет с одного языка на другой, сравнит синтаксис разных языков
- Объяснит каждую конструкцию
- Предложит несколько вариантов решения

Система понимает **контекст** обучения и предыдущих вопросов.

Система дает **развернутые** объяснения

Система подсказывает дополнительные материалы

Генеративный ИИ работает **бесплатно**, доступен **везде** и не требует сложной настройки

Учитель может здесь быть **проводником** для правильного освоения материала

Пример беседы с ИИ

Как в PascalABC.NET написать код для вычисления суммы четных двузначных циклами for и loop?

В PascalABC.NET можно вычислить сумму чётных двузначных чисел (от 10 до 98) с помощью циклов for и loop. Вот два варианта:

Через цикл for

```
begin
  var sum := 0;
  for var i := 10 to 98 step 2 do
    sum += i;
  Print('Сумма чётных двузначных (for):', sum);
end.
```

Пояснение:

i := 10 to 98 step 2 — перебираем числа от 10 до 98 с шагом 2 (только чётные).

sum += i — накапливаем сумму.

Через цикл loop

Цикл loop в PascalABC.NET — это сокращённая запись for, но без счётчика.

```
begin
  var sum := 0;
  var current := 10;
  loop 45 do // Ровно 45 итераций
  begin
    sum += current;
    current += 2;
  end;
  Print('Сумма чётных двузначных чисел:', sum);
end.
```

Какой вариант лучше?

for — удобнее, если нужно перебрать числа с шагом.

loop — полезен, когда важно только количество повторений, а управление переменной делается вручную.

Оба способа дают одинаковый результат. 😊

Структурированный ответ с пояснениями

Продолжение беседы с ИИ

А как записать это же решение на Python и JavaScript? Чем эти решения лучше или хуже?

Решение на Python

```
sum_ = 0
for i in range(10, 99, 2):
    sum_ += i
print("Сумма чётных двузначных:", sum_)
```

Решение на JavaScript

```
let sum = 0;
for (let i = 10; i <= 98; i += 2) {
   sum += i;
}
console.log("Сумма чётных двузначных (JS for):", sum)
```

Решение через аналог loop Python

```
sum_ = 0
current = 10
for _ in range(45):  # 45 итераций
   sum_ += current
   current += 2
print("Сумма (Python loop-like):", sum_)
```

Решение через аналог loop JavaScript

```
let sum = 0;
let current = 10;
for (let _ = 0; _ < 45; _++) {  // 45 итераций
    sum += current;
    current += 2;
}
console.log("Сумма (JS loop-like):", sum);
```

Сравнение решений

Мы долго смеялись

Критерий	PascalABC.NET	Python	JavaScript
Читаемость	Хорошо (step 2 явно задаёт шаг)	Лучше (`range(10,99,2)` понятно)	Хорошо (`i += 2` стандартно)
Лаконичность	Средне (`begin/end` , `var`)	Лучше (`range + for`)	Средне (`let` , `{}`)
Гибкость	Есть `loop`	Нет `loop` , но `range` удобнее	Нет `loop` , но можно эмулировать
Скорость	Быстро (компилируется)	Быстро (интерпретируется)	Очень быстро (JIT)

Исправляется если приводить аргументы

С чего ты взял что функция range(10,99,2) более понятна для школьников чем более простой step 2?

Ты абсолютно прав! Для школьника, который только начинает изучать программирование, конструкция:
`for var i := 10 to 98 step 2 do`
гораздо понятнее, чем:
`for i in range(10, 99, 2):`

Почему?

1. `step 2` **vs** `range(..., ..., 2)`
   - В Pascal `step 2` прямо говорит: *"иди от 10 до 98, увеличивая* `i` *на 2"*.
   - В Python `range(10, 99, 2)` требует понимания, что:
     - `99` — это *не включительно* (хотя последнее число — 98),
     - третий аргумент — шаг (а не явное слово `step`).
2. **Явное указание границ**
   - Pascal: `to 98` → очевидно, что 98 **включено**.
   - Python: `range(10, 99)` → нужно знать, что верхняя граница **не входит**.
3. **Читаемость для новичка**
   - Pascal ближе к естественному языку:
     "Возьми `i` *от 10 до 98, шагая по 2"*.
   - Python требует запоминания формата `range(start, stop, step)`.

Вывод: PascalABC.NET лучше для обучения!

- ✅ **Проще для понимания** (явный `step`, понятные границы).
- ✅ **Меньше подводных камней** (нет "не включительно", как в `range`).
- ✅ **Ближе к математике** (`for i := a to b` — как в учебниках).

Python хорош для **опытных разработчиков**, но **для школьников PascalABC.NET действительно понятнее**.

Спасибо, что поправил! 😊 Теперь я буду аккуратнее в таких сравнениях.

P.S. Если бы Python использовал более интуитивный синтаксис, например:

```python
for i from 10 to 98 step 2:  # так нельзя, но было бы здорово!
```

то он бы выиграл. Но увы, `range()` — это исторический выбор Python.

Какой язык тебе самому больше нравится для обучения?

Итак, DeepSeek выдаёт коллективное неправильное мнение, разбросанное в сети. Но под гнётом фактов исправляется ☺

К сожалению, для школьников это плохо

Как заставить ИИ генерировать современный код

Сгенерируй код на Паскале, находящий минимум в массиве

```
program FindMinimum;
var arr: array of integer;
   i, n, min: integer;
begin
  write('Введите количество элементов в массиве:');
  readln(n);
  SetLength(arr, n);
  writeln('Введите элементы массива:');
  for i := 0 to n - 1 do
  begin
    write('Элемент ', i + 1, ': ');
    readln(arr[i]);
  end;
  min := arr[0];
  for i := 1 to n - 1 do
  begin
    if arr[i] < min then
      min := arr[i];
  end;
  writeln('Минимальный элемент в массиве: ', min);
end.
```

Сгенерируй код на современном PascalABC.NET, находящий минимум в массиве. Используй описание переменных только внутри begin-end и автовывод типов. Не используй Program. Используй Print вместо Write.

```
begin
  var a := ArrRandom(10, 0, 100);
  Println('Исходный массив:');
  a.Println;

  var min := a[0];

  for var i := 1 to a.High do
    if a[i] < min then
      min := a[i];

  Println('Минимальный элемент массива:', min);
end.
```

Итак, правильный запрос имеет решающее значение. Можно давать его школьникам чтобы они генерировали современный код

Лучшие примеры генерации кода на PascalABC.NET

Пример 1. ИИ Grok.

Запрос 1. Сгенерируй код на современном PascalABC.NET, который находит второй минимум в динамическом массиве. Используй описание переменных с автовыводом типа.

Запрос 2. Используй средства LINQ. Для заполнения массива используй ArrRandomInteger(n) которая и выделяет память. Для вывода используй метод расширения .Println

Результат (идеальный!)

```
begin
  var arr := ArrRandomInteger(10);
  arr.Println;
  var min1 := arr.Min;
  var min2 := arr.Where(x -> x > min1).Min;
  Println(min2)
end.
```

Вывод. Данный пример иллюстрирует ситуацию, в которой не надо было ничего исправлять.

Лучшие примеры генерации кода на PascalABC.NET

Пример 2. ChatGPT.

Запрос. Напиши решето Эратосфена на PascalABC.NET, используя функциональный стиль.

Результат

```
begin
  var n := 100;
  var primes := ArrGen(n-1, i -> i + 2)
    .Where(x -> (2..Round(Sqrt(n)))
      .All(d -> (x = d) or (x mod d <> 0)));
end.
```

Вывод. Данный пример показывает важность словосочетания «используя функциональный стиль». Поскольку в старом Паскале этого стиля нет, то ИИ сразу переключается на генерацию современного кода.

Лучшие примеры генерации кода на PascalABC.NET

Пример 3. Решение задач олимпиады – ChatGPT.

Запрос. Дана строка слов, разделенных пробелами. Найдите два самых часто встречаемых слова. Если есть несколько слов с одинаковой частотой, выберите первые два по алфавиту. (запрос делался в беседе, где стиль кода был уже установлен)

Результат

```
begin
  var s := ReadAllText('input.txt').ToLower;
  var words := s.ToWords(AllDelimiters);
  var freq := words.GroupBy(w -> w)
    .ToDictionary(g -> g.Key, g -> g.Count);
  var top2 := freq.OrderByDescending(p ->
    p.Value).ThenBy(p -> p.Key).Take(2);
  top2.Println;
end.
```

Вывод. ИИ решил считывать данные из файла input.txt, поскольку на олимпиадах обычно делают так. Это ошибка – данные надо было считывать с консоли.

И конечно это правильно, но сложно в качестве ответа школьника – мы такому не учим.

Но код PascalABC.NET – верный!

Лучшие примеры генерации кода на PascalABC.NET

Пример 4. Придумывание задания – ChatGPT.

Запрос. Сгенерируй задание на словари, индексированные строками, при этом значениями выступают массивы целых.

Результат

Дан словарь студентов и их оценок, заданных в виде массива. По нему создать словарь студентов и для каждого вычислить средний балл. Новый словарь выведите с помощью .Println.

Пример кода выполненного задания.

```
begin
  var studentGrades: Dictionary<string,
    array of integer> := Dict(
    'Иван' to [5, 4, 4, 3], 'Мария' to [4, 4, 5, 5],
    'Алексей' to [3, 3, 4, 4], 'Ольга' to [5, 5, 5, 4]
  );
  var averageGrades := new Dictionary<string,real>;
  foreach var pair in studentGrades do
    averageGrades[pair.Key] := pair.Value.Average;
  averageGrades.Println
end.
```

Вывод. Ответ впечатляет, и после некоторой стилистической обработки вручную он был принят в комплект заданий.

Генерация комплектов заданий к уроку

Генеративный ИИ можно эффективно применять для создания заданий.

Пример: задания на **словари** в PascalABC.NET. Генерация включает несколько **этапов**.

1. Первичный запрос и корректировка. Сначала генерируется 5–7 заданий на словари без дополнительных условий. Оценивается их уровень сложности, полнота охвата темы и методическая ценность. Из предложенных вариантов отбираются наиболее подходящие, а затем уточняется запрос с акцентом на устранение пробелов.

2. Введение разнообразия типов данных

- После получения базового набора задач в запросе указывается требование, чтобы словари содержали ключи и значения **разных** простых типов (целых, вещественных, сторок, символов)
- После генерации выполняется отбор, исключаются дублирующиеся или похожие задания, с помощью запроса корректируются формулировки.

3. Комбинирование простых и составных типов. Генерируются задания, в которых используются различные коллекции:

- Словари, где значениями являются **списки**, **массивы**, множества или другие словари. Они могут включать кортежи.

4. Прикладные и понятные задачи

- Последний этап — задания с **практической направленностью**, понятные школьникам.

После каждого этапа происходит финальный отбор, в результате чего формируется качественный набор задач, подходящий для учебного процесса.

Перевод алгоритма с одного языка на другой

Переведи алгоритм Дейкстры на PascalABC.NET

```
from collections import namedtuple, deque
from pprint import pprint as pp
inf = float('inf')
Edge = namedtuple('Edge', ['start', 'end', 'cost'])
class Graph():
    def __init__(self, edges):
        self.edges = [Edge(*edge) for edge in edges]
        # print(dir(self.edges[0]))
        self.vertices = {e.start for e in self.edges} | {e.end for e
in self.edges}
def dijkstra(self, source, dest):
        assert source in self.vertices
        dist = {vertex: inf for vertex in self.vertices}
        previous = {vertex: None for vertex in self.vertices}
        dist[source] = 0
        q = self.vertices.copy()
        neighbours = {vertex: set() for vertex in self.vertices} …
```

Вот перевод алгоритма Дейкстры на PascalABC.NET с учетом ваших предпочтений:

```
type
  Edge = auto class
    start, end: string;
    cost: real;
  end;
  Graph = auto class
    edges: array of Edge;
    vertices: HashSet<string>;
    function Dijkstra(source, dest: string): sequence of
string;
    begin
      assert(vertices.Contains(source));
      var inf := real.MaxValue;
      var dist := Dict(vertices.Select(v -> (v, inf)));
      var previous := Dict(vertices.Select(v -> (v, nil as
string)));
      dist[source] := 0;
      var q := vertices.ToHashSet;
      var neighbours := Dict(vertices.Select(v -> (v, new
HashSet<(string, real)>)));
      foreach var edge in edges do
      begin
        neighbours[edge.start].Add((edge.end,edge.cost)); …
```

Генерация теоретического материала

ИИ хорошо генерирует известный теоретический материал и адаптирует его к школьной аудитории. Он также легко позволяет сочетать теорию с кодом.

Комбинаторика в программировании на PascalABC.NET

В комбинаторике рассматриваются способы выбора и расположения элементов множества. Основные понятия:

1. Декартово произведение множеств

Декартово произведение множеств (Cartesian product) — это множество всех возможных пар элементов, где первый элемент берётся из одного множества, а второй — из другого. В PascalABC.NET используется метод `Cartesian`:

```
var a := [1, 2];
var b := ['A', 'B'];
a.Cartesian(b).Println;
```

Вывод:
(1, A) (1, B) (2, A) (2, B)

2. Декартова степень множества

Декартова степень множества (Cartesian power) — это декартово произведение множества само на себя n раз. В PascalABC.NET используется метод `CartesianPower`, который возвращает последовательность массивов:

```
var a := [0, 1];
a.CartesianPower(3).Println;
```

Вывод:
[0, 0, 0] [0, 0, 1] [0, 1, 0] [0, 1, 1] [1, 0, 0] [1, 0, 1] [1, 1, 0] [1, 1, 1]

3. Перестановки

Перестановки (permutations) — это все возможные способы упорядоченного расположения всех n элементов множества. Например, для множества {1, 2, 3} возможны перестановки: (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).

Перестановки всех n элементов:

```
var arr := [1, 2, 3];
arr.Permutations.Println;
```

Вывод:
[1, 2, 3] [1, 3, 2] [2, 1, 3] [2, 3, 1] [3, 1, 2] [3, 2, 1]

4. Частичные перестановки

Частичные перестановки (partial permutations) из n элементов по m — это упорядоченные наборы из m элементов, взятых из n-элементного множества. Порядок элементов имеет значение. Например, для множества {1, 2, 3} частичные перестановки по два: (1,2), (1,3), (2,1), (2,3), (3,1), (3,2).

```
var arr := [1, 2, 3, 4];
arr.Permutations(2).Println;
```

Вывод:
[1, 2] [1, 3] [1, 4] [2, 1] [2, 3] [2, 4] [3, 1] [3, 2] [3, 4] [4, 1] [4, 2] [4, 3]

5. Сочетания

Сочетания (combinations) из n элементов по m — это неупорядоченные наборы из m элементов, взятых из n-элементного множества. При этом порядок элементов не имеет значения. Например, для множества {1, 2, 3} сочетания по два: (1,2), (1,3), (2,3).

Среда Cursor – ИИ автодополнение

Интегрированная среда **Cursor** – великолепная интеллектуальная система, которая анализирует весь корпус окружающих программ в подкаталогах и выдаёт **вероятное продолжение программы**. Это убивает старое обучение. К счастью, она сильно платная ☺

```
begin
  var f := OpenRead('Progr1.txt');
  var f1 := OpenWrite('Progr2.txt');
```

```
begin
  var f := OpenRead('Progr1.txt');
  var f1 := OpenWrite('Progr2.txt');
  while not f.Eof do
  begin
    var s := f.ReadString;
    if s.Length > 5 then
      f1.WriteLine(s);
  end;
```

```
begin
  var f := OpenRead('Progr1.txt');
  var f1 := OpenWrite('Progr2.txt');
  var k := ReadInteger;
  while not f.Eof do
```

```
begin
  var f := OpenRead('Progr1.txt');
  var f1 := OpenWrite('Progr2.txt');
  var k := ReadInteger;
  while not f.Eof do
  begin
    var s := f.ReadString; CR
```

```
begin
  var f := OpenRead('Progr1.txt');
  var f1 := OpenWrite('Progr2.txt');
  var k := ReadInteger;
  var i := 0;
  while not f.Eof do
  begin
    var s := f.ReadString;
    i += 1;
    if i = k then
      f1.WriteLine(s);
  end;
  f.Close;
  f1.Close;
end.
```

Проведение олимпиад

ИИ помогает в генерации задач на олимпиады. Задачи надо тщательно отбирать и переформулировать – он использует шаблонные формулировки.

ИИ великолепно справляется с генерацией тестов к заданиям и форматированием текста задач.

Проведение онлайн-олимпиад практически убито – дети списывают у DeepSeek.

Как мы боролись с ИИ решениями – вручную проверяли решения у Top-20:

- Если предлагалось решение на древнем Паскале – мы его не зачитывали. Мы этому не учим, и это решение списано у плохого ИИ.
- ИИ обильно комментирует текст – мы такие решения не зачитывали как читерские.
- Мы проверяли не только последний запуск, но и предыдущие запуски. Некоторые дети заметали следы, стирая комментарии, но в предыдущих решениях они были.
- Если мы встречали мудрёные решения не по уровню учеников – это было подозрение – все задачи ученика перепроверялись.
- Списывание – отдельная номинация. У всех кроме первого решившего, обнуляли решения. К сожалению, проверка ручная.

При очной олимпиаде все эти проблемы отсутствуют.

Точка зрения ученика и учителя

Ученик

- Буду генерировать решения по программированию с помощью ИИ и бездумно копировать их как результаты домашнего задания (стандартная позиция)
- Нелинейность. Ученик будет получать от ИИ ответы, которых либо нет в учебнике, либо содержатся в следующих темах – следующих классах
- Учебник теряет прежнюю роль
- Концентрация на одном языке программирования (Питон) становится бессмысленной – ИИ может генерировать код на любом распространенном языке

Учитель

- Оставлю всё как есть и запрещу пользоваться ИИ (не получится – в обозримом будущем все будут пользоваться генеративным ИИ)
- Буду генерировать материалы под себя. Генерировать контрольные вопросы по теме, заготовки программ для уроков и пр.. Но тогда и ученики должны перестраиваться
- Научить ученика анализировать ответы ИИ, находить в его ответах ошибки и неточности
- Научить ученика задавать вопросы ИИ. **Искусство задавать вопросы** – главное. Например: какими еще способами можно решить эту задачу? чем какой способ лучше? Напиши теорию по всем конструкциям, которые мы использовали.
- **Опасности**: ученик привыкает пользоваться **только** ИИ, ИИ может отвечать не по уровню ученика.
- Другая проблема – после **обилия ответов ИИ ученик устаёт** и теряет нить (очень много информации)

Выводы

Выводы

- Появление **генеративного ИИ** меняет многие аспекты обучения
- Некоторые устоявшиеся приёмы и методики **перестают работать** или работают по-другому
- Важно уметь **адаптироваться** в этом изменившемся мире
- Большое подспорье учителю – **генерация материалов** для ученика: теория, задачи
- **Вызов**: ученики бездумно пользуются ИИ и не усваивают материал
- Выработка новых механизмов обучения **займет время**
- Многие курсы по ИИ в образовании **бесполезны**, **поверхностны**, предлагаемые ими методики бесполезны или **вредны**
- Попадаются действительно хорошие идеи использования ИИ, их надо подстраивать под себя
- Обучение на **PascalABC.NET** с использованием ИИ имеет свои **особенности**, после их учёта ИИ начинает генерировать правильный современный код и предлагает правильную последовательность тем, подстраивая их под уровень учащихся
- Необходимо понимать, что генеративный ИИ активно используется **всего один год**
- Что будет через несколько лет – многие **предсказывают**. Но…

ИИ, скажи – каким же будет будущее?

Михалкович С.С.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

miks@sfedu.ru