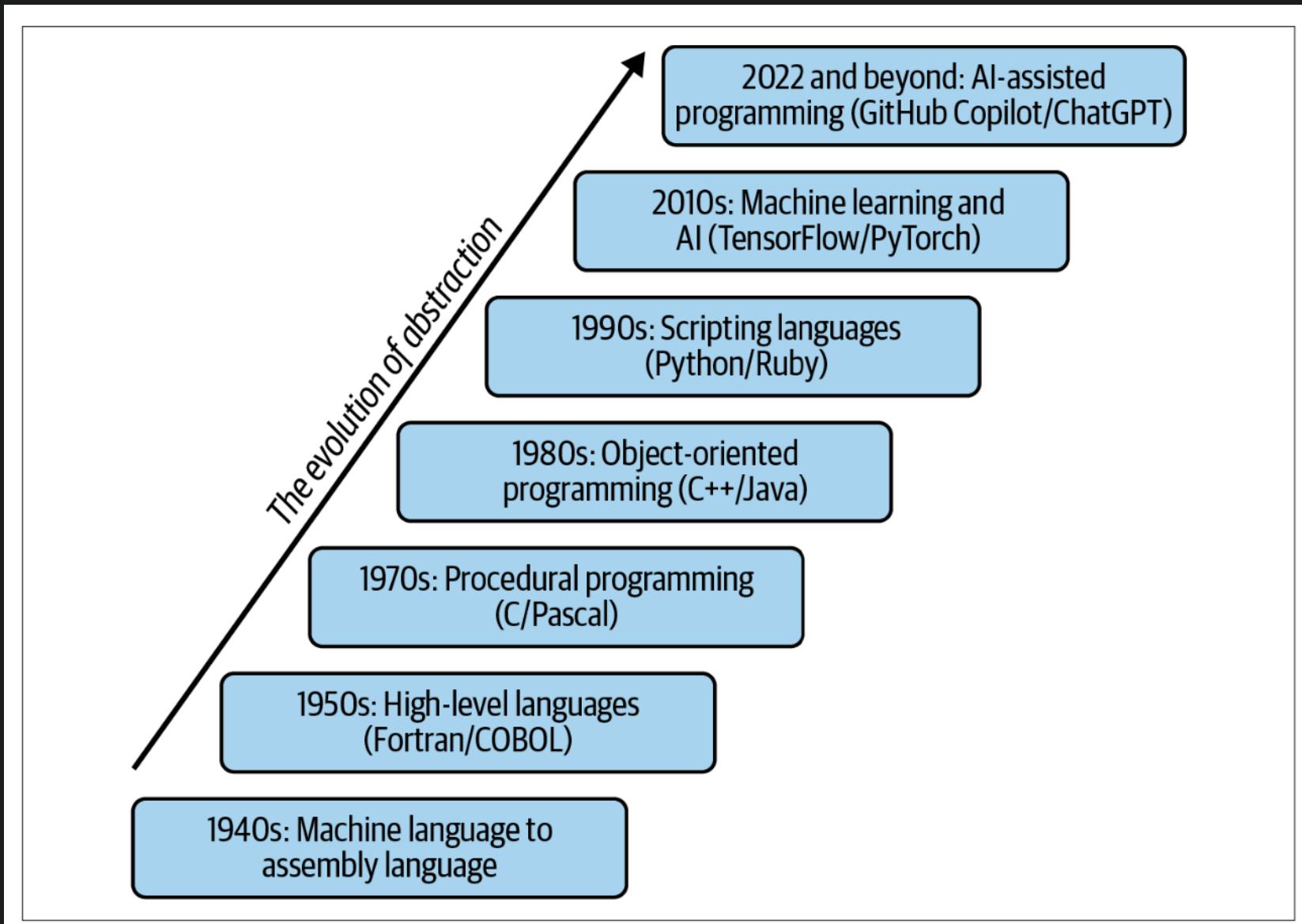




Практика применения AI-ассистента

# Эволюция разработки



Рутинные детали скрываются и разработчики могут сосредоточиться на самом важном.

Получают новые возможности, увеличивают технологический рычаг.

# Разработчики говорят о приросте производительности



88%

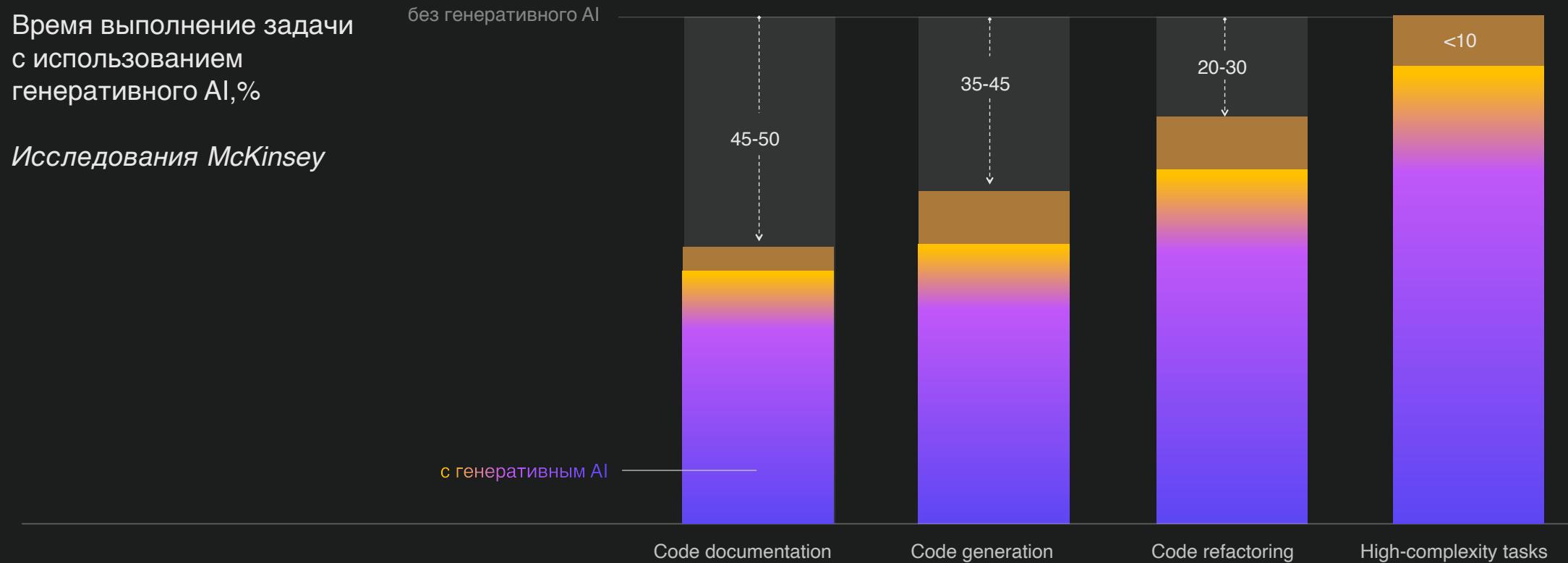
Исследования GitHub Copilot\*



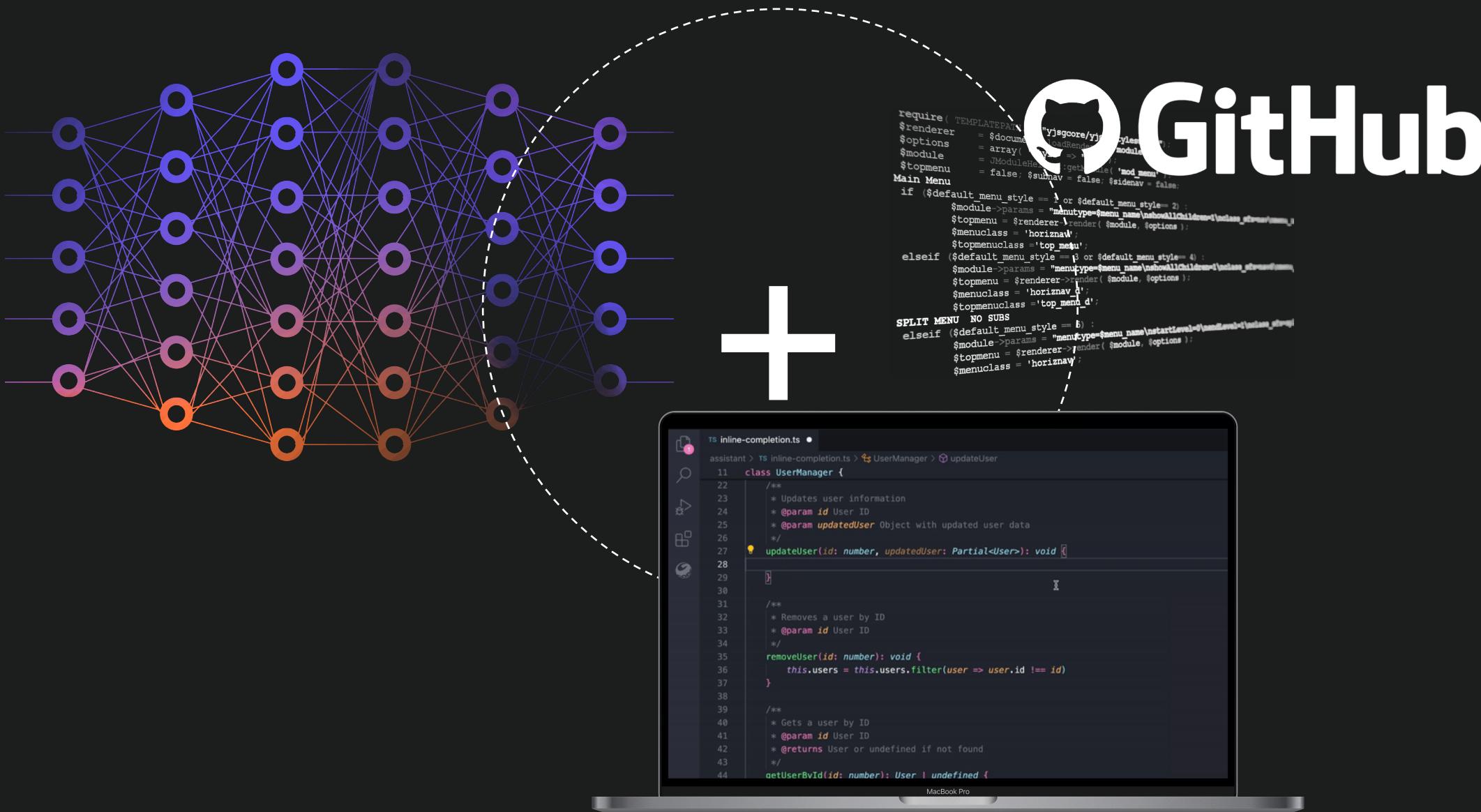
91%

Опрос в Сбере. Июль-Август 2024

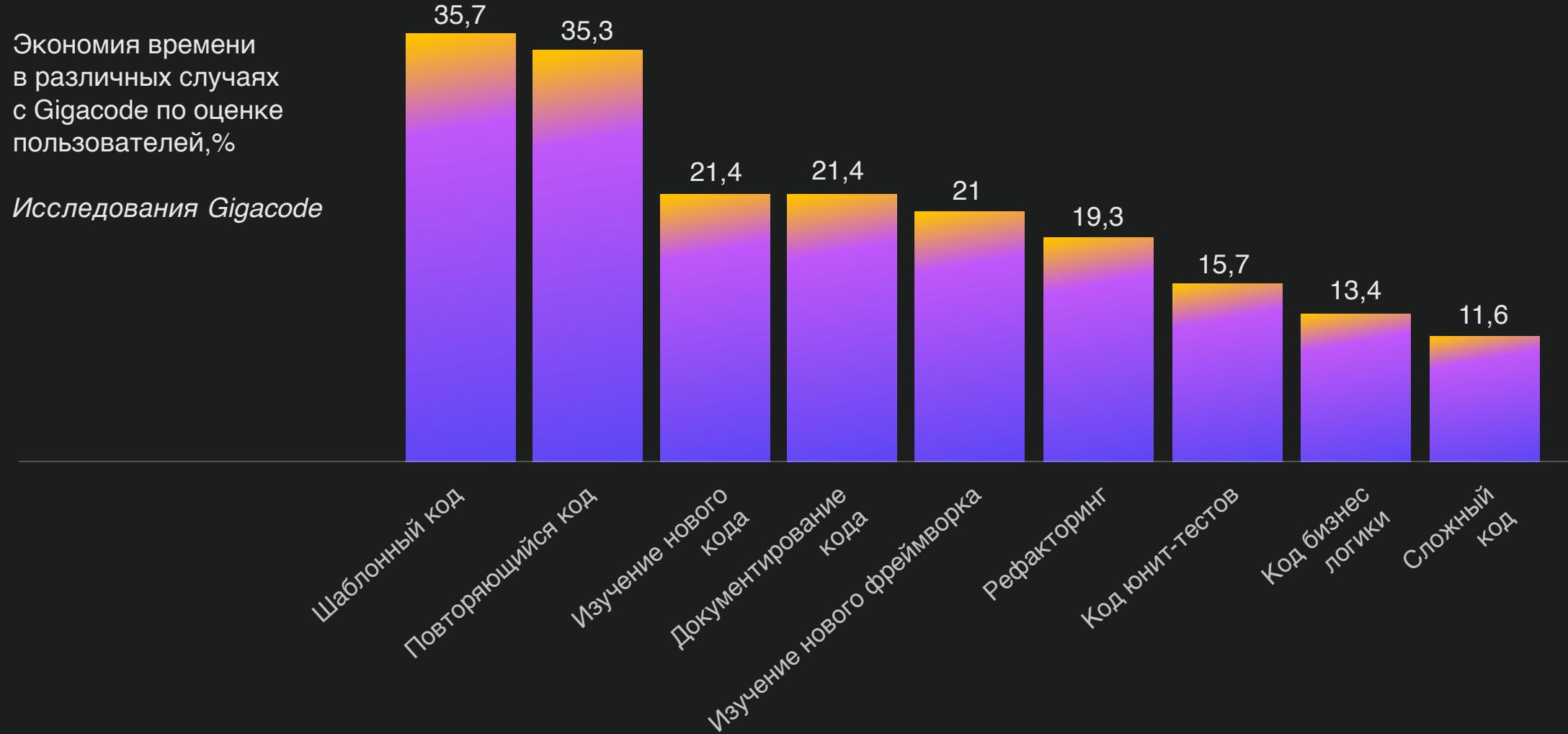
# Помощь AI неоднородна и зависит от типа задач



# Что такое AI-ассистент в программировании

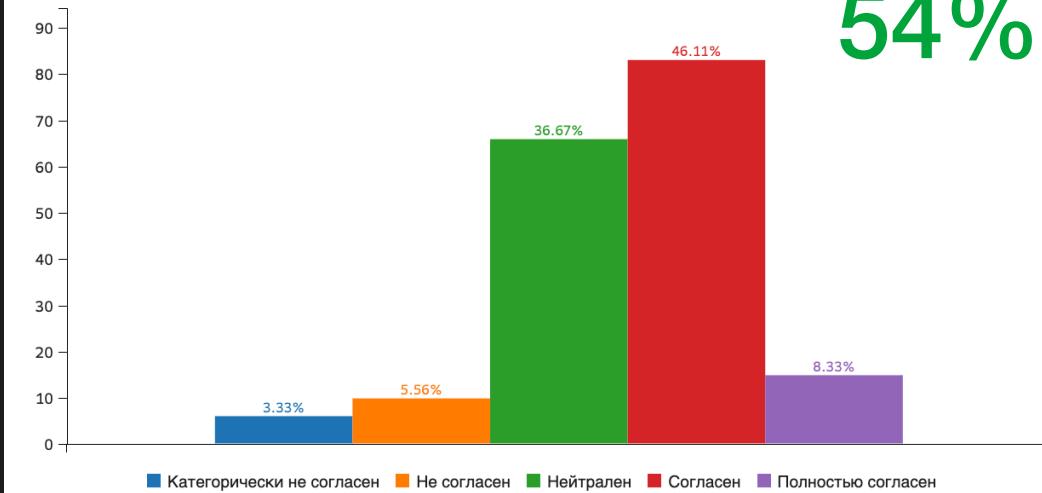


# Это подтверждается и нашими результатами



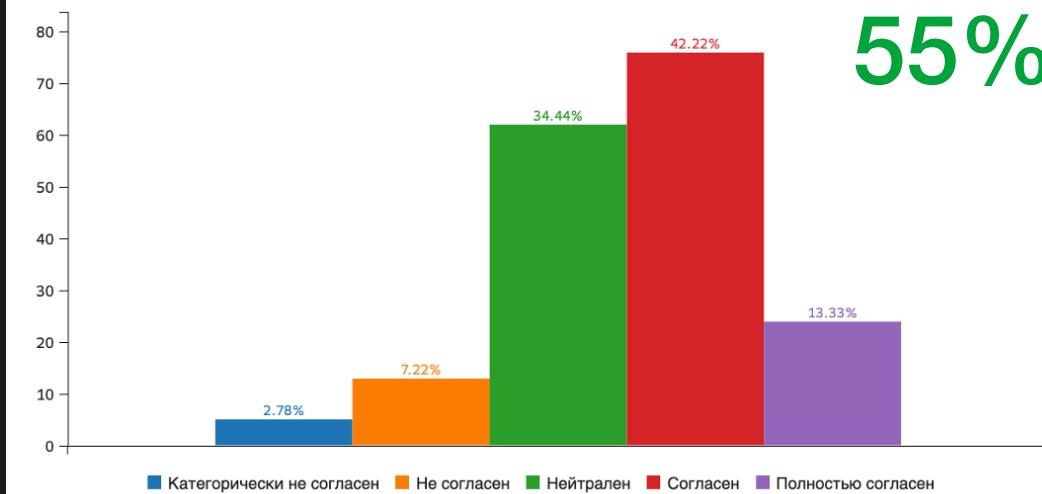
# GigaCode повышает удовлетворенность

Получаю больше удовольствия от кодирования

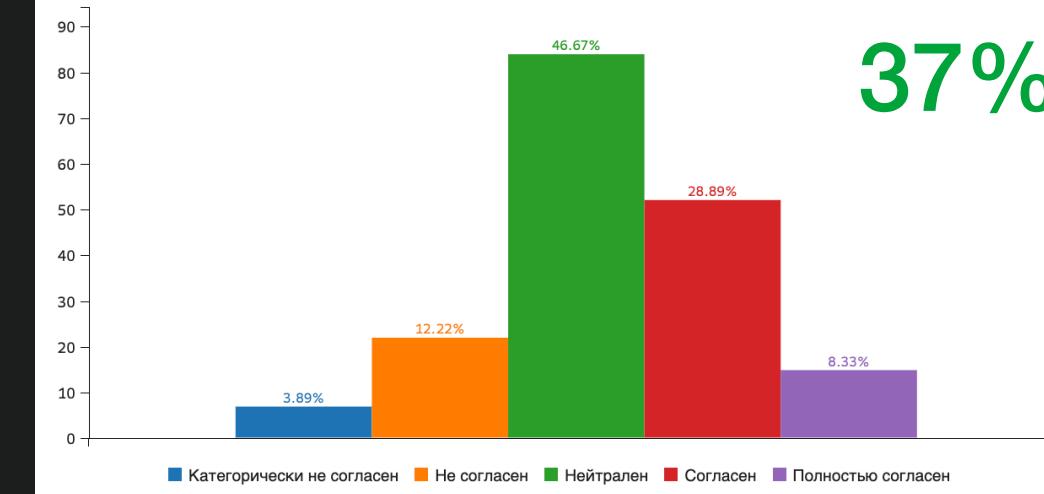


Более 50% разработчиков более удовлетворены своей работой и меньше устают, более 1/3 получает возможность концентрации на более важных задачах

Меньше устаю от повторяющихся задач

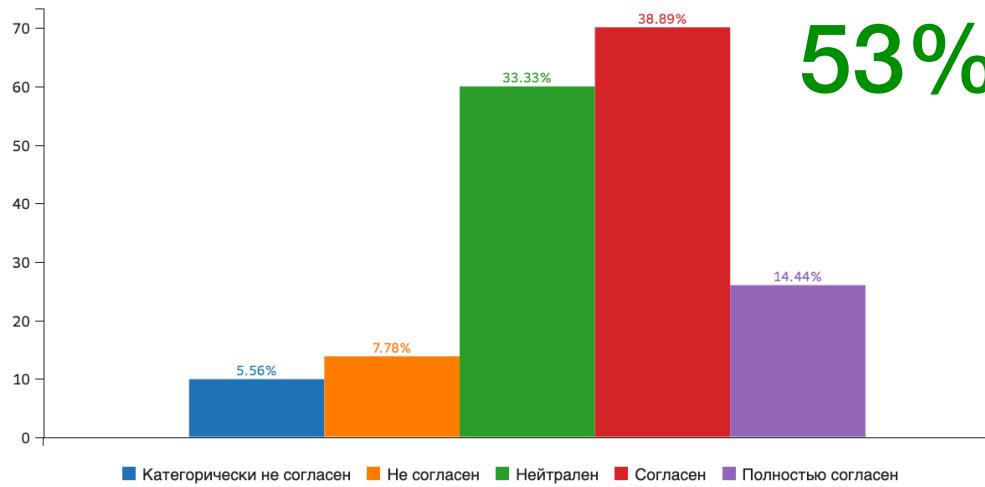


Фокусируюсь на более интересных и значимых задачах



# GigaCode оптимизирует рабочее время разработчиков

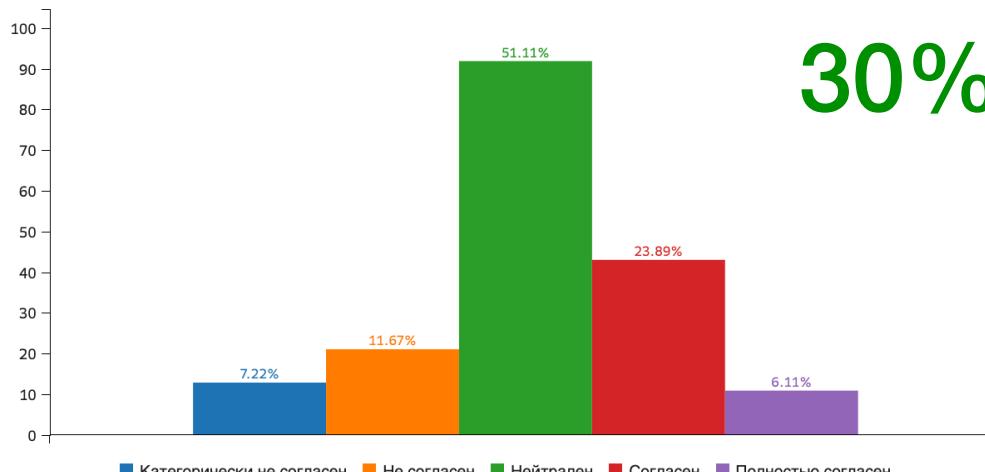
Трачу меньше времени на поиск информации и примеров кода



53%

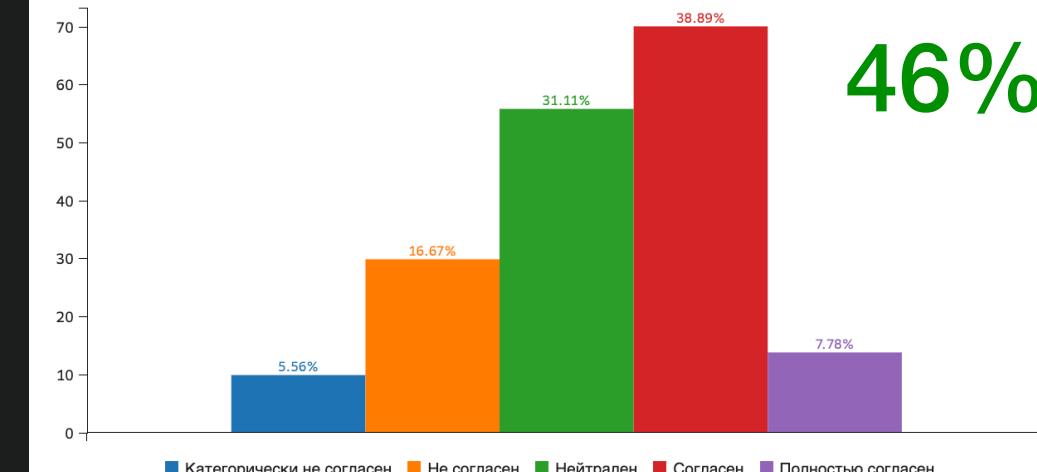
а также повышает их квалификацию

Меньше переключаюсь между задачами и больше нахожусь в состоянии «потока»



30%

Учуясь на подсказках, которые предлагает мне GigaCode



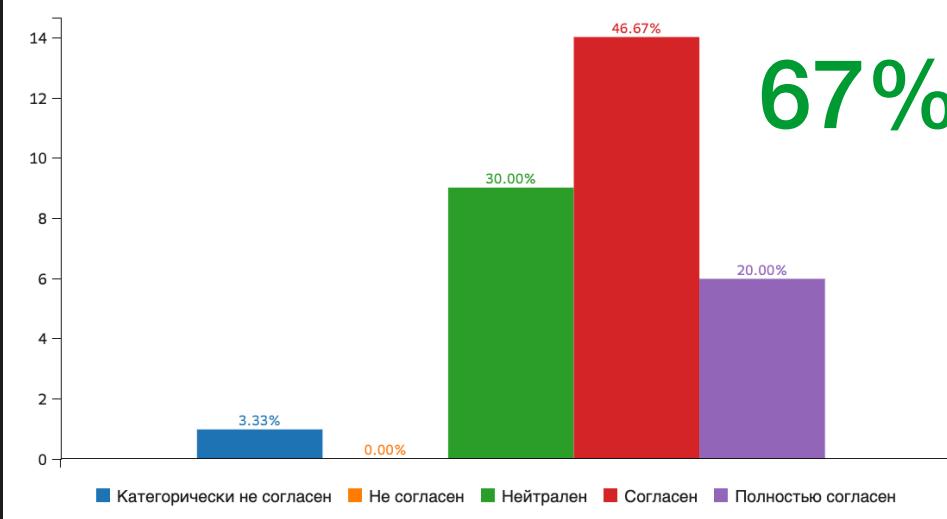
46%

# GigaCode значительно повышает эффективность и помогает начинающим разработчикам

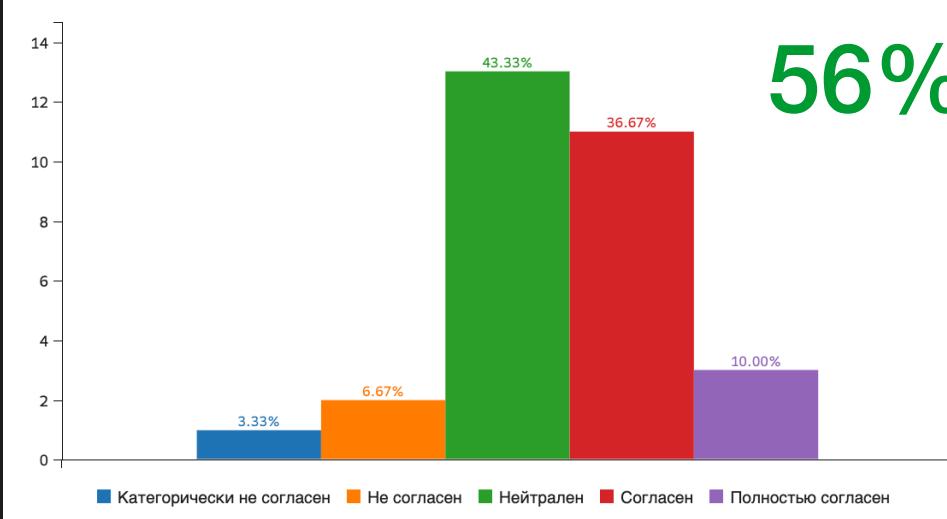
9

Junior

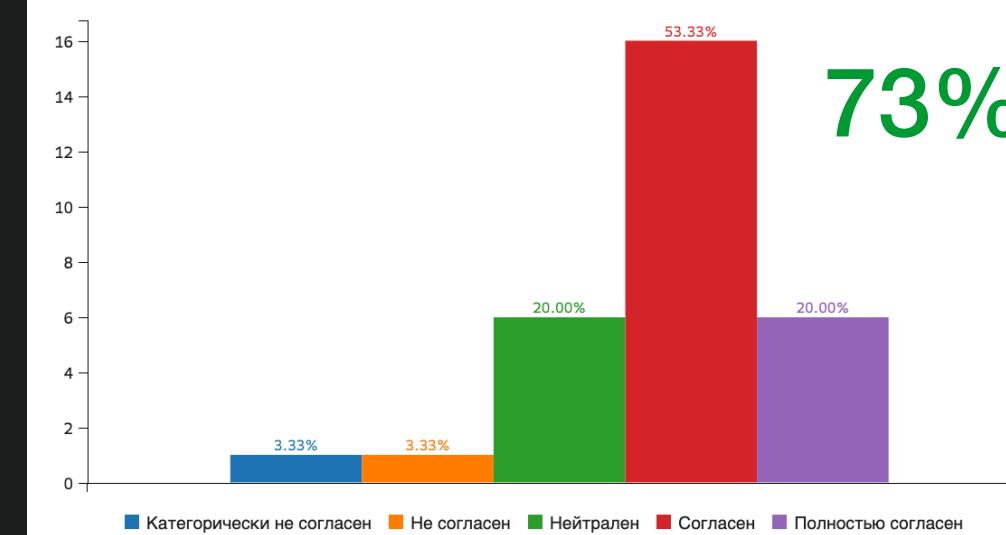
Трачу меньше времени на поиск информации и примеров кода



Меньше переключаюсь между задачами и больше нахожусь в состоянии «потока»



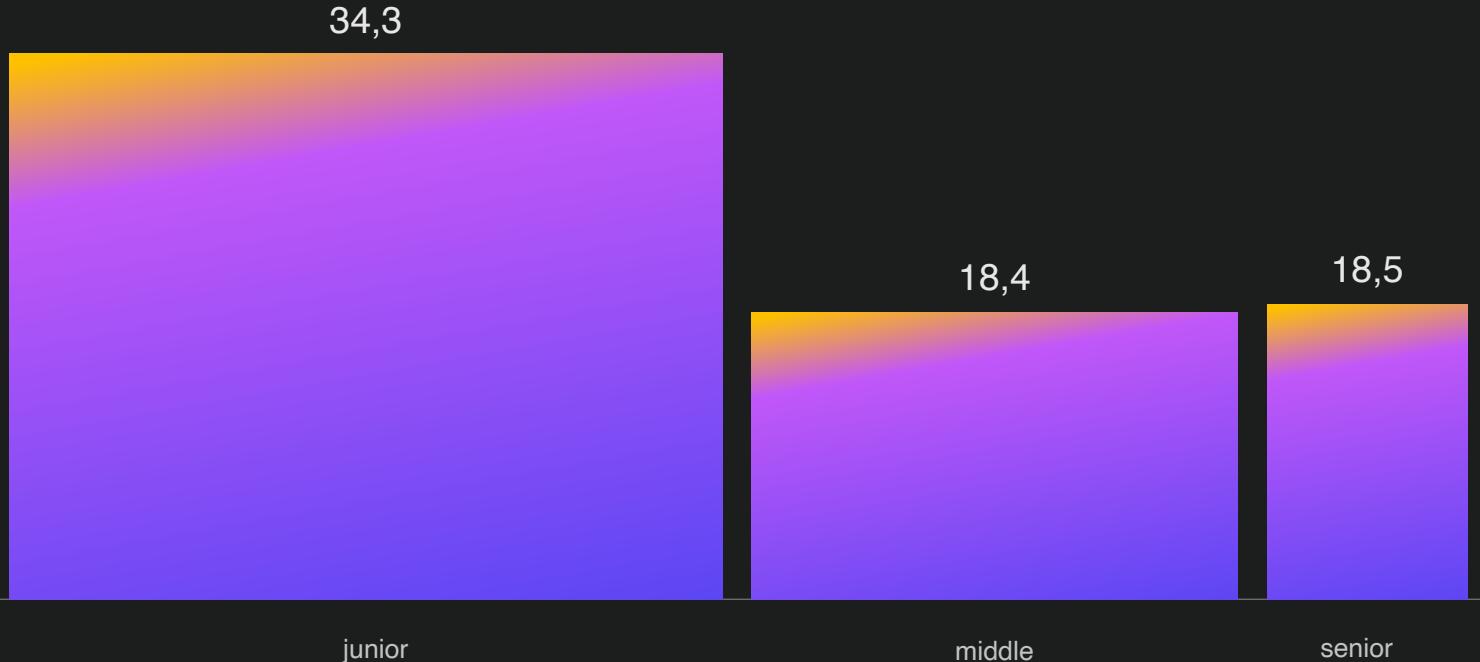
Учусь на подсказках, которые предлагает мне GigaCode



# Эффективность GigaCode среди разработчиков различных уровней

Распределение общей  
экономии времени  
по уровню разработчика  
по мнению самих  
разработчиков, %

*Исследования Gigacode*



# Точки входа в ассистента GigaCode

## Inline Code Assistant

Автодополнение, продолжающее кода после курсора.

- ✓ Реализует как однострочные, так и мультилайн подсказки
- ✓ Подсказывает в месте написания кода, уменьшая количество рутины

## Chat with GigaCode

Интеллектуальный чат, находящийся в отдельной вкладке *Chat* в IDE.

- ✓ Помогает быстро находить ответы на вопросы, не выходя из среды IDE
- ✓ Благодаря чату можно освоить новые сферы и изучить новые технологии

# Автодополнение кода: inline suggestions, multiline, lookup

## Inline Code Assistant

Интегрированное в редактор кода  
автодополнение, подсказывающее  
вероятные продолжения строки исходя  
из контекста текущего файла  
и проекта сразу после курсора

### Однострочная подсказка

```
1 const http = require('http')
2 const fs = require('fs')
3 const port = 8080
4
5 const server = http.createServer((request, response) => {
6   res.writeHead(200, {'Content-Type': 'text/html'})
7   res.write
8   res.end('Hello, world!')
```

### Многострочная подсказка

```
private static class MyWelcomeHandler implements HttpHandler {
    @Override
    public void handle(HttpExchange httpExchange) throws IOException {
        httpExchange.sendResponseHeaders(200, "Welcome to Sber!".length());
        OutputStream os = httpExchange.getResponseBody();
        os.write("Welcome to Sber!".getBytes());
        os.close();
    }
}
```

### Comments to code

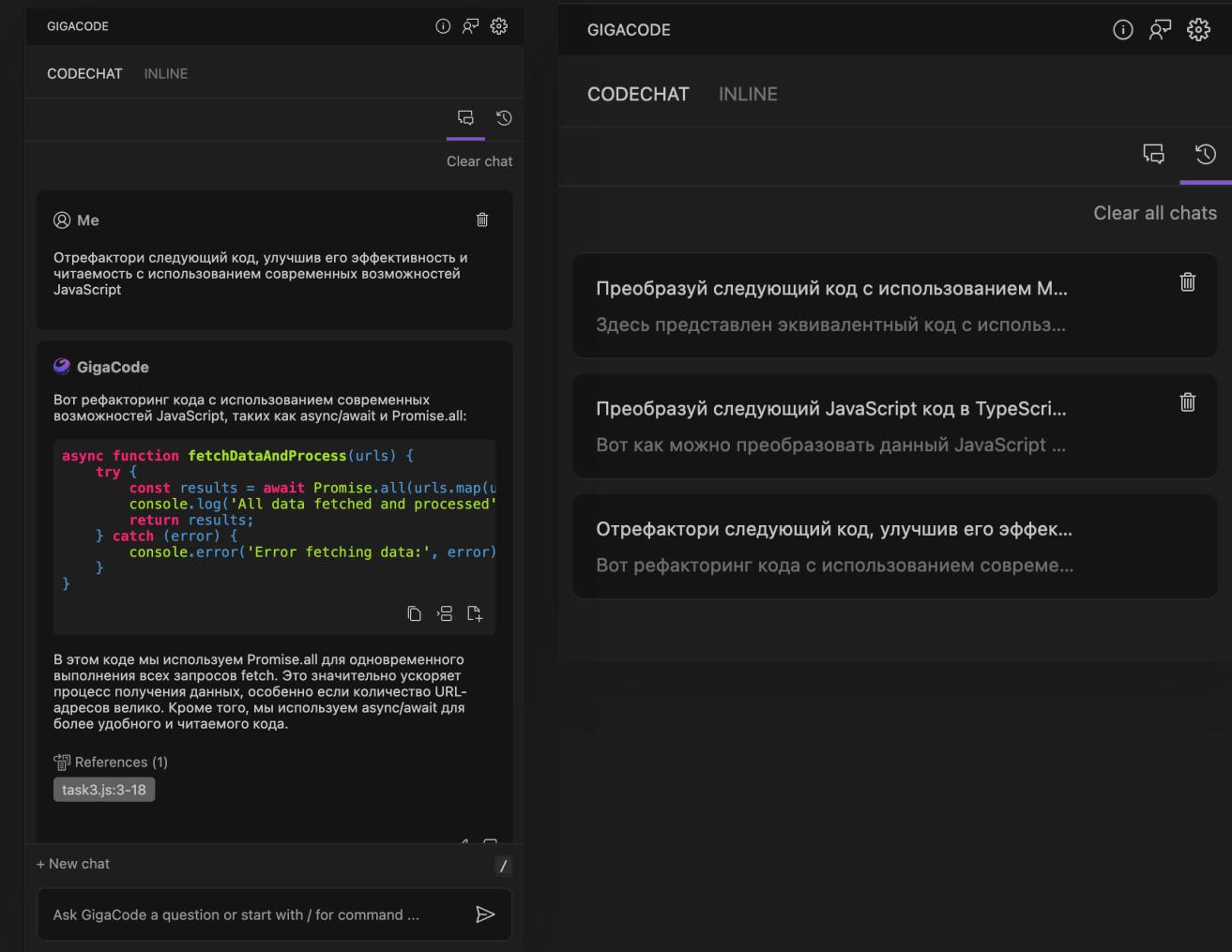
```
1 # write a function that generates a random password with a specific length
2 def random_password(length):
3     import random, string
4     password = ""
        for i in range(length):
            password += random.choice(string.ascii_letters + string.digits)
    return password
```

MacBook Pro

# Чат с ассистентом

## Chat with GigaCode

Чат с ассистентом, обладающий как стандартным набором возможностей, так и работой с текущим окном кода. Помимо этого имеет набор коротких команд: объяснение кода, генерация документации и тестов



The screenshot shows the GigaCode interface. On the left, a code editor window titled 'CODECHAT' displays the following JavaScript code:

```
async function fetchDataAndProcess(urls) {
  try {
    const results = await Promise.all(urls.map(url => fetch(url).then(response => response.json())));
    console.log('All data fetched and processed');
    return results;
  } catch (error) {
    console.error('Error fetching data:', error);
  }
}
```

Below the code, a note explains: "В этом коде мы используем Promise.all для одновременного выполнения всех запросов fetch. Это значительно ускоряет процесс получения данных, особенно если количество URL-адресов велико. Кроме того, мы используем async/await для более удобного и читаемого кода."

On the right, a list of recent chats is shown:

- Clear chat
- Me: Отрефактори следующий код, улучшив его эффективность и читаемость с использованием современных возможностей JavaScript
- GigaCode: Вот рефакторинг кода с использованием современных возможностей JavaScript, таких как async/await и Promise.all:
 

```
async function fetchDataAndProcess(urls) {
  try {
    const results = await Promise.all(urls.map(url => fetch(url).then(response => response.json())));
    console.log('All data fetched and processed');
    return results;
  } catch (error) {
    console.error('Error fetching data:', error);
  }
}
```
- Clear all chats
- Преобразуй следующий код с использованием M...: Здесь представлен эквивалентный код с использ...
- Преобразуй следующий JavaScript код в TypeScri...: Вот как можно преобразовать данный JavaScript ...
- Отрефактори следующий код, улучшив его эфек...: Вот рефакторинг кода с использованием соврем...

At the bottom of the interface, there is a search bar with the placeholder "Ask GigaCode a question or start with / for command ...".

# Короткие команды: explain, explain step by step, doc, test

Пример  
использования

```
GigaCode: explain | explain step by step | doc | test
1  function transformData(input) {
2    if (Array.isArray(input)) {
3      return input.map(item => item.trim().toUpperCase())
4    } else if (typeof input === 'string') {
5      return input.trim().toUpperCase()
6    } else {
7      throw new Error('Invalid input type')
8    }
9 }
```

TEST LIBRARIES

/Jest

/Mocha

/Jasmine

/Ava

/Tape

+ New chat

/test prog\_lang:js;lang:eng;

Me

/explain\_stepbystep lang:ru;

GigaCode

- Функция fetchDataAndProcess(urls) принимает массив URL в качестве аргумента.
- Создается пустой массив results, в который будут добавляться результаты запросов.
- Начинается цикл for, который проходит по каждому URL в массиве urls.
- Для каждого URL вызывается функция fetch, которая отправляет HTTP-запрос на указанный URL.
- Полученный ответ от сервера обрабатывается с помощью метода .then(response => response.json()). Этот метод преобразует ответ в формат JSON.
- Полученные данные (data) добавляются в массив results с помощью метода .push(data).
- Если длина массива results равна длине массива urls, то выводится сообщение 'All data fetched and processed' в консоль. Это означает, что все данные были получены и обработаны.
- Если во время выполнения кода произошла ошибка, она обрабатывается с помощью метода .catch(error => console.error('Error fetching data:', error)). Эта ошибка выводится в консоль.
- После обработки всех URL функция fetchDataAndProcess(urls) завершает свою работу.

References (1)

task3.js:3-18

MacBook Pro

# Подведем итоги интеграции ИИ в процессы разработки

**Снижение  
времени на  
рутинные задачи**

**Общее повышение  
скорости разработки**

**Снижение кривой  
обучения для новых  
разработчиков**

**Повышение  
удовлетворенности  
при работе**



<https://gigacode.ru>