Научный семинар по исследованиям цифровой экономики
«Искусственный интеллект и учебный процесс»

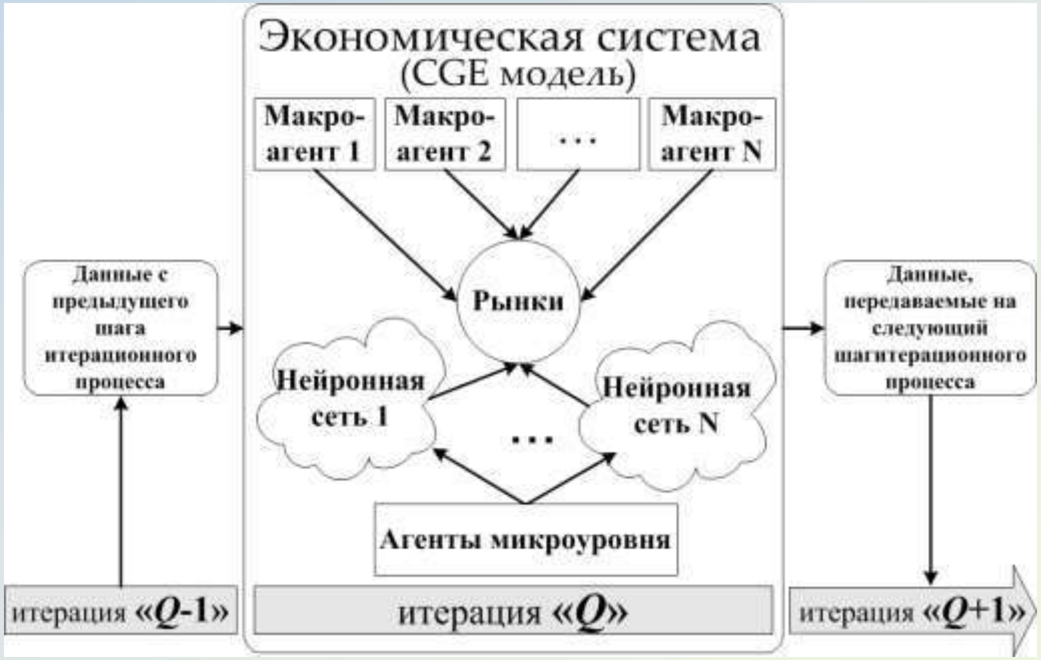
Технологии ИИ для моделирования социально-экономических систем

Бахтизин А.Р.

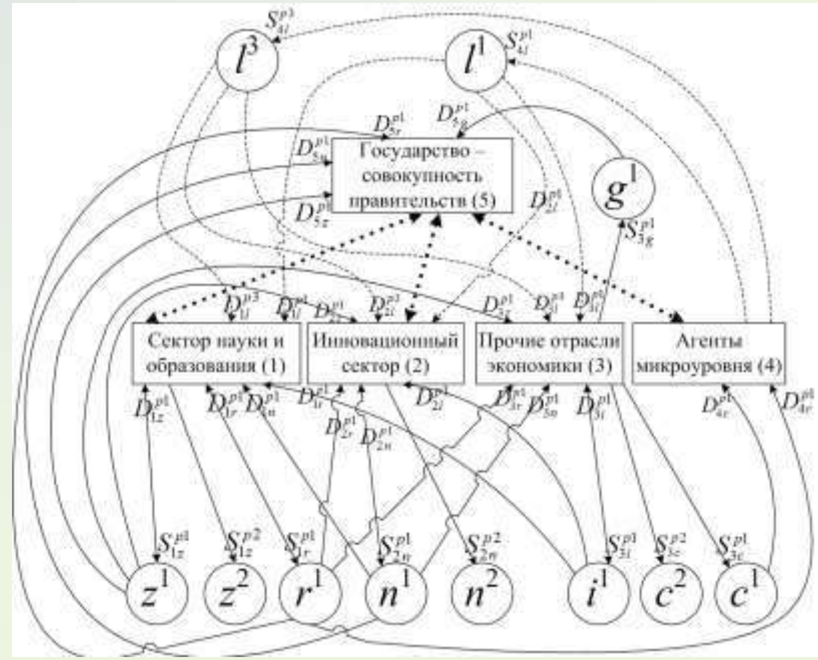
Москва, 05.04.2023

Основные направления ИИ в экономике:

1. Нейронные сети
2. Экспертные системы
3. Нечеткая логика
4. Генетические алгоритмы



Концептуальный взгляд на взаимодействие макро- и микроуровней



Укрупненная схема взаимосвязей между основными компонентами модели

Нейронная сеть (три слоя, внутренний – 10 нейронов)

$$N_1^j = \frac{\sum_{k=1}^{10}\left(\frac{1}{1+e^{-\left(\left(U_1^j \cdot r_{(1)scale}^{in} + r_{(1)shift}^{in}\right)\cdot w_1^{2k} - \tau_1^{2k}\right)}} \cdot w_1^{3k}\right) - \tau_1^3 - r_{(1)shift}^{out}}{r_{(1)scale}^{out}}$$

Generative Pre-trained Transformers (GPT)

Генеративные предварительно обученные трансформеры

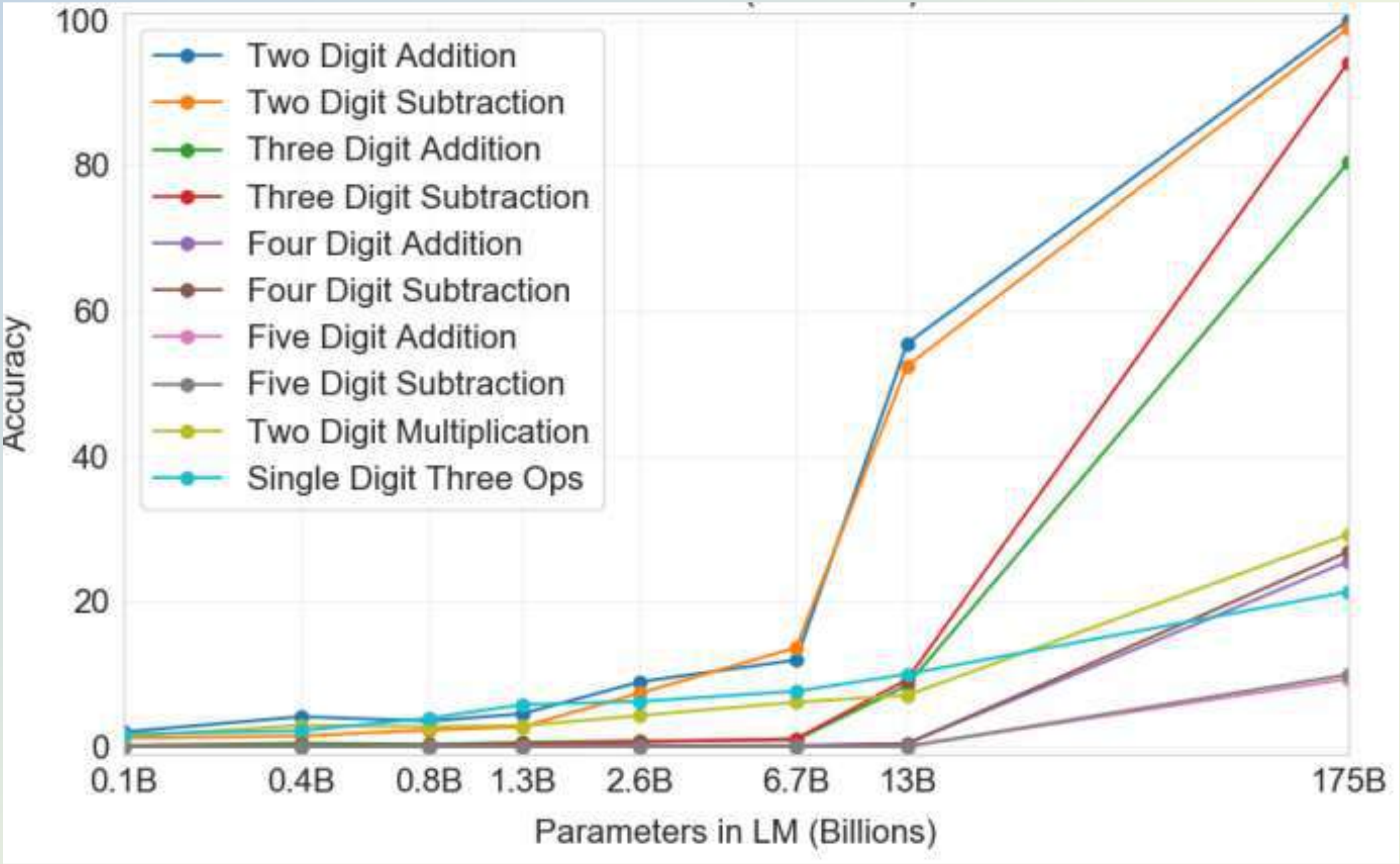
(предшественник - технология интеллектуального ввода текста для мобильных телефонов Т9)

Языковые модели	**Количество параметров**	**Тренировочные данные**	**Дата выпуска**
GPT-1	0,117 миллиарда	**4,5 ГБ** текста из 7000 книг разных жанров	11 июня 2018 г.
GPT-2	1,5 миллиарда	**40 ГБ** текста, 8 миллионов документов	14 февраля 2019 г.
GPT-3	175 миллиардов	**570 ГБ**, в основном CommonCrawl, WebText, английская Википедия	11 июня 2020 г.
ChatGPT (на базе GPT-3.5)	175 миллиардов	Дополнительное обучение	30 ноября 2022 г.
GPT-4	Неизвестно	Неизвестно	14 марта 2023 г.

OpenAI: доступны веса и технические детали нейронной сети **GPT-2**, технические подробности **GPT-3**, для **GPT-4** пока нет информации.

GPT-4 демонстрирует способности в нескольких стандартизированных тестах. В ходе тестирования модель получила 1410 баллов по Академическому оценочному тесту (***94-й процентиль***), 163 балла по Law School Admission Test (***88-й процентиль***) и 298 баллов по Единому экзамену на адвоката (***90-й процентиль***).

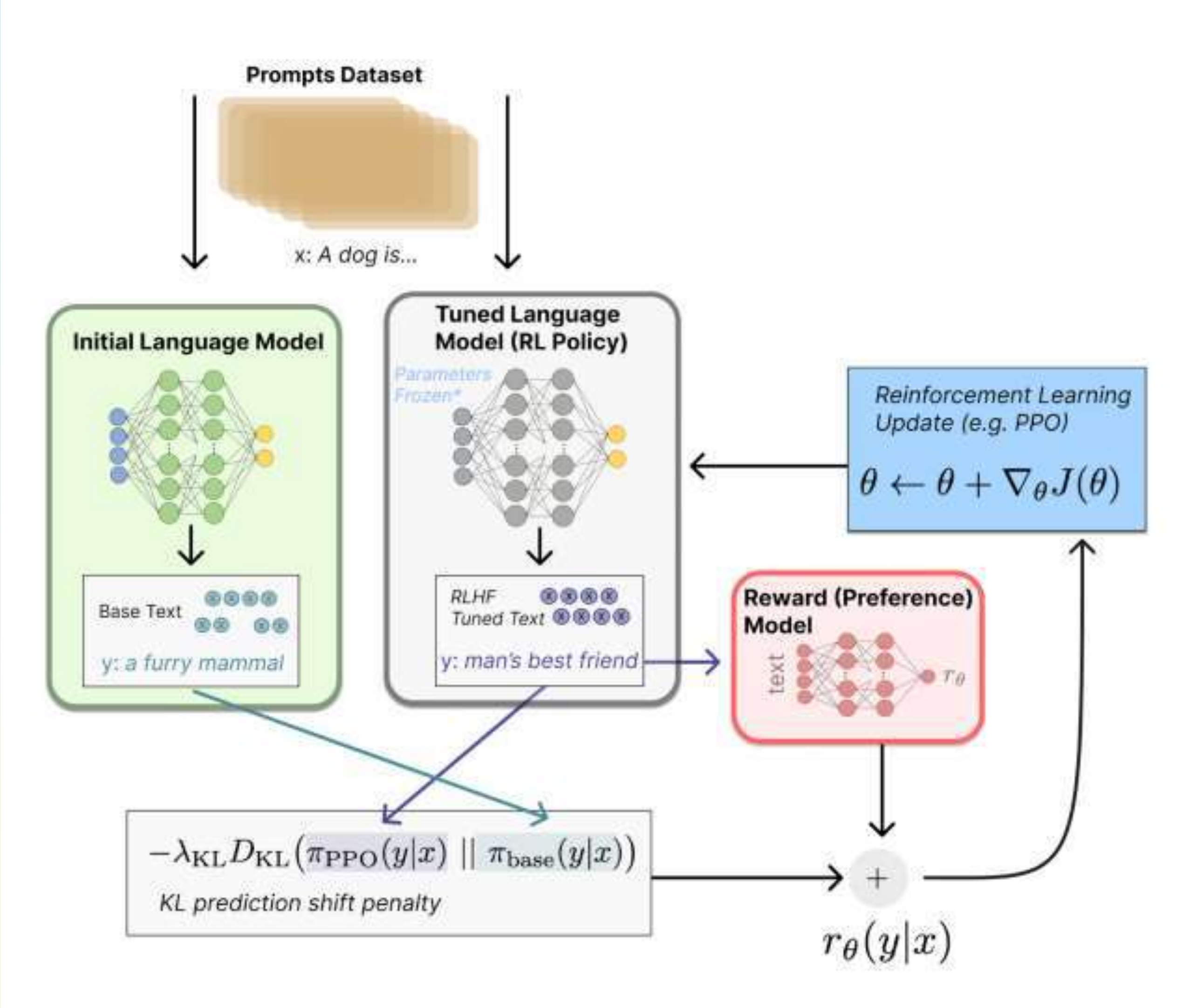
Точность решения 10 арифметических задач моделями различных размеров: точность резко возрастает при увеличении параметров



Tom B. Brown et al. Language Models are Few-Shot Learners
https://doi.org/10.48550/arXiv.2005.14165

Одно из знаковых событий 2022 года в IT мире – запуск **ChatGPT**, **Generative Pre-trained Transformer** *(разработка OpenAI) 30 ноября 2022 г.:* **https://chat.openai.com**

- *175 млрд. параметров*
- *Разработан с использованием методов обучения с учителем и с подкреплением*

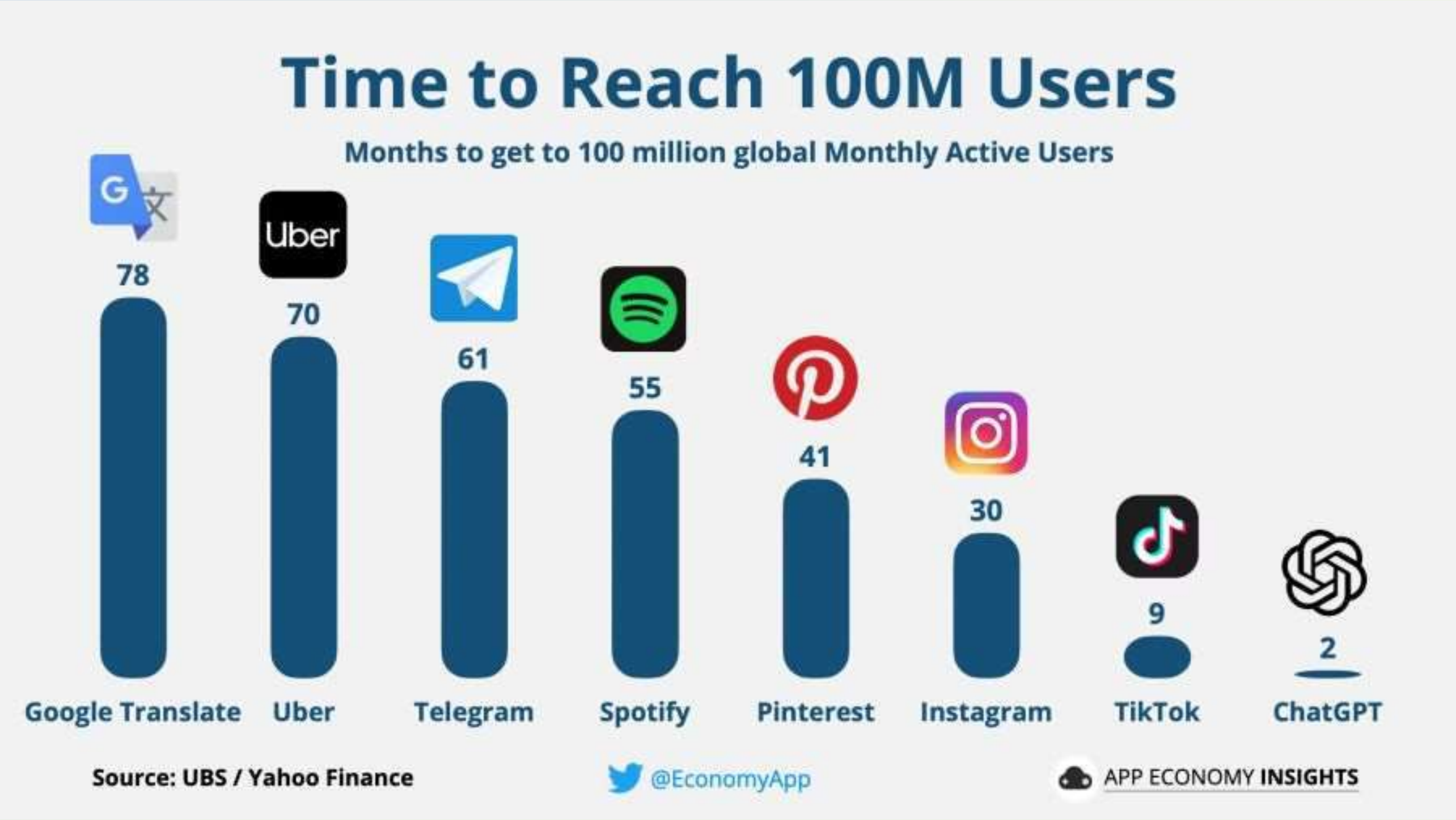


https://huggingface.co/blog/rlhf

Рекорд по скорости привлечения новых пользователей:

1 млн за 5 дней

100 млн за 2 месяца



AL can you generate c# code for agent based model?

Yes, I can generate C# code for an agent-based model. Here is an example of how you might create an agent-class and a model class in C#:

Copy code

```
using System;
using System.Collections.Generic;

class Agent
{
    public int ID { get; set; }
    public double Wealth { get; set; }

    public Agent(int id)
    {
        ID = id;
        Wealth = 1;
    }
}

class Model
{
    private List<Agent> agents;
    private Random rnd;

    public Model(int numAgents)
    {
        agents = new List<Agent>();
        rnd = new Random();

        for (int i = 0; i < numAgents; i++)
        {
            Agent a = new Agent(i);
            agents.Add(a);
        }
    }

    public void Step()
    {
        int randomIndex = rnd.Next(0, agents.Count);
        Agent selectedAgent = agents[randomIndex];
        // code to make the selected agent perform its behavior
    }
}
```

This is just a simple example of an ABM that defines an agent class `Agent` and a model class `Model`. In this example, the agents are created with an initial wealth of 1 and each time step, an agent is randomly selected to perform its behavior.

It's important to note that this is just an example, you'll need to adjust the code according to your specific model and requirements. Also, ABM are complex models and you should have a good understanding of the problem you want to model and the behavior of the agents to create a realistic and useful model.

Please let me know if you have any specific question or if you want me to generate code for other specific ABM.

write a description of this model no less than 2000 characters

This demographic agent-based model (ABM) simulates the behavior and interactions of individual agents within a population. The agents in the model can represent individuals, households, or other units of analysis, and their behavior and interactions are based on demographic processes, such as births, deaths, migration, and aging.

The model is implemented in C# and has two classes: **Agent** and **Population**. The **Agent** class has two properties: **Age** and **IsAlive**. The **Age** property keeps track of the age of the agent and the **IsAlive** property indicates whether the agent is alive or not. The **Population** class has a list of **Agent** objects, a **currentYear** variable, a **maxAge** variable, and a **deathProbability** variable. The **currentYear** variable keeps track of the current year in the simulation, the **maxAge** variable represents the maximum age an agent can reach, and the **deathProbability** variable is the probability of an agent dying in a given year.

The **Simulate()** method is responsible for advancing the simulation by one year for each agent. It increments the age of each agent and checks if they have reached the maximum age or if a randomly generated number is less than the death probability. If either of these conditions is met, the agent's **IsAlive** property is set to false. This process is repeated until there are no more agents alive in the population.

The **Main** method creates a population of 1000 agents with a maximum age of 80 and a death probability of 0.01. It then calls the **Simulate()** method, which runs the simulation until there are no more agents alive. The program then outputs the year when the simulation ends.

This is a basic example of a demographic ABM, and it can be extended to include more demographic processes such as births, migration, etc. The model can also be modified to include different types of agents with different characteristics and behaviors, and to simulate more realistic populations. The ABM approach allows for a more granular and individualized simulation of demographic processes, as opposed to aggregate models. It can be useful for studying the impacts of different demographic scenarios on a population, and for understanding the complex interactions between individuals and their environment.

write a description of this model no less than 5000

This demographic agent-based model (ABM) simulates the behavior and interactions of individual agents within a population. The agents in the model can represent individuals, households, or other units of analysis, and their behavior and interactions are based on demographic processes, such as births, deaths, migration, and aging. The ABM approach allows for a more granular and individualized simulation of demographic processes, as opposed to aggregate models. It can be useful for studying the impacts of different demographic scenarios on a population, and for understanding the complex interactions between individuals and their environment.
The model is implemented in C# and has two classes: Agent and Population. The Agent class has two properties: Age and IsAlive. The Age property keeps track of the age of the agent and the IsAlive property indicates whether the agent is alive or not.

The Population class has a list of Agent objects, a currentYear variable, a maxAge variable, and a deathProbability variable. The currentYear variable keeps track of the current year in the simulation, the maxAge variable represents the maximum age an agent can reach, and the deathProbability variable is the probability of an agent dying in a given year. This variable can be set to a constant or can be a function of age and/or other factors such as health, socioeconomic status, and environmental conditions.

The Simulate() method is responsible for advancing the simulation by one year for each agent. It increments the age of each agent and checks if they have reached the maximum age or if a randomly generated number is less than the death probability. If either of these conditions is met, the agent's IsAlive property is set to false. This process is repeated until there are no more agents alive in the population.

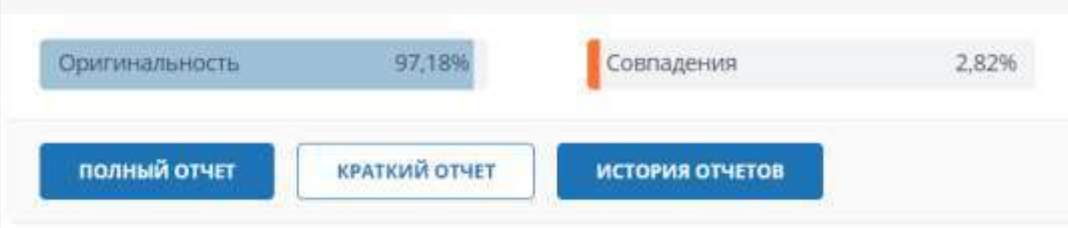
The Main method creates a population of 1000 agents with a maximum age of 80 and a death probability of 0.01. It then calls the Simulate() method, which runs the simulation until there are no more agents alive. The program then outputs the year when the simulation ends.

This basic example of a demographic ABM can be further improved and extended in several ways. For example, the model can include births by adding a birth rate variable and creating new agents at each time step. Additionally, migration can be included by allowing agents to move in or out of the population based on certain conditions. Furthermore, more realistic populations can be simulated by introducing different types of agents with different characteristics and behaviors, such as gender, socioeconomic status, and health status.

Moreover, the model can be enhanced to include more realistic demographic processes such as marriage, divorce, and family formation. These processes can be modeled by introducing new agents and allowing agents to form relationships with one another. Additionally, the model can be expanded to include education, employment, and other social-economic variables that affect demographic processes.

In addition, the model can be linked to other models or data sources, such as land use or economic models, to simulate the impact of changes in those areas on population demographics.

Overall, this demographic agent-based model provides a flexible and powerful tool for simulating and analyzing demographic processes at the individual level. It can be used to study the impacts of different demographic scenarios on a population, and to understand the complex interactions between individuals and their environment.



Международная конференция по машинному обучению (ICML):

Статьи, включающие текст, сгенерированный с помощью крупномасштабной языковой модели (LLM), такой как ChatGPT , запрещены, если созданный текст не представлен как часть экспериментального анализа статьи.

https://icml.cc/Conferences/2023/llm-policy

Запрет государственных школ Нью-Йорка (6 января 2023)

New York City public schools ban access to AI tool that could help students cheat

New York City public schools will ban students and teachers from using ChatGPT, a powerful new AI chatbot tool, on the district's networks and devices, an official confirmed to CNN on Thursday.

https://edition.cnn.com/2023/01/05/tech/chatgpt-nyc-school-ban/index.html

---

Edward Tian: I spent New Years building GPTZero — an app that can quickly and efficiently detect whether an essay is ChatGPT or human written

3:17 AM · 3 янв. 2023 г.·**7,4 млн** просмотров

Анализ предложений на предмет их равномерности. Люди пишут неравномерно (короткие и длинные предложения), ИИ более однороден

AL

can you generate more complex code?

Yes, I can generate more complex code for agent-based models, depending on the requirements of the model. For example, if you need to include additional features such as communication between agents, different types of agents with different behaviors, or a more sophisticated schedule for activating agents, the code can become more complex.

It's important to note that creating a complex ABM requires a good understanding of the problem you want to model, the behavior of the agents, and the interactions between them. It also requires advanced programming skills and knowledge of the programming language being used.

Here's an example of a more complex agent-based model in C# that implements a simple form of communication between agents:

Copy code

```
using System;
using System.Collections.Generic;

class Agent
{
    public int ID { get; set; }
    public double Wealth { get; set; }
    public List<Agent> Neighbors { get; set; }

    public Agent(int id)
    {
        ID = id;
        Wealth = 1;
        Neighbors = new List<Agent>();
    }

    public void SendMessage(Agent recipient, string message)
    {
        recipient.ReceiveMessage(this, message);
    }

    public void ReceiveMessage(Agent sender, string message)
    {
        Console.WriteLine("Agent {0} received message '{1}' from agent {2}",
            ID, message, sender.ID);
    }
}

class Model
{
    private List<Agent> agents;
    private Random rnd;

    public Model(int numAgents)
    {
        agents = new List<Agent>();
        rnd = new Random();

        for (int i = 0; i < numAgents; i++)
        {
            Agent a = new Agent(i);
            agents.Add(a);
        }

        // Connect agents randomly
        for (int i = 0; i < numAgents; i++)
        {
            Agent a = agents[i];
            int numNeighbors = rnd.Next(1, numAgents / 2);
            for (int j = 0; j < numNeighbors; j++)
            {
                Agent b = agents[rnd.Next(0, numAgents)];
                if (a != b && !a.Neighbors.Contains(b))
                {
                    a.Neighbors.Add(b);
                }
            }
        }
    }

    public void Step()
    {
        int randomIndex = rnd.Next(0, agents.Count);
        Agent selectedAgent = agents[randomIndex];
        // code to make the selected agent perform its behavior
        // and send a message to one of its neighbors
        int neighborIndex = rnd.Next(0, selectedAgent.Neighbors.Count);
        Agent selectedNeighbor = selectedAgent.Neighbors[neighborIndex];
        selectedAgent.SendMessage(selectedNeighbor, "Hello, neighbor!");
    }
}
```

This example includes a class `Agent` that has a list of neighbors, and a method `SendMessage` to send a message to one of its neighbors. The model class `Model` creates a number of agents and randomly connects them as neighbors. Each time step, one agent is randomly selected, performs its behavior and sends a message to one of its neighbors.

can you realize demographic agent based model?

Yes, I can help you create a demographic agent-based model (ABM). In a demographic ABM, agents represent individuals or households, and their behavior and interactions are used to simulate population dynamics, such as births, deaths, migrations and other demographic events.

Here's an example of a basic demographic ABM in Python using the `mesa` library:

realize it using c#

Yes, I can help you create a demographic agent-based model (ABM) in C#. In a demographic ABM, agents represent individuals or households, and their behavior and interactions are used to simulate population dynamics, such as births, deaths, migrations, and other demographic events.

Here's an example of a basic demographic ABM in C# that simulates the aging of a population:

Copy code

```
using System;
using System.Collections.Generic;

class PersonAgent
{
    public int ID { get; set; }
    public int Age { get; set; }
    public string Gender { get; set; }
    public bool IsAlive { get; set; }

    public PersonAgent(int id)
    {
        ID = id;
        Age = 0;
        Gender = "male";
        IsAlive = true;
    }

    public void Step()
    {
        Age++;
        if (Age >= 80)
        {
            IsAlive = false;
        }
    }
}

class PopulationModel
{
    private List<PersonAgent> population;
    private Random rnd;

    public PopulationModel(int numAgents)
    {
        population = new List<PersonAgent>();
        rnd = new Random();

        for (int i = 0; i < numAgents; i++)
        {
            PersonAgent a = new PersonAgent(i);
            a.Gender = rnd.Next(0, 2) == 0 ? "male" : "female";
            population.Add(a);
        }
    }

    public void
```

AL

can you make parallel version of it?

Yes, I can help you create a parallel version of a demographic agent-based model (ABM) in C#. There are several ways to make an ABM parallel, depending on the specific requirements of the model and the hardware available.

One way to make an ABM parallel is to use multi-threading, which allows you to run multiple threads in parallel on a single machine. In C#, you can use the `System.Threading` namespace to create and manage threads. Here is an example of how you might use multi-threading to make the previous demographic ABM parallel:

```
using System;
using System.Collections.Generic;
using System.Threading;

class PersonAgent
{
    public int ID { get; set; }
    public int Age { get; set; }
    public string Gender { get; set; }
    public bool IsAlive { get; set; }

    public PersonAgent(int id)
    {
        ID = id;
        Age = 0;
        Gender = "male";
        IsAlive = true;
    }

    public void Step()
    {
        Age++;
        if (Age >= 80)
        {
            IsAlive = false;
        }
    }
}

class PopulationModel
{
    private List<PersonAgent> population;
    private Random rnd;

    public PopulationModel(int numAgents)
    {
        population = new List<PersonAgent>();
        rnd = new Random();

        for (int i = 0; i < numAgents; i++)
        {
            PersonAgent a = new PersonAgent(i);
            a.Gender = rnd.Next(0, 2) == 0 ? "male" : "female";
            population.Add(a);
        }
    }

    public void Run(int numThreads)
    {
        // Split the population into numThreads chunks
        int chunkSize = population.Count / numThreads;
        List<List<PersonAgent>> chunks = new List<List<PersonAgent>>();
        for (int i = 0; i < numThreads; i++)
        {
            int startIndex = i * chunkSize;
            int endIndex = startIndex + chunkSize - 1;
            if (i == numThreads - 1)
            {
```

can you add procedure that realize migration method in this code?

Yes, I can help you add a migration method to the demographic agent-based model (ABM) in C#. Migration can be modeled in many ways in an ABM, depending on the specifics of the problem and the available data. Here is an example of how you might add a migration method to the previous ABM that randomly selects an agent and moves it to a different location:

```
using System;
using System.Collections.Generic;
using System.Threading;

class PersonAgent
{
    public int ID { get; set; }
    public int Age { get; set; }
    public string Gender { get; set; }
    public bool IsAlive { get; set; }
    public string Location { get; set; }

    public PersonAgent(int id)
    {
        ID = id;
        Age = 0;
        Gender = "male";
        IsAlive = true;
        Location = "A";
    }

    public void Step()
    {
        Age++;
        if (Age >= 80)
        {
            IsAlive = false;
        }
    }

    public void Migrate()
    {
        List<string> locations = new List<string> {"A", "B", "C", "D", "E"};
        locations.Remove(Location);
        int randomIndex = rnd.Next(0, locations.Count);
        Location = locations[randomIndex];
    }
}

class PopulationModel
{
    private List<PersonAgent> population;
    private Random rnd;

    public PopulationModel(int numAgents)
    {
        population = new List<PersonAgent>();
        rnd = new Random();

        for (int i = 0; i < numAgents; i++)
        {
            PersonAgent a = new PersonAgent(i);
            a.Gender = rnd.Next(0, 2) == 0 ? "male" : "female";
            population.Add(a);
        }
    }

    public void Run(int numThreads)
```

This is a simple example of an agent-based model that simulates a population of agents that grow older over time. The `Agent` class has properties for age and whether the agent is alive, as well as methods for growing older, giving birth, and migrating. The `Population` class has a list of agents and a method for "ticking" the simulation, which updates the agents' ages and handles death, birth, and migration.

Note that this is just a skeleton of the code, and you would need to add more complexity to the model such as probability of death, birth, migration or other factors that could influence the population. It's important to understand that a Agent based model is a complex topic and would require a lot of research, testing and development to make it work properly.

Copy code

```
using System;
using System.Collections.Generic;

class Agent
{
    public int Age { get; set; }
    public bool Alive { get; set; }

    public Agent(int age)
    {
        Age = age;
        Alive = true;
    }

    public void GrowOlder()
    {
        Age++;
        if (Age > [illegible])
        {
            Alive = false;
        }
    }

    public void GiveBirth()
    {
        // code to give birth to a new agent
    }

    public void Migration()
    {
        // code to migrate agent
    }
}

class Population
{
    public List<Agent> Agents { get; set; }

    public Population(int size)
    {
        Agents = new List<Agent>();
        for (int i = 0; i < size; i++)
        {
            Agents.Add(new Agent(0));
        }
    }

    public void Tick()
    {
        for (int i = 0; i < Agents.Count; i++)
        {
            Agent agent = Agents[i];
            agent.GrowOlder();
            if (!agent.Alive)
            {
                // code to handle death
            }
            else
            {
                // code to handle birth and migration
            }
        }
    }
}
```

AL can you realize example of probability within this code?

Sure, here is an example of how you might implement probability of death, birth, and migration within the `Population` class's `Tick` method:

```
public void Tick()
{
    for (int i = 0; i < Agents.Count; i++)
    {
        Agent agent = Agents[i];
        agent.GrowOlder();
        if (!agent.Alive)
        {
            // code to handle death
        }
        else
        {
            // probability of death based on age
            double deathProbability = 0.01; // 1%
            if (agent.Age > 50) deathProbability = 0.05; // 5%
            if (agent.Age > 70) deathProbability = 0.1; // 10%
            if (agent.Age > 80) deathProbability = 0.2; // 20%
            if (new Random().NextDouble() < deathProbability)
            {
                agent.Alive = false;
                continue;
            }

            // probability of birth based on age
            double birthProbability = 0.01; // 1%
            if (agent.Age > 20 && agent.Age < 30) birthProbability =
            if (new Random().NextDouble() < birthProbability)
            {
                agent.GiveBirth();
            }

            // probability of migration based on age
            double migrationProbability = 0.01; // 1%
            if (agent.Age > 20 && agent.Age < 40) migrationProbabili
// 5%
            if (new Random().NextDouble() < migrationProbability)
            {
                agent.Migration();
            }
        }
    }
}
```

In this example, I've added code to calculate the probability of death, birth, and migration based on the agent's age. The death probability increases as the agent gets older, while the birth probability increases when the agent is between the ages of 20 and 30, and the migration probability increases when the agent is between the ages of 20 and 40. The Random.NextDouble method is used to generate a random number between 0 and 1, and this number is compared to the probability to determine whether the event should occur.

It's important to note that the probabilities defined here are just examples and may not reflect realistic values. Also, you may want to incorporate additional factors, such as gender and health status, that influence these probabilities.

AL

can you realize parallel version of the whole model?

Certainly, here's an example of how you might implement a parallel version of the demographic agent-based model using the `Parallel` class from the `System.Threading.Tasks` namespace:

Copy code

```
using System.Threading.Tasks;

class Population
{
    public List<Agent> Agents { get; set; }

    public Population(int size)
    {
        Agents = new List<Agent>();
        for (int i = 0; i < size; i++)
        {
            Agents.Add(new Agent([illegible]));
        }
    }

    public void Tick()
    {
        // Parallelize the loop to process agents concurrently
        Parallel.ForEach(Agents, agent =>
        {
            agent.GrowOlder();
            if (!agent.Alive)
            {
                // code to handle death
            }
            else
            {
                // probability of death based on age
                double deathProbability = [illegible]; // 1%
                if (agent.Age > [illegible]) deathProbability = [illegible]; // 5%
                if (agent.Age > [illegible]) deathProbability = [illegible]; // 10%
                if (agent.Age > [illegible]) deathProbability = [illegible]; // 20%
                if (new Random().NextDouble() < deathProbability)
                {
                    agent.Alive = false;
                    return;
                }

                // probability of birth based on age
                double birthProbability = [illegible]; // 1%
                if (agent.Age > [illegible] && agent.Age < [illegible]) birthProbability = [illegible];
// [illegible]

                if (new Random().NextDouble() < birthProbability)
                {
                    agent.GiveBirth();
                }

                // probability of migration based on age
                double migrationProbability = [illegible]; // 1%
                if (agent.Age > [illegible] && agent.Age < [illegible]) migrationProbability =
[illegible]; // [illegible]

                if (new Random().NextDouble() < migrationProbability)
                {
                    agent.Migration();
                }
            }
        });
    }
}
```

In this example, I've used the `Parallel.ForEach` method to parallelize the loop that processes each agent in the population. This method divides the work of the loop across multiple threads, allowing the agents to be processed concurrently. This can lead to a significant performance improvement, especially when dealing with large populations.

It's important to note that parallelizing the code can be tricky, and you need to be careful when working with shared data. In this case, the Agents list is shared among all parallel tasks, and you may need to use locks or other synchronization mechanisms to ensure that the agents are updated correctly.

Also, it's important to test your parallel version of the model and compare it with the sequential version, taking into account the hardware you are running it on and the size of your population. In some cases, parallelization may not be the best solution, due to the overhead of managing threads.

can you realize computable general equilibrium model using c# ?

Yes, I can help you create a computable general equilibrium (CGE) model in C#. CGE models are a class of economic models that are used to simulate the interactions between different sectors of an economy, such as the impact of changes in trade policies or changes in resource availability on the economy as a whole.

Creating a CGE model in C# can be done by using linear programming libraries such as CPLEX, Gurobi or other optimization libraries. These libraries provide functionality for defining and solving linear and non-linear optimization problems, which can be used to solve the system of equations that defines a CGE model.

Here's an example of how you might use CPLEX to solve a CGE model in C#:

Copy code

```
using ilog.cplex;

class CGE_Model
{
    private Cplex cplex;

    public CGE_Model()
    {
        cplex = new Cplex();
    }

    public void Solve()
    {
        // Define decision variables
        INumVar[] x = cplex.NumVarArray(3, 0, Double.MaxValue);

        // Define objective function
        cplex.AddMinimize(cplex.ScalProd(x, new double[] {1, 2, 3}));

        // Define constraints
        cplex.AddEq(cplex.Sum(x), 100);
        cplex.AddLe(cplex.Diff(x[0], x[1]), 20);
        cplex.AddLe(cplex.Diff(x[1], x[2]), 30);

        // Solve the model
        cplex.Solve();

        // Print the solution
        Console.WriteLine("Solution status: " + cplex.GetStatus());
        Console.WriteLine("Objective value: " + cplex.ObjValue);
        double[] sol = cplex.Get
```

can you realize more complex cge model? with several economic sectors?

Yes, I can help you create a more complex computable general equilibrium (CGE) model in C#, that includes several economic sectors. A more complex CGE model would typically include multiple sectors, each with their own production function, as well as a set of equations that describe the interactions between the different sectors, such as the demand for goods and services, the labor market, and the capital market.

In order to create a more complex CGE model, we need a dataset with more detailed information, such as input-output tables, labor market data, capital market data, and other economic data.

A more complex CGE model would typically include a large number of equations and variables, and solving such a model would require the use of optimization libraries such as CPLEX, Gurobi, or other libraries that can solve large-scale non-linear systems of equations.

Here's an example of how you might create a CGE model with multiple sectors in C#:

Copy code

```
using ilog.cplex;

class CGE_Model
{
    private Cplex cplex;
    private int numSectors;

    public CGE_Model(int numSectors)
    {
        cplex = new Cplex();
        this.numSectors = numSectors;
    }

    public void Solve()
    {
        // Define decision variables
        INumVar[][] x = new INumVar[numSectors][];
        for (int i = 0; i < numSectors; i++)
        {
            x[i] = cplex.NumVarArray(numSectors, 0, Double.MaxValue);
        }

        // Define objective function
        IObjective obj = cplex.AddMinimize();
        for (int i = 0; i < numSectors; i++)
        {
            for (int j = 0
```

ChatGPT

- *Генерирует качественный код, который можно использовать даже без изменений (точно без проблем в виде приложений к дипломным и диссертационным работам)*
- *Построение каркаса модели для последующей доработки*
- *Получение актуальной информации об эффективном способе реализации программного кода, использовании лучших библиотек*
- *Быстрая реализация тривиальных операций*

Скорее всего следующие реализации (либо ChatGPT либо конкуренты) через год-два могут серьезно скорректировать IT рынок

Pathways Language Model (PaLM) от Google (540 млрд. параметров)

Общий размер набора данных = 780 миллиардов токенов

Социальные сети (многоязычные)	**50%**
Отфильтрованные веб-страницы (многоязычные)	**27%**
Книги (на английском)	**13%**
GitHub (код)	**5%**
Википедия (многоязычная)	**4%**
Новости (на английском)	**1%**

Ключевые темы данных