Journal homepage: https://itce.com.ua/en

Information Technologies and Computer Engineering **Vol. 22, No. 1. 2025**

Article's History: Received: 10.01.2025 Revised: 20.03.2025 Accepted: 24.04.2025

UDC 004.04:519.876.5 DOI: 10.63341/vitce/1.2025.42

Artificial intelligence techniques for real-time visualisation of big data graph models

Andrii Banyk*

Postgraduate Student
Uzhhorod National University
88000, 3 Narodna Sq., Uzhhorod, Ukraine
https://orcid.org/0000-0002-8991-310X

Pavlo Mulesa

Doctor of Technical Sciences, Associate Professor
Uzhhorod National University
88000, 3 Narodna Sq., Uzhhorod, Ukraine
https://orcid.org/0000-0002-3437-8082

Abstract. The purpose of the study was to develop approaches to the use of artificial intelligence to improve the processes of interactive visualisation of graph structures of big data in real time, considering the optimisation of computing resources. During the study, graphs were constructed for analysing relationships in big data, and computational intelligence methods were used to optimise the processing and visualisation of graphs in an interactive format. The results of the study included the development of programmes for building graph structures in Python in the Visual Studio Code environment and their further visualisation in Unity using C# in Visual Studio. First, a visualisation of a random Erdos-Renyi-type graph was shown, which was then recreated in Unity 3D space. Using Python libraries, graph generation and interactive web visualisation were implemented. Machine learning methods were used to optimise the location of nodes in graphs, in particular, autoencoders and principal components to reduce dimension. A demonstration of the Barbashi-Albert model allowed seeing the clustering of nodes and their relationships in real time. In addition, interactive visualisation was demonstrated, where nodes were located in 2D space according to the results of the principal components analysis. The use of the Louvain algorithm helped to perform clustering and visualise the structure of communities. The results showed that the use of neural networks significantly improves the accuracy and efficiency of node placement in graphs, and reduces computational complexity. The results obtained can be useful for scientific research involving the analysis of large graph structures and requiring interactive data visualisation

Keywords: machine learning; interactive format; clustering and principal components; Python libraries; Unity capabilities

Introduction

Graph visualisation is an important tool for analysing complex relationships in big data (BD). Graph models are widely used in various fields, including social networks, bioinformatics, financial analysis, and cybersecurity, as they allow effectively displaying the structure and relationships between objects. The development of artificial intelligence (AI) and machine learning (ML) methods contributes to improving the processing of graph structures, in particular, by automatically detecting patterns and clustering nodes. The use of AI allows analysing complex network data, increasing the efficiency of their processing and presentation. In addition, interactive visualisation approaches make it easier to investigate graph structures and improve information perception. One of the main problems in visualising large-volume graphs is optimising the location of nodes, since conventional placement methods are often computationally expensive and not suitable for interactive analysis. Another challenge is efficient cluster detection in graph structures, since existing methods may not consider the specifics of large graphs, which makes them difficult to interpret.

V.I. Pankiv & O.L. Storozhuk (2024) studied the use of deep learning (DL) for real-time BD analysis, highlighting the effectiveness of neural networks for pattern

Suggested Citation:
Banyk, A., & Mulesa, P. (2025). Artificial intelligence techniques for real-time visualisation of big data graph models. *Information Technologies and Computer Engineering*, 22(1), 42-54. doi: 10.63341/vitce/1.2025.42

*Corresponding author

recognition and identifying significant insights. They paid special attention to convolutional, recurrent, and generative adversarial networks. Results of the study by I.M. Soroka *et al.* (2024) showed the impact of large AI models on various health sectors, in particular, bioinformatics, medical diagnostics, and medical imaging, and noted the significant potential of such models in working with BD. In addition, M. Miroshnyk *et al*. (2023), R. Mykolaichuk & A. Mykolaichuk (2024) investigated the use of large language models and autonomous agents to automate data processing, which addresses issues related to effective BD analysis.

On the other hand, S. Duan & Y. Zhao (2024) applied knowledge analysis based on visualisation technologies and used mathematical and statistical analysis techniques, including CiteSpace, to analyse large amounts of data. Similarly, J. Liu *et al.* (2019) applied knowledge graph analysis in CiteSpace to visualise AI trends in education. They identified areas such as smart education, BD in education, and AI integration into learning processes. Moreover, S. Yin *et al.* (2024) investigated the role of visualisation in BD mining and the problems of conventional visualisation methods. They also emphasised the importance of explicable AI in improving visual analysis. D. Çınar (2024) examined the role of BDS in the development of artificial general intelligence (AGI), in particular, their impact on pattern recognition, adaptive learning, and generalisation, and analysed the problems associated with their processing.

In turn, X. Haiyang *et al.* (2024) devoted their study to creating a knowledge graph model for BD solubility, which includes methods for extracting, integrating, and logically inferring knowledge. The proposed approach to constructing a knowledge graph is similar to methods for visualising BD graph structures in real time, in particular, in terms of relationship analysis, structure optimisation, and interactive data representation. D. Riva & C. Rossetti (2024) also considered knowledge graphs and modern visual analysis techniques, particularly in combination with graph embedding techniques. The main challenges identified were the development of intuitive interfaces, BD processing, and clarity of query languages. Additionally, research by J. Yang (2024) investigated financial data processing using AI and BD, focusing on predictive analytics and intelligent decision support techniques. Special attention is paid to algorithms for analysing complex relationships in large data sets, which coincides with approaches to visualisation and optimisation of graph structures in real time.

Previous studies have predominantly focused on specific applied domains (such as medicine, education, and bioinformatics) and have overlooked the particularities of interactive graph visualisation in business modelling, as well as the lack of solutions integrating AI with platforms like Unity for processing large-scale graph structures in real time. This research was aimed at developing ways to use AI for interactive visualisation of graph structures in BD, which was not considered in the mentioned articles. The objectives of the study were to construct graphs for link analysis in BD, to use computational intelligence to optimise graph processing and visualisation in an interactive format, and to analyse the possibilities of integrating AI algorithms in Unity to create graphical representations of complex data structures.

Materials and Methods

The main requirements for BD graph models were identified, including scalability, structure flexibility, computational efficiency, data and algorithm quality, and the ability to interactively visualise and integrate with other systems. Methods for constructing graphs, in particular, deterministic, dynamic, and random ones, including the Erdos-Renyi model, were analysed. In the Visual Studio (VS) Code environment, Python implemented the generation of graph structures using the networkx libraries, which were used to create a large random graph, and pyvis – for visualising it as an interactive web page. The code generated a graph with 1,000 vertices and a coupling probability of 0.01, and saved it to the large_graph.html file and output interactive visualisation.

To integrate the Erdos-Renyi graph into Unity, an approach was developed to transform graph structures into 3D models. To do this, the authors used the Unity environment, which displayed the graph, and Visual Studio, where C# code was written in the GraphGenerator.cs file. Two libraries were used in the development process: UnityEngine was the main Unity library that contained classes for working with objects, graphics components, and scenes, and System.Collections.Generic, which provided the use of collections to store nodes and edges of a graph. The programme set parameters for nodes and edges, in particular, the probability of their connection and the radius of their location. Lists for storing and creating graph elements were implemented, and the LineRenderer were configured to correctly display the connections between nodes. As a result, the graph was generated and visualised in Unity 3D environment. In addition, graph visualisations in Unity and the browser environment were compared, highlighting their advantages and limitations.

The next step was to optimise graph processing and visualisation using AI, focusing on ML methods. To do this, a programme was created in VS Code in Python that generated a graph based on the Barbashi-Albert model. The networkx and pyvis libraries were used for graph creation and visualisation, numpy was used for working with the adjacency matrix, and scikit-learn was used for data analysis. In particular, principal component analysis (PCA) was used to reduce the graph dimension before clustering, and the KMeans algorithm was used to group nodes by similarity features. The programme converted the graph to an adjacency matrix, clustered nodes, and assigned them colours according to the selected groups. Additionally, a physical model for the location of nodes in space was implemented. The resulting visualisation was saved in the ai_graph_visualisation.html file. Other programmes have also been written in VS Code in Python, including an implementation of the PCA method for visualising a graph in

the pca_graph.html file. The networkx, numpy, pyvis, and scikit-learn libraries were used for implementation. First, the Barbashi-Albert graph was created, after which its adjacency matrix was calculated.

Then PCA was applied to reduce the dimension to 2D, and the resulting graph was visualised. The optimised_graph.html file provided an example of using a neural network to optimise the location of nodes in space. In addition to numpy and pyvis, the following libraries were used: torch – for creating and training a network; torch.nn – for building an autoencoder that reduces and restores node coordinates; and torch.optim – for optimisation using the Adam algorithm. The coordinates of 500 nodes in 2D space and an autoencoder were set, which was used to improve their location. After training the network, optimised coordinates were obtained, which were visualised in the graph by adding nodes and edges.

The code for clustering the graph using the Louvain algorithm was also presented, the result of which was displayed in a separate matplotlib window. For this purpose, the networkx and community libraries were used to identify communities in the graph, and matplotlib.pyplot for visualising and displaying the cluster structure of a graph. The programme generated a random graph, applied the Louvain algorithm to identify communities, assigned colours according to clusters, and visualised the resulting structure.

Results

Plotting graphs for analysing relationships in big data and reproducing them in Unity Engine

Graph models play a key role in representing complex relationships between objects in BD. They provide an effective framework for analysing connectivity, identifying hidden patterns, and optimising information processing processes. To use graph structures in the context of BD, it is necessary to define the basic requirements for their construction and processing (Table 1).

Table 1. Basic requirements for BD graph models

Requirement	Description
Scalability	Ability to work with graphs containing a large number of vertices and edges
	Efficient memory usage and optimisation of computing resources
Structure flexibility	Support for different types of links
	Ability to dynamically update the graph without the need for its complete reconstruction
Interactive visualisation capability	Fast rendering and updating of graphical data representation
	Support for scaling and changing the level of detail in interactive analysis
Computational efficiency	Optimised algorithms for pathfinding, clustering, and centrality analysis
	Support for parallel computing for working with large graphs
Ability to integrate with other systems	Compatibility with databases and data processing platforms
	Using the application programming interface (API) to interact with other analytics and visualisation tools
Quality of data and algorithms	Methods for cleaning and normalising data before plotting graphs
	Using AI to optimise graph data structuring and analysis

Source: compiled by the authors

Methods for plotting graphs to represent complex relationships in BD varied depending on the nature of the data and the purpose of the analysis. One method is to use deterministic graphs, such as connectivity graphs or tree graphs, which are used to represent structures with defined and stable relationships between elements. Such graphs can be useful in modelling technical or logistics systems, where the relationships between elements usually do not change randomly. Another approach is to build dynamic graphs in which the relationships between vertices change over time. This method is useful for mapping the evolution of networks or systems where relationships change depending on certain events or conditions. Such graphs are often used in economic, environmental, or social research.

Another important part of plotting graphs is determining how to visualise them. Using interactive visualisation tools allow creating visual models that simplify the analysis of relationships in large amounts of data. Graph visualisation helps to see the structure and also identify important nodes, central elements, and hidden patterns in networks. Moreover, a random Erdos-Renyi graph was a classic example in which each pair of vertices was connected by an edge with a certain probability. This type of graph was well suited for modelling systems with random connections found in areas such as social networks, biological systems, or the Internet. An example of VS Code code that uses Python libraries to build a random graph of the Erdos-Renyi type and its interactive visualisation is shown (Fig. 1).

graph_visualization.py > ...

```
import networkx as nx
from pyvis.network import Network

G = nx.erdos_renyi_graph(n=1000, p=0.01)

net = Network(notebook=True)
net.from_nx(G)

net.force_atlas_2based()

net.show("large_graph.html")
```

Figure 1. Code for constructing a random Erdos-Renyi graph in VS Code

Source: compiled by the authors

In the file graph_visualisation.py code has been written to create and render a large graph using Python libraries. Networkx creates a random Erdos-Renyi graph, where each pair of 1,000 vertices connects with a certain probability, which is determined by the parameter p = 0.01. This allows creating a graph where the probability of coupling between any two vertices is 1%. Next, the pyvis library is used to visualise this graph, which creates an interactive graph view that makes it easier to analyse the data structure. The force_atlas_2based method uses an algorithm that organises the placement of vertices in space in such a way as to avoid overlapping them and ensure easy perception. Next, the result is saved in HyperText Markup Language (HTML) format to the large_graph.html file that can be opened in the browser for viewing (Fig. 2).

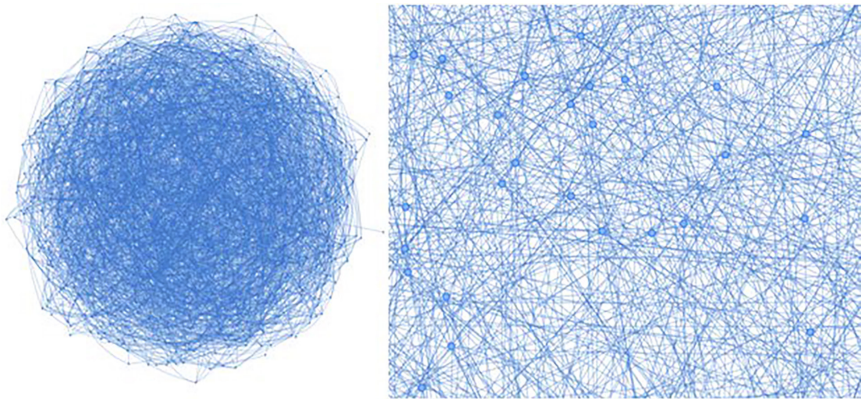

Figure 2. Graph of the Erdos-Renyi type in the large_graph.html file
Source: compiled by the authors

In other words, the code created an Erdos-Renyi graph, which is a classic example of a random graph. In such a graph, each pair of vertices is connected by an edge with a certain probability. This allows modelling structures where relationships between elements can be random, which is useful when analysing large and complex networks. In addition, visualisation is implemented in an interactive format that allows the user to scale the graph, move vertices, explore connections, and improve network structure analysis. This approach allows transfering the concept of random graphs to the Unity environment, using the capabilities of three-dimensional visualisation and interactive control. Unlike the Python implementation, where graphs are created and analysed as web interfaces, Unity creates a 3D model that can be rotated, scaled, and modified in real time (Fig. 3).

```
bly-CSharp                    GraphGenerator                    Start()
1   using UnityEngine;
2   using System.Collections.Generic;
3
    Скрипт Unity | Ссылок: 0
4   public class GraphGenerator : MonoBehaviour
5   {
6       public int nodeCount = 500;
7       public float connectionProbability = 0.01f;
8       public float graphRadius = 10f;
9
10      private List<GameObject> nodes = new List<GameObject>();
11      private List<LineRenderer> edges = new List<LineRenderer>();
12
        Сообщение Unity | Ссылок: 0
13      void Start()
14      {
15          GenerateGraph();
16      }
17
18
        Ссылок: 1
19      void GenerateGraph()
20      {
21
22          for (int i = 0; i < nodeCount; i++)
23          {
24              GameObject node = GameObject.CreatePrimitive(PrimitiveType.Sphere);
25              node.transform.position = new Vector3(Random.Range(-graphRadius, graphRadius),
26                                                    Random.Range(-graphRadius, graphRadius),
27                                                    Random.Range(-graphRadius, graphRadius));
28              node.transform.localScale = new Vector3(0.2f, 0.2f, 0.2f);
29              nodes.Add(node);
30          }
```

Figure 3. Code snippet for generating and visualising a random Erdos-Renyi graph in Unity (Visual Studio)
Source: compiled by the authors

The programme created a graph in the Unity environment, using SPHERE objects for nodes and LineRenderer for connections between them. At the beginning, the Start() method is executed, which calls the GenerateGraph() function. It starts with creating nodes: each node is a sphere that is located within a certain radius at random coordinates in three-dimensional space. All nodes are added to the nodes list. After creating nodes, the programme passes through each pair of nodes and, with a certain probability, creates an edge between them. To do this, a LineRenderer object connecting two points in space is created. All edges are stored in the edges list so that they can be changed or deleted if necessary. After running the script in Unity, 500 spheres appear in the scene, which are randomly located in space and connected by white lines that form a graph. The probability of connecting nodes is set to 0.01, so each node has a small number of connections, which brings the graph closer to the Erdos-Renyi model. The result visualises a random graph in three-dimensional space, which can be used for further analysis, modelling, or interactive interaction in Unity (Fig. 4).

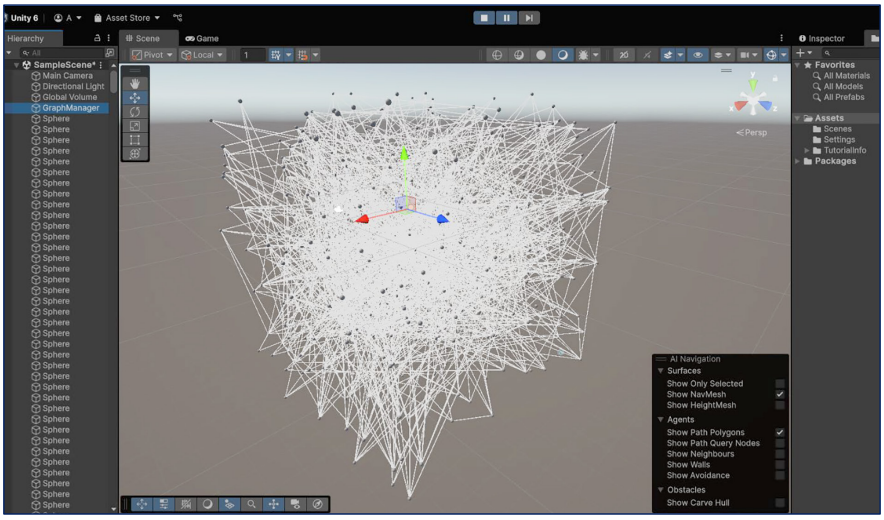

Figure 4. Code snippet for generating and visualising a random Erdos-Renyi graph in Unity (Visual Studio)
Source: compiled by the authors

In general, real-time visualisation of BD graph structures is an important task for understanding complex relationships between objects and analysing their dynamics. Unity Engine provides a powerful toolkit for interactive representation of graphs in three-dimensional space, which allows viewing the model statically, but also actively interacting with it, changing angles, scales, and exploring individual nodes and connections. Unlike conventional graph visualisation methods, which are presented as static images, Unity allows implementing a full-fledged three-dimensional scene with the possibility of interactive interaction. However, the use of browser technologies, in particular, pyvis in Python, provides a convenient way to display large graphs with dynamic updates and the ability to integrate AI algorithms. A comparison of these approaches is provided for better understanding (Table 2).

Table 2. Comparison of graph rendering in Unity and the browser environment

Parameter	Unity (3D visualisation)	Browser environment (pyvis in Python)
Interactivity	High: ability to rotate, zoom, and select nodes in 3D space.	High: supports zooming, node selection, and dynamic animations in 2D.
Visual complexity	Allows full-fledged 3D visualisation, complex effects, and lighting.	Supports interactive graphs, but in 2D, which sometimes makes it difficult to understand.
Performance	High for optimisation, but requires a powerful GPU (Graphical Processing Unit).	Efficient operation in the browser, especially for large graphs.
Scalability	Limited by the number of objects in the scene and GPU performance.	Works well with large graphs, especially with optimised algorithms.
Easy to implement	Requires the use of C# and working with 3D objects via the Unity API, which complicates development.	Easily implemented through libraries such as networkx and pyvis.
Ability to update the graph	Supports dynamic structure changes, but requires additional code.	Easily updated by refreshing the page or calling Python scripts.
Integration with other systems	Limited: it is more difficult to connect APIs and external databases.	Easily integrates with databases, ML, and AI algorithms.
Availability	Unity requires installation, is more difficult to distribute, but can be built for different platforms.	Can be saved and shared as HTML files.
Scope of application	Well suited for game applications and simulations.	Optimal for analytics, research, and working with BD.

Source: compiled by the authors

Unity is therefore an effective solution for creating 3D visualisations, but for working with large graph structures in real time, it is more appropriate to use browser technologies that provide convenience, scalability, and integration with AI techniques. In addition, browser-based tools such as pyvis make it easy to interact with graphs without the need to install additional software, making it easier for a wide range of users to access the visualisation. Unity provides more opportunities for detailed processing of 3D graphs, including complex animation effects and physical interactions between nodes, and Assembly for different platforms is possible. Therefore, the choice of tool depends on the specific requirements for interactivity, performance, and complexity of processing the graph structure.

Optimisation of graph processing and visualisation using computational intelligence

Visualising large graph structures is a complex task, as it requires simultaneous consideration of scalability, interactivity, and efficient use of computing resources. The use of AI techniques allows optimising this process, reducing computational costs, and improving the quality of representation of graph models. A programme that creates a graph using the Barbashi-Albert model is often used to simulate real networks, such as social networks or the Internet (Fig. 5). It generates a graph with a thousand nodes, where each new node is connected to the existing ones according to a certain scheme. In order to process this graph and perform analysis, the programme converts it to an adjacency matrix, which allows working with it as numeric data.

```python
import networkx as nx
from pyvis.network import Network
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

G = nx.barabasi_albert_graph(n=1000, m=5)

adj_matrix = nx.to_numpy_array(G)

pca = PCA(n_components=16)
embeddings = pca.fit_transform(adj_matrix)

num_clusters = 5
kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
labels = kmeans.fit_predict(embeddings)

net = Network(notebook=True, height="750px", width="100%", bgcolor="white", font_color="black")

colors = ["red", "blue", "green", "yellow", "purple"]
for i, node in enumerate(G.nodes()):
    net.add_node(node, label=str(node), color=colors[labels[i] % len(colors)])

for edge in G.edges():
    net.add_edge(edge[0], edge[1])

net.force_atlas_2based()

net.show("ai_graph_visualization.html")
```

Figure 5. Code for constructing the Barbashi-Albert graph in VS Code
Source: compiled by the authors

Using ML methods, in particular PCA, the programme reduces the dimension of graph data. This allows simplifying the graph structure for further analysis. Then, using the K-Means algorithm, the graph nodes are grouped into several groups, which allows distinguishing similar nodes by their connections and interactions. The result is a visual graph visualisation that allows seeing the relationships between nodes and their clustering in real time (Fig. 6).

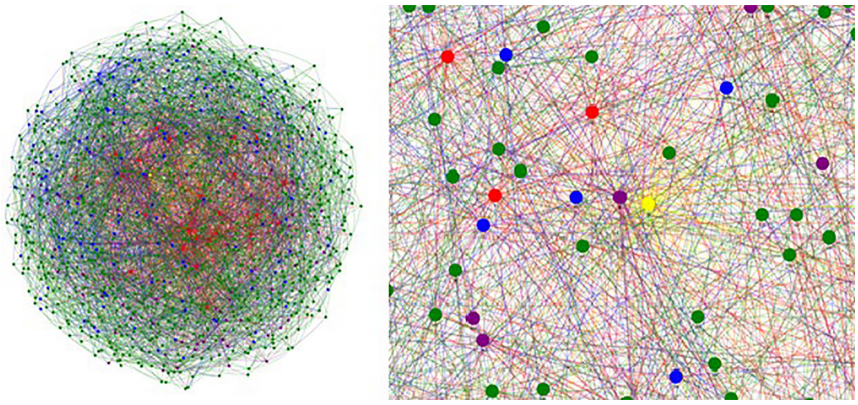

Figure 6. Visualisation of the Barbashi-Albert graph using ML methods
Source: compiled by the authors

The programme uses the ForceAtlas2 physical algorithm to organise nodes in graph space based on physical forces, which makes visualisation more understandable. The result is saved as an HTML file, and the visualisation is interactive, allowing the user to interact with the graph. An important aspect is that this application is based on ML methods, in particular, PCA and K-Means Clustering. It assigns a separate colour to each cluster, which helps to clearly distinguish groups of nodes.

Another AI method for visualising graph models should indicate dimensionality reduction. It allows representing complex graph structures in fewer dimensions, which significantly reduces the computational cost of visualisation. For example, the t-Distributed Stochastic Neighbour Embedding (t-SNE) method allows grouping nodes in social networks by community, and Uniform Manifest Approximation and Projection (UMAP) is used to analyse biological networks. Laplacian Eigenmaps, which work effectively with geospatial data, or DeepWalk and Node2Vec, node vectorisation methods used in recommendation systems, are also used for graph structures. An example of a PCA method for graph visualisation is shown in Figure 7.

```python
import networkx as nx
import numpy as np
from sklearn.decomposition import PCA
from pyvis.network import Network

G = nx.barabasi_albert_graph(n=500, m=3)

adj_matrix = nx.to_numpy_array(G)

pca = PCA(n_components=2)
embeddings = pca.fit_transform(adj_matrix)

net = Network(height="750px", width="100%", notebook=True)
for i, (x, y) in enumerate(embeddings):
    net.add_node(i, x=x, y=y, label=str(i))

for edge in G.edges():
    net.add_edge(edge[0], edge[1])

net.show("pca_graph.html")
```

Figure 7. Code for plotting a graph using the PCA method in VS Code
Source: compiled by the authors

This application creates a scalable network based on the Barbashi-Albert model, which displays real networks with nodes with different numbers of connections (the "rich get richer" principle). A graph is generated with 500 nodes, where each new node joins 3 existing ones. Next, an adjacency matrix is obtained from the graph, which displays the connections between nodes in the form of a numerical matrix. Since such a matrix has a high dimension, the PCA method is used to reduce the dimension to two dimensions, which allows representing a graph in 2D space. Using the pyvis library, an interactive graph visualisation is created. Nodes get coordinates according to the calculated PCA values, and the relationships between them are added according to the original graph structure. After executing the programme, an HTML file is generated, which can be opened in a browser to view the interactive network (Fig. 8).

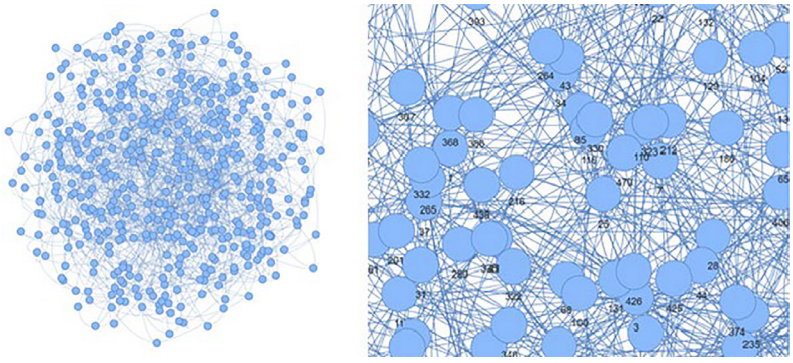

Figure 8. Graph visualisation for the data dimensionality reduction method
Source: compiled by the authors

As a result of the programme execution, an interactive visualisation of the graph is obtained, where nodes are located in two-dimensional space in accordance with the principal components defined by the PCA method. This allows simplifying the display of a complex network structure, while preserving the basic relationships between

nodes. Interacting with the graph in the browser allows exploring its structure, analysing clusters, and identifying the key nodes that have the most connections.

The next method is to optimise the location of nodes in space. Physical force algorithms such as ForceAtlas2 or Fruchterman-Reingold simulate physical interactions between vertices, providing a natural graph layout. However, for large graphs, these methods can be computationally expensive. To speed up calculations, used Barnes-Hut Simulation, which reduces the complexity of calculations for nodes. Additionally, ML techniques are used, such as graph Neural Networks (GNNs), which allow training optimal node representations, or Autoencoders, which help to reduce the dimension of space and improve the location of the graph. An example is the use of a neural network to optimise the location of nodes in space (Fig. 9).

```python
embeddings = np.random.rand(500, 2)

embeddings = torch.tensor(embeddings, dtype=torch.float32, requires_grad=True)

class GraphAutoencoder(nn.Module):
    def __init__(self):
        super(GraphAutoencoder, self).__init__()
        self.encoder = nn.Linear(2, 16)
        self.decoder = nn.Linear(16, 2)

    def forward(self, x):
        x = torch.relu(self.encoder(x))
        x = self.decoder(x)
        return x

model = GraphAutoencoder()

optimizer = optim.Adam(model.parameters(), lr=0.01)
loss_function = nn.MSELoss()

for epoch in range(200):
    optimizer.zero_grad()
    output = model(embeddings)
    loss = loss_function(output, embeddings)
    loss.backward()
    optimizer.step()

optimized_embeddings = output.detach().numpy()
```

Figure 9. Code snippet for building an autoencoder graph in VS Code
Source: compiled by the authors

The purpose of this code is to improve the placement of graph nodes on the 2D plane by training an autoencoder that corrects their coordinates. This can be useful if there is a need for a convenient visual representation of large graphs, where it is important that nodes do not overlap and are distributed clearly and efficiently. The programme starts by generating random coordinates for 500 nodes representing a graph in 2D space. These coordinates are converted to tensors for further work with them in PyTorch. Then a neural network is created – an autoencoder, which consists of two layers. First layer reduces the dimension of coordinates from 2 to 16, and the second one restores them to 2D space. During training, the network tries to minimise the difference between the original coordinates and the ones it generates, thereby improving their location.

Training lasts 200 stages, at each of which the network updates its parameters so that the coordinates of nodes become more optimised for better visualisation. As a result, optimised node coordinates are obtained, which can be used to plot the graph. After that, the programme creates an interactive visualisation of the graph using the pyvis library, where nodes are placed according to optimised coordinates, and relationships between them are added to create a regular graph structure (Fig. 10).

Figure 10. Example of a graph for a method for optimising the location of nodes
Source: compiled by the authors

All nodes are connected by edges, which shows the relationships between graph elements. The resulting file can be viewed in a browser, which helps to visualise and interact with the graph in real time. This approach can significantly

improve the visual appeal and interpretability of complex graphs, especially when working with large data sets, where the clarity and clarity of Node locations is important.

For its part, the method of clustering graph structures for improved visualisation is important. It consists in splitting the graph into separate groups (clustering), which allows analysing and visualising it more efficiently. The main algorithms include: K-Means – a simple method for dividing vertices into clusters; Louvain – for finding communities in graphs; Spectral Clustering – application of linear algebra methods for clustering. An example of using Louvain for graph clustering is shown in Figure 11.

```python
import networkx as nx
import community
import matplotlib.pyplot as plt

G = nx.erdos_renyi_graph(500, 0.05)

partition = community.best_partition(G)

colors = [partition[node] for node in G.nodes()]

plt.figure(figsize=(10, 10))
nx.draw(G, pos=nx.spring_layout(G, seed=42), node_color=colors, cmap=plt.cm.Set1, with_labels=False, node_size=50)
plt.title("Кластеризація графа методом Louvain")
plt.show()
```

Figure 11. Clustering a graph using the Louvain method in VS Code
Source: compiled by the authors

This code demonstrates graph clustering using the Louvain algorithm, which is used to find communities in graph structures. The main idea of the method is to identify subgroups of nodes that have strong internal connections, but relatively weak connections to other parts of the graph. This helps to better visualise and analyse complex networks. First, a random graph is created using the Erdos-Renyi model, where each of the 500 nodes has a probability of 0.05 being connected to other nodes. The Louvain algorithm then splits the graph into clusters, returning a dictionary where each node corresponds to a specific cluster. Based on this cluster breakdown, each node is assigned a colour according to its group. The visualisation uses the location of nodes determined by the spring_layout algorithm, which simulates the repulsive and attractive forces between vertices, creating a more understandable and ordered image. The graph is drawn with coloured cluster designations, which clearly shows how nodes are grouped together. The result is an image of a clustered network, where each colour represents a separate community (Fig. 12).

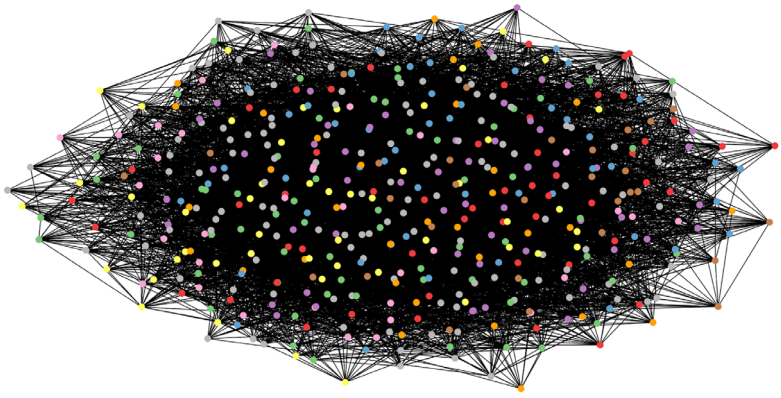

Figure 12. Visualisation of clustering of graph structures
Source: compiled by the authors

In addition to the approaches considered, several methods for speeding up the rendering of large graphs that allow efficient visualisation of complex structures using the GPU and parallel computing can be distinguished. For example, the Rapids cuGraph or Deep Graph Library (DGL) libraries, migrate calculations to TensorFlow/PyTorch, and the Web Graphics Library (WebGL), such as three.js or deck.gl. There is also an approach to predicting changes in dynamic graphs, where Graph Convolutional Networks (GNNS) are used to predict the appearance of new connections or changes in edge weights. This method includes GCN, Graph Attention Networks (GAT), and Temporal Graph Networks (TGNs).

As a result of applying computational intelligence techniques to graph model visualisation, a significant improvement has been achieved in the efficiency of processing large graphs in real time. The use of algorithms such as PCA and K-Means Clustering helped to reduce the dimension of data and classify nodes, which helped to analyse and visualise them. In addition, optimising the location

of nodes using physical algorithms and neural networks helped to achieve better graph structure, which increased the interpretability of complex networks. Louvain clustering revealed communities in graphs, which improved visualisation and analysis of relationships between network elements. Overall, the integration of AI into graph visualisation processes has made them more dynamic, understandable, and efficient for BD analysis.

Discussion

The conclusions of this study are consistent with the findings of G. Gheorghe & P. Lorenz (2023), as they point to the use of AI to work with BD. However, if the researchers was focused on intelligent classification methods, high-performance calculations, and mathematical modelling of BD, then the current study focuses on interactive visualisation of BD graph structures in real time and optimisation of their display using ML algorithms. In particular, it used clustering methods, dimensionality reduction, and optimisation of the location of nodes in space using autoencoders, which allows increasing the efficiency of visual representation of complex graph models. The results of this study also complemented the findings of A.H. Gandomi *et al.* (2023), because they used AI techniques to analyse BD. However, while this article focused on using ML, DL, and natural language processing (NLP) algorithms to identify patterns and make decisions, the current study focuses on AI methods for plotting BD graphs in real time. For this purpose, classification and clustering are applied, including integration of neural networks, which allow automatically adjusting the location of nodes to improve the convenience and accuracy of data representation in three-dimensional space.

The results obtained confirm the conclusions of L. Di & E. Yu (2023), because they emphasise the importance of visualisation when working with BD. While the researchers focused on data storage and processing using structured query language (SQL) and NoSQL systems, and graph processing, the current study focused on AI techniques to improve node placement in graphs. This reduces computational costs and improves the accuracy of visualising complex graph models. In addition, in contrast to the mentioned study, the research used ML algorithms to improve data structuring, which contributed to a more accurate representation of the relationships between BD elements. Overall, this study focused on visualising graph structures, which partially confirms the findings of S. Panyaram (2024), who also used AI techniques for dynamic data, but with a focus on finance and healthcare. That is, the current study focused on improving the visual representation of graphs using neural networks for more accurate and convenient location of nodes, which is the main difference from the mentioned study.

In this article, PCA was used to reduce the graph dimension before clustering, which simplified analysis and improved visualisation. The Barbashi-Albert model, the KMeans algorithm for grouping nodes, and an autoencoder for optimising their location were used. A. Noshi & S. Gasmi (2024) also applied PCA, but in the context of improving the structure of knowledge maps, focusing on NLP methods and semantic analysis. In contrast to their research, this study was aimed at Interactive optimisation of graphs in real time using AI, which allows not only to improve the structure of graph models, but also to automatically adjust the location of nodes, reducing computational costs and improving the accuracy of visualisation. In addition to generating and interactively optimising Barbashi-Albert graphs using ML, Erdos-Renyi graphs were also implemented in the study. Clustering, dimensionality reduction, and autoencoders were used to improve visualisation, which increased the accuracy and efficiency of node placement. In turn, L. Bellmann *et al.* (2024) presented an attribute associative graph for analysing medical data, which is based on statistical metrics and does not require programming. Their approach focuses on fixed determination of relationships between sample attributes, while in the current study, graph structures are dynamically adapted to data using AI techniques, which provides more flexible and accurate visualisation of complex relationships.

Similar to the study by J. Zhao *et al.* (2024), which improved dimensionality reduction algorithms (t-SNE and UMAP) by combining their aspects for data analysis, the current study also applied dimensionality reduction, but with the aim of optimising the visualisation of graph models. In the mentioned study, t-SNE and UMAP were used to cluster nodes in social and biological networks, while the current study also analyses other dimensionality reduction techniques, in particular PCA, which allow for more efficient visualisation of complex graph structures while reducing computational costs. It should also be noted that this study focused on optimising the visualisation of BD graphs using AI techniques that can reduce computational costs and improve the efficiency of visualising complex graph structures. S. Janicijevic & V. Nikolic (2021) focused on interaction metrics such as degree centrality, approximation, internode centrality, and PageRank to evaluate graph visualisation, and the evolution of BD visualisation techniques. Although these approaches require special attention to the specific properties of graph structures, the methods used in the current study provide more efficient management of computational resources for visualising large graphs.

The results demonstrate improvements in the efficiency of visualisation of large graph structures and optimisation of their processing using AI methods and ML algorithms, in particular, in the context of clustering using the Louvain algorithm and using an autoencoder for node placement. They are consistent with the findings of M. Yang (2024), where the integration of AI and BD analysis technologies for cybersecurity provides similar stability and efficiency benefits. That is, both approaches confirm the importance of AI in improving data processing and security. On the other hand, D. Zion & B. Tripathy (2020) focused on the importance of using specialised tools for visualising large graphs, highlighting the limitations of conventional systems when working with dynamic and large data sets.

Thus, in the current study, specific AI methods are used to optimise the visualisation of graphs in real time, which allows dynamically adjusting the location of nodes, and the above research is more focused on general visualisation tools, without focusing on interactive optimisation and the use of ML to improve the display of graph structures.

The current results are consistent with the approaches described by M. Devi & S.R. Kasireddy (2019), because the use of graph models in the context of BD analysis is an important aspect for achieving business goals. As in their study, the current study shows that the use of graph tools can effectively identify hidden relationships between data, in particular in social networks, which is useful for improving marketing and sales strategies. The results of this study also indicate the efficiency of clustering and dimensionality reduction. Comparing the study by T. Kliestik *et al.* (2024), similarities in the use of technologies to improve the analysis of large amounts of data can be noted. However, while the current study focuses on graph model visualisation, the aforementioned research focuses on the use of digital doubles and forecasting in an industrial environment. Although both studies use BD to optimise processes, the results of the current study confirm the potential of real-time interactive visualisation, which can be applied to improve link analysis in BD and support decision-making in various areas.

It is worth noting that the current study developed programmes for plotting graphs in Python using ML algorithms, and for visualising them in real time in Unity using C#. On the other hand, M.A. Ruiz Estrada (2023) examines the use of AI and multidimensional graphs for modelling socio-economic processes, where 3D visualisation models are also used. Both approaches integrate AI to improve BD processing, but while the current study focuses on real-time graph visualisation, the researcher focuses on using graphs for policy modelling and crisis analysis. Therefore, the results of both studies are consistent, AI is used for BD analysis but with a focus on policy modelling. In addition, the current study focuses on interactive visualisation of large graph structures, in particular, through the integration of AI techniques to optimise node placement and reduce computational costs. Compared to the study by F. Gebretsadik & R. Patgiri (2023), which examines common problems of large graphs such as visualisation, data distribution, and knowledge integration, the current study focuses on optimising graph processing through computational intelligence techniques and their implementation in various environments, such as Unity Engine. This approach provides improved interactivity and visualisation efficiency through the use of neural networks to optimise graphs.

Similar to the research by S.K. Devineni (2024), which examines interactive data visualisation using AI, in particular ML and virtual/augmented reality (AR/VR) applications, the current study focuses on interactive graph visualisation using AI, but in the context of optimising the processing of large graphs in real time. This study uses clustering techniques, dimensionality reduction, and neural networks to arrange nodes, which allows for greater efficiency in graph processing. While this article focuses on multidimensional data and the use of AR/VR, the current results focus more on the technical implementation of graph structure visualisation and optimisation for interactive BD work. Thus, the results of this study make an important contribution to understanding the application of AI techniques to improve the visualisation of BD graph structures and can be useful for further developments in the field of BD analysis and the creation of effective interfaces for processing and visualising such data.

Conclusions

During the study, significant results were achieved in creating and visualising graph structures. The ability to generate random Erdos-Renyi graphs and use them to analyse relationships between data has been demonstrated. For this purpose, an approach to graph construction in Python in VS Code has been developed, which allows generating graphs for analysing relationships in BD, and their 3D visualisation in Unity has also been implemented. The results confirmed the effectiveness of using Unity and Python for visualising large graph structures, improving the quality of interfaces for data analysis.

The use of various AI techniques, including ML algorithms, has improved the efficiency and accuracy of graph processing in research. The constructed Barbashi-Albert graph demonstrated the clustering of nodes and their relationships in real time. To reduce the dimension of graphs, PCA and autoencoder methods were used, which reduced the complexity of visualising high-dimensional graphs. Further clustering of nodes using the Louvain algorithm helped to identify communities in graphs, which significantly improved their interpretation and helped to achieve better visualisation results. Moreover, the use of neural networks has helped to optimise the location of nodes, reducing computational complexity and improving real-time accuracy. This contributed to the high interactivity required for real-time visualisation. Based on the use of various Python libraries, such as networkx, pyvis, torch, and scikit-learn, powerful tools for graph analysis and visualisation were created, which provided significant improvements in working with BD. During the study, graphs were generated using random models, such as the Erdos-Renyi and Barbashi-Albert models, which limited the ability to test methods on real data sets.

Further research should focus on optimising and improving parallel processing methods for scaling graph models. In addition, it is advisable to consider using more complex neural networks to improve the accuracy of node placement and testing on complex graphs. It is also worth studying the integration of other AI methods to improve clustering and predict the dynamics of graph structures in real time, which will significantly improve the practical application of the results obtained.

Acknowledgements

None.

Funding

The study received no funding.

Conflict of Interest

None.

References

[1] Bellmann, L., Wiederhold, A.J., Trübe, L., Twerenbold, R., Ückert, F., & Gottfried, K. (2024). Introducing attribute association graphs to facilitate medical data exploration: Development and evaluation using epidemiological study data. *JMIR Medical Informatics,* 12, article number e49865. doi: 10.2196/49865.

[2] Çınar, D. (2024). The role of artificial intelligence and big data analytics in business management: A review of decision – making and strategic planning. *Journal of Tourism Economics and Business Research,* 6(2), 219-229.

[3] Devi, M., & Kasireddy, S.R. (2019). Graph analysis and visualization of social network big data. In *Social network forensics, cyber security, and machine learning. springerbriefs in applied sciences and technology* (pp. 93-104). Singapore: Springer. doi: 10.1007/978-981-13-1456-8_8.

[4] Devineni, S.K. (2024). AI-enhanced data visualization: Transforming complex data into actionable insights. *Journal of Technology and Systems,* 6(3), 52-77. doi: 10.47941/jts.1911.

[5] Di, L., & Yu, E. (2023). Big data analytic platforms. In *Remote sensing big data* (pp. 171-194). Cham: Springer. doi: 10.1007/978-3-031-33932-5_10.

[6] Duan, S., & Zhao, Y. (2024). Knowledge graph analysis for chronic diseases nursing based on visualization technology and literature big data. *Scalable Computing Practice and Experience*, 25(3), 1728-1747. doi: 10.12694/scpe.v25i3.2664.

[7] Gandomi, A.H., Chen, F., & Abualigah, L. (2023). Big data analytics using artificial intelligence. *Electronics,* 12(4), article number 957. doi: 10.3390/electronics12040957.

[8] Gebretsadik, F., & Patgiri, R. (2023). The major challenges of big graph and their solutions: A review. *Advances in Computers,* 128, 399-421. doi: 10.1016/bs.adcom.2021.10.010.

[9] Gheorghe, G., & Lorenz, P. (Eds.). (2023). *Proceedings of the 3rd international conference on artificial intelligence, big data and algorithms. Advances in Artificial intelligence, big data and algorithms*. Amsterdam: IOS Press BV.

[10] Haiyang, X., Ruomei, Y., Yan, W., Lixin, G., & Li, M. (2024). Knowledge graph for solubility big data: Construction and applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* 15(1), article number e1570. doi: 10.1002/widm.1570.

[11] Janicijevic, S., & Nikolic, V. (2021). Graph structures for data visualizations. *Serbian Journal of Engineering Management*, 6(2), 24-31.

[12] Kliestik, T., Kral, P., Bugaj, M., & Ďurana, P. (2024). Generative artificial intelligence of things systems, multisensory immersive extended reality technologies, and algorithmic big data simulation and modelling tools in digital twin industrial metaverse. *Equilibrium. Quarterly Journal of Economics and Economic Policy,* 19(2), 429-461. doi: 10.24136/eq.3108.

[13] Liu, J., Xie, R., & Song, A. (2019). Analysis on research frontiers and hotspots of "Artificial intelligence plus education" in China – visualization research based on citespace V. *IOP Conference Series Materials Science and Engineering,* 569, article number 052073. doi: 10.1088/1757-899X/569/5/052073.

[14] Miroshnyk, M., Shkil, O., Rakhlis, D., Pshenychnyi, K., & Miroshnyk, A. (2023). Event processing model for simulation of real-time logic control devices. *Bulletin of Cherkasy State Technological University*, 28(2), 50-57. https://doi org/10.24025/2306-4412.2.2023.274840.

[15] Mykolaichuk, R., & Mykolaichuk, A. (2024). Using artificial intelligence technologies for document processing automation. *Modern Information Technologies in the Sphere of Security and Defence,* 50(2), 111-117. doi: 10.33099/2311-7249/2024-50-2-111-117.

[16] Noshi, A., & Gasmi, S. (2024). Building efficient knowledge maps through NLP and optimization in big data environments. doi: 10.13140/RG.2.2.11353.53609.

[17] Pankiv, V.I., & Storozhuk, O.L. (2024). The use of neural measurements and advanced techniques for the analysis of large quantities of data in real time. In *Forestry education and science: Current challenges and development prospects, international science-practical conference*. Lviv: National Polytechnic University of Ukraine. doi: 10.36930/conf150.5.19.

[18] Panyaram, S. (2024). Integrating artificial intelligence with big data for real-time insights and decision-making in complex systems. *FMDB Transactions on Sustainable Intelligent Networks,* 1(2), 85-95. doi: 10.69888/FTSIN.2024.000211.

[19] Riva, D., & Rossetti, C. (2024). Visualization of knowledge graphs with embeddings: An essay on recent trends and methods. *ArXiv*. doi: 10.48550/arXiv.2412.05289.

[20] Ruiz Estrada, M.A. (2023). *New artificial intelligence (AI) models for policy modelling*. Kuala Lumpur: Econographication Laboratory. doi: 10.13140/RG.2.2.30920.90887.

[21] Soroka, I.M., Mochalov, IO., & Kizim, A.V. (2024). The modern directions of implementation of the large models of artificial intelligence in health care. *Intermedical Journal,* 2, 174-180. doi: 10.32782/2786-7684/2024-2-30.

[22] Yang, J. (2024). Application of artificial intelligence and big data in financial management. *SHS Web of Conferences,* 208, article number 01006. doi: 10.1051/shsconf/202420801006.

[23] Yang, M. (2024). Design of a cybersecurity defense system based on big data and artificial intelligence. *Applied and Computational Engineering,* 87, 98-103. doi: 10.54254/2755-2721/87/20241443.
[24] Yin, S., Li, H., Sun, Y., Ibrar, M., & Teng, L. (2024). Data visualization analysis based on explainable artificial intelligence: A survey. *IJLAI Transactions on Science and Engineering,* 2(2), 13-20.
[25] Zhao, J., Pierre, J., & Konstantinidis, K.T. (2024). Approximate nearest neighbor graph provides fast and efficient embedding with applications for large-scale biological data. *NAR Genomics and Bioinformatics*, 6(4), article number lqae172. doi: 10.1093/nargab/lqae172.
[26] Zion, D., & Tripathy, B. (2020). Comparative analysis of tools for big data visualization and challenges. In S.M. Anouncia, H.A. Gohel & S. Vairamuthu (Eds.), *Data visualization* (pp. 33-52). Singapore: Springer. doi: 10.1007/978-981-15-2282-6_3.

Методи штучного інтелекту для візуалізації графових моделей великих даних у реальному часі

Андрій Баник
Аспірант
Ужгородський національний університет
88000, пл. Народна, 3, м. Ужгород, Україна
https://orcid.org/0000-0002-8991-310X
Павло Мулеса
Доктор технічних наук, доцент
Ужгородський національний університет
88000, пл. Народна, 3, м. Ужгород, Україна
https://orcid.org/0000-0002-3437-8082

Анотація. Метою дослідження була розробка підходів до використання штучного інтелекту для покращення процесів інтерактивної візуалізації графових структур великих даних у реальному часі з урахуванням оптимізації обчислювальних ресурсів. Під час дослідження були побудовані графи для аналізу зв'язків у великих даних, а також використані методи обчислювального інтелекту для оптимізації обробки та візуалізації графів в інтерактивному форматі. Результати дослідження включали розробку програм для побудови графових структур на Python у середовищі Visual Studio Code та їх подальшу візуалізацію в Unity з використанням C# у Visual Studio. Спочатку було показано візуалізацію випадкового графа типу Ердеша-Реньї, який потім було відтворено у 3D-просторі Unity. За допомогою бібліотек Python було реалізовано генерацію графів та інтерактивну веб-візуалізацію. Для оптимізації розташування вузлів у графах були використані методи машинного навчання, зокрема, автоенкодери та головні компоненти для зменшення розмірності. Демонстрація моделі Барбаші-Альберта дозволила побачити кластеризацію вузлів та їх зв'язки в реальному часі. Крім того, було продемонстровано інтерактивну візуалізацію, де вузли розташовувалися у 2D-просторі відповідно до результатів аналізу головних компонент. Використання алгоритму Лувена допомогло виконати кластеризацію та візуалізувати структуру спільнот. Результати показали, що використання нейронних мереж значно покращує точність та ефективність розміщення вузлів у графах, а також зменшує обчислювальну складність. Отримані результати можуть бути корисними для наукових досліджень, що пов'язані з аналізом великих графових структур та потребують інтерактивної візуалізації даних

Ключові слова: машинне навчання; інтерактивний формат; кластеризація та головні компоненти; бібліотеки Python; можливості Unity