

VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



INTERNSHIP 1

Machine learning for internet of thing: Person mask detector model in edge device

Lecturer: PhD. TBD

Students: 2270375 - Nguyễn Trung Thuận

Ho Chi Minh city - November 2023

Contents

List of Figures

I. Introduction

1. Machine Learning Introduction

Machine learning is an innovative application of artificial intelligence (AI) that empowers systems to learn and improve from experience autonomously without explicit programming.

The learning process begins by observing or obtaining data, such as examples, direct experiences, or instructions, to identify patterns within the data and make informed decisions based on the provided examples. The ultimate goal is to enable computers to learn automatically, free from human intervention, and adapt their actions accordingly.

1.1 Machine Learning algorithm

- **Supervised Machine Learning:** This approach involves leveraging labeled examples from past data to predict the outcome of future events [2]. By analyzing a known training dataset to extract patterns and learn from it, the learning algorithm creates a model to generate predictions about output values. With sufficient training, the system can provide accurate predictions for new inputs. It can also compare its outputs with the correct ones to identify errors and adjust the model accordingly.

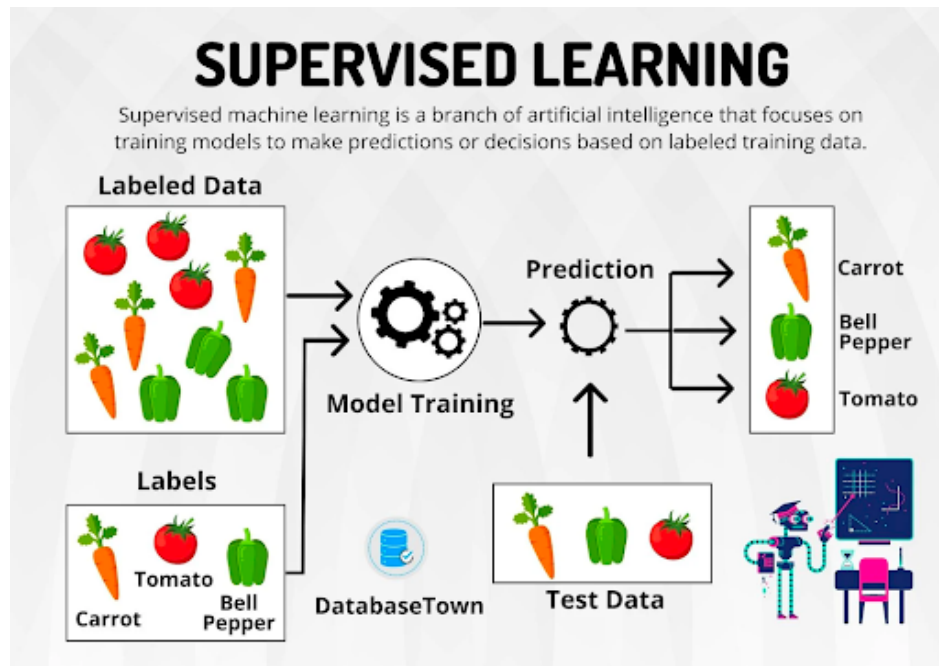


Figure 1: Supervised learning

- **Unsupervised Machine Learning:** In scenarios where training data lacks labels or classification, unsupervised learning comes into play [3]. This approach focuses on inferring hidden structures within unlabeled data to develop a function that describes the data. While this method does not aim to determine specific outputs, it explores the data to uncover valuable insights and patterns.

For example (fig 1.2), the algorithm can extract the similar properties of fruits within a bag and group fruits with similar properties into different bags (same color or similar shape)

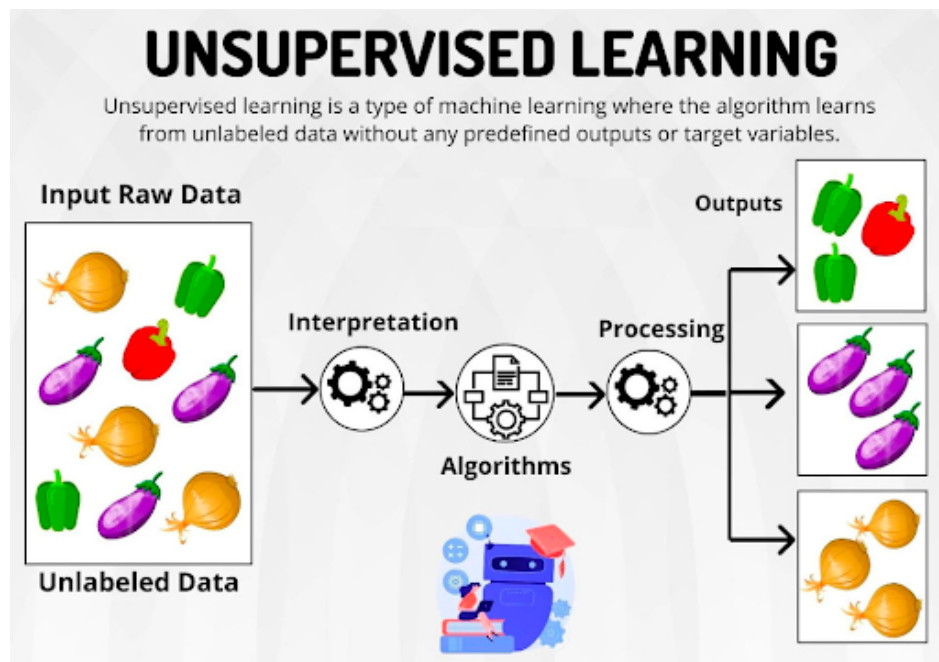


Figure 2: Unsupervised learning

- **Semi-supervised machine learning:** Combining aspects of both supervised and unsupervised learning. This approach utilizes a mix of labeled and unlabeled data for training [4]. Typically, an algorithm works on a small amount of labeled data and a larger pool of unlabeled data. By incorporating this hybrid approach, learning accuracy can significantly improve. Also, obtaining labeled data is expensive and time-consuming, as the model can leverage the large pool of unlabeled data to improve its performance

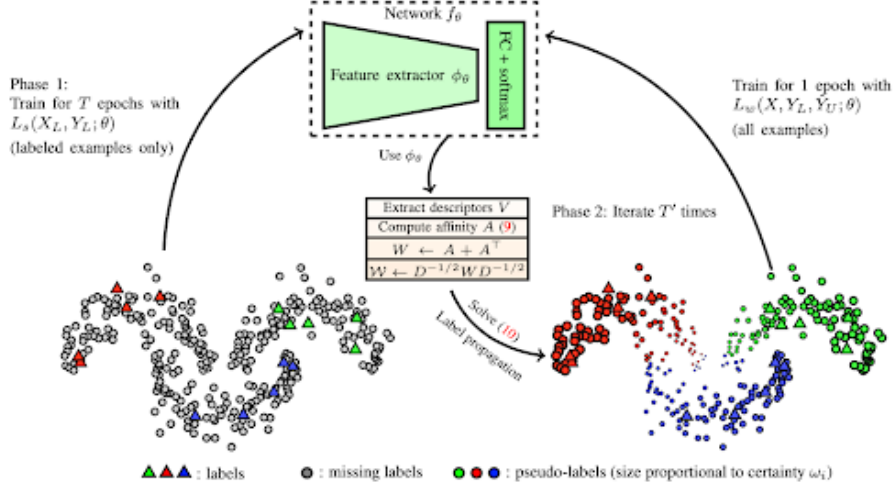


Figure 3: semi-supervised learning

- **Reinforcement learning:** This learning method involves an interactive process where an agent interacts with its environment, taking actions and receiving rewards [5]. Reinforcement learning is characterized by trial and error in making decisions and acquiring rewards. This approach enables machines and software agents to determine the optimal behavior within a specific context, maximizing their performance. Simple reward feedback guides the agent’s learning process, known as the reinforcement signal. When combined with AI and cognitive technologies, machine learning becomes a powerful tool for processing vast amounts of information, yielding faster and more accurate results to identify the most profitable opportunities or potential risks. Reinforcement learning is commonly used in areas such as robotics, game-playing, and autonomous systems.

2. Machine Learning in Everyday Applications

2.1 Virtual Personal Assistants

Virtual personal assistants like Siri, Alexa, and Google Now have become popular examples of AI-driven applications. These assistants provide information and perform tasks when prompted using voice commands. By leveraging machine learning, they refine the information they provide based on user interactions and preferences. These assistants are integrated into platforms like Amazon Echo, Google Home, and smartphone software like Samsung Bixby.

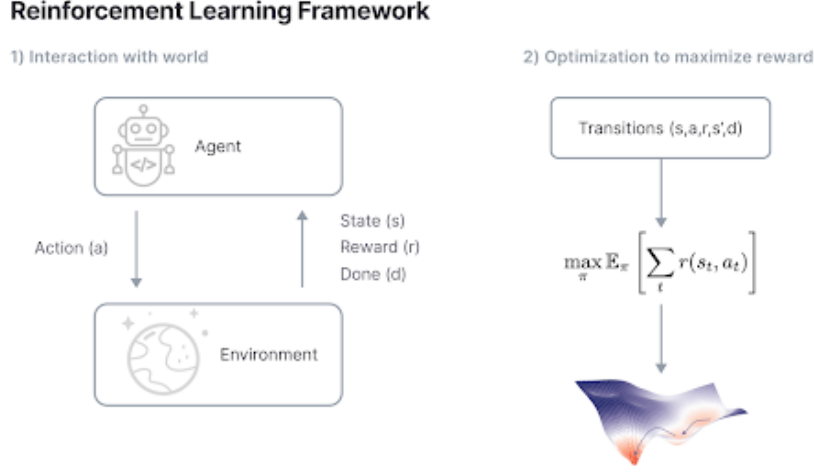


Figure 4: Reinforcement learning

2.2 Predictive in communication

Machine learning plays a crucial role in various aspects of communication. Traffic predictions rely on data from GPS navigation services to generate real-time traffic maps (for example, Google Maps) [6]. Machine learning algorithms can estimate areas prone to congestion by analyzing patterns and historical data. Online transportation networks also utilize machine learning to optimize routes, minimize detours, and predict demand, enhancing the efficiency and cost-effectiveness of shared mobility services [7].

2.3 Video surveillance

Machine learning revolutionizes video surveillance systems by enabling AI-powered detection of potential criminal activities. These systems can identify unusual behaviors, such as prolonged motionlessness or stumbling, and alert human attendants to prevent incidents. By continuously learning from data, machine learning enhances the accuracy and effectiveness of video surveillance, improving overall security [8].

2.4 Social media services

Machine learning underpins various features in social media platforms, enhancing user experiences and personalization. Examples include "People You May Know" recommendations, where machine learning analyzes user connections, profile visits, and shared interests to suggest potential friends. Face recognition algorithms leverage machine learning to identify and tag individuals in

photos. Additionally, machine learning powers computer vision techniques that identify objects in images and recommend related content [9],

2.5 Recommendation System

Machine learning is instrumental in recommendation systems used by streaming platforms, e-commerce websites, and social media platforms. These systems analyze user preferences, browsing history, and behavior to suggest personalized content, products, or connections. By continuously learning from user feedback and interactions, recommendation systems improve their accuracy and provide tailored recommendations that cater to individual user preferences [10].

3. Internet of Things

3.1 Introduction

The Internet of Things (IoT) refers to a networked system where various physical objects are interconnected and accessible via the Internet. These objects, often called "things," can range from individuals with heart monitors to automobiles with sensors. Each object is assigned an IP address, enabling them to collect and transmit data over a network without manual intervention. By leveraging embedded technology, these objects can interact with their internal states or the surrounding environment, influencing their decisions.

Typically, IoT's primary components are devices capable of connecting to the Internet and are divided into Sensors and actuators. Sensors are devices capable of detecting or measuring specific phenomena, collecting related data, and transmitting them to other entities (other IoT devices or application servers in the cloud). Some examples of IoT sensors include GPS devices, thermostats, and temperature sensors. To meet the cost-effectiveness requirements of IoT solutions, sensor nodes typically employ small-scale embedded systems, often utilizing 8-bit microcontrollers with limited storage capacity. This design enables them to operate on battery power for extended periods, sometimes lasting years. Furthermore, a wide range of networking protocols allows sensor nodes to integrate seamlessly into existing infrastructures and diverse operational conditions. This flexibility greatly facilitates the deployment of IoT solutions across various domains [11].

On the other hand, Actuators are physical devices that can execute commands transmitted by a control center or act according to some programmed conditions to create a change in the surrounding environment [12].

3.2 Applications of IoT

IoT technology enhances mobility services, bolsters public safety, and automates city household systems. Within an area of smart city, one notable application is intelligent transportation, which focuses on optimizing road infrastructure and facilitating efficient route planning for drivers. It involves innovative solutions like smart traffic signals and sensors that monitor and manage traffic systems across the road network. By facilitating smoother traffic flow and reducing congestion, these technologies contribute to improved transportation experiences. However, the scope of smart city services extends beyond transportation. It encompasses various aspects of urban life, including public safety, environmental sustainability, efficient delivery of municipal services, smart grid systems, and integrating physical infrastructure with the digital realm. [13]



Figure 5: Applications of IoT

Home automation is another section (considered a sub-section of smart city), and control systems have significantly transformed our living environments.

They have diverse applications within homes, including entertainment and smart living, surveillance, and safety management. Home automation refers to integrating IoT technology into a standard home environment to provide a secure and comfortable lifestyle. These systems rely on intelligent, self-adaptive mechanisms that analyze and evaluate user behaviors, predict future actions, and interact accordingly. Home automation systems utilize image detection and facial recognition models embedded in an intelligent control system. This control system is connected to sensors such as light, motion, water leak, smoke, and CCTV. These devices communicate with each other through a gateway distributed across a home area network. The home control system connects different subsystems collaborating to model user actions and gather environmental information, such as temperature, humidity, noise, visibility, and light intensity, to enhance learning. For instance, lighting and AC temperature can be controlled and automated based on users' needs and movements within the home environment. Home automation research extends beyond energy optimization and encompasses health monitoring and security measures. By leveraging innovative IoT technologies, users can remotely access surveillance cameras within their homes through mobile devices. Additionally, stakeholders can employ door and window sensors to ensure home safety and security from a distance.

IoT has also expanded its application to the industrial sector. Industrial IoT harnesses the capabilities of IoT technology in the business and economic sectors to automate previously complex manual operations, meeting consumer demands while reducing production costs. Various industrial domains, including warehouse operations, logistics services, supply chain management, and agricultural breeding, can benefit from machine-to-machine (M2M) intercommunication, facilitating optimal industrial operations. For example, the application of communicating sensors in agricultural systems.

This system utilizes smart agriculture technology to monitor and analyze environmental parameters using sensors such as ZigBee, EnOcean, Z-wave, and ANT, which are specifically designed for soil moisture monitoring and harvesting. These sensors are automated to assess the plant's condition and gather relevant data through an IoT platform. Sensors utilize collected information to take appropriate actions, such as determining the optimal timing for irrigation by consulting a weather forecasting service available in the Cloud. It ensures

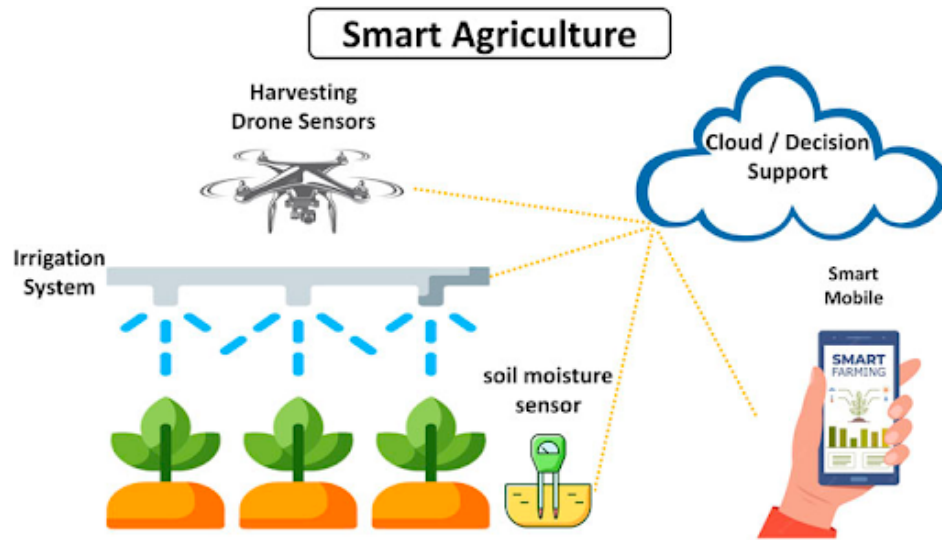


Figure 6: 2.2 Smart agricultural system

the efficient utilization of water resources while maintaining crop health. In the Healthcare domain, IoT also has a wide range of applications. IoT sensors and devices have transformed the landscape of portable and wearable medical devices, expanding their applications from fitness and wellness to medically qualified devices suitable for use in hospitals and healthcare facilities. This shift has facilitated the integration of remote patient monitoring (RPM) in healthcare settings, particularly for patients with chronic diseases. As a result, significant efforts are made to advance RPM systems by leveraging well-established IoT infrastructures and standards in the healthcare domain. These RPM systems aim to match or surpass the performance of existing monitoring and examination methods employed in hospitals and healthcare facilities. For instance, continuous heart rate monitoring and immediate detection of irregular heartbeats traditionally required patients to be hospitalized or connected to devices like Holter monitors for long-term cardiac diagnosis. However, these setups limited patient mobility due to device size and the number of connected wires.

Furthermore, hospitals expended substantial resources on providing long-term cardiac monitoring, which was often unavailable, particularly in low or middle-income countries. Remote patient monitoring systems effectively address these challenges and reduce mortality rates associated with chronic diseases such as heart disease and diabetes. IoT platforms and devices have played a significant role in accelerating the development and integration of RPM systems into existing healthcare infrastructures. Consequently, a typical RPM implementation encompasses various services, including but not limited to data acquisition,

tracking, communication, automated analysis, diagnoses, and notification systems.

4. Neural Networks Deep learning

4.1 Neural Networks Deep learning introduction

A neural network is a series of algorithms that aims to discover underlying relationships in a dataset by simulating the functioning of the human brain. Neural networks can adapt to changing input, generating optimal results without redesigning output criteria. This concept, rooted in artificial intelligence, is increasingly popular in developing intelligent systems.

In finance, neural networks are utilized in various processes such as time-series forecasting, algorithmic trading, securities classification, credit risk modeling, and constructing proprietary indicators and price derivatives.

Moreover, neural networks have been widely used for image recognition tasks such as object detection, image classification, and facial recognition. In natural language processing, it has shown remarkable performance in various tasks such as language translation, sentiment analysis, and text generation. Also, the application of neural networks has been demonstrated in speech recognition, autonomous vehicles, and medical diagnosis. The functioning of a neural network resembles that of the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and categorizes information based on a specific architecture. The network bears similarities to statistical methods like curve fitting and regression analysis [14].

A neural network consists of interconnected layers of nodes, where each node functions as a perceptron, similar to multiple linear regression. The perceptron passes the signal from multiple linear regression through an activation function, which can be nonlinear

In a multi-layered perceptron (MLP), perceptrons are organized into interconnected layers. The input layer receives input patterns, while the output layer provides classifications or output signals corresponding to the input patterns. For example, input patterns may consist of technical indicators for security, and potential outputs could be "buy," "hold," or "sell".

Hidden layers in the neural network adjust the input weightings until the network's margin of error is minimized. It is hypothesized that hidden layers extract significant features from the input data that have predictive power for

the outputs. This process, known as feature extraction, serves a purpose similar to statistical techniques like principal component analysis [15].

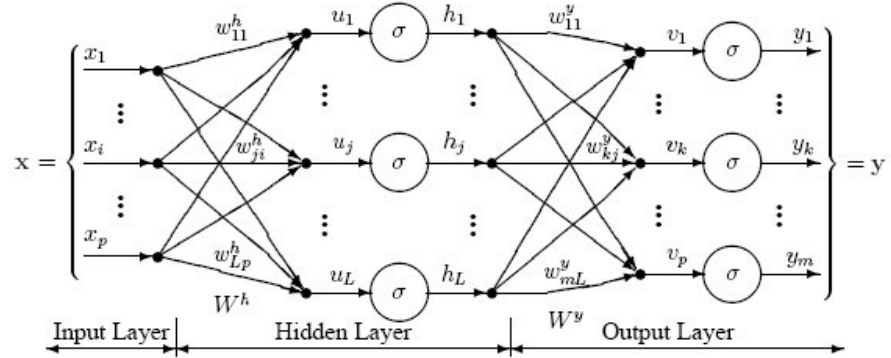


Figure 7: Multi-layer perceptron

A Deep Neural Network (DNN) is a type of artificial neural network (ANN) that consists of multiple layers between the input and output layers. The DNN employs various mathematical operations to transform the input into the desired output, accommodating both linear and non-linear relationships.

Deep learning is a specific function within the field of artificial intelligence (AI) that emulates the information processing and pattern recognition capabilities of the human brain. It falls under the umbrella of machine learning and involves the use of neural networks capable of unsupervised learning from unstructured or unlabeled data. It is also referred to as deep neural learning or deep neural network [15].

4.2 Machine learning vs Deep learning

Deep neural networks are characterized by their deeply nested network architectures, typically consisting of multiple hidden layers. These networks employ advanced neurons that utilize operations like convolutions and multiple activations within a single neuron, going beyond the simple activation functions used in traditional artificial neural networks (ANNs). It enables deep neural networks to process raw input data and automatically learn representations relevant to the learning task. This capability is commonly referred to as deep learning. In contrast, simple ANNs (such as shallow autoencoders) and other machine learning (ML) algorithms like decision trees are considered part of shallow machine

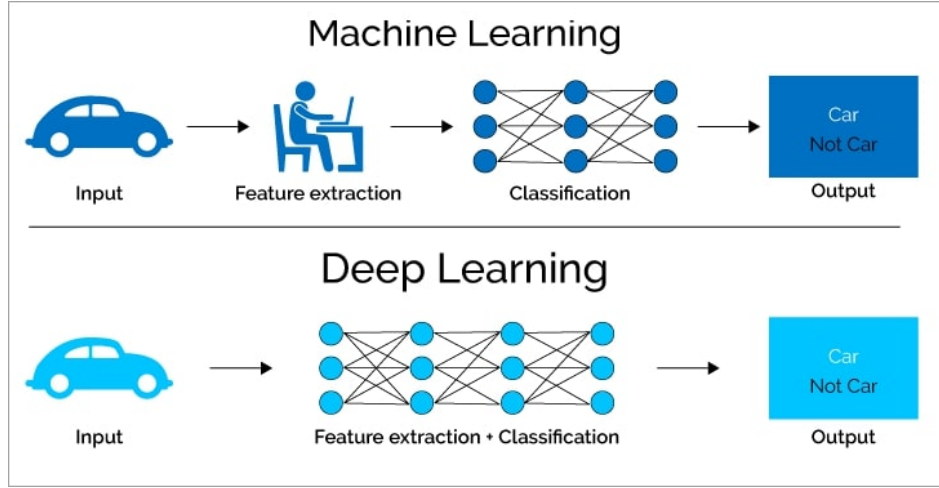


Figure 8: Deep learning vs Machine learning

learning as they lack these functionalities. While some shallow ML algorithms are inherently interpretable by humans and are considered as white boxes, the decision-making process of most advanced ML algorithms is inherently untraceable unless explicitly explained, making them black boxes.

Deep learning (DL) excels in domains with large and high-dimensional datasets, making deep neural networks outperform shallow ML algorithms in tasks involving text, image, video, speech, and audio data processing. However, when dealing with low-dimensional data inputs, especially in scenarios with limited training data availability, shallow ML approaches can still produce superior result, which are often more interpretable compared to those generated by deep neural networks [16].

4.3 Applications and challenges of AI in Internet of thing

Several case studies have showcased the successful integration of Internet of Things (IoT) and machine learning technologies in various fields including smart cities such as improving the efficiency of urban services, and enhancing citizens' overall quality of life. Deep learning algorithms combined with video analysis have been identified as practical applications in smart cities. In one study, researchers developed an IoT system based on deep learning for remote monitoring and early detection of health issues in real time. The system exhibited impressive accuracy in identifying heart conditions, achieving a remarkable accuracy rate of 0.982 [17, 18].

However, deploying deep learning methodologies within IoT frameworks presents challenges and limitations. These challenges can be broadly categorized into

ethical and privacy implications, scalability and resource constraints, and the ongoing need for research and development. Integrating deep learning into IoT has significantly advanced security and efficiency in surveillance applications. IoT devices with deep learning capabilities, such as convolutional neural networks, can effectively analyze video streams to detect threats and anomalies, improving real-time monitoring and predictive insights.

Nevertheless, these applications face significant challenges, such as the need for extensive datasets and substantial computational power. The scalability of deep learning models in the IoT poses a significant concern. IoT devices often have limited computational resources, making it challenging to deploy complex deep-learning models that require substantial data processing capabilities. It is crucial to optimize these models for deployment on resource-constrained devices, necessitating innovative solutions that strike a balance between the computational demands of deep learning algorithms and the inherent limitations of IoT hardware. [19]

II. TinyML and AI on the Edge

1. Edge computing AIoT overview

In recent years, edge computing has emerged as a promising technology with significant potential in various fields. It offers advantages such as reduced latency and cost savings. Unlike traditional cloud computing, edge computing enables data processing at the edge of the network. This approach brings data computing closer to the data source, greatly benefiting the development of time-sensitive applications. Additionally, by performing processing locally, edge computing reduces network traffic and minimizes data transmission, resulting in substantial cost savings. [20]

From the advantages of edge computing, integrating edge devices with IoT ecosystems can facilitate the shift of computation from the cloud to the network edge through collaboration among sensors, edge devices, and cloud facilities. However, edge hardware has limited resources and is platform-dependent, which restricts its ability to support advanced and complex services such as machine learning applications. It obstructs the development of a standardized machine learning framework for all IoT-edge devices.

Furthermore, while existing embedded processors can handle generic sensor data processing and web-based applications, machine learning applications rely on sophisticated hardware chips such as graphics processing units (GPUs). These chips demand significant power and memory capacity to execute deep neural network models. Thus, the current landscape challenges achieving the envisioned "cloud-to-embedded" aspect. [21]

1.1 TinyML overview

Since 2019, a new technology called TinyML has been addressing the challenges of designing power-efficient deep learning models (in the milliwatt range and below) to be integrated into embedded systems like AIoT/IIoT/IoT devices. TinyML is defined as follows: "machine learning aware architectures, frameworks, techniques, tools, and approaches which are capable of performing on-device analytics for a variety of sensing modalities (vision, audio, speech, motion, chemical, physical, textual, cognitive) at mW (or below) power range setting, while targeting predominately battery-operated embedded edge devices suitable for implementation at large scale use cases preferable in the IoT or wireless sensor network domain" (TinyML, 2021a). Thus, TinyML can be envisaged

as the c
algorithm

Figure 9: TensorFlow lite for microcontrollers

The above figure illustrates the workflow for developing deep learning algorithms. First, the dataset is created based on the chosen neural network model (CNN, RNN, etc.). In some cases, dataset preprocessing may be necessary to optimize training and improve performance. Training is typically conducted on a workstation or supercomputer server using frameworks like TensorFlow or PyTorch. The trained model's precision is usually float32, and its parameter memory footprint is significant (larger than the available flash and SRAM on a microcontroller unit).

TensorFlow Lite, within the TensorFlow framework, provides a converter to optimize the trained model by reducing the memory footprint and computational power requirement (using int8 instead of float32). [22] The TensorFlow Lite framework converter takes the trained model (saved in the xx.h5 file) and pro-

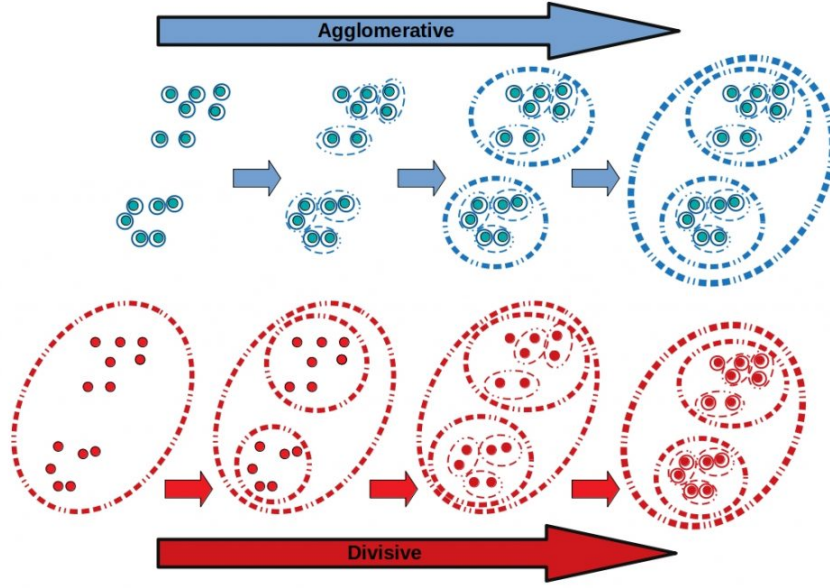
duces three optimized models: int8 quantization, integer dynamic quantization, and float16. One of these files, such as the int8 quantization tflite file, can be utilized in other tools like the X-CUBE-AI [23] extension pack of STM32CubeMx [24]. These tools can generate C codes that can be deployed and run on micro-controllers [22]

2.

III. Approaches

1. Community Detection Methods in Network Analysis: Agglomerative and Divisive Approaches

Community analytics, an essential task in network analysis, involves the identification of groups or communities within a network structure. Two primary types of methods are commonly employed for community detection: agglomerative methods and divisive methods.



In agglomerative methods (illustrated on the upper Figure 1), the process initiates with individual nodes and gradually merges them based on their similarity or connectivity, leading to hierarchical clustering.

This approach merges nodes from the bottom up, progressively forming larger communities. The agglomerative category includes algorithms such as Louvain and Leiden, which utilize modularity and hierarchical clustering.

Modularity, represented as Q , measures community strength and guides the algorithm. Higher modularity values indicate better communities, while a value below 1 suggests that each node is treated as a separate community.

The dendrogram in Figure 1 represents the hierarchy of clusters generated by hierarchical clustering. On the other hand, divisive methods (depicted on the bottom of Figure 1) start with a single partition containing all nodes and iteratively split it by removing edges with low similarity.

This process, known as the split process, aims to find smaller, more cohesive communities by progressively dividing the network. Although agglomerative

and divisive methods employ different approaches, both ultimately aim to unveil meaningful groups of nodes based on their connectivity patterns within the network.

2. Clauset-Newman-Moore greedy modularity maximization

The Clauset-Newman-Moore (CNM) greedy modularity maximization algorithm is a popular method used in community detection, a field of network analysis. It is designed to partition a given network into communities, where members of each community are more densely connected to each other compared to members of different communities. The CNM algorithm aims to maximize the modularity of the network, which is a measure of the quality of the community structure.

The algorithm was proposed by Aaron Clauset, Mark Newman, and Christopher Moore in 2004, and it has since become widely adopted due to its simplicity and effectiveness. The CNM algorithm is an agglomerative method and follows a greedy approach, iteratively merging and splitting communities to optimize the modularity score.

Network representation: Considering an undirected network/graph with N nodes and E edges. We can represent the network as a graph adjacency matrix A of size $N \times N$, where $A[i][j] = 1$ if there is an edge between nodes i and j , and $A[i][j] = 0$ otherwise.

$$A_{vw} = \begin{cases} 1, & \text{if vertices } v \text{ and } w \text{ are connected,} \\ 0, & \text{otherwise.} \end{cases}$$

Community Structure: We represent the community structure of the network using a partition vector s of length N , where $s[i]$ denotes the community membership of node i . Each node belongs to exactly one community.

The fraction of edges that fall within communities is given by:

$$\frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, c_w)$$

where:

- $\sum_{vw} A_{vw}$ represents the summation over all pairs of vertices v and w , considering the elements of the adjacency matrix A . It computes the total number of edges in the graph, denoted as m .

- $\delta(c_v, c_w)$ is the Kronecker delta function, which takes the value 1 if the

community assignments of vertices v and w (c_v and c_w , respectively) are the same, and 0 otherwise.

- $\frac{1}{2m}$ is a normalization factor, dividing by the total number of edges to ensure the measure falls within the range $[0, 1]$.

However, this measure alone is not sufficient to evaluate the quality of a community structure. It reaches its maximum value of 1 in the trivial case where all vertices belong to a single community. Additional measures like modularity is considered for a comprehensive assessment of community structure.

Modularity: The probability of an edge existing between vertices v and w , assuming random connections while respecting vertex degrees, is given by $\frac{k_v k_w}{2m}$. Here, k_v represents the degree of vertex v , k_w represents the degree of vertex w , and m is the total number of edges in the graph.

Based on this probability, we define the modularity Q as follows:

$$Q = \sum_{s=1}^m \left[\frac{l_s}{L} - \left(\frac{d_s}{2L} \right)^2 \right] \quad (1)$$

(m = number of modules, l_s = the number of edges inside module s , L = the number of edges in the network, d_s = total degree of the nodes in module s)

This process of removing an edge and calculating the modularity is iteratively repeated. The algorithm will stop when the new modularity is no longer greater than the modularity from the previous iteration. The ending modularity is usually around 0.6.

2.1 Pseudo Code

Algorithm 1 Clauset-Newman-Moore (CNM) Algorithm

Input: Network graph

Output: Community structure

- 1 Initialization: Assign each vertex to its own community
 - 2 Calculate the initial modularity value Q_{initial}
 - 3 **repeat**
 - 4 **foreach** pair of communities C_1 and C_2 **do**
 - 5 Calculate the change in modularity ΔQ by merging C_1 and C_2
 - 6 **end**
 - 7 **until** no further improvement in modularity is possible;
 - 8 Find the pair (C_1, C_2) with the maximum increase in modularity ΔQ_{max}
 - 9 Merge communities C_1 and C_2 Update the modularity value Q by adding ΔQ_{max}
 - return** Community structure
-

2.2 Complexity

Since the joining of a pair of communities between which there are no edges at all can never result in an increase in Q , we need only consider those pairs between which there are edges, of which there will at any time be at most m , where m is again the number of edges in the graph.

The change in Q upon joining two communities is given by $\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$, which can clearly be calculated in constant time. Following a join, some of the matrix elements e_{ij} must be updated by adding together the rows and columns corresponding to the joined communities, which takes worst-case time $O(n)$. Thus each step of the algorithm takes worst-case time $O(m + n)$.

There are a maximum of $n - 1$ join operations necessary to construct the complete dendrogram and hence the entire algorithm runs in time $O((m+n)n)$, or $O(n^2)$ on a sparse graph. The algorithm has the added advantage of calculating the value of Q as it goes along, making it especially simple to find the optimal community structure.

Each step: worst-case time $O(m + n)$.

A maximum of $(n - 1)$ steps to join n communities.

Thus: $O((m + n)n)$ or $O(n^2)$ for sparse graphs.

2.3 Pros and Cons

Pros

- **Ease of implementation:** Modularity Maximization (MM) is conceptually simple and can be implemented with relative ease compared to other algorithms. Several open-source libraries and software packages readily implement MM, making it accessible to a wide range of users.
- **Scalability:** MM efficiently scales to handle large networks with millions of nodes and edges. This makes it suitable for analyzing real-world networks like social media graphs, citation networks, and protein-protein interaction networks.

Cons

- **Resolution limit:** MM is sensitive to the resolution parameter, which controls the granularity of the detected communities. Choosing an appropriate resolution parameter can be challenging, as it can significantly impact the community structure. Small values tend to result in many small communities, while large values lead to a few large communities, potentially missing finer-grained structures.
- **Merging similar clusters:** MM can be biased towards merging similar clusters, even if they are not well-connected, to maximize the modularity score. This can lead to communities that are not cohesive or representative of the underlying network structure.

3. Louvain Algorithm

The Louvain algorithm is a widely used community detection algorithm that aims to identify communities or clusters within a network graph. It was developed by Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre in 2008.

The algorithm is known for its efficiency and ability to handle large-scale networks. It follows a greedy optimization approach that iteratively improves the modularity measure of the network by merging and rearranging communities. It can be used to analyze a network of 2 million nodes in only 2 minutes

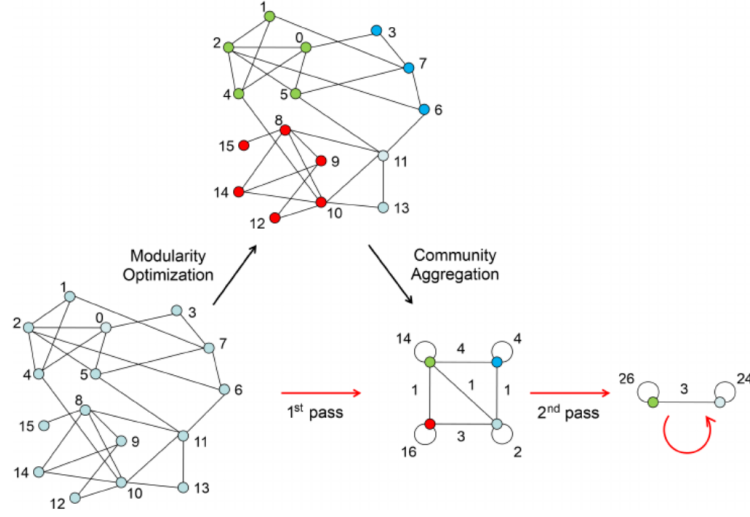


Figure 10: Louvain Algorithm

3.1 Modularity Optimization

As depicted in the figure, the first step is to optimize the modularity of the entire graph. In this example, it splits the nodes into four communities.

To find these clusters, each node is moved into its neighboring community. If the change in modularity (ΔQ) is greater than 0, it is moved into the neighboring community. Otherwise, it remains in its current community. This process is repeated until $\Delta Q = 0$ for all nodes.

3.2 Community Aggregation

After the optimization of modularity, super nodes are introduced to represent each cluster. Following the initial phase of the algorithm, numerous communities are formed.

Nevertheless, the two phases continue to iterate, resulting in the formation of larger and larger communities. The algorithm terminates only when neither of the two operations can further enhance the community structure.

3.3 Pseudo Code

Algorithm 2 Louvain Algorithm

Input: Graph $G = (V, E)$

Output: Community structure of the graph

```
10 Initialize each node as a separate community
11 repeat
12   for each node  $v \in V$  do
13     Remove  $v$  from its current community
14     Calculate the modularity gain  $\Delta Q$  for each neighboring community by
        moving  $v$ 
15     Move  $v$  to the neighboring community with the highest  $\Delta Q$ 
16   end
17   Construct the aggregated network where each community is represented by a
        node
18   Update the original graph with the new community assignments
19 until No further improvement in modularity;
```

3.4 Pros and Cons

Pros of the Louvain algorithm:

- Fast and scalable for large networks.
- Optimizes modularity, which measures community quality.
- Detects hierarchical community structures.
- Allows flexibility in resolution for different levels of detail.
- Widely used with extensive resources and documentation.

Cons of the Louvain algorithm:

- Resolution limit may merge small communities into larger ones.
- Results can be sensitive to initial community assignments.
- Assumes non-overlapping communities.
- Bias towards detecting large, cohesive communities.
- Lacks theoretical guarantees of optimality.

4. Label Propagation Algorithm

4.1 Intuition

As we will show, the advantage of this algorithm over the other methods is its simplicity and time efficiency. The algorithm uses the network structure to guide its progress and does not optimize any specific chosen measure of community strengths.

4.2 Pseudocode

1. Initialize the labels at all nodes in the network. For a given node x , $C_x(0) = x$.
2. Initialize $t = 1$.
3. Arrange the nodes in the network in a random order and set it to X .
4. For each x in X chosen in that specific order,
let $C_x(t) = f(C_{x_{i1}}(t), \dots, C_{x_{im}}(t), C_{x_{i(m+1)}}(t-1), \dots, C_{x_{ik}}(t-1))$.
 f here returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly.
5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, increment t and go to 3.

4.3 Complexity

It takes a near-linear time for the algorithm to run to its completion. Initializing every node with unique labels requires $O(n)$ time. Each iteration of the label propagation algorithm takes linear time in the number of edges $O(m)$. At each node x , we first group the neighbors according to their labels $O(d_x)$. We then pick the group of maximum size and assign its label to x , requiring a worst-case time of $O(d_x)$. This process is repeated at all nodes and hence an overall time is $O(m)$ for each iteration.

4.4 Pros and Cons

Pros

- Efficiency: Label propagation is computationally efficient, making it suitable for large networks. Its running time is generally faster compared to some other community detection algorithms.

- **Low A Priori Information Requirement:** One of its notable strengths is its low dependency on prior information about the network structure. You don't need to specify parameters beforehand, which can be advantageous in scenarios where the network characteristics are not well-known.

Cons

- **Lack of Unique Solutions:** As you mentioned, the algorithm doesn't guarantee a unique solution. This can be a drawback if you're looking for a single, definitive community structure. The results can vary across different runs of the algorithm.
- **Aggregate Solutions:** The algorithm provides an aggregate of multiple solutions, leading to a lack of specificity. This might be undesirable in situations where a precise and unique community structure is crucial.

IV. Experiments

1. Preprocessing

The dataset is quite big to handle contain 3 file:

- Books.csv: 271379 records
- Users.csv: 276271 records
- **Ratings.csv: 1149781 records**

Therefore, we are using the map-reduce technique to split the data in order to fit the computer memory.

The map reduce is using to group and count all pair of books bought by at least 2 users and running on file Ratings.csv.

Map reduce first phase:

Map phase	Reduce phase
<User-ID1, ISBN1>	<User-ID1, [ISBN1, ISBN2, ISBN3, ...]>
<User-ID1, ISBN2>	
<User-ID1, ISBN3>	

Table 1: Map reduce first phase find all books were bought by single users

Map second phase:

Map phase	
<User-ID1, [ISBN1, ISBN2, ISBN3, ...]>	<[ISBN1, ISBN2], User-ID1>
	<[ISBN1, ISBN3], User-ID1>
	<[ISBN2, ISBN3], User-ID1>

Table 2: Map phase to find all books pair of book bought by users

Reduce second phase:

Reduce phase	
<[ISBN1, ISBN2], User-ID1>	<[ISBN1, ISBN2], Count1>
<[ISBN1, ISBN3], User-ID1>	<[ISBN1, ISBN3], Count2>
<[ISBN2, ISBN3], User-ID1>	<[ISBN2, ISBN3], Count3>

Table 3: Reduce phase to find all books pair of books and number of users bought it

2. Build application

We built a web application to demonstrate the process of recommendation in book store. Web are built with VueJS as a Frontend and Flask as a framework by Python language programming. Backend is in charge of query similar books and return it to Frontend via Restful api.



Figure 11: Technologies used

2.1 Build Backend

Flask introduction: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

All required libraries:

Create file requirements.txt:

```
1 Flask==2.2.3
2 Flask-Cors==3.0.10
3 pandas==2.0.0
4 networkx==2.8.8
```

Install all libraries via command:

```
1 pip install -r requirements.txt
```

Run backend locally on port 5000

```
1 python main.py
```

2.2 Build Frontend

VueJS introduction: Vue.js is an open-source model–view–viewmodel front end JavaScript framework for building user interfaces and single-page applica-

tions. It was created by Evan You, and is maintained by him and the rest of the active core team members

2.2.1 Components

Include 3 pages

1. Homepage: Some categories, and popular items among users

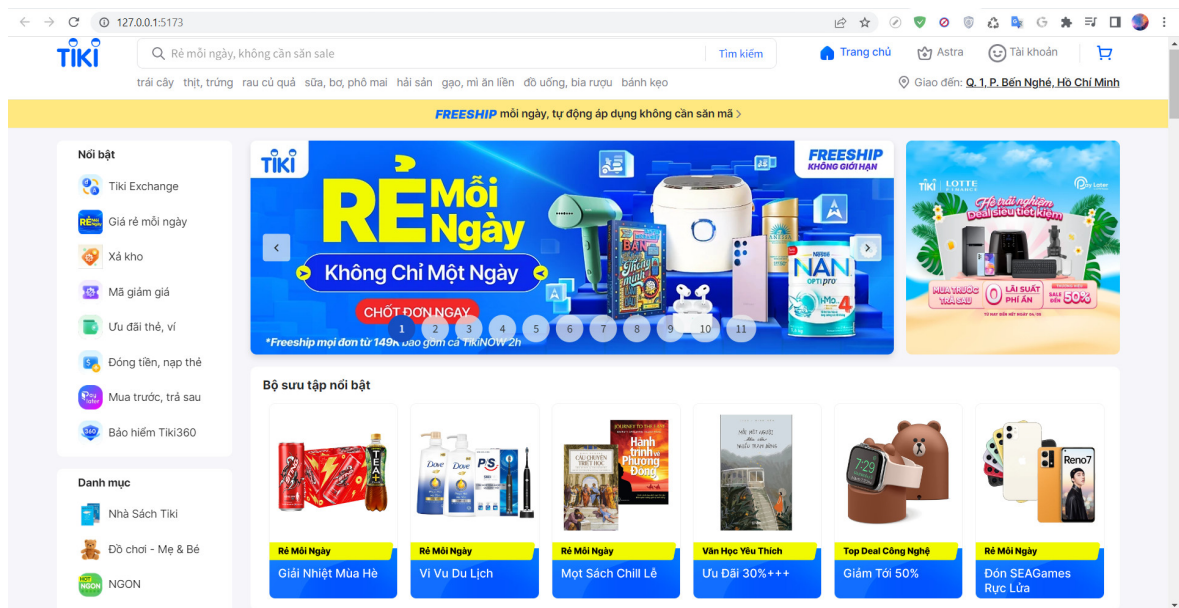


Figure 12: Homepage

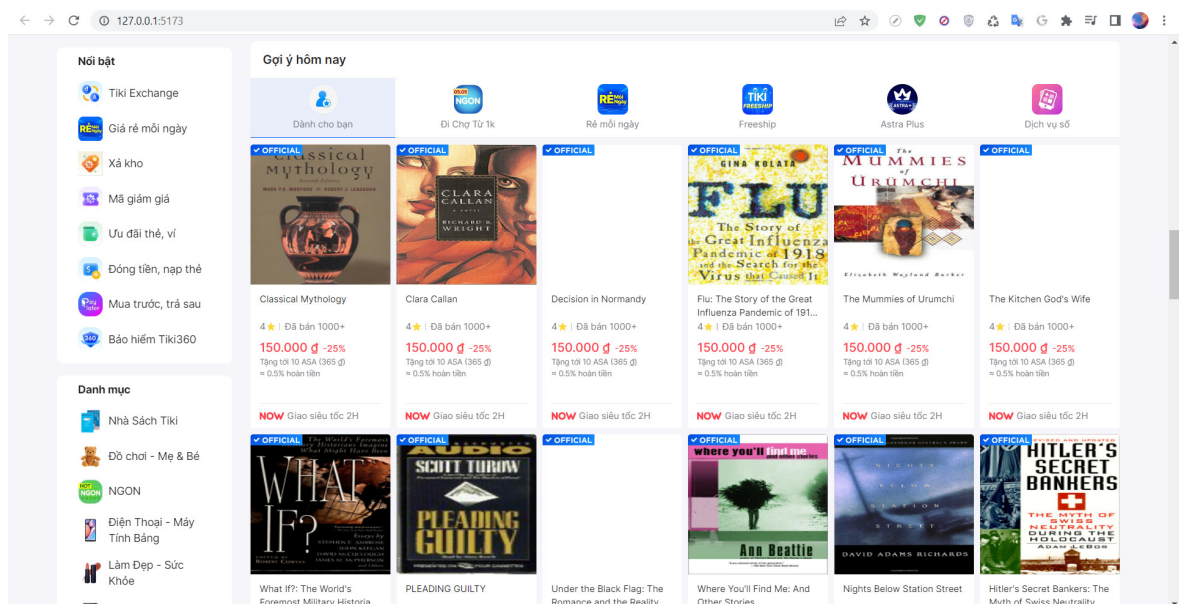


Figure 13: Book category in homepage

2. Book category page: All books in E-Commerce using pagination.

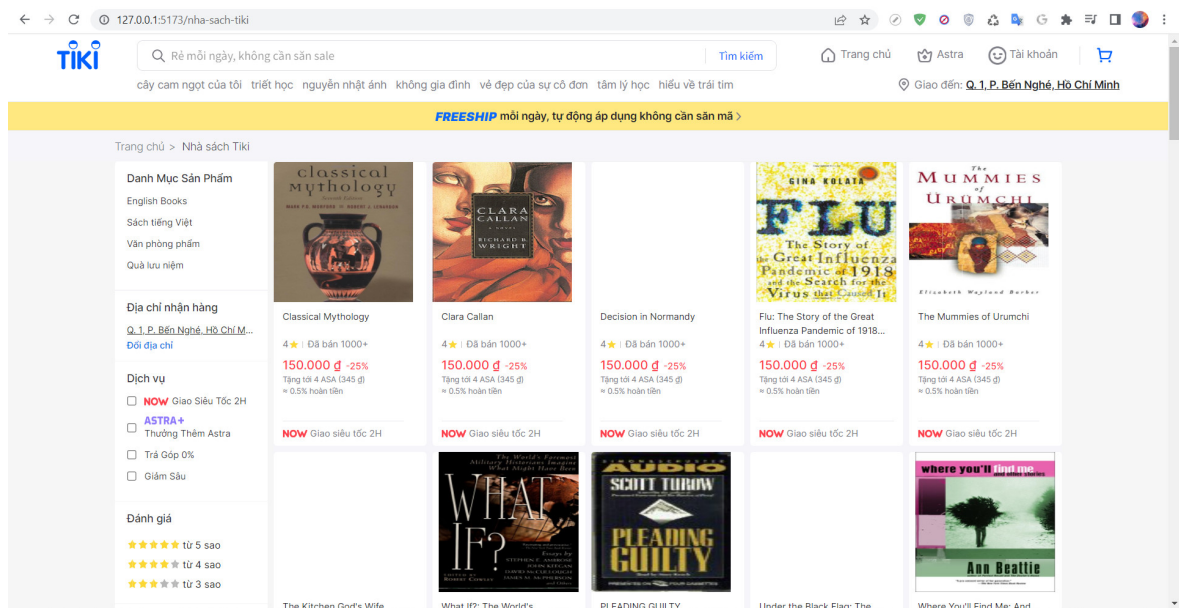


Figure 14: Book category page

3. Book detail page: Detail of a book and all similar book using Louvain, Leiden, Girvan Newman.

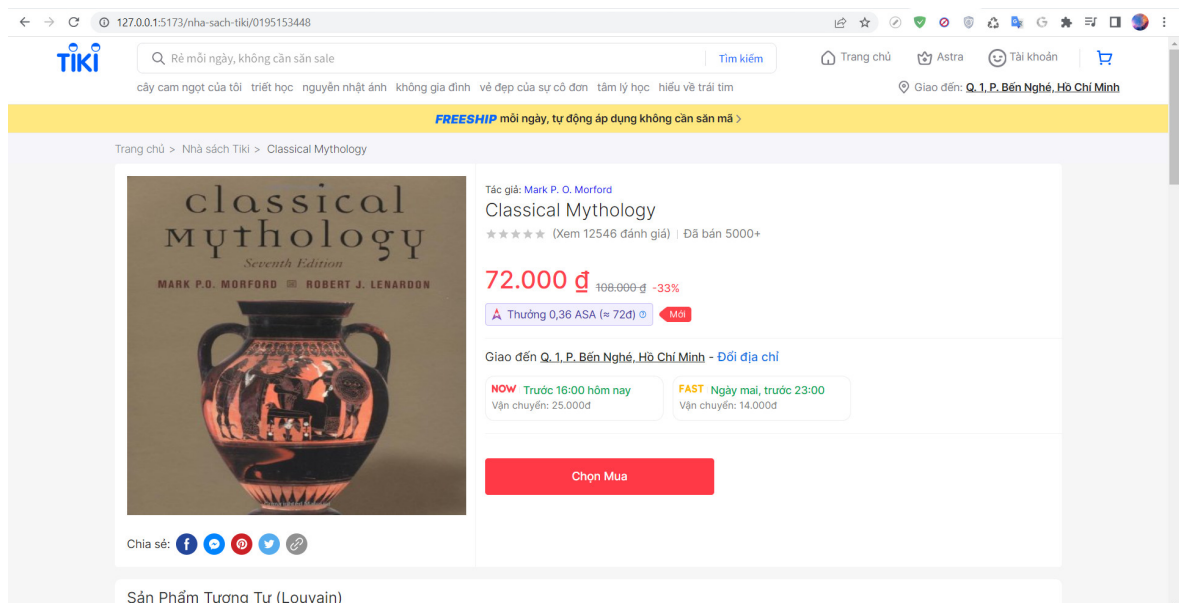


Figure 15: Book detail page

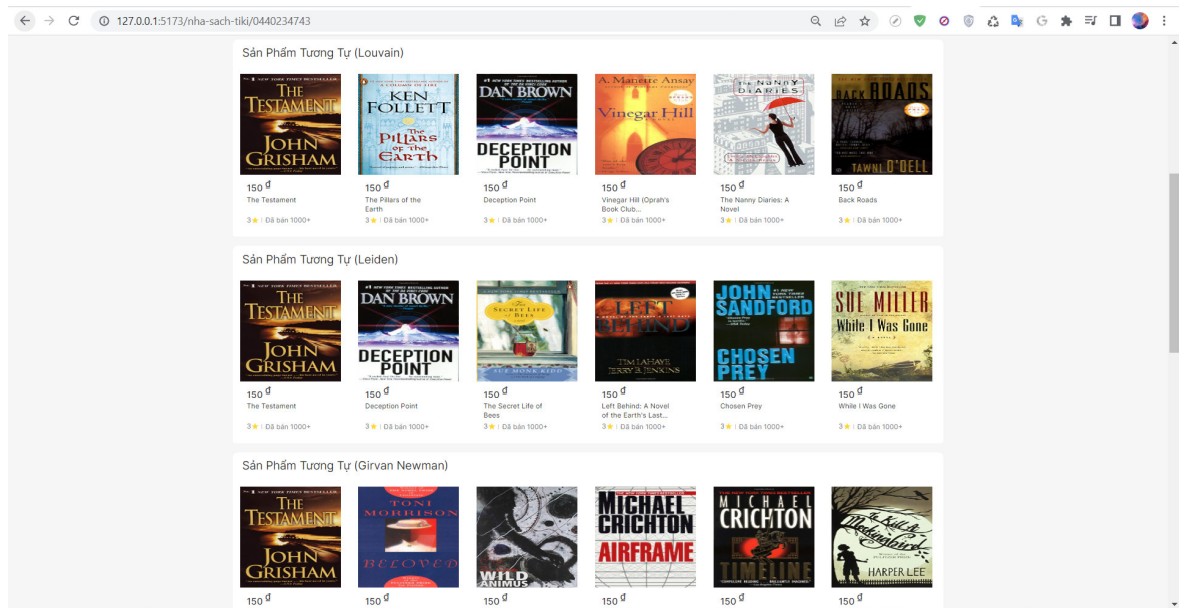


Figure 16: Book detail page recommendation algorithms

3. Modularity score comparison

To improve the efficiency of measuring algorithm modularity, we implemented a strategy of dividing the dataset into smaller subsets. Our approach involved selecting the top 500 books that appeared in all book pairs from the previous Map Reduce step. This selection criterion was chosen over random selection from all pairs to avoid generating a sparse graph that would result in significantly lower modularity scores for all three algorithms, making it challenging to measure them accurately.

By ensuring that the selected nodes had the highest number of edges, we aimed to facilitate faster execution of all three algorithms

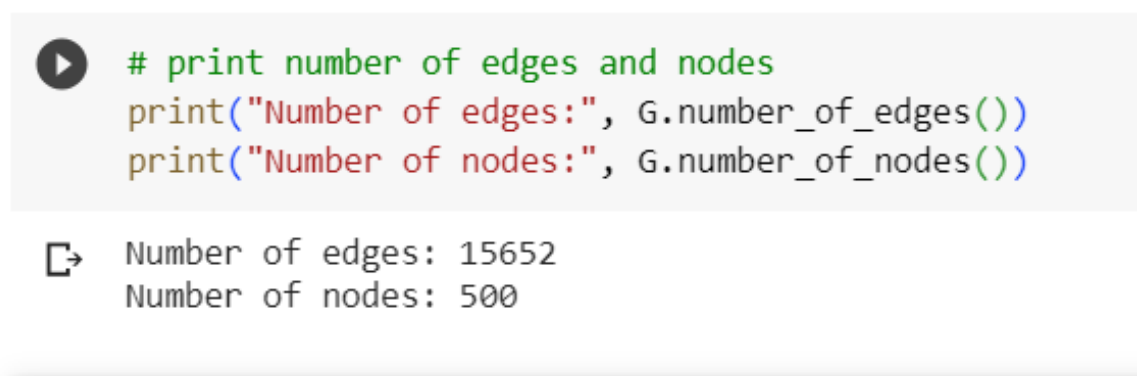
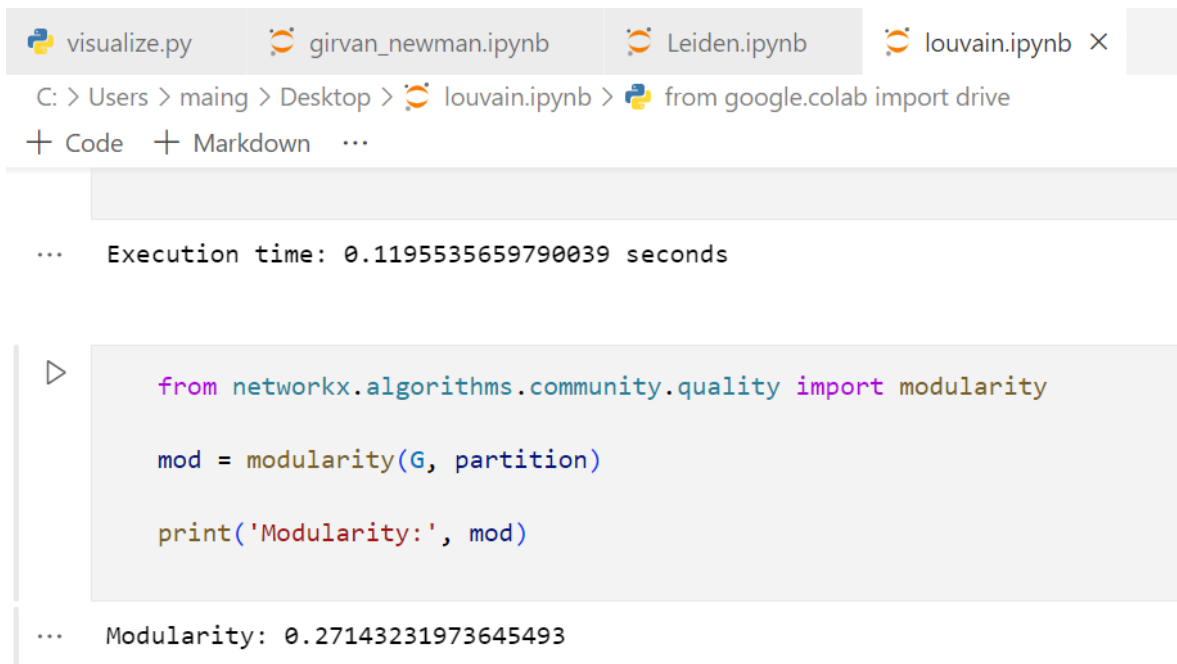


Figure 17: Number of edge and node for modularity measurement



The image shows a JupyterLab interface with four tabs: 'visualize.py', 'girvan_newman.ipynb', 'Leiden.ipynb', and 'louvain.ipynb'. The 'louvain.ipynb' tab is active. The breadcrumb path is 'C: > Users > maing > Desktop > louvain.ipynb > from google.colab import drive'. Below the path are buttons for '+ Code', '+ Markdown', and a menu icon. The code cell contains the following Python code:

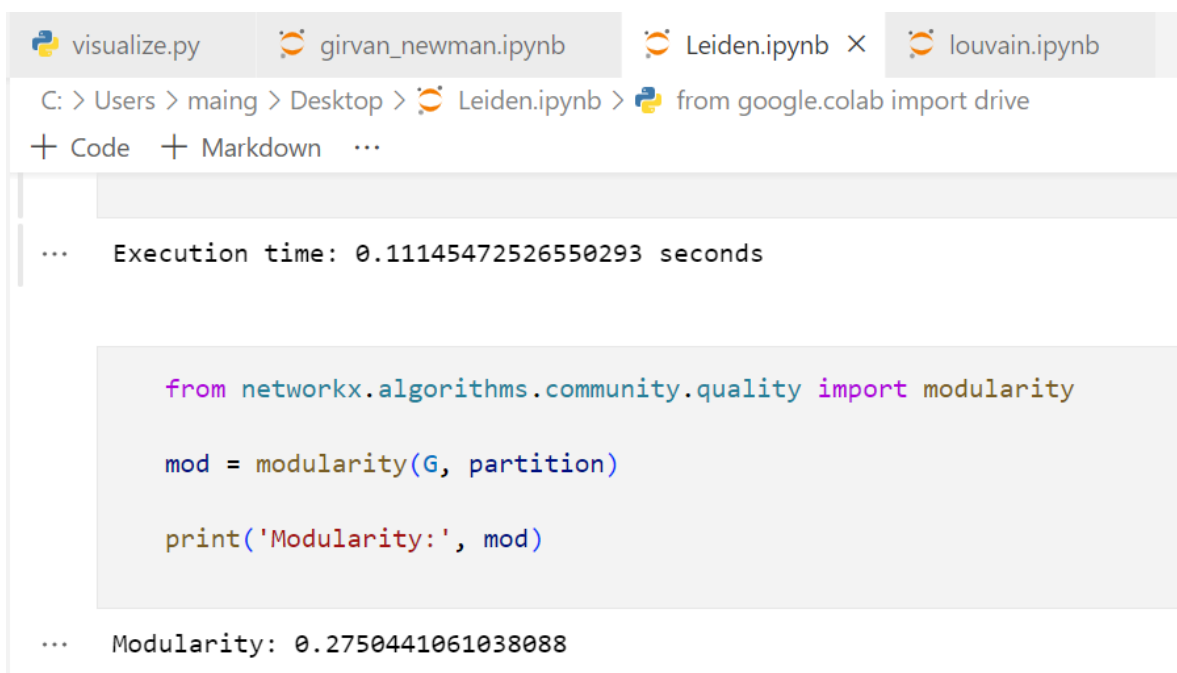
```
from networkx.algorithms.community.quality import modularity

mod = modularity(G, partition)

print('Modularity:', mod)
```

The output of the code cell is 'Modularity: 0.27143231973645493'. Above the code cell, the execution time is shown as 'Execution time: 0.1195535659790039 seconds'.

Figure 18: modularity measurement for Louvain algorithms



The image shows a JupyterLab interface with four tabs: 'visualize.py', 'girvan_newman.ipynb', 'Leiden.ipynb', and 'louvain.ipynb'. The 'Leiden.ipynb' tab is active. The breadcrumb path is 'C: > Users > maing > Desktop > Leiden.ipynb > from google.colab import drive'. Below the path are buttons for '+ Code', '+ Markdown', and a menu icon. The code cell contains the following Python code:

```
from networkx.algorithms.community.quality import modularity

mod = modularity(G, partition)

print('Modularity:', mod)
```

The output of the code cell is 'Modularity: 0.2750441061038088'. Above the code cell, the execution time is shown as 'Execution time: 0.11145472526550293 seconds'.

Figure 19: modularity measurement for Leiden algorithms

```
visualize.py  girvan_newman.ipynb  Leiden.ipynb  louvain.ipynb
C: > Users > maing > Desktop > girvan_newman.ipynb > from networkx.algorithms.communi
+ Code  + Markdown  ...

... Execution time: 137.56785249710083 seconds

[167] from networkx.algorithms.community.quality import modularity
      |
      | mod = modularity(G, communities)
      |
      | print('Modularity of the best partition:', mod)

... Modularity of the best partition: 0.0021023578934315575
```

Figure 20: modularity measurement for Girvan Newman algorithms

Louvain: The Louvain algorithm is recognized for its ability to generate community partitions with high modularity scores. Its iterative optimization process helps identify dense subgraphs and optimize modularity locally and globally.

Leiden: The Leiden algorithm slightly higher score than Louvain.

Girvan-Newman: The Girvan-Newman algorithm tends to have a lower modularity score compared to both Louvain and Leiden. This is because its divisive edge removal approach may not capture the community structure as effectively as the other two algorithms in most cases.

4. Time Efficiency comparison

To improve the efficiency of measuring algorithm modularity, we implemented a strategy of dividing the dataset into smaller subsets. Our approach involved selecting the top 500 books that appeared in all book pairs from the previous Map Reduce step. To ensured that all three algorithms were executed under similar conditions with large input size.

```
# print number of edges and nodes
print("Number of edges:", G.number_of_edges())
print("Number of nodes:", G.number_of_nodes())
```

Number of edges: 15652
Number of nodes: 500

Figure 21: Number of edge and node for time efficiency measurement

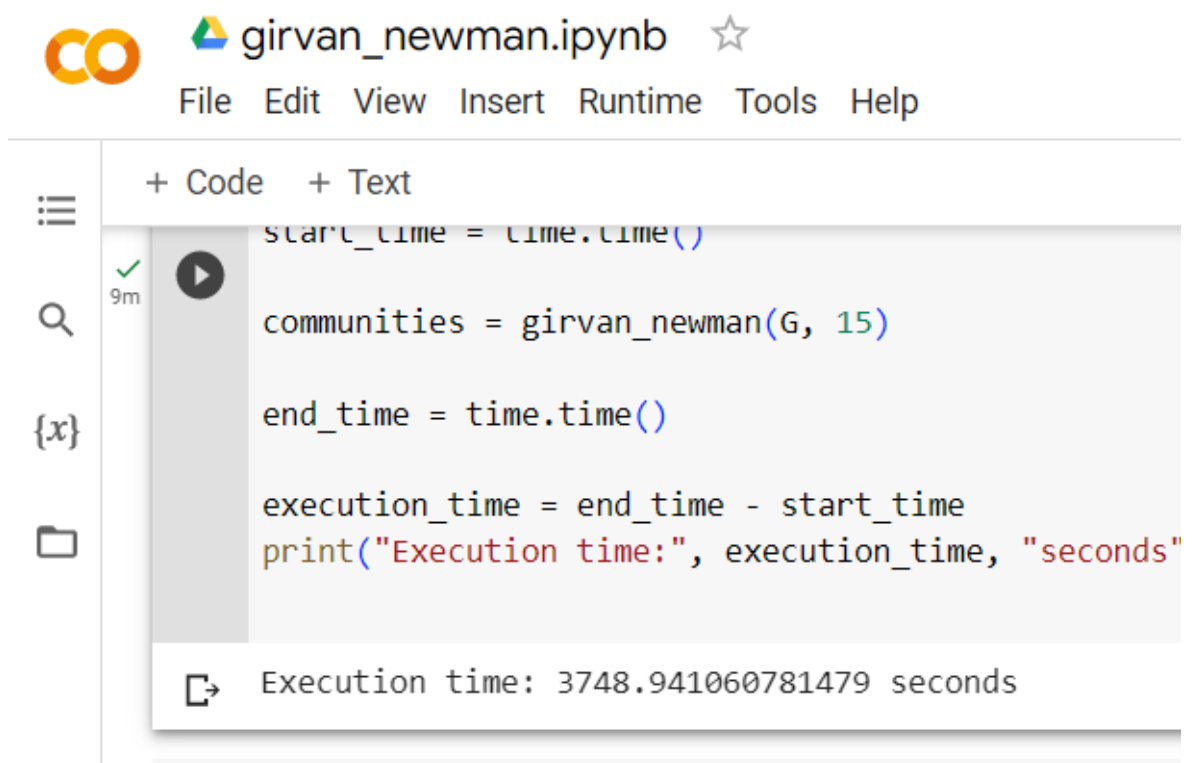
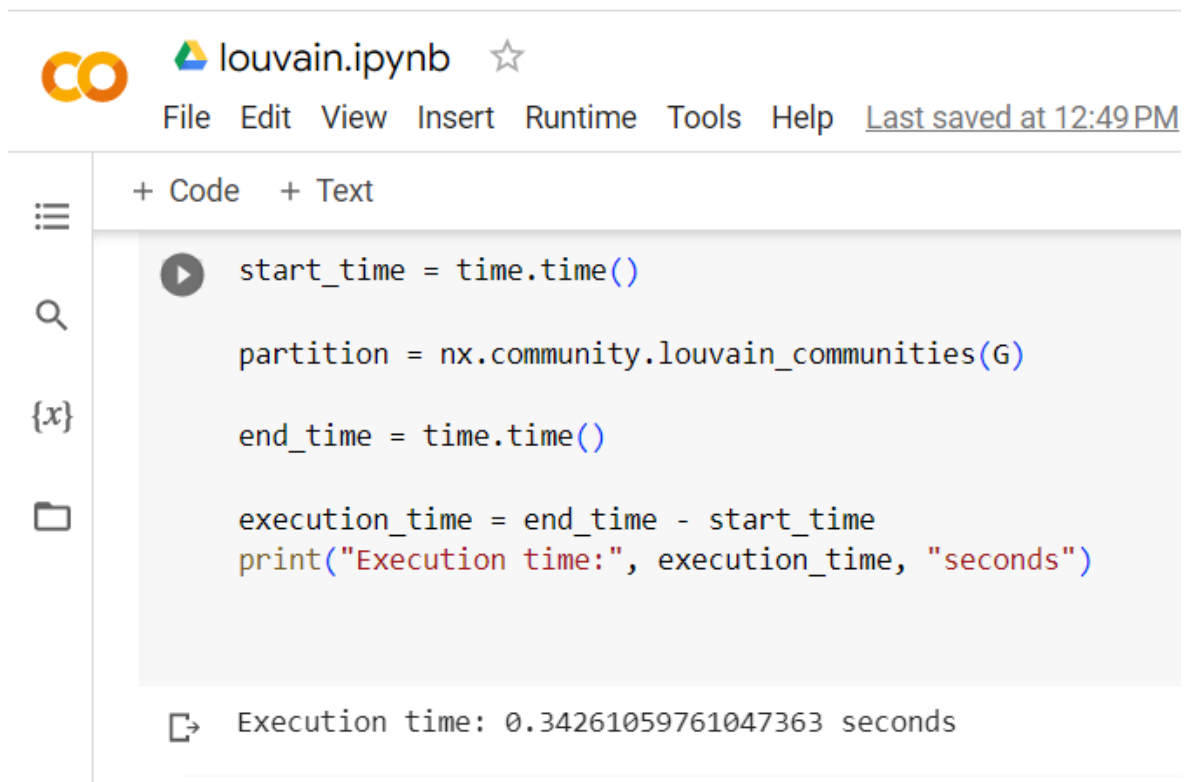


Figure 22: modularity measurement for Girvan Newman algorithms



The screenshot shows a Jupyter Notebook interface with the title 'louvain.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and 'Last saved at 12:49 PM'. The left sidebar contains icons for a menu, search, variables, and files. The main area displays a code cell with the following Python code:

```
start_time = time.time()

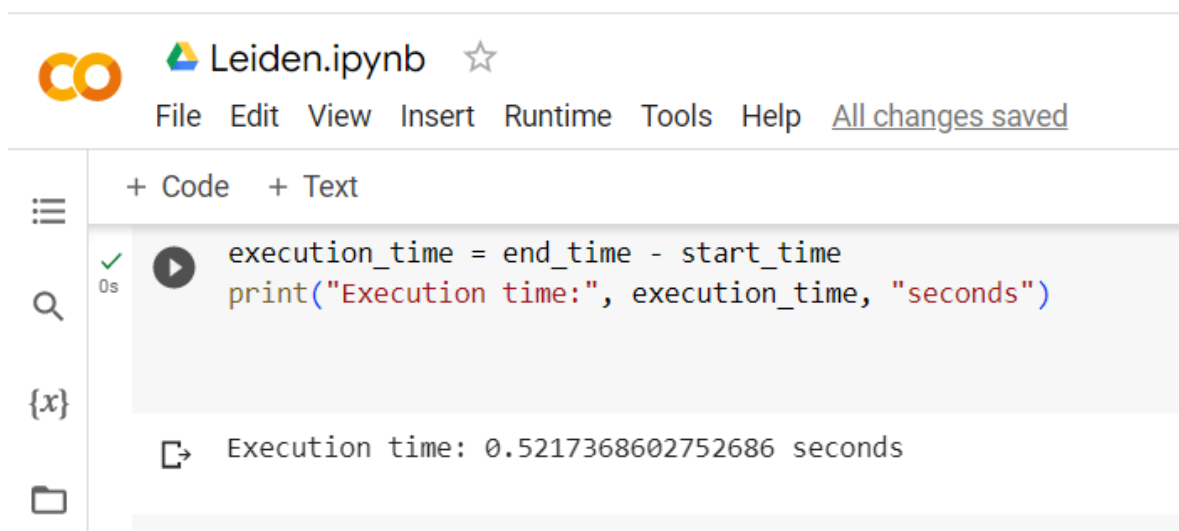
partition = nx.community.louvain_communities(G)

end_time = time.time()

execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

Below the code cell, the output is displayed: 'Execution time: 0.34261059761047363 seconds'.

Figure 23: modularity measurement for Louvain algorithms



The screenshot shows a Jupyter Notebook interface with the title 'Leiden.ipynb'. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and 'All changes saved'. The left sidebar contains icons for a menu, search, variables, and files. The main area displays a code cell with the following Python code:

```
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

Below the code cell, the output is displayed: 'Execution time: 0.5217368602752686 seconds'.

Figure 24: modularity measurement for Leiden algorithms

Leiden: The Leiden algorithm is known for its efficiency, particularly in handling large-scale networks. It utilizes techniques such as smart local moving and graph contraction to improve performance.

Louvain: The Louvain algorithm is also efficient and widely used for community detection. It employs a two-phase approach, iteratively optimizing modularity at the local and global levels. However, it can face scalability challenges

with very large networks. In this comparison, the execution times of Louvain and Leiden are similar, possibly because the Louvain algorithm has been improved with support from libraries or optimized implementations.

Girvan-Newman: The Girvan-Newman algorithm uses a divisive approach, iteratively removing edges with the highest betweenness centrality to break the graph into communities. This approach is significantly more computationally expensive compared to Louvain and Leiden (3748 seconds), especially for large networks. This is because calculating edge betweenness centrality for all edges can be time-consuming and resource-intensive.

5. Overall comparison

We have the result of each algorithms:

	Girvan Newman	Louvain	Leiden
Modularity	0.0021	0.2714	0.2750
Time(s)	3748.9411	0.3426	0.5217
Number of communities	12	21	11

In conclusion, the Louvain and Leiden algorithms generally perform well in terms of modularity score and time efficiency. They are efficient and effective for community detection in various network sizes. However, the Girvan-Newman algorithm, while effective, may not achieve as high modularity scores and much more computationally expensive. It is important to consider the specific characteristics of the network and the desired trade-offs between modularity and runtime when selecting an appropriate algorithm for community detection.

V. Conclusion

1. Achievement

The algorithms that the group used to solve the community search problem for the artist dataset above accomplished the following:

- Our team have implemented and runned 3 algorithms (Girvan Newman, Louvain, Leiden) on Python, from that, we can see some comparisions and how they works.
- Using the above data clusters, we can suggest similar books to the user while they are seeing the information of the books they want to buy, increasing the diversity of the e-commerce website such as Tiki.
- Analyzing the tastes of the readers community in specific clusters to promote better suggestions and apply them to run ads or PR is another strength that this algorithm brings.
- Besides, we also build a website to demonstrate our work and its application. The demonstration video of the website is here: <https://youtu.be/nnhuGfggjb4>

2. Drawback

Because of the limited research time, the group did not generate many algorithms or make very specific comparisons with the algorithms sought.

However, with the above three algorithms, Girvan-Newmana and Louvain, classical algorithms, and Leiden, an improved algorithm widely used today, the group has also made appropriate comparisons of performance and reliability to generalize them.

3. Future work

The team plans to use more algorithms in the future to get a thorough understanding of the approaches used by businesses and researchers to address problems.

The team also intends to work with more actual data sets in order to fully assess the algorithm' s efficacy and its potential for use in other contexts. This deals with, for instance, developing techniques for providing suggestions or advertising using these data clusters.

References.

- [1] Dataset source: <https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset>
- [2] Hastie, T., Tibshirani, R., Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science Business Media.
- [3] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.
- [4] Zhu, X., Goldberg, A. B., Nowak, R. (2009). Introduction to semi-supervised learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 3(1), 1-130.
- [5] Sutton, R. S., Barto, A. G. (2018). Reinforcement learning: An introduction. MIT Press.
- [6] Chen, H., Zhang, J., Hsu, W. (2017). Data-driven optimization for ride-sharing systems: A reinforcement learning approach. Transportation Research Part C: Emerging Technologies, 80, 48-63.
- [7] Li, Y., Zhu, J., Gong, Y. (2018). Deep reinforcement learning for person re-identification in video surveillance. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops (pp. 0-0).
- [8] Pang, C., Li, Z. (2018). Deep learning based large scale visual recommendation and search for E-commerce. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI) (pp. 0-0).
- [9] Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78-87.
- [10] Atzori, L., Iera, A., Morabito, G. (2010). The Internet of Things: A survey. Computer Networks, 54(15), 2787-2805.

- [11] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645-1660.
- [12] Elgazzar, Khalid and Khalil, Haytham and Alghamdi, Taghreed and Badr, Ahmed and Abdelkader, Ghadeer and Elewah, Abdelrahman and Buyya, Rajkumar. (2022). Revisiting the internet of things: New trends, opportunities and grand challenges
<http://dx.doi.org/10.3389/friot.2022.1073780>
- [13] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall.
- [14] Hagan, M. T., Demuth, H. B., Beale, M. H. (2014). *Neural network design*. PWS Publishing Company.
- [15] Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep learning*. MIT press.
- [16] Janiesch, C., Zschech, P. Heinrich, K. Machine learning and deep learning. *Electron Markets* 31, 685–695 (2021).
- [17] Ullah, A.; Anwar, S.M.; Li, J.; Nadeem, L.; Mahmood, T.; Rehman, A.; Saba, T. Smart cities: The role of Internet of Things and machine learning in realizing a data-centric smart environment. *Complex Intell. Syst.* 2023, 1–31
- [18] Islam, M.R.; Kabir, M.M.; Mridha, M.F.; Alfarhood, S.; Safran, M.; Che, D. Deep Learning-Based IoT System for Remote Monitoring and Early Detection of Health Issues in Real-Time. *Sensors* 2023, 23, 5204.
- [19] Elhanashi, A.; Dini, P.; Saponara, S.; Zheng, Q. Integration of Deep Learning into the IoT: A Survey of Techniques and Challenges for Real-World Applications. *Electronics* 2023, 12, 4925.
- [20] W. Yu et al., "A Survey on the Edge Computing for the Internet of Things," in *IEEE Access*, vol. 6, pp. 6900-6919, 2018, doi: 10.1109/ACCESS.2017.2778504.
- [21] F. Wang, M. Zhang, X. Wang, X. Ma and J. Liu, "Deep Learning for Edge Computing Applications: A State-of-the-Art Survey," in *IEEE Access*, vol. 8, pp. 58322-58336, 2020, doi: 10.1109/ACCESS.2020.2982411.

- [22] Rong, G., Xu, Y., Tong, X., Fan, H. (2021). An edge-cloud collaborative computing platform for building AIoT applications efficiently. *Journal of Cloud Computing*, 10(1), 1-14. <https://doi.org/10.1186/s13677-021-00250-w>
- [23] Ray, P. P. (2022). A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4), 1595-1623. <https://doi.org/10.1016/j.jksuci.2021.11.019>
- [24] Hou, K. M., Diao, X., Shi, H., Ding, H., Zhou, H., De Vault, C. (2022). Trends and Challenges in AIoT/IIoT/IoT Implementation. *Sensors*, 23(11), 5074. <https://doi.org/10.3390/s23115074>
- [25] ST. X-CUBE-AI Artificial Intelligence (AI) Software Expansion for STM32Cube; STMicroelectronics: Geneva, Switzerland, 2021; https://www.st.com/resource/en/data_brief/x-cube-ai.pdf
- [26] ST. STM32CubeMX for STM32 Configuration and Initialization C Code Generation. June 2022 UM1718 Rev 38 https://www.st.com/resource/en/data_brief/stm32cubemx.pdf
- [27] TinyML talk Felix Johnny and Fredrik Knutsson - Arm Sweden Area Group –February 8, 2021 https://cms.tinymml.org/wp-content/uploads/emea2021/tinyML_Talks_Felix_Johnny_Thomasmathibalan_and_Fredrik_Knutsson_210208.pdf