FSCO: A Secure and Adaptive Framework for Supply Chain Optimization

Tianyou Wang[1] Xing Fan[2] Wanxin Li[1*] Hao Guo[3*] Jie Zhang[1*]
[1]School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, China
[2] School of Business, Hohai University, Nanjing, China
[3]School of Software, Northwestern Polytechnical University, Xi'an, China
tianyou.wang23@student.xjtu.edu.cn, 221813060004@hhu.edu.cn
wanxin.li@xjtlu.edu.cn, haoguo@nwpu.edu.cn, jie.zhang01@xjtlu.edu.cn

2025 10th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA) | 979-8-3315-3080-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCCBDA64898.2025.11030480

***Abstract*—With the rapid growth of the e-commerce industry, the demands on logistics and transportation for timeliness and efficiency are increasing. Traditional route optimization methods may struggle with real-time traffic, configuration, and network changes, requiring more adaptive solutions. This paper proposes a novel route optimization methodology, FSCO, integrating genetic algorithms, KMeans clustering, artificial intelligence algorithms, federated learning, and blockchain technology. Genetic algorithms provide a comprehensive exploration of search spaces to identify cost-effective routes. KMeans clustering optimizes route selection by analyzing traffic data, allowing the system to adapt to real-time changes. Artificial intelligence algorithms enhance responsiveness through real-time predictions and adjustments. Federated learning enables multiple nodes to collectively optimize the dataset while preserving privacy, achieving complete data decentralization. Blockchain technology ensures data security through immutability and transparency, preventing disruptions and unauthorized manipulations. This paper details the system architecture and its operational mechanisms, highlighting key aspects and advantages, and demonstrating significant potential in addressing dynamic route adaptation, data privacy, and information security in logistics.**

***Index Terms*—genetic algorithms, KMeans clustering, federated learning, blockchain**

I. Introdction

With the advancement of society, we have entered the internet era where large B2C e-commerce platforms like Amazon and Walmart are becoming increasingly comprehensive. Events such as Amazon Prime Day and Black Friday have gained immense popularity worldwide [1]. In daily consumption, people experience the convenience and benefits of online shopping anytime, anywhere, leading to increased trust in online platforms. As a result, the scale of online shopping transactions has expanded rapidly across the globe. According to data from Statista, the global e-commerce market witnessed significant growth in 2020, with online retail sales reaching 4.28 trillion US dollars, marking a 27.6% increase from the previous year. This trend was further accelerated by the pandemic, with e-commerce's share of total global retail sales rising to 18%. By the first quarter of 2021, Amazon's net sales had surged by 44% year-over-year, reflecting the continued global growth and dominance of online shopping platforms like Amazon and Walmart.

According to the Council of Logistics Management [2], "Logistics encompasses the planning, implementation, and control of the flow and storage of goods, services, and related information from origin to consumption to meet customer requirements within the supply chain process." A critical objective in supply chain management (SCM) is to develop network flow models that not only minimize costs but also integrate sustainability and address social considerations. In SCM, the efficient determination of the shortest path between nodes is crucial across various operational scenarios. Whether it's optimizing product delivery routes between locations, solving inventory decisions within network structures, or tackling scheduling complexities, the shortest path plays a fundamental role. Beyond cost reduction, these optimized paths contribute significantly to environmental sustainability by lowering carbon dioxide [3], [4].

Utilizing multi-echelon networks to mitigate inventory shortages, the adoption of e-commerce strategies aiming to meet consumer demands for faster delivery times and expanded product choices has inadvertently reduced network resilience and increased susceptibility to disruptions driven by supply and demand fluctuations. While third-party logistics providers can efficiently support last-mile delivery operations in less complex supply chains, larger enterprises managing multi-echelon networks require sophisticated digital solutions and streamlined supply chain processes to effectively handle logistical complexities. Introduced by Accenture in the 1990s, fourth-party logistics providers enable companies to concentrate on developing value-added products by offering comprehensive logistical operations encompassing order and supplier management, as well as ensuring compliance with legal requirements [5]. Moreover, fourth-party logistics partnerships can integrate third-party logistic services strategically to optimize resource flow through efficient logistics planning and scheduling, thus minimizing costs and ensuring punctual deliveries.

Building on this foundation, this paper conceptualizes an innovative route optimization paradigm that integrates federated learning, blockchain technology, and genetic algorithms to address the limitations of conventional systems. Federated learning offers a decentralized approach to machine learning, enabling multiple logistics nodes to collaboratively develop and enhance route optimization models while keeping their data localized. This approach significantly enhances privacy

and data security. Incorporating blockchain technology adds a layer of security and transparency by establishing an immutable blockchain ledger that records all transactions and operational data within the enterprise. This ledger cannot be altered, hacked, tampered with, or accessed without authorization, ensuring data integrity and trust.

The genetic algorithm effectively navigates the complex solution space of trucking routes, providing a robust framework for exploring efficient paths that account for traffic conditions, mandated delivery times, and other logistical constraints. This combination addresses the intricate and dynamic challenges of the modern logistics market while aligning with the company's goal of making more informed decisions. This paper offers valuable insights into developing advanced logistics and transportation systems by detailing the design and implementation of this route optimization framework.

This study delves into the advanced applications of genetic algorithms, KMeans clustering, artificial intelligence algorithms, federated learning, and blockchain technology to improve delivery and supply chain processes. Our contributions focus on three primary aspects, with a particular emphasis on ensuring security and privacy in path optimization:

- We propose an integrated approach that combines federated learning and blockchain technology, driven by genetic algorithms. This model exhibits qualities such as viability, adaptability, and efficiency crucial for dynamic supply chain management. It enables rapid adjustments in real-time supply line paths to accommodate data fluctuations and unexpected events, thereby optimizing the flow of goods and communication.
- Our theoretical exploration introduces a pioneering methodology where genetic algorithms are embedded within a federated learning framework, fortified by blockchain technology to ensure security. These enhancements bolster data security and privacy, ensuring that each system interaction and transaction is securely and immutably recorded, validated, and permanently added to the blockchain. This framework promotes trust and transparency in logistics operations.
- We address contemporary flaws in logistics and supply chain management through an integrative approach. Leveraging the Hyperledger platform as a foundation for blockchain infrastructure, we conduct rigorous performance benchmarking to evaluate the model's efficiency, fault tolerance, and flexibility. These benchmarks validate the practical applicability of our approach, setting new standards for path optimization in logistics while prioritizing the highest levels of privacy protection for system users.

II. Related Work

The Dijkstra algorithm and its variants have been predominantly utilized in the literature to determine optimal routes. Originally designed to solve the shortest path problem, the Dijkstra algorithm has seen modifications to its cost function to incorporate safety in route-finding. Byon et al. [6] enhanced the Dijkstra algorithm by integrating safety parameters such as crime rate, road slope, scenic view, and ground surface elevation into the cost functions. Another method for identifying the safest route involves ranking the shortest path and its alternatives based on safety criteria [7], [8]. However, traditional Dijkstra-based methods are limited by their static nature, relying heavily on predefined parameters and struggling to adapt to dynamic environments. Building on these limitations, our framework incorporates real-time data adaptability and privacy-preserving features through federated learning and blockchain technologies.

Additionally, machine learning (ML) techniques can be leveraged not only to generate safety metrics but also to predict the safest routes. Among the ML-based approaches, deep reinforcement learning has been prominently used to determine the shortest and safest paths [9]. While ML approaches improve adaptability and prediction accuracy, they often face challenges in data privacy and scalability when applied across distributed nodes. To address these issues, we propose an integration of ML models with federated learning to ensure privacy-preserving training while enhancing global model performance.

Route-finding algorithms can be broadly categorized into reactive and predictive types based on their operational approach [10]. Reactive algorithms [10] determine paths using real-time observed data, without relying on predictions of future conditions. In contrast, predictive algorithms [11] utilize models to forecast future route conditions, allowing for more informed and anticipatory decision-making. Our approach dynamically integrates real-time updates into predictive models, ensuring both adaptability and forward-looking decision-making [12].

Route-finding algorithms can be categorized into two main types: static and dynamic, based on their ability to incorporate real-time information. Moreover, the datasets utilized by static algorithms differ from those used by dynamic algorithms. These datasets are sourced from platforms such as OpenStreetMap, Bing Maps, and Google Maps [9], [13]. In addition to geographic information, static algorithms also utilize historical data for safety assessments. For instance, studies related to crime risk in static algorithms use historical crime records. On the other hand, dynamic algorithms incorporate real-time data [14], [15]. Various sources of real-time datasets mentioned in the literature include news websites, official reports, and GPS data. Route-finding algorithms can also be classified based on their objectives. In decentralized algorithms, decision-making is handled by individual users, which allows the system to optimize for each user's specific benefits.

The logistics and transportation industry is a critical link in the economic cycle, promoting efficient integration from production to sales, and plays a decisive role in transforming economic growth models and optimizing industrial structures. The Vehicle Routing Problem (VRP), as one of the key issues in logistics system optimization, can be studied in depth to improve transportation efficiency and reduce costs effectively. VRP and its variations aim to design a route that minimizes

costs when delivering to a geographically dispersed group of customers, thereby enhancing transportation and distribution efficiency [16]. Elgharably et al. [17], through their research on the stochastic multi-objective VRP in a green environment, proposed a hybrid multi-objective search algorithm to address this problem and analyzed the potential impact of relaxing customer time windows. Hulagu et al. [18] proposed an electric vehicle routing optimization method based on a mixed-integer programming model, which considers the road network, passenger demand, and vehicle characteristics to minimize operational and charging costs. Jia et al. [19] introduced a bi-level ant colony optimization (BACO) algorithm for the capacitated electric VRP (CEVRP), which breaks down the problem into two subproblems: the capacitated VRP (CVRP) and the fixed-route vehicle charging problem (FRVCP). By coordinating the optimization of these two subproblems, the algorithm's performance in finding solutions is enhanced. Stodola et al. [20] proposed a metaheuristic algorithm based on ant colony optimization principles, incorporating techniques such as node clustering and adaptive pheromone evaporation to tackle the multi-depot VRP (MDVRP). The methods for solving VRPTW are mainly divided into traditional exact algorithms and heuristic algorithms. Exact algorithms solve VRPTW by obtaining an optimal solution through local calculations, but they are computationally complex and costly. Heuristic algorithms solve VRPTW by extensively exploring the feasible solution space to find an optimal solution, and have now become the mainstream approach for VRPTW [21], [22].Our framework addresses these gaps by integrating federated learning for decentralized data optimization, blockchain for secure and transparent data storage, and advanced genetic algorithms for efficient route selection.

III. Premilinary

A. Genetic Algorithm

Genetic algorithms (GAs) are search heuristics inspired by natural selection, used to solve optimization and search problems through evolution. Key components include:

- **Population**: A set of candidate solutions, or individuals, each encoding a potential solution to the optimization problem.
- **Fitness Function**: Evaluates each individual's quality, guiding selection. A fitness function $f(x)$ can be represented as:

$$f(x) = \text{objective function to be optimized}$$

- **Selection**: Determines which individuals contribute to the next generation, with probability expressed as:

$$P(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$$

- **Crossover**: Combines genetic material from two parents to create offspring. For one-point crossover:

$$\text{Offspring}_1 = \text{Parent}_1[0:c] + \text{Parent}_2[c:N]$$

- **Mutation**: Introduces random changes to maintain genetic diversity, defined as:

$$\text{Mutated\_Gene} = \begin{cases} \text{New\_Value} & \text{with} p \\ \text{Original\_Value} & \text{with} 1-p \end{cases}$$

- **Termination Criteria**: The algorithm stops based on criteria such as a maximum number of generations $G_{\max}$ or a satisfactory fitness level f^*.

B. KMeans Clustering

KMeans is a widely used clustering algorithm designed to partition a dataset into K clusters, where each data point is assigned to the cluster with the nearest mean. The algorithm is characterized by the following steps:

1) **Initialization**: Randomly select K initial cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_K\}$.
2) **Assignment**: For each data point x_i, assign it to the nearest cluster centroid μ_j:

$$C(i) = \arg\min_j \|x_i - \mu_j\|^2$$

where $C(i)$ is the cluster index assigned to point x_i and $\|\cdot\|$ denotes the Euclidean distance.
3) **Update**: Calculate the new centroids as the mean of the data points in each cluster:

$$\mu_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

where S_j is the set of points assigned to cluster j.
4) **Repeat**: Continue repeating the assignment and update steps until the centroids stabilize, i.e., the change in centroids is less than a threshold ϵ:

$$\|\mu_j^{(new)} - \mu_j^{(old)}\| < \epsilon$$

C. Neural Network

Neural networks are a class of machine-learning models inspired by the human brain. They consist of layers of neurons that process input features and learn complex patterns through training.

The structure of a neural network typically includes:

1) **Input Layer**: Receives the input features $\mathbf{x} = [x_1, x_2, \ldots, x_n]$.
2) **Hidden Layers**: Intermediate layers that transform the input features through learned weights $\mathbf{W}$ and activation functions f. The output of a hidden neuron h_j can be expressed as:

$$h_j = f\left(\sum_i w_{ji} x_i + b_j\right)$$

where w_{ji} is the weight from input i to neuron j, and b_j is the bias term.

3) **Output Layer**: Produces the final output, such as a classification or regression result. For a single output neuron, this can be expressed as:

$$y = f\left(\sum_j w_{oj} h_j + b_o\right)$$

where w_{oj} are the weights from the hidden layer to the output neuron, and b_o is the output bias.

D. Federated Learning

Federated learning is a distributed machine learning approach that allows multiple clients to collaboratively train a model while keeping their data decentralized.

The process includes:

1) **Local Training**: Each client i trains a local model M_i on its own data D_i:

$$M_i \leftarrow \text{Train}(M_i, D_i)$$

2) **Model Aggregation**: The updates from the local models, denoted as ΔM_i, are sent to a central server. The server aggregates these updates to create the updated global model M:

$$M \leftarrow M + \frac{1}{N}\sum_{i=1}^{N} \Delta M_i$$

where N is the number of clients.

3) **Global Model Update**: The updated global model M is sent back to the clients for the next round of training:

$$M \leftarrow \text{Update}(M)$$

E. Blockchain

Blockchain is a decentralized ledger technology that ensures data integrity and security through a distributed network of nodes.

The key components include:

1) **Blocks**: Data records grouped in blocks.
2) **Chain**: Blocks are linked together in a chronological chain using cryptographic hashes.
3) **Consensus Mechanisms**: Methods like Proof of Work (PoW) or Proof of Stake (PoS) that ensure agreement on the blockchain state across the network.

Blockchain provides transparency, immutability, and security, making it suitable for applications requiring robust data integrity.

F. Dynamic Simulation

Dynamic simulation involves adjusting system parameters or behaviors in response to real-time environmental changes, such as traffic flow.

The process includes:

1) **Real-time Simulation**: Use a scheduler to dynamically simulate changes in path weights to reflect real-time factors.
2) **Fitness Score Modification**: Periodically modify fitness scores in a CSV file using the *modify_fitness_score()* function.
3) **Process Initiation**: Regularly start federated learning and optimization processes using the *start_process()* function.

IV. FSCO Design

The Fig. 1 illustrates the FSCO system's overall design, integrating genetic algorithms, KMeans clustering, neural networks, and federated learning. Each client independently processes data using genetic algorithms and KMeans clustering to identify optimal paths. A neural network calculates the fitness scores, and results are combined via federated learning, ensuring data privacy. Dynamic simulation keeps the system adaptive to real-world changes. Additionally, blockchain technology secures the integrity of the optimized paths, facilitating transparent auditing, smart contract execution, and ensuring accountability and efficiency in supply chain management.

The FSCO system integrates several advanced technologies to optimize supply chain management. First, genetic algorithms in Subsec. IV-A are employed at each client to find the most suitable paths within their respective datasets. These paths are then clustered using KMeans clustering in Subsec. IV-B, enhancing the management and optimization of the routes.

Subsequently, the paths' fitness scores are calculated using a pretrained neural network model in Subsec. IV-C, ensuring each path's efficacy. The results of multiple rounds of genetic algorithm searches are aggregated into a global model through federated learning in Subsec. IV-D, which allows the system to incorporate optimized routes from all clients while preserving data privacy.

To mimic real-world dynamics, the system periodically updates the fitness scores of each client and adjusts the paths in real time, using dynamic simulation techniques in Subsec. IV-E. This ongoing process ensures that the FSCO system adapts to data fluctuations and unexpected events,

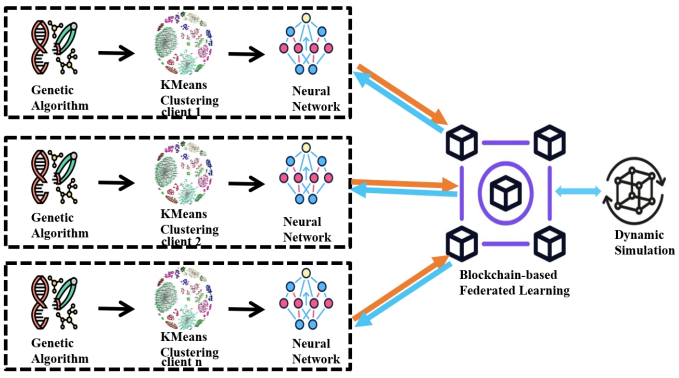

Fig. 1. The optimal paths derived from the genetic algorithm are clustered by KMeans clustering, and then scores are computed with a pre-trained neural network, which are aggregated to the global model by federated learning.

maintaining optimal performance and decision-making capabilities securely and efficiently.

The FSCO system incorporates blockchain technology in Subsec. IV-F to securely store optimized paths identified through genetic algorithms and KMeans clustering. By leveraging blockchain, the system ensures the integrity and transparency of route data, enhancing trust and reliability across the supply chain. This approach enables immutable records of route optimizations, facilitating transparent auditing and automated smart contract execution for performance-based incentives, thereby enhancing operational efficiency and accountability in supply chain management.

As shown in Fig. 2, the workflow begins with each client independently processing its data using Genetic Algorithms (①) and K-Means clustering (②) to identify optimal paths within their datasets. Following this, a Neural Network (③) calculates fitness scores for these paths, ensuring effective solution selection. These evaluations lead to an Optimized Path (④), representing refined solutions. Blockchain (⑤) is then employed to secure the integrity and transparency of the optimization process, enhancing trust across the system. Federated Learning (⑥) aggregates the results into a global model, preserving data privacy while consolidating insights from all clients. Finally, Dynamic Simulation (⑦) enables real-time adaptation to environmental changes, ensuring the system remains responsive and reliable under fluctuating conditions.

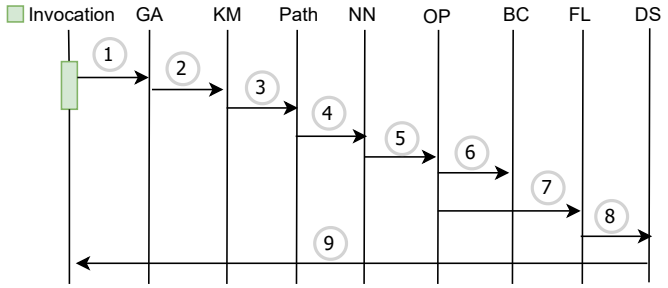

Fig. 2. GA - Genetic Algorithm, KM - K-Means, NN - Neural Network, OP - optimized path, NN - Neural Network, FL - Federated Learning, and DS - Dynamic Simulation.

A. Genetic Algorithm (Algo. 1 and Algo. 2)

Lines 1-2: These variables set up the foundational settings for the genetic algorithm, including the total population gene limits, gene range, genes per individual, population size, termination condition, and scaling factor. Lines 3-11: This procedure initializes the population by randomly selecting genes from `generange` to form individuals. It randomly samples `total_numbers` genes from `generange` to create `pop_size` individuals and returns the population (list of individuals) and `humanGene` (list of genes). Lines 12-15: Calculates the fitness of an individual based on its path optimization quality. The fitness evaluation considers multiple real-world factors such as `distance`, `cost`, and `time`, which are crucial for practical path optimization problems.

Algorithm 1 Genetic Algorithm for Path Optimization

```
1: Parameters:
2:     total_numbers, generange, selected_numbers, pop_size, terminder, N
3: procedure GENERATE_POPULATION
4:     humanGene ← random.sample(range(generange), total_numbers)
5:     population ← ∅
6:     for each individual in pop_size do
7:         individual ← random.sample(humanGene, selected_numbers)
8:         population.append(individual)
9:     end for
10:    return population, humanGene
11: end procedure
12: procedure FITNESS(individual, humanGene)
13:    distance, cost, time, penalty ← calculate_metrics(individual)
14:    return 1/(w1 · distance + w2 · cost + w3 · time + penalty)
15: end procedure
16: procedure SELECTION(population, fitness_values)
17:    return random.choices(population, fitness_values, total_numbers)
18: end procedure
19: procedure GENETIC_ALGORITHM
20:    population, humanGene ← GENERATE_POPULATION
21:    while termination condition not met do
22:        fitness_values ← [FITNESS(ind, humanGene) for ind in population]
23:        selected_population ← SELECTION(population, fitness_values)
24:        next_generation ← ∅
25:        for each pair in selected_population do
26:            child1, child2 ← ENHANCED_CROSSOVER(pair[0], pair[1])
27:            GUIDED_MUTATE(child1, humanGene)
28:            GUIDED_MUTATE(child2, humanGene)
29:            next_generation.extend([child1, child2])
30:        end for
31:        population ← next_generation
32:    end while
33:    return population
34: end procedure
```

A penalty term is introduced to handle undesirable solutions, making the algorithm more robust and adaptive. Lines 17-18: Selects individuals from the population based on their fitness values. It uses `random.choices` to probabilistically select individuals, favoring those with higher fitness values. Lines 19-34: The main procedure of the genetic algorithm. It iteratively generates improved populations. In each iteration, it calculates fitness values, selects individuals, and produces offspring using enhanced crossover and guided mutation operations. Finally, it returns the optimized population.

As shown in Algo. 2, the `enhanced_crossover` procedure uses a uniform crossover mechanism. For each gene, it selects the genetic material from either parent with a 50% probability, ensuring better diversity in the offspring (Lines 1-13). Lines 14-22: The `guided_mutate` procedure introduces an adaptive mutation rate to prevent premature convergence. The mutation operation is guided by a heuristic that incorporates knowledge of the fitness function, such as optimizing `distance`, `cost`, or `time`, to produce beneficial changes in the individual.

B. KMeans Clustering (Algo. 3)

The KMeans clustering algorithm described in Algo. 3 groups paths into K clusters for path optimization. The algorithm starts with the initialization of input parameters, including K, the number of clusters, and `max_iterations`, the maximum number of iterations allowed for convergence

Algorithm 2 Enhanced Crossover and Mutation Operations

```
1: procedure ENHANCED_CROSSOVER(parent1, parent2)
2:     child1, child2 ← ∅, ∅
3:     for each gene index i in parent1 do
4:         if random.random() < 0.5 then
5:             child1[i] ← parent1[i]
6:             child2[i] ← parent2[i]
7:         else
8:             child1[i] ← parent2[i]
9:             child2[i] ← parent1[i]
10:        end if
11:    end for
12:    return child1, child2
13: end procedure
14: procedure GUIDED_MUTATE(individual, humanGene)
15:    mutation_rate ← adaptive_mutation_rate(population) ▷ Adaptive rate
16:    for each gene in individual do
17:        if random.random() < mutation_rate then
18:            gene ← heuristic_mutation(gene, humanGene)
19:        end if
20:    end for
21:    return individual
22: end procedure
```

Algorithm 3 KMeans Clustering for Path Optimization

```
1: Parameters:
2:    K                                   ▷ {Number of clusters}
3:    max_iterations                      ▷ {Maximum number of iterations}
4: procedure KMEANS_CLUSTERING(paths, K)  ▷ {Cluster the paths into K clusters}
5:    centroids ← initialize_centroids(paths, K) ▷ {Initialize K centroids}
6:    for iteration in max_iterations do
7:        clusters ← array of empty lists          ▷ {Initialize clusters}
8:        for each path in paths do
9:            nearest_centroid ← find_nearest_centroid(path, centroids) ▷ {Find the nearest centroid for each path}
10:           clusters[nearest_centroid].append(path) ▷ {Assign path to the nearest centroid's cluster}
11:       end for
12:       new_centroids ← update_centroids(clusters) ▷ {Update centroids based on the mean of assigned paths}
13:       if convergence_reached(centroids, new_centroids) then ▷ {Check if centroids have converged}
14:           break
15:       end if
16:       centroids ← new_centroids   ▷ {Set centroids to new centroids}
17:   end for
18:   return clusters, centroids
19: end procedure
```

(Lines 1-3). The procedure `KMeans_clustering` is defined to take the list of paths and the number of clusters K as inputs (Line 4). Initial centroids are determined using the `initialize_centroids` function, which randomly selects K paths from the dataset (Line 5).The main iterative process begins in Line 6 and continues until either convergence is achieved or the maximum number of iterations is reached. At the start of each iteration, an empty array of lists, `clusters`, is initialized to store paths assigned to each cluster (Line 7). For each path, the algorithm identifies the nearest centroid using the `find_nearest_centroid` function (Line 9) and assigns the path to the corresponding cluster (Line 10). After all paths are assigned, the centroids are updated based on the mean of the paths in each cluster using the `update_centroids` function (Line 12). The algorithm then checks for convergence using the `convergence_reached` function, which compares the old centroids with the updated centroids. If convergence is detected, the loop terminates early (Lines 13-15). If not, the centroids are updated to the newly calculated values for the next iteration (Line 16).

C. Genetic Algorithm with Neural Network for Fitness Score Calculation (Algo. 4)

Lines 1 to 5: These lines define the parameters required for setting up the neural network model: input_size, hidden_sizes, output_size, and dropout_rate. These parameters determine the architecture and behavior of the neural network. Lines 6 to 19: The DeepPathNet procedure constructs a deep neural network model tailored for path cost prediction. It initializes a sequential stack of layers comprising linear transformations, rectified linear unit (ReLU) activations for non-linearity, and dropout layers to prevent overfitting. The model architecture is defined based on the provided parameters input_size, hidden_sizes, output_size, and dropout_rate. Lines 20 to 29: The train_on_client procedure iterates through a specified number of epochs to train the neural network model on given data and targets. It performs forward propagation to generate predictions, computes the loss between predicted and actual targets using a specified criterion, computes gradients using backpropagation, and updates the model parameters using the optimizer. Lines 30 to 34: The compute_fitness calculates the fitness of a path using the trained neural network model. It extracts relevant features from the path, predicts the path's cost using the neural network model, and computes the fitness score as $1/(predicted_cost + 1)$. This fitness score quantifies the path's suitability based on predicted cost. Lines 35 to 45: The integrate_with_GA_and_KMeans integrates the genetic algorithm (GA), KMeans clustering, and the neural network model for path optimization. It initializes paths and genetic material using GA, clusters paths into groups using KMeans, trains separate neural network models on each cluster to predict path costs, and updates fitness scores based on the predicted costs. This integration optimizes paths across multiple clients' datasets, reflecting dynamic changes over time.

D. Federated Learning for FSCO (Algo. 5)

Line 1-6: The algorithm initializes parameters necessary for federated learning, specifying details like the list of clients with their datasets, the global neural network model to be updated, the number of rounds for federated learning, the number of training epochs per round, the optimizer used for model training, and the loss criterion for model training. Line 8-20: Within each round of federated learning, every client independently trains a local copy of a shared neural network model using its own data. After training, each client evaluates the trained model's performance on its respective dataset, updates its internal evaluation score, and then communicates its updated model parameters to a central server or aggregator. Line 21-30: The compute_fitness function computes fitness scores for each path in a given set of paths using a trained

Algorithm 4 Neural Network Model for Path Cost Prediction

```
1: Parameters:
2:     input_size                          ▷ {Number of input features}
3:     hidden_sizes                        ▷ {List of neurons in each hidden layer}
4:     output_size                         ▷ {Number of output features}
5:     dropout_rate                        ▷ {Dropout rate for preventing overfitting}
6: procedure DEEPPATHNET(input_size, hidden_sizes, output_size, dropout_rate)
7:     layers ← ∅
8:     layers.append(Linear(input_size, hidden_sizes[0]))
9:     layers.append(ReLU())
10:    for each hidden_size in hidden_sizes[1:] do
11:        layers.append(Linear(previous_hidden_size, hidden_size))
12:        layers.append(ReLU())
13:        layers.append(Dropout(dropout_rate))
14:        previous_hidden_size ← hidden_size
15:    end for
16:    layers.append(Linear(hidden_sizes[-1], output_size))
17:    model ← Sequential(layers)
18:    return model
19: end procedure
20: procedure TRAIN_ON_CLIENT(model, data, targets, optimizer, criterion, epochs)
21:    for each epoch in epochs do
22:        optimizer.zero_grad()
23:        outputs ← model(data)
24:        loss ← criterion(outputs, targets)
25:        loss.backward()
26:        optimizer.step()
27:    end for
28:    return model
29: end procedure
30: procedure COMPUTE_FITNESS(path, model)
31:    input_features ← extract_features(path) ▷ {Extract relevant features}
32:    predicted_cost ← model(input_features)  ▷ {Predict cost using the model}
33:    return 1 / (predicted_cost + 1)         ▷ {Calculate fitness}
34: end procedure
35: procedure INTEGRATE_WITH_GA_AND_KMEANS
36:    paths, humanGene ← GA.generate_initial_population()
37:    clusters, centroids ← KMeans_clustering(paths, K)
38:    for each cluster in clusters do
39:        model ← DeepPathNet(input_size, hidden_sizes, output_size, dropout_rate)
40:        cluster_data ← concatenate_paths(cluster)
41:        cluster_targets ← compute_targets(cluster)
42:        trained_model ← train_on_client(model, cluster_data, cluster_targets, optimizer, criterion, epochs)
43:        update_fitness_scores(cluster, trained_model)
44:    end for
45: end procedure
```

Algorithm 5 Federated Learning

```
1: Parameters:
2:     clients                 ▷ List of clients with their respective datasets
3:     global_model            ▷ Global neural network model to be updated
4:     rounds                  ▷ Number of federated learning rounds
5:     epochs_per_round        ▷ Number of training epochs per round
6:     optimizer               ▷ Optimizer for model training
7:     criterion               ▷ Loss criterion for model training
8: procedure FEDERATED_LEARNING(clients, global_model, rounds, epochs_per_round, optimizer, criterion)
9:     for round ← 1 to rounds do
10:        for each client ∈ clients do
11:            local_model ← clone(global_model)
12:            data, targets ← client.get_data()
13:            trained_model ← train_on_client(local_model, data, targets, optimizer, criterion, epochs_per_round)
14:            client_fitness ← compute_fitness(client.paths, trained_model)
15:            client.update_fitness_score(client_fitness)
16:        end for
17:        global_model ← aggregate_models(clients, global_model)
18:    end for
19:    return global_model
20: end procedure
21: procedure COMPUTE_FITNESS(paths, model)
22:    fitness_scores ← []
23:    for each path ∈ paths do
24:        input_features ← extract_features(path)
25:        predicted_cost ← model(input_features)
26:        fitness_score ← 1 / (predicted_cost + 1)
27:        fitness_scores.append(fitness_score)
28:    end for
29:    return fitness_scores
30: end procedure
31: procedure AGGREGATE_MODELS(clients, global_model)
32:    aggregated_model ← clone(global_model)
33:    for each parameter in global_model.parameters do
34:        parameter_sum ← 0
35:        for each client ∈ clients do
36:            parameter_sum ← parameter_sum + client.model.parameter
37:        end for
38:        aggregated_model.parameter ← parameter_sum / len(clients)
39:    end for
40:    return aggregated_model
41: end procedure
```

model. This score is derived from the model's predictions on extracted features from each path, calculated as the inverse of the predicted cost plus one. Line 31-41: The aggregate_models function consolidates models from all participating clients to update a global neural network model. It computes new model parameters by averaging the parameters of all client models, ensuring that each client's contribution is weighted equally in the model aggregation process.

E. Dynamic Simulation with Federated Learning (Algo. 6)

Lines 1-5: The algorithm initializes a dynamic simulation process for federated learning clients. Parameters are defined: clients: List of clients participating in federated learning. simulation_duration: Total duration of the dynamic simulation in time steps. Lines 6-19: dynamic_simulation procedure: It simulates the dynamic behavior where each client updates its fitness score periodically based on a predefined interval (update_interval). During each update, clients adjust their fitness scores using the modify_fitness_score function. The main loop runs for the specified duration of the simulation (simulation_duration), during which clients' fitness scores are updated dynamically.

F. Blockchain Storage (Algo. 7)

Lines 1-8: Set the format and define the parameters section. List the parameters passed to the algorithm, including clients, global model, number of federated learning rounds, epochs per round, optimizer, criterion, and blockchain object. Lines 9-27: Perform federated learning for a set number of rounds. Lines 11-17: For each client, clone the global model, get client data, train the local model, compute fitness scores, and update the client's fitness score. Line 18: Aggregate the models to update the global model. Lines 20-25: After all training rounds, for each client, get the best path and its fitness score, get the current timestamp, and store this data in the blockchain. Line 26: Return the global model. Lines 28-31: Define the

Algorithm 6 Dynamic Simulation

```
Parameters:
    clients                    ▷ List of clients with their respective datasets
    simulation_duration        ▷ Duration of the simulation (in time steps)
    update_interval            ▷ Interval for client data and fitness score updates
    scheduler                  ▷ Scheduler for triggering client updates and model aggregation
procedure DYNAMIC_SIMULATION(clients, simulation_duration, update_interval, scheduler)
    time ← 0
    while time < simulation_duration do
        for each client ∈ clients do
            if time % update_interval == 0 then
                client.update_data()          ▷ Update client's local dataset
                client.train_local_model()    ▷ Train client's local model
                client.compute_fitness()      ▷ Compute client's fitness score
            end if
        end for
        scheduler.optimize_global_model()     ▷ Aggregate models and optimize global model
        time ← time + 1
    end while
end procedure
```

Algorithm 7 Store Optimized Paths in Blockchain

```
Parameters:
    clients                    ▷ List of clients with their respective datasets
    global_model               ▷ Global neural network model to be updated
    rounds                     ▷ Number of federated learning rounds
    epochs_per_round           ▷ Number of training epochs per round
    optimizer                  ▷ Optimizer for model training
    criterion                  ▷ Loss criterion for model training
    blockchain                 ▷ Blockchain object to store data
procedure OPTIMIZE_AND_STORE_PATHS(clients, global_model, rounds, epochs_per_round, optimizer, criterion, blockchain)
    for round ← 1 to rounds do
        for each client ∈ clients do
            local_model ← clone(global_model)
            data, targets ← client.get_data()
            trained_model ← train_on_client(local_model, data, targets, optimizer, criterion, epochs_per_round)
            client_fitness ← compute_fitness(client.paths, trained_model)
            client.update_fitness_score(client_fitness)
        end for
        global_model ← aggregate_models(clients, global_model)
    end for
    for each client ∈ clients do
        best_path ← client.get_best_path()
        fitness_score ← client.get_best_fitness_score()
        timestamp ← get_current_timestamp()
        store_path_in_blockchain(best_path, fitness_score, timestamp, blockchain)
    end for
    return global_model
end procedure
procedure STORE_PATH_IN_BLOCKCHAIN(path, fitness_score, timestamp, blockchain)
    data ← { "path": path, "fitness_score": fitness_score, "timestamp": timestamp }
    blockchain.add_block(data)
end procedure
```

store_path_in_blockchain procedure to add the best path data to the blockchain.

V. Evaluation

We evaluate the protocols on the machine with 16 vCPUs and 16GB memory, running Ubuntu 20.04. The entire protocol is implemented with the Python 3.11 version. We use the Pandas library [23] for reading and manipulating CSV files. Pandas is widely used in data analysis and manipulation tasks. We employ the Scikit-learn library [24] for K-Means clustering. Scikit-learn provides a range of machine learning algorithms for data analysis. For scheduling and executing tasks at regular intervals, we use the APScheduler library [25], particularly the 'apscheduler.schedulers.blocking.BlockingScheduler' module. APScheduler is useful for task scheduling in Python applications. For blockchain simulation, we use the Hyperledger Fabric v2.3 [26]. Hyperledger Fabric is an enterprise-grade, permissioned distributed ledger platform. Its modular architecture enables a high degree of flexibility, performance, and security, making it well-suited for our federated learning environment [27]. The evaluation result is shown in Fig. 3.

Population Size: When the population size increased from 500 to 1500, the best fitness score also gradually increased, indicating that a larger population size is beneficial for finding better solutions. However, a larger population size may lead to longer algorithm run times, necessitating a balance in practical applications. By increasing the population size, we allow the algorithm to explore a more extensive search space, enhancing its capability to avoid premature convergence and increasing the likelihood of discovering optimal solutions. It is essential to note that while a larger population can provide better results, it also requires more computational resources and time, making it crucial to find an optimal balance based on the specific problem and available resources.

Number of Training Epochs: When the number of training epochs increased from 100 to 400, there was a significant improvement in the best fitness score. This indicates that more training epochs allow the neural network model to learn more thoroughly, leading to more accurate predictions of path costs, which in turn helps the genetic algorithm find better solutions. By providing the model with more opportunities to refine its weights, we enhance its generalization capabilities, thereby improving the overall effectiveness of the genetic algorithm. However, it's important to monitor for potential overfitting, where too many epochs might cause the model to perform well on training data but poorly on unseen data.

Number of Generations: In the experiments, when the number of generations increased from 40 to 60 or 80, the best fitness score also improved. This suggests that more generations of natural selection, inheritance, and mutation enhance the optimization effect of the genetic algorithm. Each additional generation provides more opportunities for the algorithm to fine-tune solutions and converge towards optimal results. However, beyond a certain point, the marginal gains from additional generations may diminish, and computational costs will increase, so it's important to determine an optimal number of generations that balances improvement with efficiency.

Mutation Rate: In the experimental results, as the mutation rate increased, the best fitness score also increased. This indicates that higher fitness mutations are beneficial for finding better solutions, increasing population diversity, and helping the genetic algorithm escape local optima to

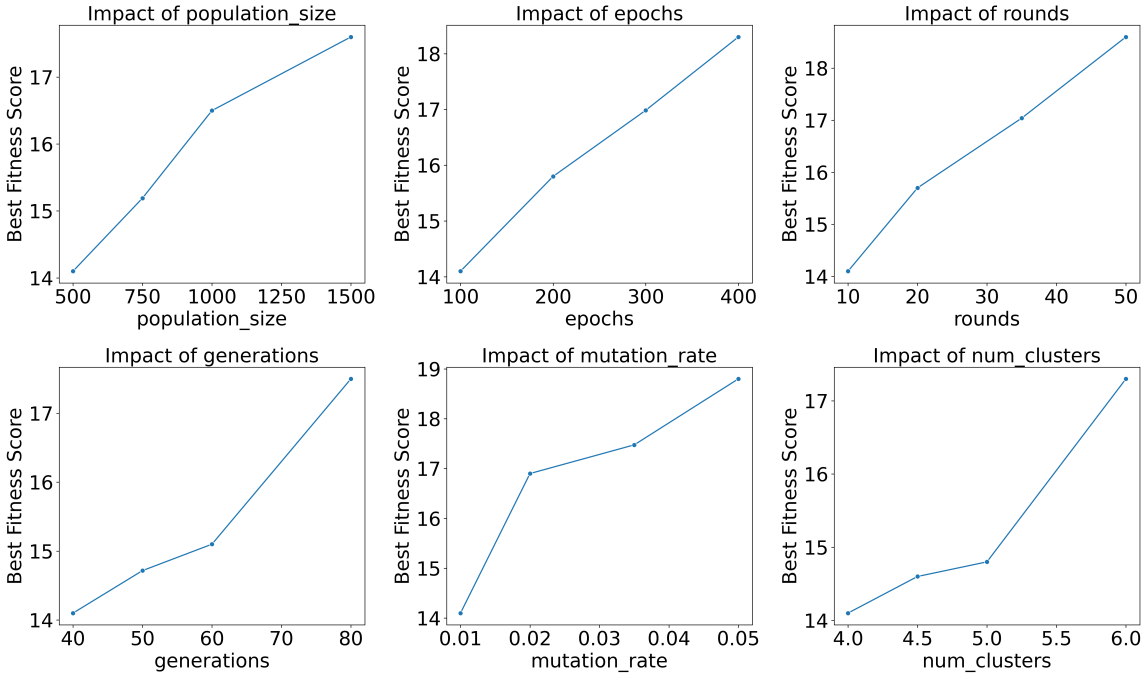

Fig. 3. FSCO evaluation result

find global optima. Nevertheless, too high a mutation rate can affect the stability of the algorithm, leading to non-convergence of results. A moderate mutation rate is crucial as it introduces variability without disrupting the overall search process, allowing the algorithm to explore new solutions while maintaining a core of high-fitness individuals.

Number of Clusters: When the number of clusters (num_clusters) increased, the best fitness score first showed an increasing trend. This indicates that the number of clusters affects the search process of the genetic algorithm, and an optimal number of clusters enables the algorithm to find better results. The clustering approach influences how solutions are grouped and evaluated, affecting the genetic algorithm's ability to explore and exploit the search space effectively. An appropriate clustering number strikes a balance between diversification and convergence, facilitating the discovery of high-quality solutions without excessive fragmentation or stagnation.

Blockchain Write and Read Phase: As shown in Fig. 4, when the data size is 250 bytes, the average latency for writing data on the chain is 2.0606 s; read on the chain is 0.926s.

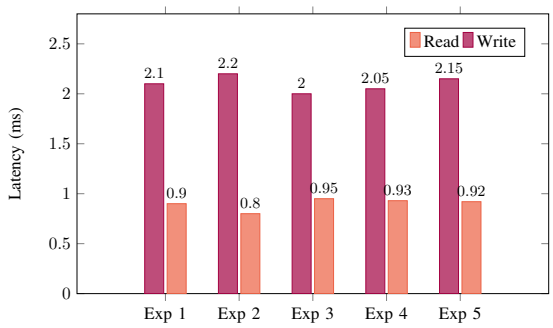

Fig. 4. Blockchain write and read evaluations with 5 experiments. (The data size is 250 bytes.)

VI. Conclusion

This paper introduces an integrated approach combining federated learning and blockchain technology, driven by genetic algorithms, to enhance dynamic supply chain management. Our model, characterized by its viability, adaptability, and efficiency, facilitates real-time adjustments in supply line paths, optimizing the flow of goods and communication. We ensure robust data security and privacy by embedding genetic algorithms within a federated learning framework and utilizing blockchain technology, promoting trust and transparency. Leveraging the Hyperledger platform, our rigorous performance benchmarks validate the model's efficiency, fault tolerance, and flexibility, setting new standards for logistics optimization while prioritizing privacy protection. This approach addresses contemporary flaws in supply chain management, offering a secure, efficient, and adaptable solution.

References

[1] F. Ali. (2021) Global e-commerce sales. Accessed: September 20, 2024. [Online]. Available: https://www.digitalcommerce360.com/article/global-ecommerce-sales/

[2] S. CSCMP, "Council of supply chain management professionals," *Retrieved from*, 2014.

[3] P. Ghadimi, C. Wang, and M. K. Lim, "Sustainable supply chain modeling and analysis: Past debate, present problems and future challenges," *Resources, conservation and recycling*, vol. 140, pp. 72–84, 2019.

[4] D. Yadav, R. Kumari, N. Kumar, and B. Sarkar, "Reduction of waste and carbon emission through the selection of items with cross-price elasticity of demand to form a sustainable supply chain with preservation technology," *Journal of Cleaner Production*, vol. 297, p. 126298, 2021.
[5] H.-J. Schramm, C. N. Czaja, M. Dittrich, and M. Mentschel, "Current advancements of and future developments for fourth party logistics in a digital future," *Logistics*, vol. 3, no. 1, p. 7, 2019.
[6] Y.-J. Byon, B. Abdulhai, and A. Shalaby, "Incorporating scenic view, slope, and crime rate into route choices: Emphasis on three-dimensional geographic information systems with digital elevation models and crime rate geospatial data," *Transportation research record*, vol. 2183, no. 1, pp. 94–102, 2010.
[7] S. Ito and Z. Koji, "Assessing a risk-avoidance navigation system based on localized torrential rain data," in *MATEC Web of Conferences*, vol. 308. EDP Sciences, 2020, p. 03006.
[8] N. Hoseinzadeh, R. Arvin, A. J. Khattak, and L. D. Han, "Integrating safety and mobility for pathfinding using big data generated by connected vehicles," *Journal of Intelligent Transportation Systems*, vol. 24, no. 4, pp. 404–420, 2020.
[9] S. Levy, W. Xiong, E. Belding, and W. Y. Wang, "Saferoute: Learning to navigate streets safely in an urban environment," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 6, pp. 1–17, 2020.
[10] E. Schmitt and H. Jula, "Vehicle route guidance systems: Classification and comparison," in *2006 IEEE Intelligent Transportation Systems Conference*. IEEE, 2006, pp. 242–247.
[11] E. Galbrun, K. Pelechrinis, and E. Terzi, "Urban navigation beyond shortest route: The case of safe paths," *Information Systems*, vol. 57, pp. 160–171, 2016.
[12] C. Meese, H. Chen, W. Li, D. Lee, H. Guo, C.-C. Shen, and M. Nejad, "Adaptive traffic prediction at the its edge with online models and blockchain-based federated learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 9, pp. 10 725–10 740, 2024.
[13] J. M. Lozano Domínguez and T. d. J. Mateo Sanguino, "Walking secure: Safe routing planning algorithm and pedestrian's crossing intention detector based on fuzzy logic app," *Sensors*, vol. 21, no. 2, p. 529, 2021.
[14] R. Kaur, V. Goyal, V. M. Guntur, A. Saini, K. Sanadhya, R. Gupta, and S. Ratra, "A navigation system for safe routing," in *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2021, pp. 240–243.
[15] T. Santhanavanich, P. Wuerstle, J. Silberer, V. Loidl, P. Rodrigues, and V. Coors, "3d safe routing navigation application for pedestrians and cyclists based on open source tools," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 6, pp. 143–147, 2020.
[16] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.
[17] N. Elgharably, S. Easa, A. Nassef, and A. El Damatty, "Stochastic multi-objective vehicle routing model in green environment with customer satisfaction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 1, pp. 1337–1355, 2022.
[18] S. Hulagu and H. B. Celikoglu, "An electric vehicle routing problem with intermediate nodes for shuttle fleets," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 1223–1235, 2020.
[19] Y.-H. Jia, Y. Mei, and M. Zhang, "A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 855–10 868, 2021.
[20] P. Stodola and J. Nohel, "Adaptive ant colony optimization with node clustering for the multidepot vehicle routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1866–1880, 2022.
[21] T. Kinoshita and T. Uchiya, "Diversity maintenance method using multiple crossover in genetic algorithm for vrptw," in *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*. IEEE, 2021, pp. 563–565.
[22] J. Duan, Z. He, and G. G. Yen, "Robust multiobjective optimization for vehicle routing problem with time windows," *IEEE Transactions on Cybernetics*, vol. 52, no. 8, pp. 8300–8314, 2021.
[23] T. P. D. Team, "Pandas: Python data analysis library," 2023. [Online]. Available: https://pandas.pydata.org/
[24] T. S. learn Development Team, "Scikit-learn: Machine learning in python," 2023. [Online]. Available: https://scikit-learn.org/
[25] T. A. D. Team, "Apscheduler: Advanced python scheduler," 2023. [Online]. Available: https://apscheduler.readthedocs.io/
[26] T. H. F. D. Team, "Hyperledger fabric v2.3: An enterprise-grade permissioned distributed ledger platform," 2023. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.3/
[27] H. Guo, C. Meese, W. Li, C.-C. Shen, and M. Nejad, "B2sfl: A bi-level blockchained architecture for secure federated learning-based traffic prediction," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 4360–4374, 2023.