Collaborative Vehicular Edge Computing Design for Delay-Sensitive Applications

Jing-Yang Voon, Yao Chiang, Cheng-Rui Jia, Hung-Yu Wei
Department of Electrical Engineering, National Taiwan University, Taiwan

***Abstract*—Vehicular edge computing (VEC) has become a promising solution in electric vehicle (EV) utilization. However, the uneven geographical distribution of service requests may lead to load imbalances among edge servers in different clusters. Thus, the integration of task offloading (TO) and resource allocation (RA) is pivotal for achieving optimal performance in edge computing systems. In this study, we explore an efficient collaborative scheme for task offloading and resource allocation across multiple edge network areas. Initially, we model the Multi-Edge System Delay (MESD) by considering the average end-to-end delay in the system. Subsequently, we introduce the concept of request redistribution using a load-balancing approach to simplify the joint TO & RA problem into a manageable RA problem. Our algorithm mathematically formulates the MESD model and employs a heuristic method to address the formulated problem. Finally, we have compared our proposed work with several baselines and the results confirm the effectiveness of the proposed mechanism.**

***Index Terms*—Edge Computing, Computational Offloading, Multi-Server Resource Allocation, Electric Vehicle.**

I. Introduction

The emergence of 6G networks is revolutionizing communication with advanced capabilities and significantly impacted the EV industry, fostering enhanced connectivity for smart and efficient electric transportation systems. However, due to limited resources, vehicles may experience a slight decline in performance when processing these applications locally. In this context, Edge Computing(EC), which aims to provide computing resources at network edges has become a promising solution in EV utilization, offering real-time processing capabilities at the network's edge for improved data analysis and swift decision-making [1]. Drawing on EC principles like dynamic resource allocation and low-latency processing, EC in EVs seamlessly integrates with the broader network, boosting electric transportation system efficiency and effectiveness.

Through the implementation of Vehicular Edge Computing (VEC), computational tasks within the system can be intelligently offloaded to nearby Edge Computing Servers (ECS). This strategic offloading enhances processing efficiency, leading to a notable reduction in end-to-end delay compared to conventional cloud-based approaches. Consequently, this approach not only optimizes system performance but also elevates users' Quality of Experience (QoE).

The convergence of TO and RA has become a focal point in academia and industry. Prior studies [2]–[4] have primarily focused on task-offloading strategies using remote servers but often neglected the challenge posed by the uneven geographical distribution of service requests, leading to load imbalances among edge servers in different clusters. Addressing this, collaborative edge-edge cooperation models [5], [6] have emerged, allowing tasks offloaded to high-load edge servers to be further delegated to adjacent low-load servers. A distinctive feature of our VEC scenario lies in the consideration of multiple network areas with varying inter-propagation speeds. The overarching objective is to minimize the MESD, highlighting the nuanced and sophisticated nature of our proposed solution.

This paper makes notable contributions in the following key aspects:

1) A comprehensive MESD model is introduced to optimize the system's average latency and end-to-end delay.
2) In addressing the intricate challenge of joint TO and RA, a TO policy is devised aligned with the IEEE 1935 standard.
3) The proposed framework is rigorously validated through simulations based on real-world traffic datasets and a sample application service.

The following sections are structured as follows: Section II introduces the proposed system model. Section III details algorithm design intricacies, while Section IV covers simulations and evaluations. The paper concludes in Section V, summarizing findings and suggesting future research directions.

II. System Model

This section begins by introducing the collaborative cluster-cluster system architecture. Subsequently, it delves into the detailed processes of TO and RA within the VEC system. Finally, the paper outlines the MESD model considered in the context of this study.

A. Network Model

Depicted in Fig. 1, our study delves into a hierarchical VEC network, extending across diverse network areas and closely reflecting the architecture proposed in [7], [8]. Within this setup, compute nodes, representing the compute-level entity defined in the IEEE 1935 standard, serve as fundamental processing elements with various processor types. Meanwhile, control nodes, denoting the control-level entity defined in IEEE 1935, manage resources within their designated edge areas. Notably, compute nodes handle service execution, while control nodes collect data for decision-making in our scenario. At the core of this architecture, the Edge/Fog Orchestrator (EFO), functioning as the orchestrator-level entity for main management and orchestration per the IEEE 1935 standard,

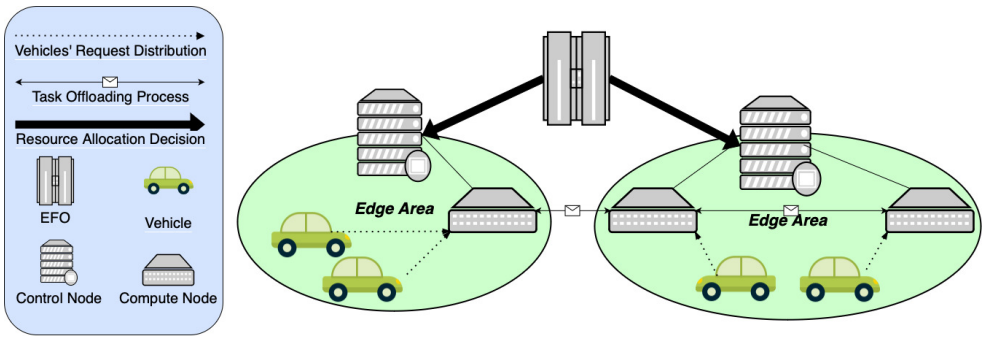

Fig. 1. The topology of the proposed VEC network scenario.

plays a pivotal role in making RA decisions to optimize MESD across the system.

Within our system, key components include an EFO, a network of Edge Clusters denoted as $C = \{1, 2, ..., c\}$, and a set of Edge Applications represented by $A = \{1, 2, ..., a\}$. Additionally, diverse resource types denoted collectively as $R = \{1, 2, ..., r\}$, are available across the entire system. The system further accommodates requests from all vehicles in the area of the ith cluster seeking the kth application's service, with Req_{ik} serving as a metric for the number of requests soliciting the kth application's service covered by Edge Cluster i. Notably, within each cluster's coverage area, the requesting services exhibit variations in processing delay, data size, and distinct inter-propagation speeds across clusters.

B. Computing & Communication Model

1) Computing Model: In acknowledging the influence of allocated computing resources on the performance of a deployed machine/container, we draw insights from recent advancements, such as those presented in [9]. Various methods, including linear regression and Support Vector Machine (SVM) models, have been proposed to assess the intricate relationship between allocated computing resources and the overall performance of a Docker container. In light of this, we conceptualize the task computing delay as a function of the allocated resource, denoted by η_k. Specifically, η_k represents the function capturing the average processing delay of application k under diverse allocations of computing resources.

Within the framework of processing delay functions, we make the simplifying assumption that these values will be truthfully provided by the service provider of the application. However, it is essential to recognize that the availability and distribution of computing resources differ across each edge cluster. Consequently, we impose the following constraints to ensure a realistic depiction of the system's dynamics:

$$C_1 : \sum_k Res_{jkr} \leq Res'_{jr}, j \in C, k \in A, r \in R \quad (1)$$

which implies that the aggregate RA decisions for all applications within a given cluster j must not surpass the available resources allocated to that cluster.

2) Communication Model: The computational tasks received within cluster i can be dynamically distributed among other edge clusters, necessitating consideration of the associated task transmission delay. Similar to the work [10], we introduce the term "unit propagation delay" denoted as e_{ij} between clusters i and j, which can be readily obtained through periodic measurements. Additionally, we assume that the average data size d_k of a deployed application k is provided by its service provider. Consequently, we formulate the model for average transmission delay for a single request being forwarded from cluster i to cluster j as the following:

$$t_{ij} = e_{ij} \cdot d_k, i \in C, j \in C, k \in A \quad (2)$$

C. MESD Model

In contrast to the several approaches in the recent works of [11], we aim to improve the system delay in a multi-edge scenario, with a particular emphasis on reducing latency for each user. The overarching concept involves identifying an adequately representative average delay for the entire system. Consequently, we calculate the average delay by considering all clusters, aiming to capture a holistic perspective of the system's performance.

$$D = avg([D_i]) = \frac{\sum_{i \in C}(D_i \cdot \sum_{k \in A} Req_{ik})}{\sum_{i \in C} \sum_{k \in A} Req_{ik}} \quad (3)$$

where D represents the average delay in the overall system, D_i is the average delay in cluster i and Req_{ik} is the amount of requests of application k in cluster i.

In the aforementioned context, we introduced the term average delay for a single cluster D_i without a precise definition. The average cluster delay is derived by calculating the overall delay through weighted averaging of the product of the forwarded requests amount N_{ijk} and its experienced delay, which is the summation of processing delay $D_{P_{ij}}$ and propagation delay $D_{T_{ij}}$ in our case. The detailed definition of the average delay for cluster i is specified below:

$$D_i = avg([D_{P_{ij}} + D_{T_{ij}}]) = \frac{\sum_{j \in C}(T_{ij} + P_{ij})}{\sum_{k \in A} Req_{ik}}, i \in C \quad (4)$$

where the variable T_{ij} is the result of multiplying the transmission delay between cluster i and cluster j by the

number of requests forwarded from cluster i to j. On the other hand, P_{ij} is the product of the processing delay and the number of requests received in cluster i, which is currently being handled in cluster j.

The propagation term, T_{ij}, has the physical meaning of cumulative propagation delay for the requests being forwarded from cluster i to j. Acknowledged by eq.(2), this relationship can be expressed by incorporating the summation term for the number of requests for application k being forwarded from cluster i to j, namely N_{ijk}.

$$T_{ij} = e_{ij} \cdot \sum_{k \in A} (N_{ijk} \cdot d_k), i \in C, j \in C \tag{5}$$

On a contrasting note, the processing term, P_{ij}, can be acquired by utilizing the performance function η_k provided by each service provider. This function takes RA decisions and concurrent serving requests as inputs and outputs a processing delay. Similarly, we introduce the summation term to calculate the overall case, encompassing the total processing delay for requests forwarded from cluster i to j.

$$P_{ij} = \sum_{k \in A} (\eta_{jk} \cdot N_{ijk}), i \in C, j \in C \tag{6}$$

At this juncture, we have transformed the problem of optimizing the overall effective delay, D, into a TO problem. The formulations are articulated through the crucial term N_{ijk} which holds significance and will be elaborated upon in the subsequent section.

III. Proposed Method And Algorithm

In this section, we commence by presenting the concept of redistributing requests within the system. Subsequently, we mathematically formulate the forwarding of requests employing a fixed forwarding policy. Finally, we integrate and address the problem by employing our proposed heuristic algorithm for an effective solution.

A. Collaborative Task Offloading(CTO)

In this section, we initially introduce the redistribution of request density through a load-balancing method. The fundamental concept is straightforward: clusters with a higher allocation of resources should handle a greater volume of requests. Consequently, the redistributed requests reflect the quantity that each cluster should effectively manage. The detailed formulation is presented below.

$$Req'_{jk} = \frac{|R_{jk}|}{\sum_{n \in C} |R_{nk}|} \sum_{n \in C} Req_{nk} \tag{7}$$

$$|R_{jk}| = \sqrt{\sum_{l \in R} R_{jkl}^2}$$

Here, Req'_{jk} symbolizes the redistributed quantity of requests for application k that ought to be managed in cluster j. Significantly, the formulated redistributed requests are functions of RA decisions that will dynamically adjust their value based on varying RA decisions. The term $|R_{jk}|$ denotes the allocated resource amount in cluster j for application k. Given the availability of multiple resource types for allocation, we formulate this term by employing the concept of vector magnitude. In other words, we can consider each available resource type as an independent vector, and quantify the resource amount by calculating the square root of the sum of the squares of each term. In addition, the notation $\sum_{n \in C} Req_{nk}$ straightforwardly signifies the total requests for application k across the entire system.

B. Requests Forwarding Formulation

Having obtained the redistributed requests, we can iteratively compute the forwarding amounts. We adopt a greedy approach to derive an optimal forwarding strategy, prioritizing clusters with higher inter-propagation speeds in the iteration process. In this approach, we consistently compute the self-offloaded part at the initial, where tasks are processed at the local cluster without forwarding, given the absence of propagation delay between any cluster and itself.

$$N_{iik} = \begin{cases} Req'_{ik}, Req_{ik} \geq Req'_{ik} \\ Req_{ik}, Req_{ik} < Req'_{ik} \end{cases} \tag{8}$$

Afterward, we formulated the rest of the terms in the ascending order of propagation speed considering a conditional term. The formulated relation considers two conditions: the clusters forwarding requests must have available request slots, and the cluster to which requests are being forwarded should have excess requests. The function enters the second part only if both conditions are simultaneously valid, namely $condition_{valid}$. In this second part, we establish another set of conditional values which the formulation always acquires the smaller term between the available request slots x on the cluster forwarding requests and the remaining excess requests y on the receiving cluster. The detailed algorithm for the formulation of the above mathematical relations is shown as the following in Algorithm.1.

$$N_{ijk} = \begin{cases} 0, Condition_{invalid} \\ min(x, y), Condition_{valid} \end{cases} \tag{9}$$

$$condition_{invalid} : Req'_{ik} + \sum_{\beta \in C - prevC_j} N_{i\beta k} \geq Req_{ik}$$
$$||Req'_{jk} \leq Req_{jk} + \sum_{\alpha \in C - prevC_i} N_{\alpha jk};$$

$$condition_{valid} : else;$$
$$x = Req'_{jk} - Req_{jk} - \sum_{\alpha \in C - prevC_i} N_{\alpha jk};$$
$$y = Req_{ik} - Req'_{ik} - \sum_{\beta \in C - prevC_j} N_{i\beta k};$$

It's important to note that the formulated strategy above is not rigid; rather, it is a dynamic function. This function takes an RA strategy as its input and produces a greedy forwarding policy under our assumptions.

Algorithm 1 Dynamical Requests Forwarding

Input: $[Req_{jk}], [Req'_{jk}], [e_{ij}]$
Output: $[N_{ijk}]$

Initialize $[N_{ijk}]$ as a 3D-array filled with zeros.
Initialize $prevC_n$ as a 2D-array filled with zeros.
for i loop through each cluster **do**
  for k loop through each application **do**
    $N_{iik} = Req'_{ik}$
    Append i into $prevC_i$
Sort $[e_{ij}]$ in ascending order
for i, j loop through $[e_{ij}]$ **do**
  if $i! = j$ **then**
    for k loop through each application **do**
      Compute N_{ijk} using Eq. 9
      Append j, i into $prevC_i$, $prevC_j$ respectively
return $[N_{ijk}]$

C. Collaborative Resource Allocation

In our discussion thus far, we have not addressed the resource constraints on each cluster and have yet to incorporate them into the mathematical relations defined above. While numerous mathematical approaches can solve optimization problems, the challenge lies in discarding invalid solutions due to resource constraints.

To address this, we introduce a filtering mechanism simulated by a sigmoid function $\sigma(ax)$. By adjusting the coefficient a to a higher value, the sigmoid function approximates a unit-step function of $u(x)$. In this way, we can incorporate the sigmoid function into our defined model to adhere to the resource constraints. Unlike a unit-step function, a sigmoid function lacks any discontinuous points, making it computationally straightforward.

To formulate the system delay equation with resource constraint, we simply add a conditional term to the delay model.

$$D^* = D + \Omega \sum_{\gamma_i \in Consts} \sigma(a\gamma_i) \tag{10}$$

$$\sigma(ax) = \frac{1}{1 + e^{-ax}}$$

where D^* represents the average system delay under resource constraint, Ω serves as a hyperparameter simulating significant delay when allocation decisions exceed resource constraints, and $Consts$ refers to the set of resource constraints within the system. For instance, the constraint function of 50 available CPUs in edge area 1, $CPU_1 \leq 50$ can be expressed as follows:

$$\sigma(a\gamma_1) = \frac{1}{1 + e^{-a(CPU_1 - 50)}}$$

Having discussed the formulated problem, it becomes apparent that it can be represented as a function taking RA decisions and requests distribution as inputs, and producing the average system delay as an output. With this mathematical definition established, various methods can be utilized to address this optimization problem.

To streamline the calculation, we propose a heuristic solution that leverages Particle Swarm Optimizer (PSO) for optimization. The formulated average delay model serves as the fitness function within the PSO solver. In this context, each particle in the PSO solver corresponds to a potential RA strategy. Utilizing the system delay model can easily translate these RA strategies into average system delays, optimizing the overall system delay.

IV. Performance Evaluation Methodology

This section provides an overview of the simulation environment, encompassing system parameters, methodology, and the dataset employed. To comprehensively evaluate the performance of our proposed scheme, we conducted extensive simulations across various scenarios. The parameters varied including the variance of requests distributed among the clusters and the amount of cumulative requests distributed among clusters.

The simulation parameters were carefully chosen following prior research, particularly studies referenced in [10] and [12]. Results from each scenario were averaged over 150 simulations to ensure robustness. In our default settings, we employed 3 clusters, each equipped with 2 resource types: CPUs randomly ranging from 5 to 15, and memory randomly ranging from 4GB to 10 GB. Additionally, the default data size of applications ranged randomly from 1MB to 5MB, while unit propagation delays fell within the range of 1MB/s to 20MB/s. Finally, the processing efficiencies of deployed applications were set to default values as follows by simulating the linear regression model acknowledged in [9].

$$\eta_{ik} = \frac{Req'_{ik}}{k_1 \cdot CPU_{ik}} + \frac{Req'_{ik}}{k_2 \cdot Memory_{ik}} \tag{11}$$

Here, k_1 and k_2 represent the weighted coefficients of each allocated resource and their default values are randomly ranging from 10 to 25 respectively. It is important to note that this performance model can be adjusted as needed to align with the characteristics of the deployed applications in real-world scenarios. For the sake of efficiency in our simulations, we have opted for a simplified model, understanding that it provides a foundational framework for our evaluation.

V. Performance Evaluation Results

This section delves into assessing the proposed framework's performance, focusing on the average system delay. To gauge the quality of RA decisions, we consider the following benchmarks:

1) No Task Offloading(NTO): This refers to the non-collaborative edge system where no cluster-cluster forwarding is allowed within the framework.
2) Game-based Task Offloading(GTO): This represents a non-collaborative framework where each cluster makes rational decisions solely for its own benefit. The idea of the scheme is embodied in [10].

3) Collaborative Task Offloading(CTO): This is the dynamical task offloading algorithm proposed in this work.

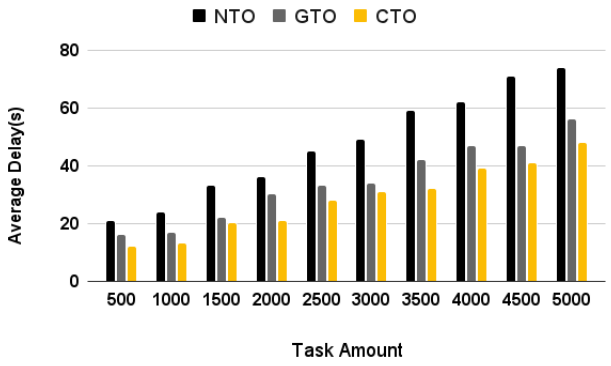

Fig. 2. Average Delay with the increase of task amount.

1) Average Delay With Different Task Amount: In Fig.2, we observe the comparison of the average delay optimized by our proposed algorithm with two reference schemes as the total requests in the overall system increase. Across all schemes, there is a notable trend of increasing average delay with the rising request volume. Particularly striking is the consistently higher average delay in the NTO scheme than others. This disparity primarily stems from non-uniform resource and request distribution among clusters, resulting in certain clusters exhibiting lower proficiency in local task processing. Consequently, this exacerbates the overall delay of the system. Conversely, our proposed CTO scheme stands out as the most effective. It optimizes resource utilization while ensuring system task delay tolerance and holding time limits, ultimately minimizing the overall average delay in the system.

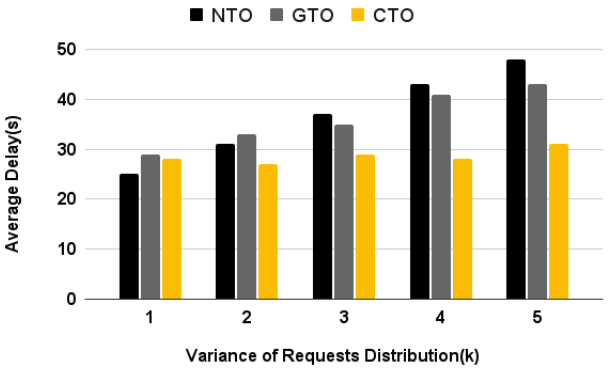

Fig. 3. Average delay with the increase of variance of requests distribution.

2) Average delay With Different Variance Of Requests Distribution: In Fig.3, we modify the variance of the requests distribution to create a less uniform distribution among the clusters, while keeping the other parameters at default values. We observe that the average delay increases for all schemes except our proposed scheme as the variance of the request distribution rises. This is attributed to the growing variance in request distribution, which amplifies the processing delay for a larger portion of requests in the more heavily loaded clusters, while marginally reducing delays for requests in less burdened clusters. Furthermore, our proposed approach, which prioritizes collaborative task offloading, effectively addresses this challenge and consistently outperforms other schemes.

VI. Conclusion

This paper explores the challenge of joint task offloading and resource allocation. By integrating propagation and processing delay, we formulate a MESD model, representing the average delay in the system. Our proposed solution involves redistributing requests among edge areas to address this issue. Simulation results demonstrate the superiority of our algorithm over alternative approaches, consistently achieving lower delays as task volume increases and maintaining consistent delays despite variations in request distribution among clusters. In future research, we will explore scenarios involving microservices and diverse network topologies, thoroughly discussing their potential limitations and scalability.

VII. Acknowledgement

Hung-Yu Wei is grateful for the funding support by National Science and Technology Council (NSTC) of Taiwan under Grant 112-2622-8-002-021- and 112-2628-E-002-025-.

References

[1] Y. Chiang et al., ”Management and Orchestration of Edge Computing for IoT: A Comprehensive Survey,” in IEEE Internet of Things Journal, vol. 10, no. 16, pp. 14307-14331, 15 Aug.15, 2023.

[2] M. Najm, M. Patra, and V. Tamarapalli, “Cost-and-delay aware dynamic resource allocation in federated vehicular clouds,” IEEE Trans. Veh. Tech- nol., vol. 70, no. 6, pp. 6159–6171, Jun. 2021.

[3] Y. Chiang, C. -H. Hsu, G. -H. Chen and H. -Y. Wei, ”Deep Q-Learning-Based Dynamic Network Slicing and Task Offloading in Edge Network,” in IEEE Transactions on Network and Service Management, vol. 20, no. 1, pp. 369-384, March 2023.

[4] L. H. Phuc, M. Kundroo, D. -H. Park, S. Kim and T. Kim, ”Node-Based Horizontal Pod Autoscaler in KubeEdge-Based Edge Computing Infrastructure,” in IEEE Access, vol. 10, pp. 134417-134426, 2022.

[5] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan and M. Xiao, ”Asynchronous Deep Reinforcement Learning for Collaborative Task Computing and On-Demand Resource Allocation in Vehicular Edge Computing,” in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 12, pp. 15513-15526, Dec. 2023.Magnetics Japan, p. 301, 1982].

[6] T. X. Tran and D. Pompili, ”Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks,” in IEEE Transactions on Vehicular Technology, vol. 68, no. 1, pp. 856-868, Jan. 2019.

[7] “IEEE standard for edge/fog manageability and orchestration,” IEEE standard 1935-2023, 2023.

[8] ETSI, “Mobile edge computing (mec); framework and reference architecture,” ETSI, DGS MEC, standard 3, 2016.

[9] K. Ye, Y. Kou, C. Lu, Y. Wang and C. -Z. Xu, ”Modeling Application Performance in Docker Containers Using Machine Learning Techniques,” 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 2018.

[10] W. Fan et al., ”Game-Based Task Offloading and Resource Allocation for Vehicular Edge Computing With Edge-Edge Cooperation,” in IEEE Transactions on Vehicular Technology, vol. 72, no. 6, pp. 7857-7870, June 2023.

[11] Y. -J. Ku, P. -H. Chiang and S. Dey, ”Real-Time QoS Optimization for Vehicular Edge Computing With Off-Grid Roadside Units,” in IEEE Transactions on Vehicular Technology, vol. 69, no. 10, pp. 11975-11991, Oct. 2020.

[12] W. Fan et al., ”Joint Task Offloading and Resource Allocation for Vehicular Edge Computing Based on V2I and V2V Modes,” in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 4, pp. 4277-4292, April 2023.