Augmented Reality: Mapping Methods and Tools for Enhancing the Human Role in Healthcare HMI

Distributed Learning with Memory: Optimizing Model Usage across Training Tasks

F. Malandrino*†, C. F. Chiasserini‡*†
*CNR-IEIIT, Italy – †CNIT, Italy – ‡Politecnico di Torino, Italy

***Abstract*—As the relevance of distributed learning to vehicular services grows, it becomes more important to perform such learning in the most effective possible manner. In this paper, we investigate the benefits stemming from a learning controller considering *multiple* learning tasks at the same time. Our performance evaluation shows that this new paradigm, which also enables model reusage across learning tasks, yields up to 60% savings on model training costs.**

I. Introduction and Scenario

As the communication and computation capabilities of both vehicles and the infrastructure supporting them grow, applications of machine learning (ML) to vehicular services multiply. At the same time, most of such ML is *distributed*, i.e., carried out by multiple mobile or fixed nodes, as exemplified in Fig. 1. The problem of optimizing a *single* distributed learning task has been widely studied in the literature, and many effective and efficient solutions thereto have been proposed. In particular, a significant body of work focuses on the federated learning (FL) paradigm and tackles the twin problems of (i) *selecting* the nodes to involve in the learning process, and (ii) *incentivizing* their cooperation. For node selection, reputation mechanisms are often used: the system tracks (through, e.g., a blockchain) which nodes provide the cooperation they promise, and excludes those that do not.

Most existing works, however, share the assumptions that:

- there is only one controller/broker, and
- each learning task can be considered in isolation.

In this work, we aim to go beyond the current assumptions, and optimize distributed learning tasks where (i) multiple brokers compete against each other, and (ii) multiple models can be used – and, crucially, *re-used* – across different learning tasks. We underline that, in our scenario and system model, the learning coordinator also acts as a *broker*. In fact, in addition to the tasks associated with distributed learning (e.g., collecting and averaging the local models in the case of FL), it is also in charge of:

- identifying the nodes to involve in the learning itself, and
- negotiating their compensation.

Also, contrary to traditional scenarios, all participants are rational, i.e., driven by their own *profit*: learning nodes participate only if they are awarded what they see as fair compensation, and the broker seeks to maximize its profit.

The ego-learner, coordinator, and helpers interact as summarized in Fig. 1. The process is initiated by the ego-learner (step 1), which provides the learning coordinator with basic statistics about its local data, e.g., number of features and learning domain, along with the target accuracy l^{M} and the maximum learning time T^{M}. Based upon such information, the coordinator determines the best model to use (step 2),

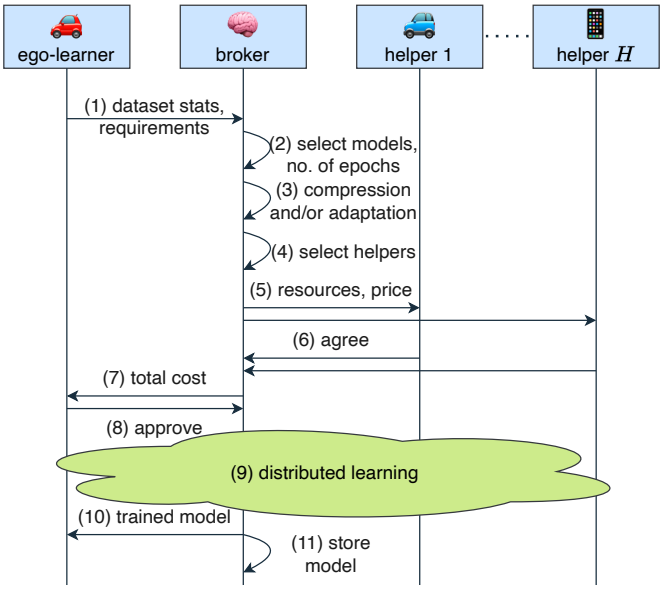

Fig. 1. Main steps of the learning process we envision, highlighting how the learning coordinator acts as a learning *broker*, in charge of choosing the best model to use (steps 2–3) and nodes (step 4), negotiating prices (steps 5–8), and storing already-trained models for future usage (step 11).

performs model compression or adaptation as needed (step 3), and selects the helpers to call for (step 4). Then, it agrees with the helpers on the resources they are going to contribute and their compensation (steps 5–6). Indeed, helpers will cooperate only if they are offered a sufficient compensation for the resources they devolve to the training. After the price is accepted by the ego-learner (steps 7–8), the training itself takes place following any distributed learning technique, possibly including model compression and/or domain adaptation [1] steps, as needed. Once training is over, the resulting model is returned to the ego-learner (step 10) and, crucially, stored by the coordinator for future use (step 11).

There are three main ways the coordinator can reuse an existing model for a new learning task, as per step 2 in Fig. 1:

- if the task and domain (e.g., recognizing license plates) are the same as that of an already-trained model, then that model can be *fine-tuned* [2], [3] and reused;
- if the task is the same but the domain is different (e.g., license plates from different countries), then *domain adaptation* techniques [1] can be applied in step 7;
- if both the task and the domain are different, some training can anyway be salvaged, e.g., the coefficients of early convolutional layers [4].

The key observation is the same in all cases: the coordinator can leverage *past* training to perform the learning task in a shorter time and/or with fewer resources, hence, at a lower cost.

II. System Model and Problem Formulation

Model elements. The main system components are as follows:

- learning nodes in $\mathcal{N}$, which includes the ego-learner;
- datasets in $\mathcal{D}$, owned by the nodes;
- models in $\mathcal{M}$, also including modified and compressed model versions.

Parameters. Each node $n \in \mathcal{N}$ is associated with a quantity of *resources* $\rho(n)$, denoting its capabilities (e.g., energy, computational). Similarly, models are associated with *a complexity level* $\chi_{\mathrm{r}}(\mu)$, indicating the quantity of resources needed to train one epoch of that model. We are also given size $\sigma(\mu)$ of model μ and bandwidth $\beta(n)$ characterizing the communication link between node n and the coordinator. It follows that the communication latency incurred by node n when transmitting/receiving an updated version of model μ is $\frac{\sigma(\mu)}{\beta(n)}$. Finally, $\chi_{\mathrm{avg}}(\mu)$ is the quantity of resources[1] needed to average instances of model μ at the coordinator.

Learning process. Given a model $\mu \in \mathcal{M}$ and a set of datasets $\bar{\mathcal{D}} \subseteq \mathcal{D}$, we define as $\lambda_{\mathrm{r}}(k, \mu, \bar{\mathcal{D}})$ the global change in the loss function achieved by training model μ over datasets $\bar{\mathcal{D}}$ at the k-th epoch. Notice that training aims at minimizing the loss, hence, λ_{r} will be negative if the training does progress.

Compressing model $\mu \in \mathcal{M}$ into model $\mu' \in \mathcal{M}$ changes the loss by $\lambda_{\mathrm{c}}(\bar{\mathcal{D}}, \mu, \mu')$, with $\bar{\mathcal{D}} \subseteq \mathcal{D}$ being the datasets being used. Similarly, performing domain adaptation, i.e., moving from model μ trained on datasets $\bar{\mathcal{D}} \subseteq \mathcal{D}$ to model μ' to use with datasets $\bar{\mathcal{D}}'$ changes the loss by $\lambda_{\mathrm{a}}(\bar{\mathcal{D}}, \bar{\mathcal{D}}', \mu, \mu')$. Both λ_{c} and λ_{a} are typically positive, as both model compression and domain adaptation may result – in the short term – in an increase of the loss.

Finally, we are given the quantity of computational resources consumed by model compression and domain adaptation, denoted, respectively, by $\chi_{\mathrm{c}}(\bar{\mathcal{D}}, \mu, \mu')$ and $\chi_{\mathrm{a}}(\bar{\mathcal{D}}, \bar{\mathcal{D}}', \mu, \mu')$. All these values are *input* information to our problem, and, as discussed later, estimating them is an orthogonal problem to the one we tackle.

Decisions and their effects. The coordinator has to make the following decisions:

- for each node $n \in \mathcal{N}$, whether or not to involve it in the learning at epoch k, expressed through binary variable $y(k, n) \in \{0, 1\}$;
- for the nodes that are chosen, for how many resources they are asked, expressed through real variable $x(k, n) \geq y(k, n)\rho(n)$;
- the number K of epochs to run;
- the model (or model version) $\mu \in \mathcal{M}$, to use for training;
- possibly, a model μ_0 that needs to be adapted and/or compressed before the training starts;
- the amount $x^c(k)$ of coordinator resources to commit at epoch k;
- the price $p(n)$ to offer to each node so as to guarantee its cooperation.

Choosing a previously-trained model, $\mu_0 \neq \mu$, permits to kickstart training. Although model adaptation and pruning may come at a cost in terms of computational resources [5], the resulting loss will often be substantially better than the one yielded by a new, untrained model [1], [5], [6].

[1]Due to the way averaging is implemented, this quantity is actually independent of the number of such instances.

Assuming, without loss of generality, that synchronous learning is adopted, and indicating as $\bar{\mathcal{D}}$ the datasets of the selected nodes, the time taken by local learning and data transmission in epoch k is given by:

$$T_n(k) = \max_{n \in \mathcal{N}} \left(\frac{\chi_{\mathrm{r}}(k, \mu(k), \bar{\mathcal{D}})}{x(k, n)} + \frac{\sigma(\mu)}{\beta(n)} \right), \tag{1}$$

i.e., the combined computation and communication time of the *slowest* selected node.

To compute the total epoch duration, we must combine (1) with the additional operations taking place at the coordinator, i.e., (i) averaging, (ii) compression (if needed), and (iii) domain adaptation (if applicable). More formally,

$$T_c(k) = \mathbb{1}_{k=1} \frac{\chi_{\mathrm{a}}(\bar{\mathcal{D}}_{\mathrm{prev}}(\mu), \bar{\mathcal{D}}, \mu, \mu)}{x^c(k)} + \frac{\chi_{\mathrm{avg}}(\mu)}{x^c(k)} + \mathbb{1}_{k=1 \wedge \mu_0 \neq \mu} \frac{\chi_{\mathrm{c}}(\bar{\mathcal{D}}, \mu_0, \mu)}{x^c(k)}. \tag{2}$$

The first and last terms of (2) can only be incurred at epoch k=1; also, they reduce to zero if (respectively) $\bar{\mathcal{D}}_{\mathrm{prev}}(\mu)$=$\bar{\mathcal{D}}$, i.e., the chosen model has been trained over the same dataset (hence, no adaptation is necessary) or μ_0=μ, hence, no compression is necessary. Combining the two contributions in (1) and (2), we can obtain the total duration of epoch k as:

$$T(k) = T_n(k) + T_c(k). \tag{3}$$

Concerning the loss $l(k)$, it can be computed recursively by accounting for all λ-contributions:

$$l(k) = l(k-1) + \lambda_{\mathrm{r}}(k, \mu(k), \bar{\mathcal{D}}) + \mathbb{1}_{k=1} \lambda_{\mathrm{a}}(\bar{\mathcal{D}}_{\mathrm{prev}}(\mu), \bar{\mathcal{D}}, \mu, \mu) + \mathbb{1}_{k=1} \lambda_{\mathrm{c}}(\bar{\mathcal{D}}, \mu_0, \mu). \tag{4}$$

Similarly to (3), the last two terms in (4) reduce to zero if no compression or adaptation is performed.

Pricing and cooperation. To determine whether they are offered a sufficient price for the resources they should devolve to the training, nodes will weight:

- the cost of their computing resources and bandwidth, against
- the price they receive from the broker, plus the value they see in training the selected model (e.g., to have it ready for future local use).

These two quantities can be expressed as:

$$\kappa(n) = \sum_{k=1}^{K} \left[x(k, n) \kappa_{\mathrm{R}}(n) + y(n, k) \kappa_{\mathrm{T}}(n) \sigma(\mu(k)) \right], \tag{5}$$

and

$$\gamma(n) = p(n) + \max_{k} y(k, n) \nu(n, \mu(k), l(k)). \tag{6}$$

It is clear that rational nodes will cooperate only if the latter outweighs the former. In (5), κ_{R} and κ_{T} represent the cost – e.g., monetary or energetic – assigned by node n to its computing and bandwidth resources (resp.). In (6), $\nu(n, \mu(k), l(k))$ represents the value that node n sees at epoch k in training (and storing) model n until loss l.

Concerning the broker, its objective is to maximize its profit, i.e., the difference between the price asked to the ego-learner (Fig. 1, step 6a) and the price it has to pay to the helpers (i.e., $\sum_n p(n)$), plus its own costs. It follows that the broker will seek to solve the problem of minimizing the

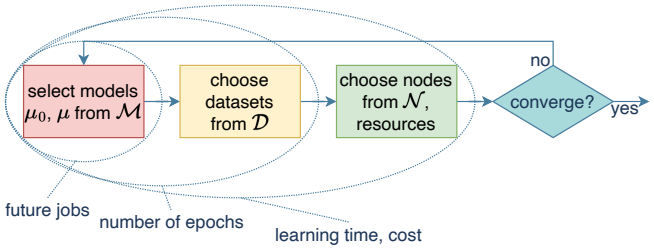

Fig. 2. The main steps of our solution concept. Ellipses enclose the decisions that affect the aspect of the learning process reported in the associated label.

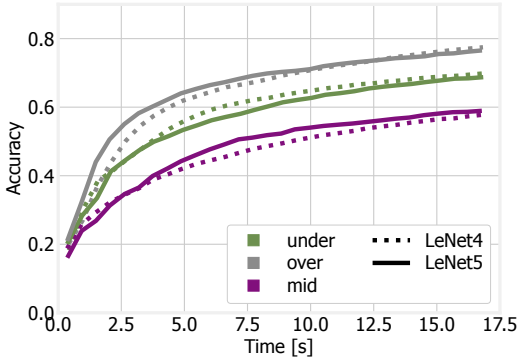

Fig. 3. First learning job: accuracy as a function of time for different datasets (line color) and model architectures (line style).

aforementioned quantity, subject to learning quality and time target, and to the fact that all helpers cooperate.

Objective and constraints. The problem solved by the broker can be summarized as follows:

$$\min_{x,y,K,\mu} \quad \sum_n p(n) + \kappa_{\text{server}} \sum_{k=1}^{K} x^c(k) \tag{7}$$

$$\text{s.t.} \quad l(K) \leq l^{\text{M}} \tag{8}$$

$$\sum_{k=1}^{K} T(k) \leq T^{\text{M}} \tag{9}$$

$$\kappa(n) - \gamma(n) \leq 0 \quad \forall n \in \mathcal{N}. \tag{10}$$

In other words, the broker seeks to minimize the price to pay to other nodes plus the cost of its own resources (7), subject to the fact that the target learning quality (8) is achieved by the target time (9), and that all nodes required to cooperate will do so (10).

Multiple servers: the need to look ahead. As it can be inferred from (7), the above problem is *myopic*, i.e., it only considers the current learning task. However, this neglects the fact that the same model may be reused in multiple learning tasks, and model selection decisions need to account for that. This is especially relevant when multiple brokers are present. Indeed, as mentioned earlier, such brokers compete against each other and seek to maximize their profit. They cannot increase the price asked from the ego-learner, as doing so would drive them out of the market [7], hence, their best strategy is to reduce costs. Look-ahead decisions allow exactly for that, by exploiting the availability of partially-trained model instances, which can be adapted to the new ego-learner with little effort [1], [6].

To express and implement look-ahead decisions, we need to define a set of *jobs*

$$\mathcal{J} = \{(t^J_{\text{arr}}, n^J, T^{\text{M,J}}, l^{\text{M,J}})\},$$

where each job $J \in \mathcal{J}$ is a 4-uple including:

- the time t^J_{arr} at which the job is requested;
- the ego-learner n^J where it originates;
- the maximum learning time $T^{\text{M,J}}$;
- the maximum acceptable value of final loss value $l^{\text{M,J}}$.

Given the above, and adding superscripts J to the decision variables and costs, we can rewrite objective (7) as:

$$\min_{x^J, y^J, K^J, \mu^J} \sum_{J \in \mathcal{J}} \sum_n p^J(n) + \kappa_{\text{server}} \sum_{k=1}^{K^J} X^J(k), \tag{11}$$

i.e., the broker's goal becomes the minimization of its *total* cost, over – in principle – *all* jobs, including future ones.

III. The Hardest-first Solution Strategy

Solving our problem directly, e.g., through a solver, is prohibitively complex; specifically, it is possible to prove that:

Property 1: The problem of minimizing (7) subject to constraints (8)–(10) is NP-hard.

The proof, based upon a reduction from the generalized assignment problem (GAP) is skipped in the interest of space.

As a consequence, our solution concept is predicated upon (i) *decoupling* the main decisions to make; (ii) making them *sequentially*, while (iii) accounting for their *mutual influence*. Specifically, the main decisions to make concern (i) the models to choose, (ii) the data sources to leverage, and (iii) the additional nodes to enroll, along with the resources they should allocate. Decisions are made through nested loops: as shown in Fig. 2, we arrange the loops so that different aspects of the learning process stay, insofar as possible, constant throughout the iterations. Specifically,

- model selection is the only decision influencing the cost of future jobs (11), hence, that decision is made in the outermost loop;
- models and datasets, determine the number of epochs needed to reach the target learning quality l^{M}, hence, datasets selection is made in the middle loop;
- all decisions influence epoch duration and cost, hence, node selection decisions are made in the inner loop.

The individual decisions can then be made following high-performance, well-tested approaches, as described next.

Model selection. Inspired by [8], which remarks that the *complexity* of DNN models grows more *slowly* than the learning performance they yield, we aim at finding the smallest model (i.e., the one with the fewest parameters) among those that yield the required learning quality. Thanks to the fact that we have to select *one* model, we can scour the set $\mathcal{M}$ very efficiently following a bisection, Newton-based approach [9].

Data source selection. Our key observation is that adding data sources has two effects on the overall learning process. First, it improves the learning quality (e.g., accuracy), which is proportional to the *logarithm* of the quantity of data [10]. At the same time, it increases the duration of each individual epoch, which is *proportional* to the total quantity of data [11]. It follows that a greedy hill-climbing approach [12], choosing the *minimum* amount of data needed for a feasible solution, yields a provably quasi-optimal solution – namely, within $1 - \frac{1}{e}$ from the optimum.

Node and resource allocation. Once model and dataset have been chosen, selecting the nodes to use and the resource

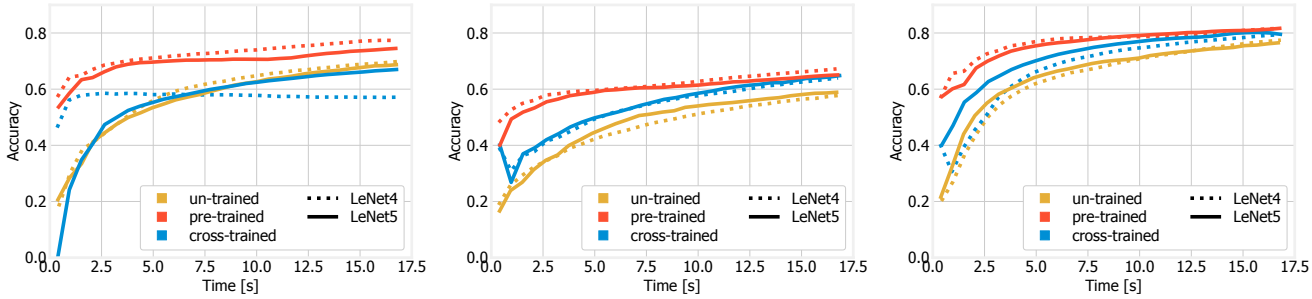

Fig. 4. Second learning job: accuracy as a function of time for different model selection strategies and model architectures (line style), when the second job uses dataset *under* (left), *mid* (middle), *over* (right).

to devote to the learning process is akin to solving a *VNF chaining* problem [13]. Hence, any of the efficient and effective solution strategies developed for VNF chaining can be leveraged. Many powerful approaches are based upon the generalized assignment problem (GAP); among those, we follow the one in [13] for our performance evaluation.

IV. Numerical Results

The goal of our performance evaluation is to show how accounting for *multiple* learning tasks – as allowed by our framework – yields significant cost savings. To this end, we consider a very relevant task for vehicular scenarios, i.e., image recognition, and perform a set of experiments using the CIFAR10 dataset and two DNN architectures belonging to the family of LeNet, namely, LeNet4 and LeNet5, having (resp.) four and five layers. We subdivide the CIFAR10 dataset in three sub-datasets, called: *under* (classes 1–5), *mid* (classes 3–7), and *over* (classes 6–10). We consider two learning jobs: in the first one, no pre-trained model is available (hence, a new model is trained from scratch); conversely, the second job can exploit the models trained during the first job (if that is beneficial). Throughout the performance evaluation, we use the learning time as a proxy for cost; this is consistent with the fact that, in all cloud environments, (virtual) machine utilization is assessed (and billed) by time.

Fig. 3 shows the evolution of accuracy over time for different datasets and model architectures. It is important to remark that time is a very good proxy for the total training cost. For some datasets, the simpler LeNet4 architecture yields a better time/accuracy than the more complex LeNet5; this is due to the fact that individual training epochs under simpler architectures are shorter, hence, more epochs can be performed in the same time.

We now move to the second job. Each plot in Fig. 4 shows the time evolution of the accuracy given the dataset used for that job; within each plot, the line style denotes the DNN architecture used in that job. Lines of different colors denote the *model selection* strategy used by the broker:

- un-trained: use a new model;
- pre-trained: use a model trained over *the same* dataset;
- cross-trained: use a model trained over *a different* dataset.

We notice that the pre-trained strategy always yields the best performance: this makes intuitive sense, as that means effectively avoiding the training process altogether. More interestingly, there is no consistently better strategy between un-trained and cross-trained; in many cases, a model trained for a different dataset may yield better performance than a new, untrained one. In some cases (see, e.g., the right plot), choosing the cross-trained strategy allows using the simpler LeNet4 architecture *in lieu* of the more complex (hence, computationally more onerous) LeNet5.

V. Conclusion

We made the key observation that distributed learning approaches should be optimized over *multiple* learning jobs, accounting for reuse opportunities when making model selection decisions. We thus envisioned an entity, called *learning broker*, in charge of making joint decisions about (i) the datasets to use, (ii) the models to train, and (iii) the nodes and resources to leverage. Multiple brokers compete with each other to maximize their own profit, hence, they seek to minimize the costs of the learning jobs. In view of the problem complexity, we proposed an iterative solution strategy, where each decision is made separately while also accounting for their mutual interaction. Our performance evaluation shows that cost savings exceeding 60% are possible when multiple subsequent learning jobs are optimized jointly, validating our motivation and solution strategy.

References

[1] H. Zhang, G. Luo, J. Li, and F.-Y. Wang, "C2fda: Coarse-to-fine domain adaptation for traffic object detection," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[2] H. Guo, Q. Yang, H. Wang, Y. Hua, T. Song, R. Ma, and H. Guan, "Spacedml: Enabling distributed machine learning in space information networks," *IEEE Network*, 2021.

[3] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, 2022.

[4] M. Riera, J.-M. Arnau, and A. González, "Computation reuse in DNNs by exploiting input similarity," in *ACM/IEEE ISCA*, 2018.

[5] H. Harutyunyan, A. S. Rawat, A. K. Menon, S. Kim, and S. Kumar, "Supervision complexity and its role in knowledge distillation," *arXiv preprint arXiv:2301.12245*, 2023.

[6] Z. Cao, Z. Li, X. Guo, and G. Wang, "Towards cross-environment human activity recognition based on radar without source data," *IEEE Transactions on Vehicular Technology*, 2021.

[7] S. Shakkottai and R. Srikant, "Economics of network pricing with multiple isps," *IEEE/ACM Transactions On Networking*, 2006.

[8] J. Sevilla, "Parameter counts in machine learning." https:towardsdatascience.comparameter-counts-in-machine-learning. Accessed: 2010-09-30.

[9] R. B. Kearfott, "Interval newton/generalized bisection when there are singularities near roots," *Annals of Operations Research*, 1990.

[10] T. Linjordet and K. Balog, "Impact of training dataset size on neural answer selection models," in *Springer ECIR*, 2019.

[11] Y. Duan and J. Wu, "Joint optimization of dnn partition and scheduling for mobile cloud computing," in *ACM ICPP*, 2021.

[12] S. H. Jacobson and E. Yücesan, "Analyzing the performance of generalized hill climbing algorithms," *Journal of Heuristics*, 2004.

[13] D. Harris and D. Raz, "Dynamic vnf placement in 5g edge nodes," in *IEEE NetSoft*, 2022.