# Loci
## Federated Continual Learning of Heterogeneous Tasks at Edge

Luopan, Yaxin; Han, Rui; Zhang, Qinglong; Zuo, Xiaojiang; Liu, Chi Harold; Wang, Guoren; Chen, Lydia Y.

**DOI**
10.1109/TPDS.2025.3531123

**Publication date**
2025

**Document Version**
Final published version

**Published in**
IEEE Transactions on Parallel and Distributed Systems

# Loci: Federated Continual Learning of Heterogeneous Tasks at Edge

Yaxin Luopan, Rui Han, Qinglong Zhang, Xiaojiang Zuo, Chi Harold Liu, *Senior Member, IEEE*, Guoren Wang, *Senior Member, IEEE*, and Lydia Y. Chen, *Senior Member, IEEE*

***Abstract*—Federated continual learning (FCL) has attracted growing attention in achieving collaborative model training among edge clients, each of which learns its local model for a sequence of tasks. Most existing FCL approaches aggregate clients' latest local models to exchange knowledge. This unfortunately deviates from real-world scenarios where each model is optimized independently using the client's own dynamic data and different clients have heterogeneous tasks. These tasks not only have distinct class labels (e.g., animals or vehicles) but also differ in input feature distributions. The aggregated model thus often shifts to a higher loss value and incurs accuracy degradation. In this article, we depart from the model-grained view of aggregation and transform it into multiple task-grained aggregations. Each aggregation allows a client to learn from other clients to improve its model accuracy on one task. To this end, we propose Loci to provide abstractions for clients' past and peer task knowledge using compact model weights, and develop a communication-efficient approach to train each client's local model by exchanging its tasks' knowledge with the most accuracy relevant one from other clients. Through its general-purpose API, Loci can be used to provide efficient on-device training for existing deep learning applications of graph, image, nature language processing, and multimodal data. Using extensive comparative evaluations, we show Loci improves the model accuracy by 32.48% without increasing training time, reduces communication cost by 83.6%, and achieves more improvements when scale (task/client number) increases.**

***Index Terms*—Federated continual learning (FCL), heterogeneous tasks, task-grained aggregation, edge computing.**

## I. Introduction

Artificial intelligence (AI) applications are widely deployed on edge devices or clients, e.g., classifying image objects [1], [2], translating text [3], and recognizing patterns in social graphs [4]. Federated learning (FL) [5] enables edge clients to collaboratively learn a global model by aggregating their local models without exchanging the raw data. Federated continuous learning (FCL) addresses scenarios where clients incrementally train models over a sequence of tasks characterized by their data distributions. For example, Fig. 1(a) illustrates a FCL scenario of $n$ edge clients in an image classification application. Each client learns its own sequence of *heterogeneous tasks*, namely distinct class labels (e.g., cars, ships, and houses in client 1, and cats, dogs, and horses in client 2) and a subspace of input feature distribution for each task. Fig. 1(c) further shows that such heterogeneous tasks widely exist in edge AI applications of different data modalities, such as graph [4], text [3], and multi-modal data [1], [2]. It is exceedingly challenging to learn AI models that can be generalized on such heterogeneous tasks, which vary across clients and over time, especially on resource strenuous edge devices.

*Challenges:* The main focus of existing FCL studies is on incorporating the knowledge from *peer clients' latest local models* [6], with an implicit assumption that these tasks are homogeneous or similar. This unfortunately deviates from the real-world application scenarios [7], [8], [9], [10], where each participating client trains the local model for *its own sequence of tasks*. At the same time, different clients have heterogeneous tasks, resulting in a large discrepancy of gradients in their loss space. As illustrated in Fig. 1(a), the aggregated model drifts to a higher loss space [11] (i.e., negative knowledge transfer) and causes accuracy degradation. To verify the above claim, Fig. 1(b)'s evaluation result shows that existing FL for homogeneous/heterogeneous models [12], [13], [14], [15], FCL [16], [17], [18], and clustered FL [19], [20], [21] all have considerable accuracy decreases in their local models after aggregating other clients' heterogeneous tasks. The *first challenge*, therefore, requires us to depart from the model-oriented view of global aggregation and manage it at a finer level of task granularity. One feasible approach of incorporating heterogeneous tasks' knowledge in global aggregation is to upload the information of clients' historical and current tasks to the central server. The *second challenge* lies in designing a scheme to support such global knowledge exchange with low communication cost. In existing approaches, the exchanged information can be training samples [21], [22], extracted knowledge [17], [18], information matrix [23], or tasks' mask parameters [24]. However, since the communication cost of extra information is proportional to model size, task number, and client number. These approaches incur *high communication cost* and become infeasible when running on edge devices with limited network bandwidth. For instance, when applying FedKNOW [18] to

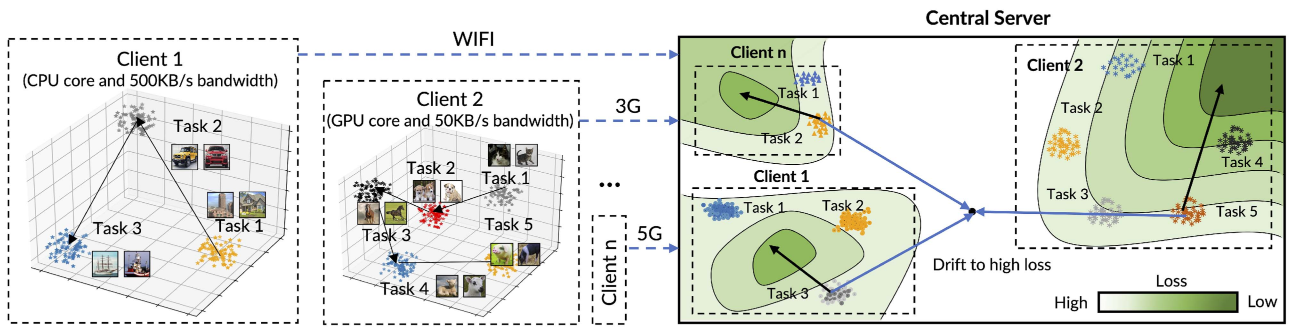
Received 6 April 2024; revised 6 January 2025; accepted 6 January 2025. Date of publication 29 January 2025; date of current version 5 March 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62272046, Grant 62132019, Grant 61872337, and Grant U21A20519, in part by the high-quality development project of the Ministry of Industry and Information Technology under Grant CEIEC-20240, and in part by the Open Project of Key Laboratory of Computing Power Network and Information Security under Grant 2023PY002. Recommended for acceptance by R. Prodan. *(Corresponding author: Rui Han.)*

Yaxin Luopan, Rui Han, Qinglong Zhang, Xiaojiang Zuo, Chi Harold Liu, and Guoren Wang are with the Beijing Institute of Technology, Beijing 100811, China (e-mail: hanrui@bit.edu.cn).

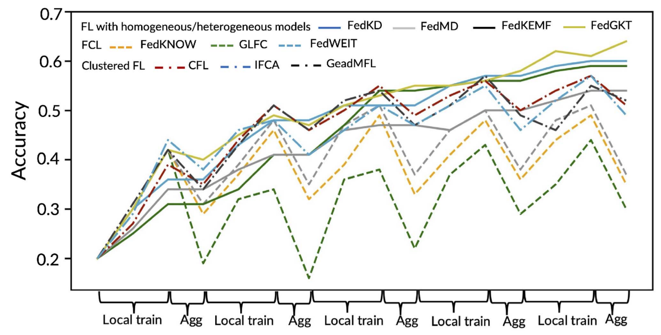
Lydia Y. Chen is with the TU Delft, 2628 CD Delft, Netherlands.

Our code is available at https://github.com/LINC-BIT/Loci.

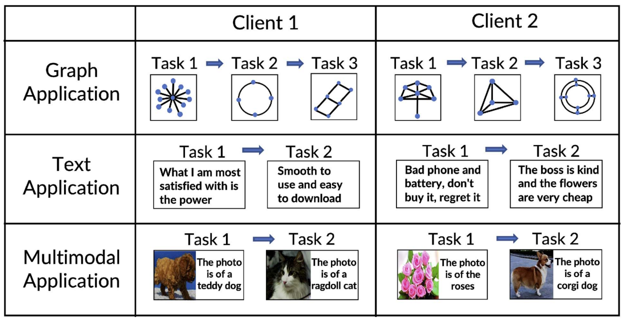
Digital Object Identifier 10.1109/TPDS.2025.3531123



(a) Heterogeneous tasks in different clients



(b) Accuracies of existing FL and FCL methods under heterogeneous tasks


Client 1
Client 2
Graph Application
Text Application
Multimodal Application
Task 1
Task 2
Task 3
What I am most satisfied with is the power
Smooth to use and easy to download
Bad phone and battery, don't buy it, regret it
The boss is kind and the flowers are very cheap
The photo is of a teddy dog
The photo is of a ragdoll cat
The photo is of the roses
The photo is of a corgi dog

(c) Heterogeneous tasks in different applications

Fig. 1. An example scenario of FCL with heterogeneous tasks.

aggregate ResNet-18 across 10 clients (each one has 10 tasks), the average communication cost is 1.18 GB per aggregation, which is several 3 to 4 orders of magnitudes larger than network bandwidths.

In this paper, we present Loci, a FL approach designed for collaboratively learning heterogeneous tasks across edge clients. To address the first challenge, Loci employs a novel *task memory palace* which transforms one global aggregation into multiple task-grained aggregations. Each aggregation allows a client to extract knowledge from other clients to improve its local model's performance on one task (that is, positive knowledge transfer to this task). To this end, it retains the compact knowledge of all clients' previously learned tasks and organizes these tasks in an index tree such that tasks close in feature values are grouped in the same node. For a task-grained aggregation, Loci quickly identifies this task's similar tasks in the task memory palace and only aggregates the task with them. The task memory palace also supports dynamic addition of newly arrived tasks with incremental tree updating (inserting or deleting leaf nodes) and combines similar tasks, hence it is scalable to large task and client numbers.

To address the second challenge, each time a client completes learning a new task, loci extracts its knowledge as a compressed network and sends it to the server-side task memory palace only once. In global aggregation, Loci uses the task memory palace to share and exchange knowledge among clients and transfers the learned knowledge to the client's local model via compact distilled models [25].

We fully implement Loci on top of PyTorch and integrate our system with MMCV (Convolution Neural Networks) [26] and Huggingface (transformers) [27]. We show that Loci enables positive knowledge transfer with low computation and communication costs. Using applications of various data modalities (image, text, graph, and multimodal), we demonstrate the effectiveness of Loci by comparing it with three categories types of state-of-the-art distributed learning techniques: FL methods with homogeneous and heterogeneous models [12], [13], [14], [15], FCL methods [16], [17], [18], and clustered FL methods [19], [20], [21]. Extensive experiments on four types of commodity edge devices (Jetson TX2, Xavier NX, AGX, and Raspberry Pi) show: (i) Loci improves model accuracy by 32.48% using similar or less model training time, in which the searching of similar tasks only takes less than 0.1% of entire training time. In challenging scenarios of 50 tasks or 200 clients, our approach achieves a significant accuracy improvement (57.43%); (ii) When completing the same training jobs, Loci reduces communication time by 83.6%; (iii) Loci achieves both 44.48% accuracy improvement and 67.14% communication cost reduction in decentralized/Gossip-based model training; and (iv) Loci is applicable to different continual learning algorithms and follows the convergence proof curves. We make the following contributions in this paper:

- We present Loci, the first-of-kind FL system on heterogeneous tasks with evolving data distributions.
- We design the task memory palace to quickly search for peer and historical tasks that are influential for improving each client's local model accuracy.
- We show how our task memory palace allows clients to cooperate independently and avoid high communication costs.

- We demonstrate how Loci scales up to a large number of heterogeneous tasks and clients.
- We extensively evaluate the effectiveness of Loci with popular benchmarks in CV, NLP, graph, and multimodal applications.

## II. Background and Related Work

In this section, we first introduce heterogeneous tasks with evolving input distributions and their corresponding distance metrics in four prevalent AI applications (Section II-A), and then discuss related work (Section II-B).

### A. Heterogeneous Tasks

*Definition 1. Task and heterogeneous task:* Each task $\mathcal{T}_i$ consists of a set of input data $X_i = \{x_{i,1}, x_{i,2}, \ldots\}$ and a set of corresponding class labels $Y_i = \{y_{i,1}, y_{i,2}, \ldots\}$. Any two different tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ meet the following two conditions: (1) their input data $X_i$ and $X_j$ are not identical and independently distributed (Non-IID); (2) their labels $Y_i$ and $Y_j$ are not identical. We refer to these two tasks as *heterogeneous* tasks (denoted as $\mathcal{T}_i \neq \mathcal{T}_j$) due to their non-identical data distributions and disjoint label sets.

Heterogeneous tasks widely exist in diverse data modalities and applications and this paper studies four examples:

- *Graph applications:* A task consists of multiple graphs, where each graph contains multiple nodes and edges. The heterogeneity between two tasks is characterized by their inconsistent edges, nodes, and topological structures in the graphs. For instance, task 1's star-shaped structure and task $n$'s polygon-like structure in Fig. 1(a), and the heterogeneity/distance between them is calculated as the discrepancy between the nodes/edges of graphs.
- *Computer vision (CV) applications:* Two heterogeneous tasks have different local features in their images, e.g., houses in task 1 and cats in task $n$ (Fig. 1(a)). Their heterogeneity is calculated as the discrepancy of the extracted local features.
- *Text applications:* Two heterogeneous tasks have different vocabularies and underlying semantics. In Fig. 1(a)'s example, task 1's texts discuss cars and task $n$'s texts focus on phones. Hence their heterogeneity is calculated as the discrepancy between vocabularies and semantics.
- *Multimodal applications:* A task contains multiple types of data such as graphs, images, and texts. Therefore, the heterogeneity between two tasks is calculated as the sum of the heterogeneity of each data type they contain.

### B. Related Work

FL is a collaborative learning paradigm designed to train models on distributed edge devices without directly sharing raw data, thereby ensuring data privacy and security [5]. Recently, FL has attracted significant successes in various AI applications, typical examples include: (i) graph applications: due to the high degree of correlation between nodes in graph data, current research [28] has designed specific federated graph frameworks to efficiently transfer node information between different clients. (ii) NLP applications: most text-related tasks rely on large language models, e.g., GPT [29], and FL methods focus on compression, pruning, and parallel computation techniques. (iii) Computer vision (CV) applications: most FL methods study image classification applications [12], object detection and semantic segmentation applications [30]. (iv) Multimodal applications: these applications require models capable of processing multiple different types of data, such as image-text matching. FL methods have effectively used representation learning methods to efficiently transfer client information [31]. At present, numerous efforts are contributed to address incremental tasks and heterogeneous models in clients' non-IID datasets.

*FL with heterogeneous models:* These methods aim to aggregate models of distinct architectures submitted by different clients [32]. In addressing the challenges posed by model heterogeneity, the concept of knowledge distillation has been widely applied in federated learning. FedMD [33] is the first to use public and private datasets to obtain information about heterogeneous models on individual clients. It uses the logical outputs of local models on the public dataset as global knowledge. Then FedDF [34] refines a set of client-side teacher models into a server-side student model. RHFL [35], [36] learns the knowledge distribution of other clients by aligning models' feedback on unrelated public data. [37] further reduces communication cost from clients to the server using a compression approach. FedGEN [38] considers server-independent public datasets and employs data-free distillation for federated training. FedGKT [14] performs dual-end knowledge distillation on client and server models. Recently, FedKD [39] designs various distillation losses based on the network layer on the client-side. In addition, FedKEMF [40] considers merging all teacher networks during aggregation, and uses a common dataset to distill a better server-side global network. Some other methods combine transfer learning and knowledge distillation to address the challenges of model heterogeneity [41], [42]. In conclusion, current FL methods assume that all local models learn the same task.

*FCL:* Developed from the standard FL methods such as FedAvg [12], FCL methods are developed to learn incremental classes/tasks in individual clients while tackling catastrophic forgetting problems, which are extensively studied in traditional memory rehearsal [43], [44], regularization-based techniques [7], [8], and dynamic architectures [9], [10], [45]. In a FL environment, these methods can be divided into two types. First, federated incremental class learning (FCIL) employs a hypernetwork to cope with the learning of new classes [46], or develops an enhancer knowledge distillation method to mitigate the imbalance between old and new knowledge [16], [47], [48]. Second, FCL methods allow each client to learn its private task sequence and exchange knowledge with other clients via a global Fisher matrix [23], model divisions methods (base parameters and adaptive parameters) [17], and signature task knowledge [18]. In global aggregation, however, both types of methods implicitly assume that all clients submit their latest local models, which are trained for their own task sequences.

*Clustered FL* methods aggregate models of similar training data. They divide participating clients into different

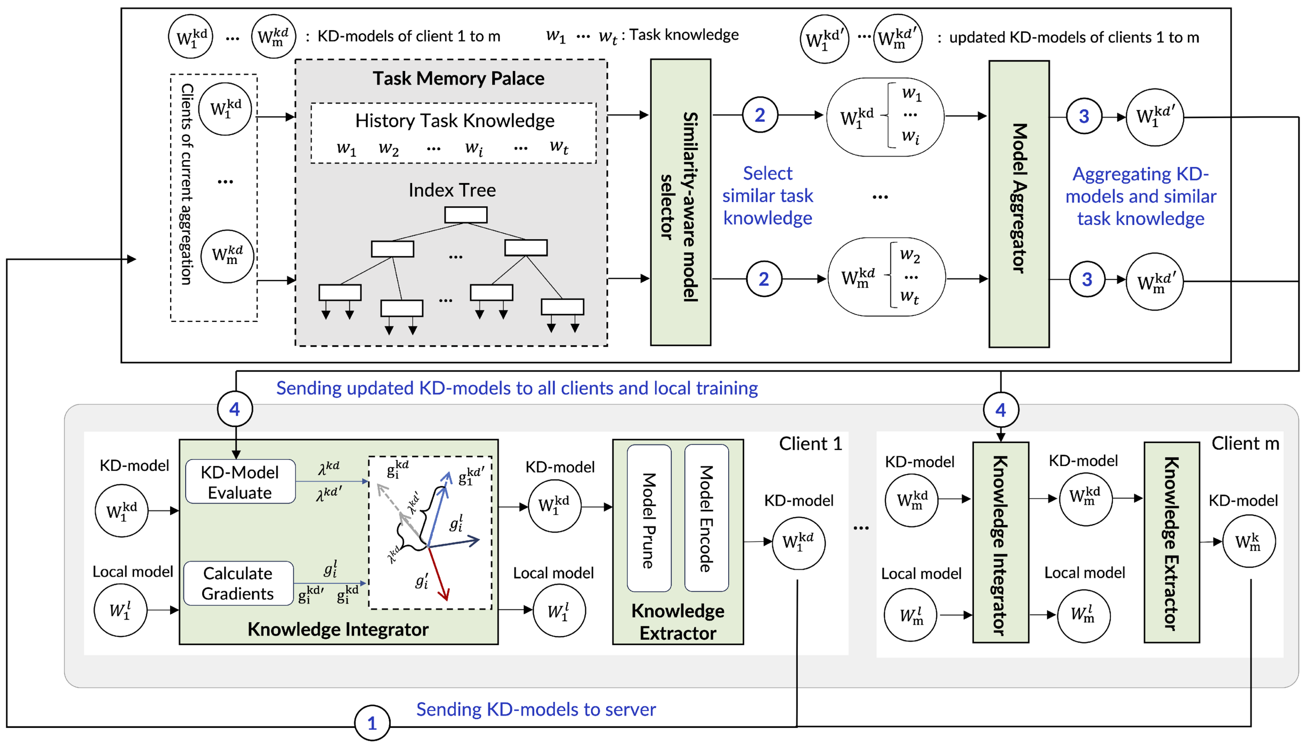

KD-models of client 1 to m
$w_1 \cdots w_t$: Task knowledge
updated KD-models of clients 1 to m
Clients of current aggregation
Task Memory Palace
History Task Knowledge
Index Tree
Similarity-aware model selector
Select similar task knowledge
Model Aggregator
Aggregating KD-models and similar task knowledge
Sending updated KD-models to all clients and local training
KD-model
Local model
KD-Model Evaluate
Calculate Gradients
Knowledge Integrator
Model Prune
Model Encode
Knowledge Extractor
Client 1
Client m
Sending KD-models to server

Fig. 2. Loci process and its two design objectives. We assume that there are $c$ clients participating in the training and $m$ ($m < c$) clients are selected to participate in the aggregation at the server side in each round.

clusters/groups, where each one contains clients with similar training data. Specifically, CFL [19] utilizes cosine-similarity to measure clients' models/gradients in clustering. FICA [20] employs a weight sharing mechanism to reduce the overhead of client clustering. PFE [49] trains each client to learn a sparse representation model before clustering. FLIC [50] designs an incremental clustering strategy for asynchronous FL. GradMFL [21] combines clustering with gradient memorization techniques to constrain the direction of client model updates. FlexCFL [51] groups clients based on their optimization objectives and decomposes the global optimization objective into sub-objectives of groups. ACFL [52] introduces a mean-shift clustering algorithm for clients and only selects a part of clients from each cluster in training with an auction mechanism. ClusterFLADS [53] first extracts neurons from each layer and then uses PCA dimensionality reduction to calculate the similarity of client models. Based on the assumption that the clients in the same group have similar tasks, these methods directly aggregate these clients' latest local models.

## III. Design of Loci

### A. Overview

We design Loci to enable accurate and efficient aggregation for heterogeneous tasks on federated edge devices. As shown in Fig. 2, Loci employs the *task memory palace* to address model heterogeneity across different clients. This data structure consists of two parts: (1) all clients' task knowledge is retained as compact model weights; and (2) an index tree groups similar tasks together and allows quick search of each client's most accurate relevant tasks. Using task memory palace, Loci integrates each client's model with the knowledge of past and peer tasks into a single model using five steps.

At step 1, the *knowledge extractor* employs the standard knowledge distillation method [54] to approximate the output probability distribution of each client's local model and transforms it into a compact network, termed *knowledge distillation (KD) model*. The KD-model is then sent to the server. At step 2, the server-side *heterogeneous-aware model selector* searches the task knowledge palace and finds the $k$ most similar task knowledge for each client. At step 3, the *model aggregator* sequentially aggregates each client's KD-model and the $k$ task knowledge using optimal transport [55], which ensures that the parameters aggregated at each position have similar functionality and thus maximizes the acquisition of beneficial information to enhance model accuracy. Subsequently, step 4 sends the updated KD-models to all clients. Each client first evaluates its KD-model before and after aggregation and obtains the evaluation parameters $\lambda^{kd}$ and $\lambda^{kd'}$. Its *knowledge integrator* then uses the updated KD-model, and retrains the local model as the student model using a few iterations. At each iteration, the integrator computes the task loss and distillation loss of the local model, and uses the relative sizes of $\lambda^{kd}$ and $\lambda^{kd'}$ as hyperparameters in the calculation of the final gradient $g_i'$, thus preventing the local model from negative knowledge transfer. Finally, after the training of the current task completes, the *knowledge extractor* extracts its knowledge and sends it to the server.

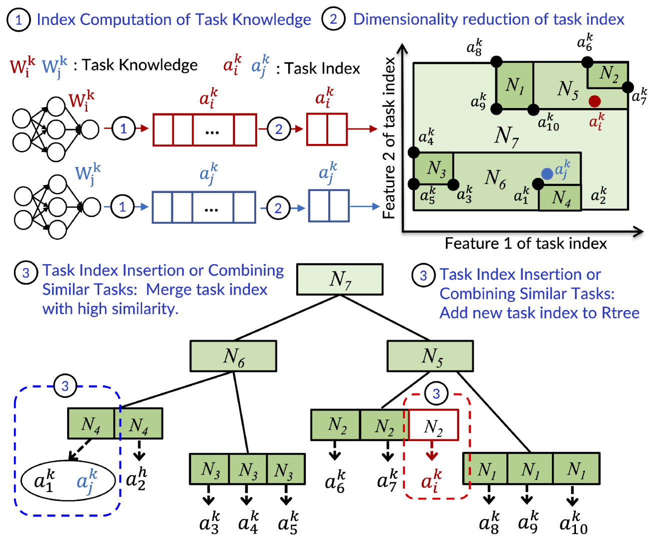

① Index Computation of Task Knowledge
② Dimensionality reduction of task index
$W_i^k$ $W_j^k$ : Task Knowledge
$a_i^k$ $a_j^k$ : Task Index
Feature 2 of task index
Feature 1 of task index
③ Task Index Insertion or Combining Similar Tasks: Merge task index with high similarity.
③ Task Index Insertion or Combining Similar Tasks: Add new task index to Rtree

Fig. 3. An example of adding new task knowledge to task memory palace.

During the whole training process, the KD-model used in Loci is much smaller than the client's local model, thus the computational cost is low when training the local model using the KD-model (as the teacher model). Moreover, the *knowledge extractor* prunes and encodes the trained KD-model into compact knowledge and transmits it to the server with low communication cost [56], [57], because the task-specific model weights are highly sparse and they are only communicated once upon creation. The extracted knowledge maintains most of the task's information for two reasons. First, it filters and deletes irrelevant model weights, while the weights retained in client $i$'s task knowledge $w_i$ can accurately recover the information of task $t+1$. Second, the pruning phase is architecture-independent and can be completed quickly on edge devices. Expanding the above knowledge extraction and the recovery process with structured pruning techniques (such as L1 norm or L2 norm filter pruning [58]) is feasible. We now introduce the task memory palace and four components of Loci.

## B. Task Memory Palace

In Loci, task memory palace stores the knowledge of all clients' tasks and employs an RTree to organize the knowledge in a bottom-up fashion, as illustrated in Fig. 3. Task memory palace is defined as a 3-tuple $(\mathbf{W}, \mathbf{A}, \mathbf{N})$, where $\mathbf{W} = \{w_1, \ldots w_t\}$ represents the knowledge of $k$ learned tasks, $\mathbf{A} = \{a_1, \ldots a_t\}$ is these tasks' indexes (each index contains two floating numbers and represents a point in the 2D plane), and $\mathbf{N}$ represents the minimum bounding rectangles (MBRs) used to form the Rtree. That is, each Rtree leaf node holds the index of a task's knowledge or a group of similar tasks' knowledge. Note that task knowledge has over 10 k dimensionalities but its index's dimensionality is usually less than 10 to support fast Rtree construction, updating, and searching. Task memory palace is designed with four objectives.

*Compact task knowledge:* Task memory palace transforms each task's corresponding model weights into its knowledge using model pruning techniques (e.g., compressed column storage). The knowledge corresponds to a small proportion of the weights with the highest values to retain the most important information and reduce memory footprint.

*Fast search of accuracy relevant task knowledge:* Task memory palace employs an Rtree to organize all task knowledge in a hierarchical way and groups similar task knowledge into the same node. Specifically, the construction of the RTree first groups similar task knowledge in *leaf* nodes, and then groups similar leaf nodes into one non-leaf nodes. Each non-leaf node contains pointers to multiple child nodes and their corresponding MBRs, which help efficiently index and search for relevant task knowledge. In Fig. 3's example, the spatially closed task knowledge $a_6$ and $a_7$ are group in leaf node $N_2$. Leaf nodes $N_1$ and $N_2$, which are close in the task knowledge, are grouped in non-leaf node $N_5$. Hence using the Rtree, task memory palace is able to quickly search a task's similar knowledge even under large client and task numbers.

*Updating of Rtree with new task knowledge:* Task memory palace supports incremental task knowledge updating via fast tree node addition/deletion. Specifically, the addition process has three steps. Step 1 transforms task knowledge $w_i$ into a lower-dimensional task index $a_i^k = f(w_i, X)$ and step 2 further employs PCA dimensionality reduction to reduce this index's dimensionality, because Rtree works effectively in low-dimensional space. Finally, step 3 searches $w_i$'s most relevant knowledge from the RTree.

Note that Loci uses multi-head cross-entropy loss $\mathcal{L}_{MCE}$ to estimate the relationship between these two task indices: $\mathcal{L}_{MCE}(a_i^k, a_j^k)$. If this relationship value exceeds a threshold $\phi$, the knowledge of two tasks is merged in the same node; otherwise, a new node is inserted into the Rtree to store the new task index $a_i^k$. Fig. 3 shows an example. Step 1 transforms the task knowledge $w_i, w_j$ into task index $a_i^k$ and $a_j^k$ and step 2 reduces the dimensionality of them to 2. Finally, step 3 shows two conditions of new task knowledge addition: if $a_j^k$ (represented by the black point) is similar to $a_1^k$, two tasks are merged; otherwise, $a_i^k$ (represented by the red point) is added as a new Rtree node.

The task memory palace is designed for low computation overheads and memory footprints. First, it employs an RTree to ensure logarithmic time complexity in searching, updating, and merging task knowledge. When task memory space has $N$ task knowledge, the time complexity is just $\mathcal{O}(logN)$. Second, it only stores compressed task knowledge and dimensionality-reduced task indexes. Each task knowledge and each index have less than 5% and 0.05% of the original model's memory footprint. Third, it merges similar task knowledge to support continual learning of large numbers of tasks.

## C. Similarity-Aware Model Selector

The selector is designed to identify similar task knowledge for each client's submitted model. It takes the $m$ KD-models $\mathbf{W}^{kd} = \{\mathrm{W}_1^{\mathrm{kd}}, \mathrm{W}_2^{\mathrm{kd}}, \ldots, \mathrm{W}_{\mathrm{m}}^{\mathrm{kd}}\}$ from the current clients and the task knowledge in the task memory palace as input and outputs $m$ sets. Each set comprises a client's KD-model $\mathrm{W}_{\mathrm{i}}^{\mathrm{kd}} \in \mathbf{W}^{kd}$

and $h(k < n)$ task knowledge $\{w_1, w_2, \ldots, w_h\}$ that have the highest similarity to $\mathrm{W_i^{kd}}$. The selector compares the similarity of task knowledge obtained from the task memory palace to ensure that they are sufficiently similar. For this, Loci provides two similarity measurement methods.

*Similarity measurement based on weight:* Based on the parameter space theory [59], this method measures the similarity between two models in their parameter space. That is, two models are similar if the geometry distance between their parameters is small. Specifically, the weight metric ($S_w()$) calculates the similarity between the parameters of the two models on a per-layer basis, taking into account their corresponding parameters. Given the disparity in the sizes of the parameters across the layers of a neural network, Loci averages the similarities across all layers. Specifically, assuming that the model has $L$ layers and $||$ denotes the magnitude of a vector, the weight-based similarity measurement $S_w(\mathrm{W_i^{kd}}, w_j)$ is calculated as:

$$S_w\left(\mathrm{W_i^{kd}}, w_j\right) = \sum_{\ell=1}^{L} \frac{|\mathrm{W_i^{kd,(\ell)}} - w_j^{(\ell)}|}{|\mathrm{W_i^{kd,(\ell)}}|} \quad (1)$$

*Similarity measurement based on activation vector:* Based on representation learning theory [60], which states that large models can be mapped to low-dimensional feature space while preserving their behaviors, this method first transforms original models into activation vectors of fewer dimensions. It then calculates the distance between two vectors to measure the similarity of corresponding models. Specifically, the vector metric ($S_a()$) is designed to compare the output of activation vectors between two models. Specifically, for a client's KD-model $\mathrm{W_i^{kd}}$ and a task knowledge $w_j$, their corresponding $S_a(\mathrm{W_i^{kd}}, w_j)$ is calculated as:

$$S_a\left(\mathrm{W_i^{kd}}, w_j\right) = \mathcal{L}_{MCE}\left(f\left(\mathrm{W_i^{kd}}, X\right), f(w_j, X)\right) \quad (2)$$

Where $X$ represents some sample features stored on the server, $f(\mathrm{W_i^{kd}}, X)$ and $f(w_j, X)$ denote the activation vectors of $\mathrm{W_i^{kd}}$ and $w_j$ respectively, and $\mathcal{L}_{MCE}()$ is the multi-head cross-entropy loss function.

## D. Model Aggregator

The model aggregator aims to transfer the information from each client's selected task knowledge to its local KD-model to improve its accuracy. In transfer, the vanilla averaging may not map the parameters directly to each other and lead to a large discrepancy in neurons at the same position. For instance, the $p$th neuron in Model A may be more similar to the $q$th neuron rather than the $p$th neuron in Model B. To this end, the model aggregator employs optimal transport [55] to align the task knowledge to the client's KD-model layer by layer, and then performs vanilla averaging of the model parameters. The optimal transport method [55] is able to find the optimal mapping between two probability distributions. In the aggregator, these two distributions are the KD-model and the task knowledge, and the optimal mapping corresponds to the minimum difference of parameters between them. Hence after mapping, similar model parameters are moved to the same group and then aggregated.

This alignment can be represented by a permutation matrix. Formally, suppose the task knowledge $w_j$ corresponding to the KD-model $\mathrm{W_i^{kd}}$, which includes $L$ layers of parameters. After the parameters of the first $\ell - 1$ layers are aligned, the parameter arrangement for the $\ell$th layer $W_j^{k,(\ell)}$ is calculated as:

$$\widetilde{w}_j^{(\ell)} = \mathrm{diag}\left(1/\boldsymbol{\beta}^{(\ell)}\right) \cdot \mathbf{T}^{(\ell)^\top} \cdot \widehat{w}_j^{(\ell)} \quad (3)$$

Where $\boldsymbol{\beta}^{(\ell)} = \mathbf{1}_{o^{(\ell)}}/o^{(\ell)}$ represents a random probability vector, $o^{(\ell)}$ is the output size of the current layer; $\mathbf{T}^{(\ell)}$ denotes the transportation matrix for the current layer, which can be obtained by solving an Optimal Transport (OT) problem [61]; $\widehat{w}_j^{(\ell)}$ represents the variable obtained by projecting the $\ell$th layer of the task knowledge into the client's KD-model. The computation of $\widehat{w}_j^{(\ell)}$ is necessary because the reordering in the previous layer $(\ell - 1)$ mutates the parameters in layer $\ell$. Hence the computations is based on layer $(\ell - 1)$'s parameters $w_j^{(\ell-1)}$ and their corresponding optimal transport matrix $\mathbf{T}^{(\ell-1)}$:

$$\widehat{w}_j^{(\ell)} = w_j^{(\ell-1)} \cdot \mathbf{T}^{(\ell-1)} \cdot \mathrm{diag}\left(1/\boldsymbol{\beta}^{(\ell-1)}\right) \quad (4)$$

After alignment, the client's $k$ task knowledge are adjusted to $\{\widetilde{w}_1, \widetilde{w}_2, \ldots, \widetilde{w}_h^k\}$. The final aggregated model $\mathrm{W_i^{kd'}}$ is computed as:

$$\mathrm{W_i^{kd'}} = \frac{1}{2} * \mathrm{W_i^{kd}} + \frac{1}{2k} * \left(\widetilde{w}_1^k + \widetilde{w}_2 + \cdots + \widetilde{w}_h\right) \quad (5)$$

## E. Knowledge Integrator

The integrator runs on resource-constrained edge devices and it aims to prevent negative knowledge transfer from the updated KD-model to the client's local model. To this end, the integrator compares the KD-model before and after global aggregation, and transfers the knowledge of better one to the local model $\mathrm{W_i^l}$ with three steps. Step 1 uses local samples $X_i, Y_i$ to calculate top 1 scores of $\mathrm{W_i^{kd}}, \mathrm{W_i^{kd'}}$ and get $\lambda^{kd}$ and $\lambda^{kd'}$. Step 2 calculates the gradients according to (6).

$$\begin{aligned} g_i^l &= \nabla\mathcal{L}_{CE}\left(f\left(\mathrm{W_i^l}, X_i\right); Y_i\right) \\ g_i^{kd} &= \nabla\mathcal{L}_{KL}\left(f\left(\mathrm{W_i^l}, X_i\right); f\left(\mathrm{W_i^{kd}}, X_i\right)\right) \\ g_i^{kd'} &= \nabla\mathcal{L}_{KL}\left(f\left(\mathrm{W_i^l}, X_i\right); f\left(\mathrm{W_i^{kd'}}, X_i\right)\right) \end{aligned} \quad (6)$$

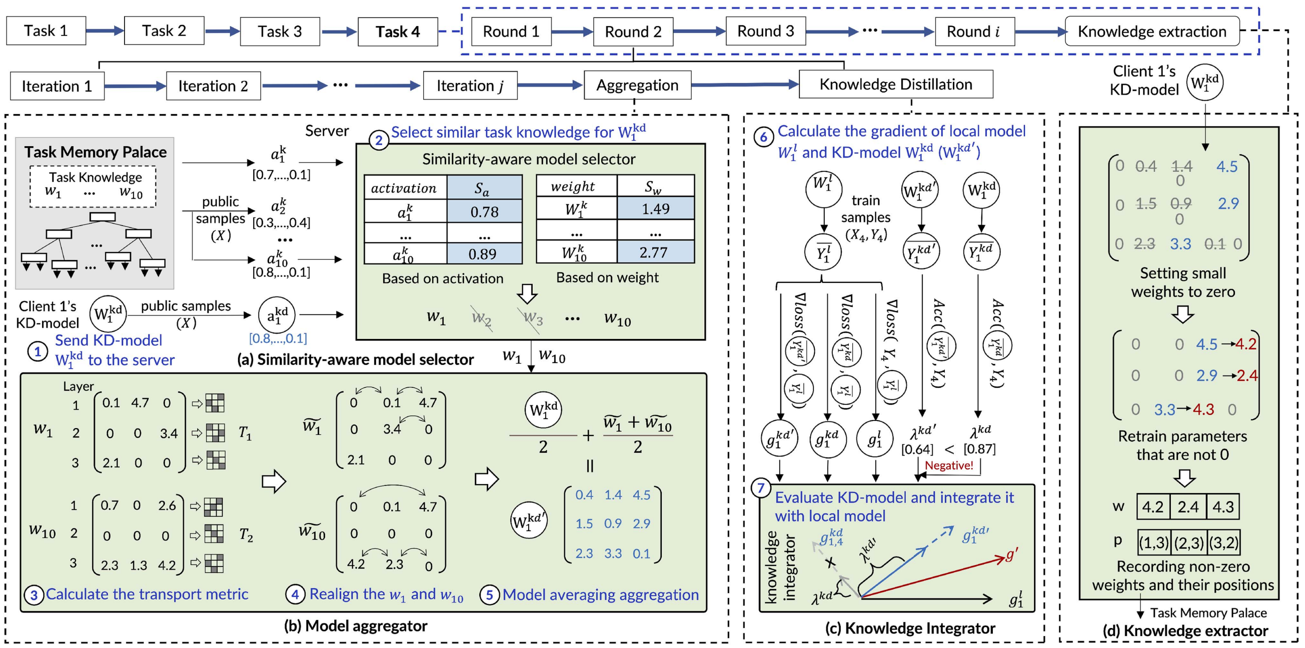
where $\mathcal{L}_{CE}()$ refers to the cross-entropy loss function, and $\mathcal{L}_{KL}()$ represents the Kullback-Leibler (KL) divergence loss. Based on the comparison between $\lambda^{kd}$ and $\lambda^{kd'}$, step 3 judges the integration as negative knowledge transfer if $\lambda^{kd'} < \lambda^{kd}$. Subsequently, the KD-model with a higher value is chosen to participate in the final gradient computation according to (7).

$$g_i' = g_i^l + \gamma * \lambda^{kd} * g_i^{kd} + (1 - \gamma) * \lambda^{kd'} * g_i^{kd'} \quad (7)$$

where $\gamma$ is used to determine whether $\lambda_{kd} \leq \lambda_{kd'}$ or not. If so, the value of $\gamma$ is set to 0; otherwise, it is set to 1.

## F. Knowledge Extractor

The extractor is designed to quickly maintain most of the task's information in the obtained knowledge. For such an



Task 1
Task 2
Task 3
Task 4
Round 1
Round 2
Round 3
Round i
Knowledge extraction
Iteration 1
Iteration 2
Iteration j
Aggregation
Knowledge Distillation
Client 1's KD-model
Server
Task Memory Palace
Task Knowledge
Select similar task knowledge for W1kd
Similarity-aware model selector
Based on activation
Based on weight
public samples (X)
Send KD-model W1kd to the server
(a) Similarity-aware model selector
Calculate the transport metric
Realign the w1 and w10
Model averaging aggregation
(b) Model aggregator
Calculate the gradient of local model W1l and KD-model W1kd (W1kd')
train samples (X4, Y4)
Negative!
Evaluate KD-model and integrate it with local model
knowledge integrator
(c) Knowledge Integrator
Setting small weights to zero
Retrain parameters that are not 0
Recording non-zero weights and their positions
Task Memory Palace
(d) Knowledge extractor

Fig. 4. An example process of learning task 4 at client 1. (a) and (b) show how to aggregate client 1's local model; (c) shows how to distill aggregated knowledge from this local model; and (d) shows how to extract the task knowledge.

issue, the extraction process utilizes the weight-based pruning technique [56], [57] to remove most of the model weights whose absolute values are lower than a given threshold and retain the remaining ones (e.g., 10%) as the signature task knowledge. For example, this pruning technique can remove over 90% of the weights in VGG-16 [62] and causes negligible accuracy losses. This process has four steps: step 1 first trains the KD-model $\mathrm{W}_{\mathrm{i}}^{\mathrm{kd}}$ until convergence; step 2 selects a ratio $\rho$ (e.g., 10%) of weights with the highest values; and step 3 fine-tunes the weights in $\mathrm{W}_{\mathrm{i}}^{\mathrm{kd}}$ while keeping other model weights unchanged. Finally, step 4 encodes the pruned KD-model, recording the magnitude and position of non-zero weights.

Note that the extractor is only executed once after a task's training completes and sends the compact knowledge of this task to the task memory palace. That is, the model pruning in the extractor works independently with the client's local training and hence has no influence on the local model's accuracy.

## G. Running Example

*This section presents an example of learning a new task with Loci:* Fig. 4 demonstrates client 1's learning process of task 4 (tasks 1 and 3 have been learned). This process includes $i$ rounds and each round consists of $j$ local training iterations and one aggregation. At step 1 (round 2), the server receives client 1's KD-model $\mathrm{W}_1^{\mathrm{kd}}$ and forwards it to the similarity-aware model selector. At step 2, the selector first computes the indexes $a_1^k$ to $a_{10}^k$ of the task memory palace, and then calculates their similarity to $\mathrm{W}_1^{\mathrm{kd}}$ based on activation (2) or weight (1). This step selects the two most similar task knowledge and transmits them to the model aggregator. In the following three steps, the aggregator first calculates the optimal transport matrices $T_1$ and $T_{10}$ for similar task knowledge $w_1$ and $w_{10}$(step 3). Subsequently, it uses these matrices to realign the parameters of $w_1$ and $w_{10}$ and produces $\widetilde{w}_1$ and $\widetilde{w}_{10}$ according to (3) and (4) (step 4). Finally, the aggregator generates the aggregated model $\mathrm{W}_1^{\mathrm{kd}'}$ using federated averaging and sends it back to client 1 (step 5).

After completing round 2, Loci prevents negative knowledge transfer when integrating the KD-models with client 1's local model. Specifically, at step 6, the knowledge integrator*first calculates the prediction results of the KD models $\mathrm{W}_1^{\mathrm{kd}}$ and $\mathrm{W}_1^{\mathrm{kd}'}$. It then computes gradient $g_1^l$, $g_1^{kd}$, and $g_1^{kd'}$ according to (6). At step 7, the integrator calculates the gradient $g_1^l$ using these three gradients (7) and updates the local model. Finally, the* knowledge extractor prunes and encodes client 1's KD-model $\mathrm{W}_1^{\mathrm{kd}}$ and sends it back to the task memory palace.

## IV. Implemention

*Portability:* We implemented Loci on python version 3.7.10. We added 2372 lines of code (LoC), modified 175 LoC, and removed 279 LoC of the mainline code. It's index tree and PCA dimensionality reduction is implemented using rtree 4.6.3 and scikit-learn 0.1.2. Meanwhile, Loci employs 10 regular edge devices (including four Jetsons and six Raspberry Pis) as clients, and the parameter interaction among different clients is based on flwr 0.17.0.

*Applicability to DNNs:* Loci has designed the *DNNAdapter* interface to better adapt and manage various DNNs. This interface comprises the following three key functions: 1. loadModel(): this function is responsible for loading the DNN and initializes all parameters within the model. 2. setModel(): this function provides various elements required during the training of different

DNNs. These elements include but are not limited to the number of iterations, learning rate, and so on. 3. unifiedForward(): this function serves to standardize the parameter transmission in the 'forward()' method of different DNNs. Regardless of how the original parameters in a DNN's 'forward()' are designed, unifiedForward() ensures a consistent and simplified forward propagation interface for clients. The current implementation of Loci supports prevalent models across four applications: for CV applications, it supports ResNet, MobileNet, and ViT series models; for NLP applications, it supports RNN, TextCNN, and Transformer series models; for graph applications, it supports GCN and GAT series models; and for multimodal applications, it supports CLIP series models.

Furthermore, Loci not only supports DNNs with multiple training strategies but is also compatible with various federated learning frameworks. To achieve this goal, we designed the *CLUser* interface. This interface integrates various continual learning methods, allowing DNNs to select and use the different training strategies. In addition, we also implemented the *FrameworkIntegrator*, which offers compatibility with several mainstream federated learning frameworks, such as Flower [63] and DisPFL [64], ensuring that Loci can flexibly adapt to different federated learning scenarios and requirements.

## V. Evaluation

### A. Experimental Settings

*Testbed:* To demonstrate the cross-platform capability of Loci, we selected four commodity edge platforms. The Jetson TX2 is equipped with a 256-core NVIDIA Pascal GPU and 8 GB memory. The Jetson Xavier NX features a 384-core NVIDIA Volta GPU and 16 GB memory. The Jetson AGX incorporates a 512-core Volta GPU with Tensor Cores and 32 GB memory. The Raspberry Pi 4 Model B has a quad-core 64-bit ARM Cortex-A72 processor and 4 GB (8 GB) memory. All platforms operate on Ubuntu 18.04.5 LTS and run DNNs in PyTorch 1.9.0 (Python 3.6.9).

*Datasets and DNN models:* We select nine representative FL and continual learning benchmarks to evaluate Loci [74]. Following their settings, the training and test points in each dataset are divided into different tasks without overlap. In the NLP application, the ASC dataset is constructed by merging the HL5Domains [68], Liu3Domains [69], Ding9Domains [70], and SemEval14 datasets. Table I lists the dataset, client task information, and model for different applications. Specifically, for CV applications, we deploy the 6-CNN and 10-CNN models on 4 Raspberry Pi devices, and ResNet and ViT series models on 6 Jetson devices. For NLP applications, we deploy TextCNN and RNN models on 4 Raspberry Pi devices, and deploy LSTM, Bert and 2 GPT series models on 6 Jetson devices. Both GPT series models are trained using LoRA. For graph applications, we deploy 2-layer GCN models on 4 Raspberry Pi devices, and GCN and GAT models on 6 Jetson devices. Finally, for multimodal applications, we deploy CLIP-RN50 models on 2 Raspberry Pi devices, and CLIP-RN50 and CLIP-RN101 models on 3 Jetson devices. In addition, when using LoRA to fine-tune the GPT series models, only a small percentage of parameters are updated

TABLE I
DNN and Dataset Setting for Different Applications

| Application | Dataset | Task Number | Model |
|---|---|---|---|
| CV | Cifar100 [63] | 10 | Densenet, 6-CNN, 10-CNN, TinyPiT, WideResnet, Resnet18,MobilenetV2 |
| | MiniImageNet [64] | 10 | |
| | TinyImageNet [65] | 20 | |
| NLP | Online-Shopping | 10 | GPT2, GPTNeo, Bert, TextCNN-1 TextCNN-2, RNN, LSTM, MoELSTM |
| | ASC [66], [67], [68] | 19 | |
| | DSC [69] | 10 | |
| Graph | MiniGC | 10 | GCN-2, GCN-3, GCN-4, GCN-5, GCN-6, GAT-3, GAT-4 |
| | Reddit [70] | 8 | |
| Multimodal | Cifar100-Text [71] | 5 | CLIP-ViT-B/3, CLIP-RN50, CLIP-RN101 |

in Loci: for the GPT2 model with 125 M parameters and the GPTNeo model with 127 M parameters, the percentage is about 1% and hence about 1.2 M parameters are trained.

*Task and dataset assignment:* Following the setting of FedRep [15], we set the distributions of tasks and datasets to ensure their heterogeneity across clients. In evaluation, each client has a random sequence of tasks, where each task includes 2 to 5 classes, and each class's data is composed of 5% to 10% of the training samples. The number of each client's tasks is shown in Table I.

*Compared baselines:* Three categories of state-of-the-art FL techniques are compared. *FL methods with homogeneous/heterogeneous models:* (1) FedMD [12] uses public datasets to update the distillation model during aggregation. (2) FedKD [13] designs various distillation losses based on the network layer on the client-side. (3) FedKEMF [15] considers merging all teacher networks during aggregation, and uses a common dataset to distill a better server-side global network. (4) FedGKT [14] designs a variant of the alternating minimization approach to train small models on edge nodes and periodically transfer their knowledge by knowledge distillation to a large server-side model.

*FCL methods:* (1) GLFC [16] designs the multi distillation loss to balance the forgetting of old classes and distills consistent inter-class relations across tasks. (2) FedWEIT [17] uses masks to divide model weights into two categories: base weights and adaptive weights. It stores the adaptive weights of all past tasks from every client in a knowledge base, allowing clients to easily access previous information. (3) FedKNOW [18] uses model weight as task knowledge to obtain the past task information and uses quadratic programming to calculate the gradient. (4) FedViT [75] retains training samples of vision transformers and uses them in new task training. (5) TFCL [76] uses knowledge aggregation and model decomposition to learn similar tasks within clients. (6) AFFCL [77] uses a generative model to estimate data distribution and employs feature replay and knowledge consolidation to prevent catastrophic forgetting.

*Clustered FL methods:* (1) CFL [19] divides clients into clusters based on the cosine-similarity of their gradients/parameters, and performs global aggregation for clients in the same cluster.

(2) IFCA [20] estimates the clustering identity of clients, optimizes the model parameters for each cluster, and also allows parameter sharing among different clusters; (3) GradMFL [21] introduces a hierarchical cluster to organize clients and supports knowledge transfer among different hierarchies.

*Evaluation metrics:* In evaluation, we use both model accuracy and training time (measured in hours). The accuracy metric is the top-1 accuracy on test data points: the top predicted class (the one with the largest probability) is the same as the actual class label. In continual learning, the reported accuracy for task $t_m$ is derived from the mean accuracy across all $m$ tasks learned up to that point.

## *B. Evaluations of Model Accuracy Under Heterogeneous Tasks*

In this evaluation, we follow the setting of hyperparameters in the widely used benchmark of lost domain generalization [78]. For each dataset, this benchmark searches the optimal hyperparameters that produce the highest accuracy. The detailed settings have two parts:

The first part is the *common settings for all methods*. The evaluation is conducted on a cluster of heterogeneous edge devices: 2 Jetson TX2, 2 Jetson AGX, 2 Raspberry Pi(8 GB), and 4 Raspberry Pi(4 GB) devices for CV, NLP and graph applications; and 2 Jetson AGX, 1 Jetson TX2, and 2 Rasberry Pi(8 GB) devices for the multimodal application. In hyperparameter search, we set the search range for the aggregation rounds between 5 to 15 and for training iterations between 5 to 150. For all datasets in different applications, we set the number of global aggregation rounds as 1. For FedKEMF, FedKD, and Loci, each round includes 5 local iterations and 5 global knowledge integration iterations. For FedMD, each round comprises 5 training iterations on the public dataset and 5 iterations on the client's private dataset. For other methods, each round has 10 local training iterations.

The second is the *settings for specific applications*. For the hyperparameter search pertaining to each baseline method, we determine the bounds of the search space as 1/2 and 2 times the parameter value from its original evaluation. In the FL methods with heterogeneous models and Loci, we choose GEM as the local continual learning technique in clients. For IFCA, the initial number of clusters is set to 3. For Loci, the search space for the number $h$ of candidate models and the number $k$ of the most similar candidate models during aggregation is defined as $\{10, 20, 40\}$ and $\{5, 10, 15\}$, respectively.

*Specific hyperparameter in task memory palace:* Loci merges similar tasks in the task memory palace to improve its scalability. This process has two hyperparameters: (1) a similarity function to calculate the similarity between tasks; and (2) a similarity threshold to determine whether two tasks should be merged. We note that commonly used similarity functions are mean squared error (MSE), KL divergence, and multi-head cross-entropy. Loci chooses multi-head cross-entropy, because MSE is mainly used for regression problems and KL divergence is more sensitive to data. Our evaluation of Table I's 9 datasets shows that KL divergence has large or small values when processing datasets of large discrepancy, thus may lead to inaccurate calculations of task similarities. In contrast, multi-head cross-entropy has moderate values in task similarity calculation and hence it is robust to data discrepancy.

TABLE II
COMPARISON OF ACCURACY (%) AND ACCURACY IMPROVEMENT (%) ON DIFFERENT APPLICATIONS AND DEVICES

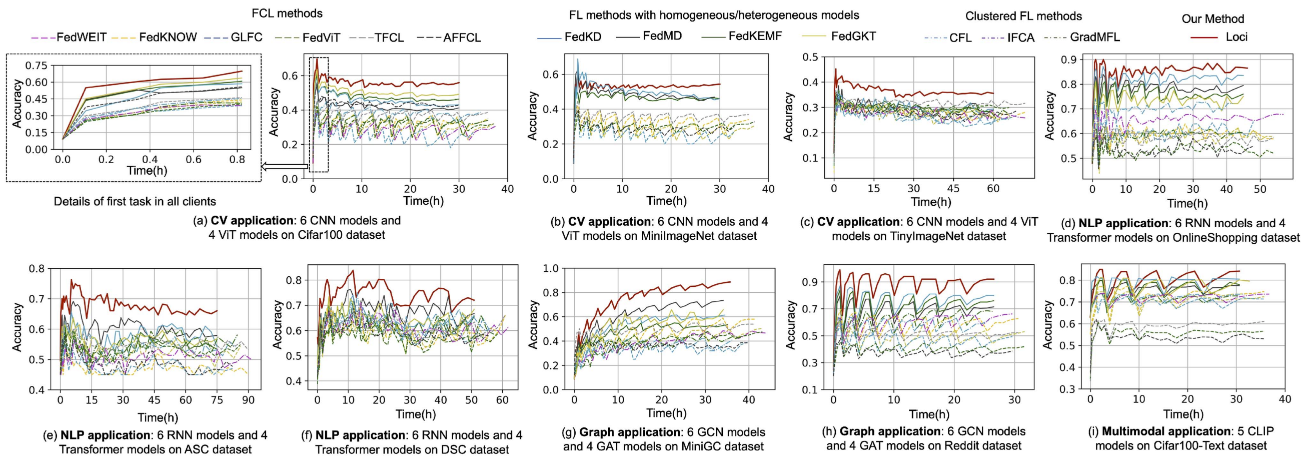
| | CV | | NLP | | Graph | | Multimodal | |
|---|---|---|---|---|---|---|---|---|
| **device** | Acc | Imp | Acc | Imp | Acc | Imp | Acc | Imp |
| Jetson-AGX | 50.2 | 34.2 | 76.1 | 21.1 | 90.1 | 55.2 | 84.4 | 16.4 |
| Jetson-NX | 49.7 | 35.7 | 74.9 | 20.9 | 89.6 | 54.9 | 84.2 | 15.9 |
| Jetson-TX2 | 48.4 | 36.1 | 74.3 | 21.3 | 89.7 | 54.3 | 83.9 | 16.3 |
| Raspberry | 47.9 | 38.2 | 74.3 | 21.8 | 89.4 | 56.7 | 83.7 | 17.9 |

In addition, Loci's similarity threshold is domain-specific. Its value depends on the dimension of the compressed vector, which varies across different applications (domains) and models. For each application, we first test the impact of different thresholds on accuracy under a fixed storage limit (1 GB), and then select the threshold that achieves the highest accuracy. For CV, NLP, Graph, and multimodal applications, we set the task similarity thresholds as 0.3, 0.2, 0.35, and 0.4, respectively.

*Comparison results:* Fig. 5 demonstrates the comparison results of Loci and baseline techniques. We have two key observations.

*Impact of negative knowledge transfer:* All baseline methods suffer from negative transfer due to task heterogeneity. Specifically, the six FCL methods are designed to aggregate models with the same network structure, thus leading to the lowest accuracies. The three clustered FL methods aggregate local models of similar tasks, and result in low accuracies under scenarios with a large number of tasks (Fig. 5(c)) or complex tasks (Fig: 5(i)). FedKD, FedMD, FedGKT and FedKEMF have higher accuracies, because they can use knowledge distillation to aggregate the models with heterogeneous network structures. In contrast, Loci prevents negative knowledge transfer using task-oriented aggregation, thus avoiding unrelated task information in affecting local model training.

*Impact of heterogeneous edge devices:* The FCL methods take 14.13% longer training time than other methods because they have larger local models and thus have a slower local training process on Raspberry Pi (the slowest device with CPU cores). In addition, CFL, IFCA, and GradMFL require re-partitioning of client clusters before each aggregation, thus causing extra computational time (approximately 20% of the total training time). In contrast, Loci takes shorter time than baseline methods, because it employs compact networks to transmit task knowledge. Table II lists the converged accuracy (Acc) and accuracy improvement (Imp) of Loci compared to the baseline methods on four edge devices. We can see that Loci still maintains high accuracies on Raspberry PI with the smallest amount of resources and achieves more accuracy improvements compared to baseline methods. This is because some methods, such as FedWEIT, have large memory footprints and cause out-of-memory errors. We can also observe that Loci achieves the largest accuracy



(a) **CV application**: 6 CNN models and 4 ViT models on Cifar100 dataset
(b) **CV application**: 6 CNN models and 4 ViT models on MiniImageNet dataset
(c) **CV application**: 6 CNN models and 4 ViT models on TinyImageNet dataset
(d) **NLP application**: 6 RNN models and 4 Transformer models on OnlineShopping dataset
(e) **NLP application**: 6 RNN models and 4 Transformer models on ASC dataset
(f) **NLP application**: 6 RNN models and 4 Transformer models on DSC dataset
(g) **Graph application**: 6 GCN models and 4 GAT models on MiniGC dataset
(h) **Graph application**: 6 GCN models and 4 GAT models on Reddit dataset
(i) **Multimodal application**: 5 CLIP models on Cifar100-Text dataset

Fig. 5. Comparison of model accuracy and training time between Loci and 13 baseline methods.

improvement in graph applications. This is because graph data has sparse and complex relationships between nodes and edges, worsening the accuracy degradation due to the negative transfer among heterogeneous tasks. In contrast, Loci transfers knowledge among similar tasks and thus effectively mitigates such problem.

Compared to baseline methods, Loci improves accuracy by an average of 36.57%, 21.4%, 55.18%, and 16.76% in CV, NLP, graph, and multimodal applications, respectively. Overall, Loci improves accuracy by an average of 32.48%.

## C. Evaluation of Communication and Computation Cost

This section's evaluation compares different methods' communication costs based on the gRPC framework [79]. In system implementation, gRPC serializes the communication content (e.g., model parameters or samples) into binary for network transmission and measures the cost as the data size of transmitted content. Note that for GPT series models, only part of the parameters participating in LoRA training are transmitted.

Table III reports the comparison results. We can observe that the methods using knowledge distillation (FedMD, FedKD, FedKEMF, and FedGKT) have smaller communication costs than other baseline methods, because they upload entire local models in aggregation. In particular, FedWEIT causes the largest communication cost because it needs to transmit all clients' task knowledge to each client. GradMFL has the second largest cost because it transmits additional information of clients' historical training samples. In graph applications, FedMD has the largest communication cost because it transmits clients' local samples and outputs to the server, and both contents have large data size in graphs. In contrast, Loci has the lowest communication costs (83.6% lower on average) because it only transmits tasks' specific knowledge as compact networks.

TABLE III
COMPARISON OF COMMUNICATION COST IN DIFFERENT APPLICATIONS

| | CV | NLP | Graph | Multi-modal |
|---|---|---|---|---|
| FedWEIT | 64.9GB | 4.07GB | 59.4MB | 112GB |
| FedKNOW | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| GLFC | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| FedViT | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| TFCL | 30.9GB | 1.14GB | 22.4MB | 52.1GB |
| AFFCL | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| CFL | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| IFCA | 46.4GB | 2.26GB | 33.6MB | 78.1GB |
| GradMFL | 51.3GB | 2.58GB | 42.5MB | 88.7GB |
| FedKD | 10.8GB | 0.56GB | 4.64MB | 25.6GB |
| FedMD | 4.77GB | 0.44GB | 337MB | 11.8GB |
| FedKEMF | 10.8GB | 0.56GB | 4.64MB | 25.6GB |
| FedGKT | 4.78GB | 0.59GB | 3.76MB | 10.7GB |
| Loci | **4.63GB** | **0.22GB** | **1.87MB** | **10.25GB** |

*Discussion of network bandwidths:* In a distributed training environment, network bandwidth is a key factor that influences communication time. Based on the evaluation of the Cifar100 dataset, we test all methods' total communication time under four distinct network bandwidths ranging from 50 KB/s to 1 MB/s. As shown in Fig. 7, When the bandwidth is limited (100 KB or 200 KB), methods (that is, GLFC, FedWEIT, FedKNOW, CFL, IFCA, GradMFL, TFCL) that directly upload local models take nearly 40 additional hours compared to other methods. The two methods (FedWEIT and GradMFL) with the largest communication cost take 60 hours. This communication time is close to 1.5 times of their training time. Overall, Loci consumes the least communication time compared to other baselines, and saves more communication time when the network bandwidth decreases. For example, Loci saves 25 hours when the bandwidth is 100 KB/s.

*Discussion of computational costs:* Following the setting of the previous evaluations, we break down Loci's computational cost into four parts according to its components: similarity-aware model selector, model aggregator, knowledge integrator, and knowledge extractor. Fig. 6 shows Loci's average training

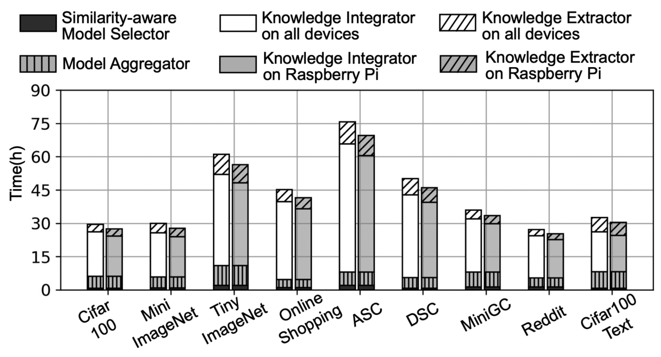


Fig. 6. Breaking down Loci's computational costs on its components.

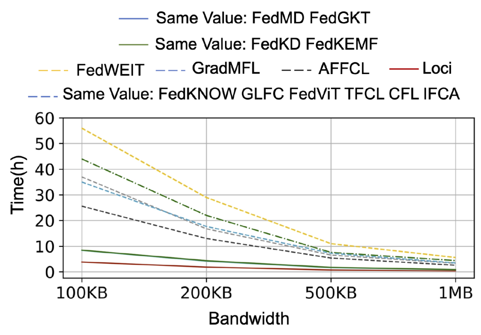


Fig. 7. Comparison of communication time under different bandwidths.

time (hour) on all datasets. We can see that the knowledge integrator takes most of the training time because it needs to perform expensive gradient calculations. The other three components search and select task knowledge using the task memory palace and they take less than 20% of the total computational cost. Fig. 6 also shows the training time on Raspberry Pi. We can see this slowest device takes 76.29% of the total training time, because it needs to compute gradients using CPU cores.

## D. Scalability to Task and Client Numbers

In this section, we compare the average accuracy and the searching time of similar models using Loci and FedKD /FedMD / FedKEMF / FedGKT. Note that we do not consider GLFC, FedWEIT and FedKNOW, because their training causes out-of-memory errors under large task/client numbers.

*Number of tasks:* In this evaluation, we combine MiniImageNet, Cifar100, FC100 and TinyImage datasets to construct a new dataset with 50 tasks, where each task has 10 classes. We follow the setting of FedRep [15] and randomly distribute this dataset to 10 clients for task heterogeneity(each client has 50 heterogeneous tasks). We evaluate Loci and four baseline methods when the number of tasks increases from 1 to 50. Fig. 8(a) shows that Loci consistently delivers higher accuracies than other baseline methods, because Loci can effectively prevent negative transfer. Fig. 8(b) shows the searching time of similar tasks in Loci and baseline methods. We can see Loci considerably reduces searching time by 50x, because it employs an index tree to organize tasks in leaf nodes and provides logarithm time complexity in querying.

*Number of clients:* In this evaluation, we combine MiniImageNet, Cifar100, FC100, and TinyImage to generate training and test sets, and employ the FedRep allocation strategy [15] to assess the impact of client number on different methods. Fig. 9(a)'s result shows that when the client number increases from 10 to 200, the accuracies of all methods linearly decrease. This is because the training set is fixed and each client's number of training samples linearly decreases when the client number increases, while the client has a fixed number (10) of tasks and test set. We can also observe that Loci achieves more accuracy improvement as client number increases. This is because more clients lead to higher task heterogeneity, and the baseline methods suffer from severe negative transfer. In addition, under large client numbers, Loci also achieves the highest accuracy (Fig. 9(b)) and saves more searching time (Fig. 9(c)). Overall, Loci searches similar models using only 5.51 seconds, and saves the searching time by over 95% compared to other methods.

## E. Discussion of Different Training Algorithms and Decentralized Setting

*Applicability to continual learning algorithms:* We test six algorithms belonging to three typical categories: (1) regularization-based techniques: EWC [7] and MAS [80]; (2) memory-based techniques: GEM [43] and MEGA [44]; and (3) dynamical architecture-Based techniques: Packnet [45] and ChannelGate [9]. We apply these methods in the local training of FedKD, FedMD, FedKEMF, FedGKT and Loci. FedWEIT, FedKNOW, and GLFC still use their proposed local training techniques. We use Cifar100 dataset with 10 tasks.

Fig. 10 shows that Loci maintains the highest accuracy compared to other methods, and it has larger accuracy improvements when using EWC, MAS, and MEGA as the local training algorithms. Some baseline methods such as FedMD, FedKD, FedKEMF and FedGKT show much lower accuracies, because the applied continual learning algorithms amplify the negative transfer. Overall, Loci improves the accuracy by an average of 48.7%.

In this evaluation, we also evaluate Loci in a decentralized setting using Cifar100 and MiniImageNet datasets. Without the central parameter server, Loci constructs a local task memory palace for each client and updates it using task knowledge from neighboring clients. We compare Loci with three latest decentralized learning methods: (1) Token gossip [81] learning enhances inter-client interactions using sampling-based compression and token techniques; (2) PENS [82] facilitates interactions between clients with similar data; and (3) DisPFL [64] combines decentralized and personalized FL to mitigate the impact of limited communication. Fig. 11 shows that Loci achieves much higher accuracies in both datasets. This indicates that the local task memory palace in each client can effectively mitigate negative knowledge transfer. DisPFL outperforms the other two methods because it introduces a localized personalized learning to alleviate the negative knowledge transfer. Overall, Loci has 45.70% higher accuracy compared to the three decentralized methods.

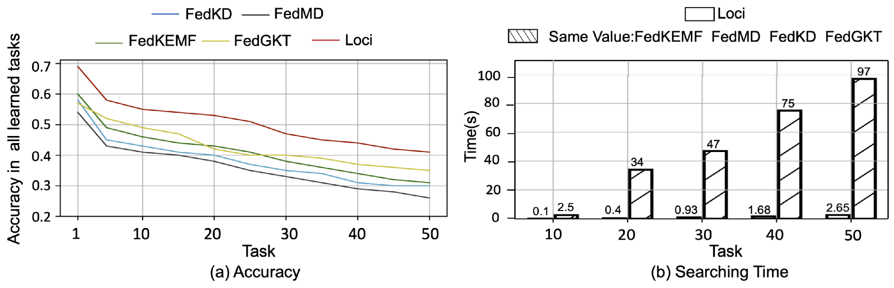


Fig. 8. Discussion of model accuracies under different numbers of tasks.

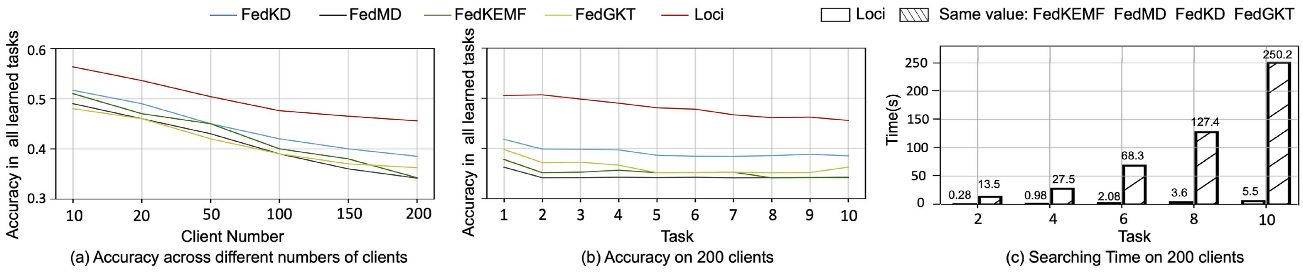


Fig. 9. Discussion of model accuracies under different numbers of clients.

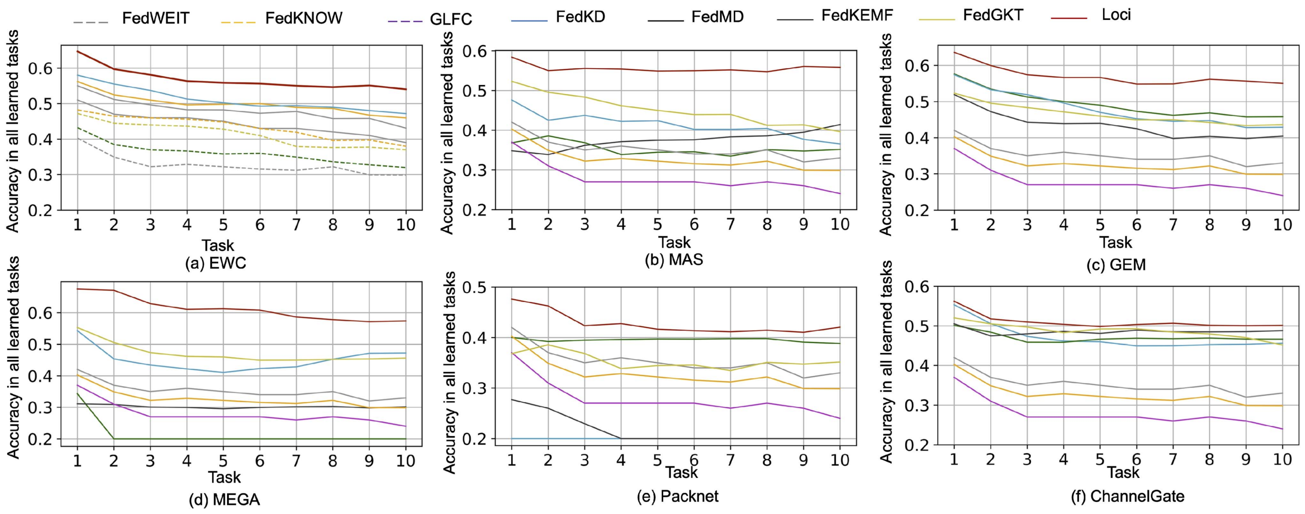


Fig. 10. Applicability of Loci to six categories of CL methods.

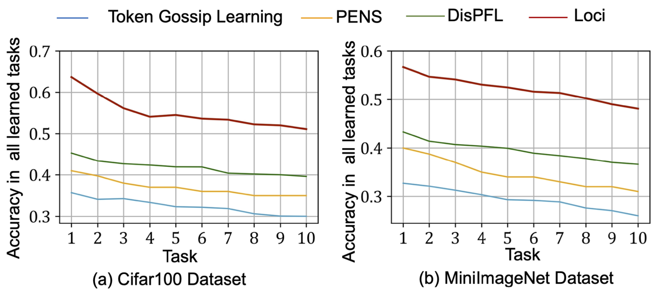


Fig. 11. Discussion of model accuracies under decentralized setting.

## F. Ablation Study

In this section, we take the Cifar100 dataset and the seven CV models as an example to test whether Loci suffers from overfitting when learning similar tasks from other clients, and discuss the design choices of its three components: task memory palace, model selector, and model aggregator.

*Discussion of overfitting under similar scenarios:* We test Cifar100 and MiniImageNet datasets and report Loci's degree of overfitting, namely the model's accuracy gap between training test sets when it repeatedly learns similar tasks. In evaluation, we equally partition the training set into 10 parts and assign each part

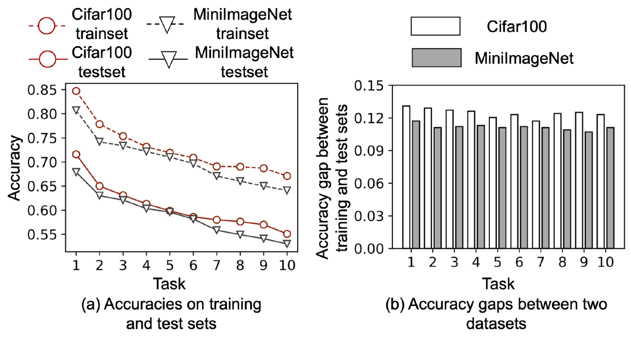


Fig. 12. Training and test accuracies when learning similar tasks.

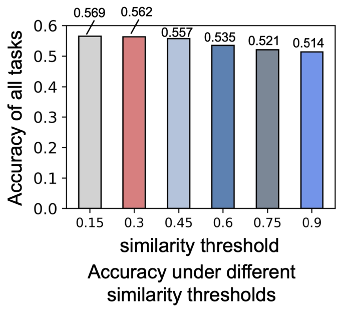


Fig. 14. Impact of similarity threshold on model accuracy.

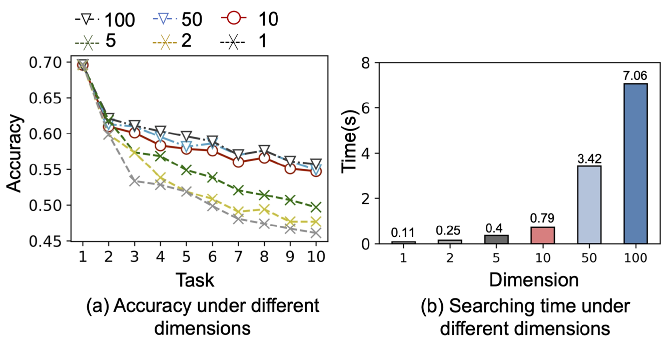


Fig. 13. Impact of compression dimension on model accuracy and task searching time.

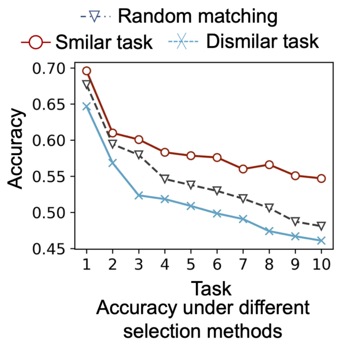


Fig. 15. Impact of task selection strategy on model accuracy.

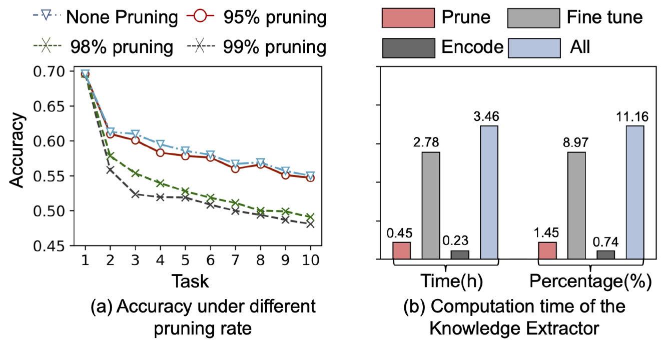


Fig. 16. Impact of pruning rate on accuracy and computation time in knowledge extractor.

to a client. That is, each client contains 10% of training samples and a sequence of similar tasks. Fig. 12's evaluation result shows that the accuracy gaps remain consistent (around 13.7%) as more similar tasks are learned, indicating our approach has no overfitting when learning similar tasks from other clients.

*Discussion of task memory palace:* We test six compression dimensions of task memory palace to discuss their impact on model accuracy and training time. Fig. 13(a)'s result shows that when the compression dimension is larger than 10, the model has similar accuracies. In contrast, when the dimension is reduced to 5, 2 or 1, the model accuracy significantly decreases. In addition, Fig. 13(b) shows that the task memory palace's searching time linearly increases with the compression dimension. Hence our approach set compression dimension to 10 to provide both high accuracy and short searching time.

*Discussion of similarity threshold:* In this evaluation, we take the Cifar100 dataset (the compression dimension is 10) as an example and test the impact of the similarity threshold on model accuracy. Fig. 14 shows that the accuracy has small fluctuations under different similarity thresholds. The results indicate that our approach has low sensitivity to the similarity threshold and can consistently find similar tasks when the similarity threshold varies.

*Discussion of model selector:* We test three different task selection strategies in our model selector: (1) selecting models with similar tasks (used by Loci); (2) selecting models with dissimilar tasks; and (3) randomly selecting models. Fig. 15's result shows that the first selection strategy indeed brings 7.72% higher accuracy than the other two strategies.

*Discussion of knowledge extractor:* Loci's knowledge extractor prunes and encodes the model. Our first evaluation tests three pruning rates and Fig. 16(a)'s result shows that 95% pruning rate has similar accuracy to no pruning, because the remaining 5% of model parameters retain most of the task knowledge. In contrast, pruning rates 98% and 99% incur considerable accuracy losses. The second evaluation tests the percentages of training time in pruning, fine-tuning, and encoding the KD-model. Fig. 16(b)'s result shows that most of the time is spent in fine tuning the compressed KD-model and all three steps take 11.16% of the total model training time.

## VI. Conclusion

This paper presents the design, implementation and evaluation of Loci, a FL framework that can effectively integrates the positive knowledge from heterogeneous tasks of different data modalities while keeping the on-device training efficient. The proposed approach assembles each client's model of the most helpful past and peer tasks, while optimally integrating this global model with the client's local model with high accuracy and low overheads. Extensive evaluations in real scenarios against latest FL techniques strongly prove the efficacy and practicality of Loci, especially for challenging learning scenarios of 50 different tasks and 200 clients.

## References

[1] E. Kristiani, C.-T. Yang, and C.-Y. Huang, "iSEC: An optimized deep learning model for image classification on edge computing," *IEEE Access*, vol. 8, pp. 27267–27276, 2020.

[2] S. Abdel Magid, F. Petrini, and B. Dezfouli, "Image classification on IoT edge devices: Profiling and modeling," *Cluster Comput.*, vol. 23, pp. 1025–1043, 2020.

[3] I. Haritaoglu, "InfoScope: Link from real world to digital information space," in *Proc. Int. Conf. Ubiquitous Comput.*, Springer, 2001, pp. 247–255.

[4] L. Zeng, C. Yang, P. Huang, Z. Zhou, S. Yu, and X. Chen, "GNN at the edge: Cost-efficient graph neural network processing over distributed edge servers," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 3, pp. 720–739, Mar. 2023.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[6] X. Yang, H. Yu, X. Gao, H. Wang, J. Zhang, and T. Li, "Federated continual learning via knowledge fusion: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 8, pp. 3832–3850, Aug. 2024.

[7] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, 2017.

[8] S. Jung, H. Ahn, S. Cha, and T. Moon, "Continual learning with node-importance based adaptive group sparse regularization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 3647–3658.

[9] D. Abati, J. Tomczak, T. Blankevoort, S. Calderara, R. Cucchiara, and B. E. Bejnordi, "Conditional channel gated networks for task-aware continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3931–3940.

[10] G. Cao et al., "E2-AEN: End-to-end incremental learning with adaptively expandable network," 2022, *arXiv:2207.06754*.

[11] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, Dec. 2019.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, PMLR, 2017, pp. 1273–1282.

[13] Y. Deng, M. M. Kamani, and M. Mahdavi, "Adaptive personalized federated learning," 2020, *arXiv: 2003.13461*.

[14] C. He, M. Annavaram, and S. Avestimehr, "Group knowledge transfer: Federated learning of large CNNs at the edge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 14068–14080.

[15] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 2089–2099.

[16] J. Dong et al., "Federated class-incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10164–10173.

[17] J. Yoon, W. Jeong, G. Lee, E. Yang, and S. J. Hwang, "Federated continual learning with weighted inter-client transfer," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 12073–12086.

[18] Y. Luopan, R. Han, Q. Zhang, C. H. Liu, G. Wang, and L. Y. Chen, "FedKNOW: Federated continual learning with signature task knowledge integration at edge," in *Proc. IEEE Int. Conf. Data Eng.*, 2023, pp. 341–354.

[19] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3710–3722, Aug. 2021.

[20] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 19586–19597.

[21] G. Tong, G. Li, J. Wu, and J. Li, "GradMFL: Gradient memory-based federated learning for hierarchical knowledge transferring over non-IID data," in *Proc. Int. Conf. Algorithms Architectures Parallel Process.*, Springer, 2021, pp. 612–626.

[22] Z. Wang, Y. Zhang, X. Xu, Z. Fu, H. Yang, and W. Du, "Federated probability memory recall for federated continual learning," *Inf. Sci.*, vol. 629, pp. 551–565, 2023.

[23] X. Yao and L. Sun, "Continual local training for better initialization of federated models," in *Proc. IEEE Int. Conf. Image Process.*, 2020, pp. 1736–1740.

[24] Y. Chaudhary, P. Rai, M. Schubert, H. Schütze, and P. Gupta, "Federated continual learning for text classification via selective inter-client transfer," 2022, *arXiv:2210.06101*.

[25] J. Dong et al., "Dual learning with dynamic knowledge distillation for partially relevant video retrieval," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 11302–11312.

[26] M. Contributors, "MMCV: OpenMMLab computer vision foundation," 2018. [Online]. Available: https://github.com/open-mmlab/mmcv

[27] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, Association for Computational Linguistics, 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[28] L. Peng, N. Wang, N. Dvornek, X. Zhu, and X. Li, "FedNI: Federated graph learning with network inpainting for population-based disease prediction," *IEEE Trans. Med. Imag.*, vol. 42, no. 7, pp. 2032–2043, Jul. 2023.

[29] J. Stremmel and A. Singh, "Pretraining federated text models for next word prediction," in *Proc. Future Inf. Commun. Conf.*, Springer, 2021, pp. 477–488.

[30] Y. Liu et al., "FedVision: An online visual object detection platform powered by federated learning," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 13172–13179.

[31] J. Chen and A. Zhang, "FedMSplit: Correlation-adaptive federated multi-task learning across multimodal split networks," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2022, pp. 87–96.

[32] G. K. Gudur, B. S. Balaji, and S. K. Perepu, "Resource-constrained federated learning with heterogeneous labels and models," 2020, *arXiv: 2011.03206*.

[33] D. Li and J. Wang, "FedMD: Heterogenous federated learning via model distillation," 2019, *arXiv: 1910.03581*.

[34] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 2351–2363.

[35] X. Fang and M. Ye, "Robust federated learning with noisy and heterogeneous clients," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10072–10081.

[36] W. Huang, M. Ye, and B. Du, "Learn from others and be yourself in heterogeneous federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10143–10153.

[37] F. Sattler, A. Marban, R. Rischke, and W. Samek, "Communication-efficient federated distillation," 2020, *arXiv: 2012.00632*.

[38] Z. Zhu, J. Hong, and J. Zhou, "Data-free knowledge distillation for heterogeneous federated learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 12878–12889.

[39] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nature Commun.*, vol. 13, no. 1, 2022, Art. no. 2032.

[40] S. Yu, W. Qian, and A. Jannesari, "Resource-aware federated learning using knowledge extraction and multi-model fusion," 2022, *arXiv:2208.07978*.

[41] C. Li, G. Li, and P. K. Varshney, "Decentralized federated learning via mutual knowledge transfer," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1136–1147, Jan. 2022.

[42] J. Zhang, S. Guo, X. Ma, H. Wang, W. Xu, and F. Wu, "Parameterized knowledge transfer for personalized federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 10092–10104.

[43] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6470–6479.

[44] K. Raghavan and P. Balaprakash, "Formalizing the generalization-forgetting trade-off in continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, Art. no. 1322.
[45] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.
[46] S. M. Hendryx, D. R. Kc, B. Walls, and C. T. Morrison, "Federated reconnaissance: Efficient, distributed, class-incremental learning," 2021, *arXiv:2109.00150*.
[47] C. Liu, X. Qu, J. Wang, and J. Xiao, "FedET: A communication-efficient federated class-incremental learning framework based on enhanced transformer," 2023, *arXiv:2306.15347*.
[48] J. Dong, Y. Cong, G. Sun, Y. Zhang, B. Schiele, and D. Dai, "No one left behind: Real-world federated class-incremental learning," 2023, *arXiv:2302.00903*.
[49] B. Liu, Y. Guo, and X. Chen, "PFA: Privacy-preserving federated adaptation for effective model personalization," in *Proc. Web Conf.*, 2021, pp. 923–934.
[50] F. E. Castellon, A. Mayoue, J.-H. Sublemontier, and C. Gouy-Pailler, "Federated learning with incremental clustering for heterogeneous data," in *Proc. 2022 Int. Joint Conf. Neural Netw.*, 2022, pp. 1–8.
[51] M. Duan et al., "Flexible clustered federated learning for client-level data distribution shift," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2661–2674, Nov. 2022.
[52] R. Lu et al., "Auction-based cluster federated learning in mobile edge computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1145–1158, Apr. 2023.
[53] J. Fan, K. Wu, G. Tang, Y. Zhou, and S. Huang, "Taking advantage of the mistakes: Rethinking clustered federated learning for IoT anomaly detection," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 6, pp. 862–876, Jun. 2024.
[54] G. Hinton et al., "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[55] S. P. Singh and M. Jaggi, "Model fusion via optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 22045–22055.
[56] S. Han et al., "DSD: Dense-sparse-dense training for deep neural networks," 2016, *arXiv:1607.04381*.
[57] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[58] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.
[59] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, no. 3, pp. 107–115, 2021.
[60] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.
[61] L. V. Kantorovich, "On the translocation of masses," *J. Math. Sci.*, vol. 133, no. 4, pp. 1381–1382, 2006.
[62] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *Proc. IAPR Asian Conf. Pattern Recognit.*, 2015, pp. 730–734.
[63] D. J. Beutel et al., "Flower: A friendly federated learning research framework," 2020, *arXiv: 2007.14390*.
[64] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, "DisPFL: Towards communication-efficient personalized federated learning via decentralized sparse training," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2022, pp. 4587–4604.
[65] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Tech. Rep., 2009.
[66] O. Vinyals et al., "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3637–3645.
[67] Y. Le and X. Yang, "Tiny ImageNet visual recognition challenge," *CS 231 N*, vol. 7, no. 7, pp. 3–8, 2015.
[68] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2004, pp. 168–177.
[69] X. Ding, B. Liu, and P. S. Yu, "A holistic lexicon-based approach to opinion mining," in *Proc. 2008 Int. Conf. Web Search Data Mining*, 2008, pp. 231–240.
[70] Q. Liu, Z. Gao, B. Liu, and Y. Zhang, "Automated rule selection for aspect extraction in opinion mining," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 1291–1297.
[71] Z. Ke, B. Liu, H. Wang, and L. Shu, "Continual learning with knowledge transfer for sentiment classification," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, Ghent, Belgium, Springer, 2021, pp. 683–698.
[72] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.
[73] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 8748–8763.
[74] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2001–2010.
[75] X. Zuo et al., "FedViT: Federated continual learning of vision transformer at edge," *Future Gener. Comput. Syst.*, vol. 154, pp. 1–15, 2024.
[76] Q. Wang, B. Liu, and Y. Li, "Traceable federated continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 12872–12881.
[77] A. Wuerkaixi et al., "Accurate forgetting for heterogeneous federated continual learning," 2025, *arXiv:2502.14205*.
[78] I. Gulrajani and D. Lopez-Paz, "In search of lost domain generalization," 2020, *arXiv: 2007.01434*.
[79] Google, "GRPC: A high performance, open source universal RPC framework," 2015. [Online]. Available: https://grpc.io/
[80] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 139–154.
[81] I. Hegeds, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *J. Parallel Distrib. Comput.*, vol. 148, pp. 109–124, 2021.
[82] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, "Decentralized federated learning of deep neural networks on non-IID data," 2021, *arXiv:2107.08517*.

**Yaxin Luopan** is currently working toward the master's degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on edge intelligence and deep learning applications.

**Rui Han** received the MSc degree with honor from Tsinghua University, China, in 2010, and the PhD degree from Imperial College London, U.K., in 2014. He is an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests are system optimization for cloud data center workloads (in particular highly parallel services and deep learning applications). He has more than 40 publications in these areas, including papers at MobiCOM, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, INFOCOM, and ICDCS.

**Qinglong Zhang** is currently working toward the PhD degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on edge intelligence and deep learning applications.

**Xiaojiang Zuo** is currently working toward the PhD degree with the School of Computer Science and Technology, Beijing Institute of Technology. His work focuses on optimization of federated learning and deep learning workloads.

**Chi Harold Liu** (Senior Member, IEEE) received the BEng degree from Tsinghua University, Beijing, China, and the PhD degree from Imperial College London, London, U.K. He is currently a full professor and the vice dean with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing. Before that, he worked for IBM Research - China and Deutsche Telekom Laboratories, Berlin, Germany, and IBM T. J. Watson Research Center. He is now an associate editor for IEEE TRANS. NETWORK SCIENCE AND ENGINEERING. His current research interests include the Big Data analytics, mobile computing, and deep learning. He is a fellow of IET, and a fellow of Royal Society of the Arts.

**Guoren Wang** (Senior Member, IEEE) received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, Shenyang, China, in 1988, 1991, and 1996, respectively. He is now a full professor and director of the Institute of Data Science and Knowledge Engineering, Department of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He was an assistant president for Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 150 research papers in top conferences and journals like IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ICDE, SIGMOD, VLDB, etc.

**Lydia Y. Chen** (Senior Member, IEEE) received the BA degree from National Taiwan University, and the PhD degree from Pennsylvania State University. She is an associate professor with the Department of Computer Science, Technology University Delft. Prior to joining TU Delft, she was a research staff member with the IBM Zurich Research Lab from 2007 to 2018. Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. She has published more than 80 papers in journals, e.g., IEEE TRANSACTIONS ON DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON SERVICE COMPUTING, and conference proceedings, e.g., INFOCOM, Sigmetrics, DSN, and Eurosys. She was a co-recipient of the best paper awards with CCgrid and eEnergy.