The Journal of Systems & Software 191 (2022) 111357

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Architectural patterns for the design of federated learning systems☆

Sin Kit Lo [a,b,*], Qinghua Lu [a,b], Liming Zhu [a,b], Hye-Young Paik [b], Xiwei Xu [a,b], Chen Wang [a]

[a] Data61, CSIRO, Australia
[b] University of New South Wales, Australia

ARTICLE INFO

Article history:
Received 7 January 2021
Received in revised form 11 December 2021
Accepted 29 April 2022
Available online 7 May 2022

Keywords:
Federated learning
Pattern
Software architecture
Machine learning
Artificial intelligence

ABSTRACT

Federated learning has received fast-growing interests from academia and industry to tackle the challenges of data hungriness and privacy in machine learning. A federated learning system can be viewed as a large-scale distributed system with different components and stakeholders as numerous client devices participate in federated learning. Designing a federated learning system requires software system design thinking apart from the machine learning knowledge. Although much effort has been put into federated learning from the machine learning technique aspects, the software architecture design concerns in building federated learning systems have been largely ignored. Therefore, in this paper, we present a collection of architectural patterns to deal with the design challenges of federated learning systems. Architectural patterns present reusable solutions to a commonly occurring problem within a given context during software architecture design. The presented patterns are based on the results of a systematic literature review and include three client management patterns, four model management patterns, three model training patterns, four model aggregation patterns, and one configuration pattern. The patterns are associated to the particular state transitions in a federated learning model lifecycle, serving as a guidance for effective use of the patterns in the design of federated learning systems.

1. Introduction

Federated learning overview

The ever-growing use of big data systems, industrial-scale IoT platforms, and smart devices contribute to the exponential growth in data dimensions (Lo et al., 2019) and accelerated the adoption of machine learning in many areas. However, many machine learning systems suffer from insufficient training data. The reason is mainly due to the increasing concerns on data privacy, e.g., restrictions on data sharing with external systems for machine learning purposes. For instance, the General Data Protection Regulation (GDPR) (EU, 2020) stipulates a range of data protection measures, and data privacy is now one of the most important ethical principles expected of machine learning systems (Jobin et al., 2019). Furthermore, raw data collected are unable to be used directly for model training for most circumstances. The raw data needs to be pre-processed before being used for model training and data sharing restrictions increase the difficulty to obtain training data. Moreover, concept drift (Li et al., 2020) also occurs when new data is constantly collected, replacing the outdated data. This makes the model trained on previous data degrades at a much faster rate. Hence, a new technique that can swiftly produce models that adapt to the concept drift when different data is discovered in clients is essential.

To effectively address the lack of training data limitations, concept drift, and the data-sharing restriction while still enabling effective data inferences by the data-hungry machine learning models, Google introduced the concept of federated learning (McMahan et al., 2017) in 2016. Fig. 1 illustrates the overview of federated learning. There are three stakeholders in a federated learning system: (1) learning coordinator (i.e., system owner), (2) contributor client — data contributor & local model trainer, and (3) user client — model user. Note that a contributor client can also be a user client. There are two types of system nodes (i.e., hardware components): (1) central server, (2) client device. The functioning of FL is described in the next paragraph by means of an example.

A motivation example

Imagine we use federated learning to train the next-word prediction model in a mobile phone keyboard application. The learning coordinator is the provider of the keyboard application,

☆ Editor: Raffaela Mirandola.

* Corresponding author.
E-mail addresses: Kit.Lo@data61.csiro.au (S.K. Lo), Qinghua.Lu@data61.csiro.au (Q. Lu), liming.zhu@data61.csiro.au (L. Zhu), h.paik@unsw.edu.au (H.-Y. Paik), Xiwei.Xu@data61.csiro.au (X. Xu), Chen.Wang@data61.csiro.au (C. Wang).

https://doi.org/10.1016/j.jss.2022.111357
0164-1212/

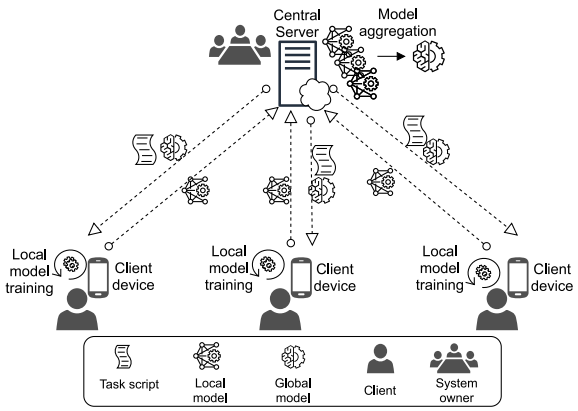

Fig. 1. Federated learning overview.

while contributor clients are the mobile phone users. The user clients will be the new or existing mobile phone users of the keyboard application. The differences in ownership, geolocation, and usage pattern cause the local data to possess non-IID[1] characteristics, which is a design challenge of federated learning systems. The federated learning process starts when a training task is created by the learning coordinator. For instance, the keyboard application provider produces and embeds an initial global model into the keyboard applications. The initial global model (includes task scripts & training hyperparameters) is broadcast to the participating client devices. After receiving the initial model, the model training is performed locally across the client devices, without the centralised collection of raw client data. Here, the smartphones that have the keyboard application installed receive the model to be trained. The client devices typically run on different operating systems and have diverse communication and computation resources, which is defined as the *system heterogeneity* challenges.

Each training round takes one step of gradient descent on the current model using each client's local data. In this case, the smartphones optimise the model using the keyboard typing data. After each round, the model update is submitted by each participating client device to the central server. The central server collects all the model updates and performs model aggregation to form a new version of the global model. The new global model is re-distributed to the client devices for the next training round. This entire process iterates until the global model converges. As a result, *communication and computation efficiency* are crucial as many local computation and communication rounds are required for the global model to converge. Moreover, due to the limited resources available on each device, the device owners might be reluctant to participate in the federated learning process. Therefore, *client motivatability* becomes a design challenge. Furthermore, the central server communicates with multiple devices for the model exchange which makes it vulnerable to the *single-point-of-failure*. The *trustworthiness* of the central server and the possibility of adversarial nodes participating in the training process also creates *system reliability* and *security* challenges. After the completion of training, the learning coordinator stores the converged model and deploys it to the user clients. For instance, the keyboard application provider stores the converged model and embeds it to the latest version of the application for existing or new application users. Here, the different versions of the local models associated with the global model created need to be maintained.

[1] Non-Identically and Independently Distribution: Highly-skewed and personalised data distribution that vary heavily between different clients and affects the model performance and generalisation (Sattler et al., 2020).

Design challenges

A federated learning system, as a large-scale distributed system, presents more architectural design challenges (Lo et al., 2021b,a), especially when dealing with the interactions between the central server and client devices and managing tradeoffs of software quality attributes. The main design challenges are summarised as follows.

- Global models might have low accuracy, and lack generality when client devices generate non-IID data. Centralised process of the data is adopted by conventional machine learning to deal with data heterogeneity but the inherent privacy-preserving nature of federated learning render such techniques inappropriate.
- To generate high-quality global models that are adaptive to concept drift, multiple rounds of communication are required to exchange local model updates, which could incur high communication costs.
- Client devices have limited resources to perform the multiple rounds of model training and communications required by the system, which may affect the model quality.
- As numerous client devices participate in federated learning, it is challenging to coordinate the learning process and ensure model provenance, system reliability and security.

How to select appropriate designs to fulfil different software quality requirements and design constraints is non-trivial for such a complex distributed system. Although much effort has been put into federated learning from the machine learning techniques side, there is still a gap on the architectural design considerations of the federated learning systems. A systematic guidance on architecture design of federated learning systems is required to better leverage the existing solutions and promote federated learning to enterprise-level adoption.

Research contribution

In this paper, we present a collection of patterns for the design of federated learning systems. In software engineering, an architectural pattern is a reusable solution to a problem that occurs commonly within a given context in software design (Beck and Cunningham, 1987). Our pattern collection includes three client management patterns, four model management patterns, three model training patterns, four model aggregation patterns, and one configuration pattern. We define the lifecycle of a model in a federated learning system and associate each identified pattern to a particular state transition in the lifecycle.

The main contribution of this paper includes:

- The collection of architectural patterns provides a design solution pool for practitioners to select from for real-world federated learning system development.
- The federated learning model lifecycle with architectural pattern annotations, which serves as a systematic guide for practitioners during the design and development of a federated learning system.

Paper structure

The remainder of the study is organised as follows. Section 2 introduces the research methodology. Section 3 provides an overview of the patterns in the federated learning lifecycle, followed by the detailed discussions on each pattern. Section 4 summarises and discusses some repeating challenges of federated learning systems and Section 5 presents the related work. Finally, Section 6 concludes the paper.

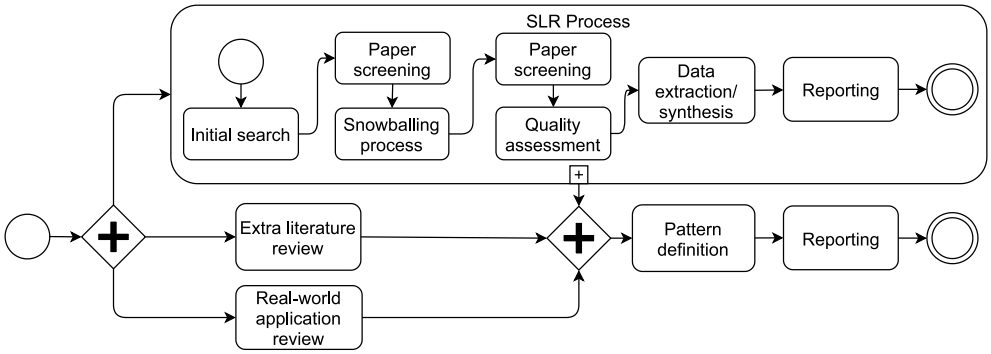

Fig. 2. Pattern collection process.

Table 1
Number of sources of per patterns.

Category	Pattern	SLR papers	ML/FL papers	Real-world applications
Client management patterns	Pattern 1: Client registry	0	0	3
	Pattern 2: Client selector	4	2	1
	Pattern 3: Client cluster	2	2	2
Model management patterns	Pattern 4: Message compressor	8	5	0
	Pattern 5: Model co-versioning registry	0	0	4
	Pattern 6: Model replacement trigger	0	1	3
	Pattern 7: Deployment selector	0	0	3
Model training patterns	Pattern 8: Multi-task model trainer	2	2	2
	Pattern 9: Heterogeneous data handler	1	2	0
	Pattern 10: Incentive registry	18	1	0
Model aggregation patterns	Pattern 11: Asynchronous aggregator	4	1	0
	Pattern 12: Decentralised aggregator	5	3	0
	Pattern 13: Hierarchical aggregator	4	2	0
	Pattern 14: Secure aggregator	31	0	3
Configuration patterns	Pattern 15: Training configurator	0	0	3

2. Methodology

Fig. 2 illustrates the federated learning pattern extraction and collection process. Firstly, the patterns are collected based on the results of our previous systematic literature review (SLR) on federated learning (Lo et al., 2021b). SLR and situational method engineering (SME) (Brinkkemper, 1996) are some of the renowned systematic methodologies for derivation of pattern languages. For instance, several pattern derivations on cloud migration and software architecture have used SLR (e.g., Zdun, 2007; Ahmad et al., 2014; Jamshidi et al., 2017). Moreover, Balalaie et al. (2018) have derived the pattern languages in the context of cloud-native and microservices using situational method engineering.

For this work, we have adopted the SLR method as the currently available materials and research works on federated learning are still highly academic-based. Secondly, we intend to propose design patterns for software architectural design aspects of building federated learning systems rather than for their development/engineering processes. This is because, during the SLR work, we have identified many architectural design challenges and lack of systematic design approaches to federated learning. Furthermore, while SME has the benefit of offering a systematic methodology for selecting appropriate method components from a repository of reusable method components, it is more suitable for pattern extraction of an information system development (ISD) process (Mirbel and Ralyté, 2006).

The SLR was performed according to Kitchenham's SLR guideline (Kitchenham and Charters, 2007), and the number of final studied papers is 231. During the SLR, we developed a comprehensive mapping between federated learning challenges and approaches. Additionally, we reviewed 20 machine learning and federated learning papers published after the SLR search cut-off date (31st Jan 2020) and 24 real-world applications. The additional literature review on machine learning and federated learning, and the review of the real-world applications are conducted based on our past real-world project implementation experience. Table 1 shows the mapping of sources number for each pattern generated from the SLR, extra literature, and real-world applications. Table 2 records all the extra literature papers and real-world applications with their respective patterns. Each real-world application in the table is embedded with the hyperlink to their respective website or git repositories.

We discussed the proposed patterns according to the pattern form presented in Meszaros and Doble (1997). The form comprehensively describes the patterns by discussing the **Context**, **Problem**, **Forces**, **Solution**, **Consequences**, **Related patterns**, and **Known-uses** of the pattern.

The **Context** is the description of the situation where a problem occurs, in which the solution proposed is applicable, or the problem is solvable by the pattern. **Problem** comprehensively elicits the challenges and limitations that occur under the defined context. **Forces** describe the reasons and causes for a specific design or pattern decision to be made to resolve the problem. **Solution** describes how the problem and the conflict of forces can be resolved by a specific pattern. **Consequences** reason about the impact of applying a solution, specifically on the contradictions among the benefits, costs, drawbacks, tradeoffs, and liabilities of the solution. **Related patterns** record the other patterns from this paper that are related to the current pattern. **Known-uses** refer to empirical evidence that the solution has been used in the real world.

3. Federated learning patterns

Fig. 3 illustrates the lifecycle of a model in a federated learning system. The lifecycle covers the deployment of the completely trained global model to the client devices (i.e., model users)

Table 2
Extra literature and real-world applications.

Pattern	Extra literature	Real-world applications
Client registry	–	IBM Federated Learning, doc.ai, Siemens Mindsphere Asset Manager
Client selector	Wang et al. (2019), Zhang et al. (2021)	FedCS
Client cluster	Chai et al. (2020), Huang et al. (2019)	IFCA, CFL
Message compressor	Haddadpour et al. (2021), Jiang et al. (2022), Konečný et al. (2017), Sattler et al. (2020), Mills et al. (2020)	–
Model co-versioning registry	–	DVC, Managed MLflow, Replicate.ai, Pachyderm
Model replacement trigger	Bowden et al. (2012)	Microsoft Azure ML Designer, Amazon SageMaker, Alibaba ML Platform
Deployment selector	–	Azure Machine Learning, Google Cloud, Amazon SageMaker
Multi-task model trainer	Corinzia and Buhmann (2019), Bai et al. (2009)	MultiModel, MT-DNN
Heterogeneous data handler	Ahn et al. (2019), Jeong et al. (2018)	–
Incentive registry	Liu et al. (2020)	–
Asynchronous aggregator	Xie et al. (2020)	–
Decentralised aggregator	Jiang and Hu (2020), Roy et al. (2019), Warnat-Herresthal et al. (2021)	–
Hierarchical aggregator	Briggs et al. (2020), Liu et al. (2020)	–
Secure aggregator	–	TensorFlow Privacy Library, ZamaAI, SEAL
Training configurator	–	AutoML, FritzAI, MakeML

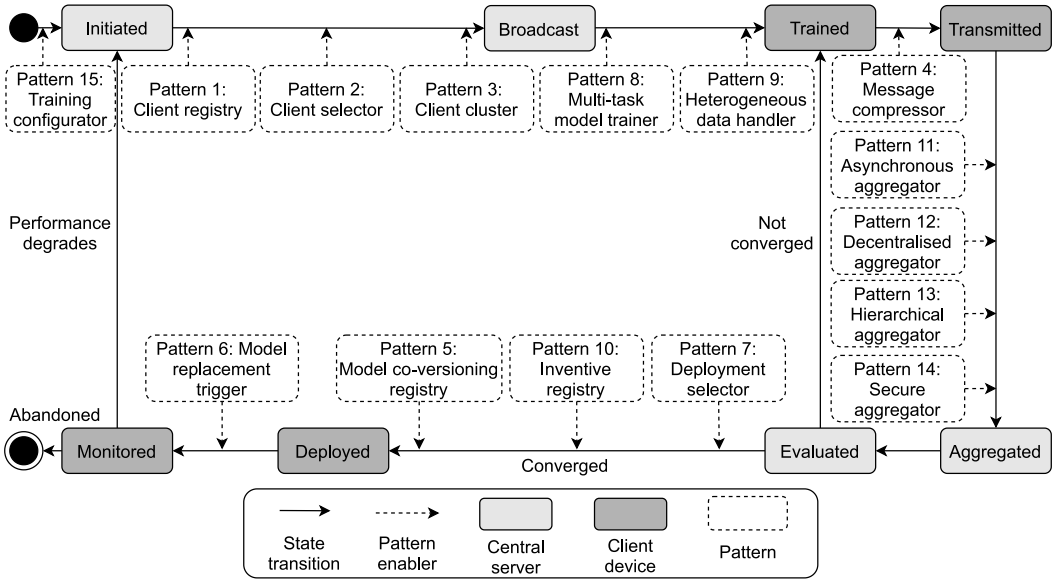

Fig. 3. A Model's lifecycle in federated learning.

for data inference. The deployment process involves the communication between the central server and the client devices. We categorise the federated learning patterns as shown in Table 3 to provide an overview. There are four main groups: (1) client management patterns, (2) model management patterns, (3) model training patterns, (4) model aggregation patterns, and (5) configuration pattern.

3.1. Client management patterns

Client management patterns describe the patterns that manage the client devices' information and their interaction with the central server. A *client registry* manages the information of all the participating client devices. *Client selector* selects client devices for a specific training task, while *client cluster* increases the model performance and training efficiency through grouping client devices based on the similarity of certain characteristics (e.g., available resources, data distribution).

3.1.1. ***Pattern 1: Client registry***

Summary: A client registry maintains the information of all the participating client devices for client management. According to Fig. 4, the client registry is maintained in the central server. The central server sends the request for information to the client devices. The client devices then send the requested information together with the first local model updates.

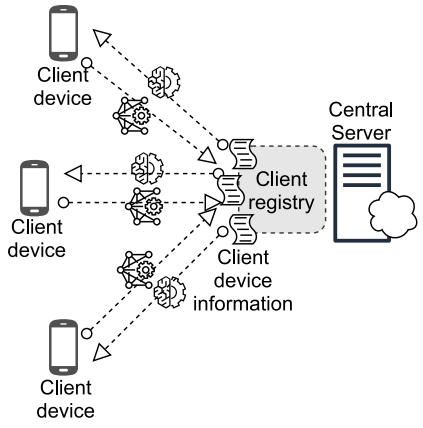

Fig. 4. Client registry.

Table 3
Overview of architectural patterns for federated learning.

Category	Name	Summary
Client management patterns	Client registry	Maintains the information of all the participating client devices for client management.
	Client selector	Actively selects the client devices for a certain round of training according to the predefined criteria to increase model performance and system efficiency.
	Client cluster	Groups the client devices (i.e., model trainers) based on their similarity of certain characteristics (e.g., available resources, data distribution, features, geolocation) to increase the model performance and training efficiency.
Model management patterns	Message compressor	Compresses and reduces the message data size before every round of model exchange to increase the communication efficiency.
	Model co-versioning registry	Stores and aligns the local models from each client with the corresponding global model versions for model provenance and model performance tracking.
	Model replacement trigger	Triggers model replacement when the degradation in model performance is detected.
	Deployment selector	Selects and matches the converged global models to suitable client devices to maximise the global models' performance for different applications and tasks.
Model training patterns	Multi-task model trainer	Utilises data from separate but related models on local client devices to improve learning efficiency and model performance.
	Heterogeneous data handler	Solves the non-IID and skewed data distribution issues through data volume and data class addition while maintaining the local data privacy.
	Incentive registry	Measures and records the performance and contributions of each client and provides incentives to motivate clients' participation.
Model aggregation patterns	Asynchronous aggregator	Performs aggregation asynchronously whenever a model update arrives without waiting for all the model updates every round to reduce aggregation latency.
	Decentralised aggregator	Removes the central server from the system and decentralises its role to prevent single-point-of-failure and increase system reliability.
	Hierarchical aggregator	Adds an edge layer to perform partial aggregation of local models from closely-related client devices to improve model quality and system efficiency.
	Secure aggregator	The adoption of secure multiparty computation (MPC) protocols that manages the model exchange and aggregation security to protect model security.
Configuration pattern	Training configurator	A user-friendly platform that allows users to request, configure and deploy FL training processes and models without the need to code or program.

Context: Client management is centralised, and global and local models are exchanged between the central server the massive number of distributed client devices with dynamic connectivity and diverse resources.

Problem: It is challenging for a federated learning system to track any dishonest, failed, or dropout node. This is crucial to secure the central server and client devices from adversarial threats. Moreover, to effectively align the model training process of each client device for each aggregation round, a record of the connection and training information of each client device that has interacted with the central server is required.

Forces:

- *Updatability*. The ability to keep track of the participating devices is necessary to ensure the information recorded is up-to-date.
- *Data privacy*. The records of client information expose the clients to data privacy issues. For instance, the device usage pattern of users may be inferred from the device connection up-time, device information, resources, etc.

Solution: A client registry records all the information of client devices that are connected to the system from the first time. The information includes device ID, connection up & downtime, device resource information (computation, communication, power & storage). The access to the client registry could be restricted according to the agreement between the central server and participating client devices.

Consequences:

Benefits:

- *Maintainability*. The client registry enables the system to effectively manage the dynamically connecting and disconnecting clients.
- *Reliability*. The client registry provides status tracking for all the devices, which is essential for problematic node identification.

Drawbacks:

- *Data privacy*. The recording of the device information on the central server leads to client data privacy issues.
- *Cost*. The maintenance of client device information requires extra communication cost and storage cost, which further surges when the number of client devices increases.

Related patterns: *Model Co-Versioning Registry, Client Selector, Client Cluster, Asynchronous Aggregator, Hierarchical Aggregator*

Known uses:

- *IBM Federated Learning*[2]: *Party Stack* component manages the client parties of IBM federated learning framework, that contains sub-components such as protocol handler, connection, model, local training, and data handler for client devices registration and management.

2 https://github.com/IBM/federated-learning-lib

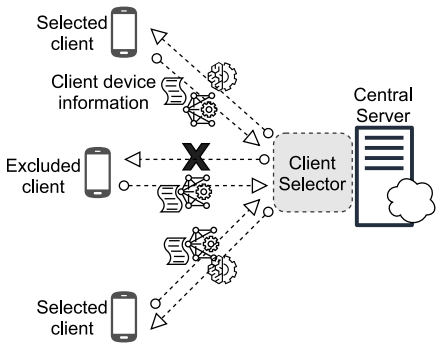

Fig. 5. Client selector.

- *doc.ai*[3]: Client registry is designed for medical research applications to ensure that updates received apply to a current version of the global model, and not a deprecated global model.
- *Siemens Mindsphere Asset Manager*[4]: To support the collaboration of federated learning clients in industrial IoT environment, *Industrial Metadata Management* is introduced as a device metadata and asset data manager.

3.1.2. Pattern 2: Client selector

Summary: A client selector actively selects the client devices for a certain round of training according to the predefined criteria to increase model performance and system efficiency. As shown in Fig. 5, the central server assesses the performance of each client according to the information received. Based on the assessment results, the second client is excluded from receiving the global model.

Context: Multiple rounds of model exchanges are performed and communication cost becomes a bottleneck. Furthermore, multiple iterations of aggregations are performed and consume high computation resources.

Problem: The central server is burdensome to accommodate the communication with massive number of widely-distributed client devices every round.

Forces:

- *Latency*. Client devices have system heterogeneity (difference in computation, communication, & energy resources) that affect the local model training and global model aggregation time.
- *Model quality*. Local data are statistically heterogeneous (different data distribution/quality) which produce local models that overfit the local data.

Solution: Selecting client devices with predefined criteria can optimise the formation of the global model. The client selector on the central server performs client selection every round to include the best fitting client devices for global model aggregation. The selection criteria can be configured as follows:

- Resource-based: The central server assesses the resources available on each client devices every training round and selects the client devices with the satisfied resource status (e.g., WiFi connection, pending status, sleep time)
- Data-based: The central server examines the information of the data collected by each client, specifically on the number of data classes, distribution of data sample volume per class, and data quality. Based on these assessments, the model training process includes devices with high-quality data, higher data volume per class, and excludes the devices with low-quality data, or data that are highly heterogeneous in comparison with other devices.
- Performance-based: Performance-based client selection can be conducted through local model performance assessment (e.g., performance of the latest local model or the historical records of local model performance).

Consequences:

Benefits:

- *Resource optimisation*. The client selection optimises the resource usage of the central server to compute and communicate with suitable client devices for each aggregation round, without wasting resources to aggregate the low-quality models.
- *System performance*. Selecting clients with sufficient power and network bandwidth greatly reduces the chances of clients dropping out and lowers the communication latency.
- *Model performance*. Selecting clients with the higher local model performance or lower data heterogeneity increases the global model quality.

Drawbacks:

- *Model generality*. The exclusion of models from certain client devices may lead to the missing of essential data features and the loss in the global model generalisability.
- *Data privacy*. The central server needs to acquire the system and resource information (processor's capacity, network availability, bandwidth, online status, etc.) every round to perform devices ranking and selection. Access to client devices' information creates data privacy issues.
- *Computation cost*. Extra resources are spent on transferring of the required information for selection decision-making.

Related patterns: *Client Registry, Deployment Selector*

Known uses:

- *Google's FedAvg*: *FedAvg* (McMahan et al., 2017) algorithm includes client selection that randomly selects a subset of clients for each round based on predefined environment conditions and device specification of the client devices.
- In *IBM's Helios* (Xu et al., 2021), there is a training consumption profiling function that fully profiles the resource consumption for model training on client devices. Based on that profiling, a resource-aware soft-training scheme is designed to accelerate local model training on heterogeneous devices and prevent stragglers from delaying the collaborative learning process.
- *FedCS* suggested by OMRON SINIC X Corporation[5] sets a certain deadline for clients to upload the model updates.
- *Communication-Mitigated Federated Learning (CMFL)* (Wang et al., 2019) excludes the irrelevant local updates by identifying the relevance of a client update by comparing its global tendency of model updating with all the other clients.
- *CDW_FedAvg* (Zhang et al., 2021) takes the centroid distance between the positive and negative classes of each client dataset into account for aggregation to exclude those with extreme outliers.

[3] https://doc.ai/

[4] https://documentation.mindsphere.io/resources/html/asset-manager/en-US/index.html

[5] https://www.omron.com/sinicx/

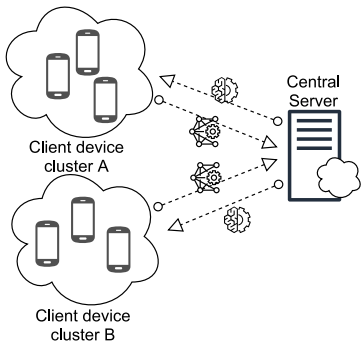

Fig. 6. Client cluster.

3.1.3. Pattern 3: Client cluster

Summary: A client cluster groups the client devices (i.e., model trainers) based on their similarity of certain characteristics (e.g., available resources, data distribution, features, geolocation) to increase the model performance and training efficiency. In Fig. 6, the client devices are clustered into 2 groups by the central server, and the central server will broadcast the global model that is more related to the clusters accordingly.

Context: The system trains models over client devices which have diverse communication and computation resources, resulted in statistical and system heterogeneity challenges.

Problem: Federated learning models generated under non-IID data properties are deemed to be less generalised. This is due to the lack of significantly representative data labels from the client devices. Furthermore, local models may drift significantly from each other.

Forces: The problem requires to balance the following forces:

- *Computation cost and training time.* More computation costs and longer training time are required to overcome the non-IID issue of client devices.
- *Data privacy.* Data privacy contradicts with the access to the entire or parts of the client's raw data is needed by the learning coordinator to resolve the non-IID issue which creates data privacy risks.

Solution: Client devices are clustered into different groups according to their properties (e.g., data distribution, features similarities, gradient loss). By creating clusters of clients with similar data patterns, the global model generated will have better performance for the non-IID-severe client network, without accessing the local data.

Consequences:

Benefits:

- *Model quality.* The global model created by client clusters can have a higher model performance for highly personalised prediction tasks.
- *Convergence speed.* The consequent deployed global models can have faster convergence speed as the models of the same cluster can identify the gradient's minima much faster when the clients' data distribution and IIDness are similar.

Drawbacks:

- *Computation cost.* The central server consumes extra computation cost and time for client clustering and relationship quantification.
- *Data privacy.* The learning coordinator (i.e., central server) requires extra client device information (e.g., data distribution, feature similarities, gradient loss) to perform clustering. This exposes the client devices to the possible risk of private data leakage.

Related patterns: *Client Registry, Client Selector, Deployment Selector*

Known uses:

- Iterative Federated Clustering Algorithm (*IFCA*)[6] is a framework introduced by UC Berkley and Google to cluster client devices based on the loss values of the client's gradient.
- Clustered Federated Learning (*CFL*)[7] uses a cosine similarity-based clustering method that creates a bi-partitioning to group client devices with the same data generating distribution into the same cluster.
- *TiFL* (Chai et al., 2020) is a tier-based federated learning system that adaptively groups client devices with similar training time per round to mitigate the heterogeneity problem without affecting the model accuracy.
- Patient clustering in a federated learning system is implemented by Massachusetts General Hospital to improve efficiency in predicting mortality and hospital stay time (Huang et al., 2019).

3.2. Model management patterns

Model management patterns include patterns that handle model transmission, deployment, and governance. A *message compressor* reduces the transmitted message size. A *model co-versioning registry* records all local model versions from each client and aligns them with their corresponding global model. A *model replacement trigger* initiates a new model training task when the converged global model's performance degrades. A *deployment selector* deploys the global model to the selected clients to improve the model quality for personalised tasks.

3.2.1. Pattern 4: Message compressor

Summary: A message compressor reduces the message data size before every round of model exchange to increase the communication efficiency. Fig. 7 illustrates the operation of the pattern on both ends of the system (client device and central server).

Context: Multiple rounds of model exchanges occurs between a central server and many client devices to complete the model training.

Problem: Communication cost for model communication (e.g., transferring model parameters or gradients) is often a critical bottleneck when the system scales up, especially for bandwidth-limited client devices.

Forces: The problem requires to balance the following forces:

- *Computation cost.* High computation costs are required by the central server to aggregate all the bulky model parameters collected every round.
- *Communication cost.* Communication costs are scarce to communicate the model parameters and gradients between resource-limited client devices and the central server.

[6] https://github.com/jichan3751/ifca
[7] https://github.com/felisat/clustered-federated-learning

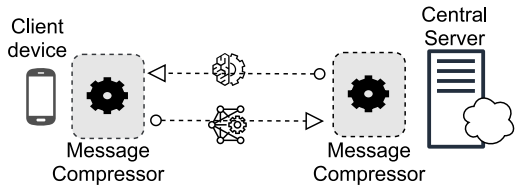

Fig. 7. Message compressor.

Solution: The model parameters and the training task script as one message package is compressed before being transferred between the central server and client devices.

Consequences:

Benefits:

- *Communication efficiency.* The compression of model parameters reduces the communication cost and network throughput for model exchanges.

Drawbacks:

- *Computation cost.* Extra computation is required for message compression and decompression every round.
- *Loss of information.* The downsizing of the model parameters might cause the loss of essential information.

Related patterns: *Client Registry, Model Co-Versioning Registry*

Known uses:

- *Google Sketched Update* (Konečný et al., 2017): Google proposes two communication efficient update approaches: structured update and sketched update. Structured update directly learns an update from a restricted space that can be parametrised using a smaller number of variables, whereas sketched update compresses the model before sending it to the central server.
- *IBM PruneFL* (Jiang et al., 2022) adaptively prunes the distributed parameters of the models, including initial pruning at a selected client and further pruning as part of the federated learning process.
- *FedCom* (Haddadpour et al., 2021) compresses messages for uplink communication from the client device to the central server. The central server produces a convex combination of the previous global model and the average of updated local models to retain the essential information of the compressed model parameters.

3.2.2. Pattern 5: Model co-versioning registry

Summary: A model co-versioning registry records all local model versions from each client and aligns them with their corresponding global model. This enables the tracing of model quality and adversarial client devices at any specific point of the training to improve system accountability. Fig. 8 shows that the registry collects and maps the local model updates to the associated global model versions.

Context: Multiple new versions of local models are generated from different client devices and one global model aggregated each round. For instance, a federated learning task that runs for 100 rounds on 100 devices will create 10,000 local models and 100 global models in total.

Problem: With high numbers of local models created each round, it is challenging to keep track of which local models contributed to the global model of a specific round. Furthermore, the system needs to handle the challenges of asynchronous updates, client dropouts, model selection, etc.

Forces: The problem requires to balance the following forces:

- *Updatability.* The system needs to keep track of the local and global models concerning each client device's updates (application's version or device OS/firmware updates) and ensure that the information recorded is up-to-date.
- *Immutability.* The records and storage of the models co-versions and client IDs needs to be immutable.

Solution: A model co-versioning registry records the local model version of each client device and the global model it corresponds to. This enables seamless synchronous and asynchronous model updates and aggregation. Furthermore, the model co-versioning registry enables the early-stopping of complex model training (stop training when the local model overfits and retrieve the best performing model previously). This can be done by observing the performance of the aggregated global model. Moreover, to provide accountable model provenance and co-versioning, blockchain is considered as one alternative to the central server due to immutability and decentralisation properties.

Consequences:

Benefits:

- *Model quality.* The mapping of local models with their corresponding version of the global model allows the study of the effect of each local model quality on the global model.
- *Accountability.* System accountability improves as stakeholders can trace the local models that correspond to the current or previous global model versions.
- *System security.* It enables the detection of adversarial or dishonest clients and tracks the local models that poisons the global model or causes system failure.

Drawbacks:

- *Storage cost.* Extra storage cost is incurred to store all the local and global models and their mutual relationships. The record also needs to be easily retrievable and it is challenging if the central server host more task.

Related patterns: *Client Registry*

Known uses:

- *DVC*[8] is an online machine learning version control platform built to make models shareable and reproducible.
- *Managed MLflow*[9] on Databricks is a centralised model store that provides chronological model lineage, model versioning, stage transitions, and descriptions.
- *Replicate.ai*[10] is an open-source version control platform for machine learning that automatically tracks code, hyperparameters, training data, weights, metrics, and system dependencies.
- *Pachyderm*[11] is an online machine learning pipeline platform that uses containers to execute the different steps of the pipeline and also solves the data versioning provenance issues.

8 https://dvc.org/
9 https://databricks.com/product/managed-mlflow
10 https://replicate.ai/
11 https://www.pachyderm.com/

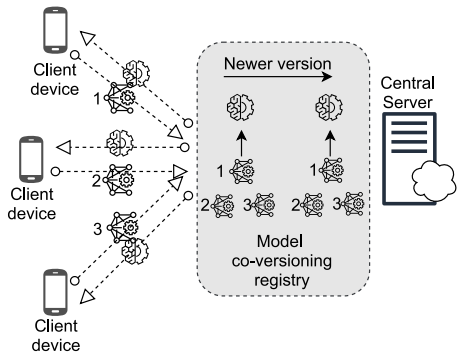

Fig. 8. Model co-versioning registry.

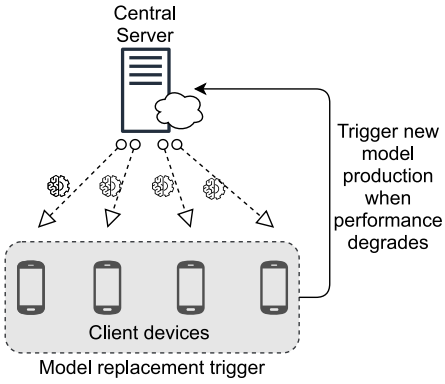

Fig. 9. Model replacement trigger.

3.2.3. Pattern 6: Model replacement trigger

Summary: Fig. 9 illustrates a model replacement trigger that initiates a new model training task when the current global model's performance drops below the threshold value or when a degrade on model prediction accuracy is detected.

Context: The client devices use converged global models for inference or prediction.

Problem: As new data is introduced to the system, the global model accuracy might reduce gradually. Eventually, with the degrading performance, the model is no longer be suitable for the application.

Forces: The problem requires to balance the following forces:

- *Model quality.* The global model deployed might experience a performance drop when new data are used for inference and prediction.
- *Performance degradation detection.* The system needs to effectively determine the reason for the global model's performance degradation before deciding whether to activate a new global model generation.

Solution: A model replacement trigger initiates a new model training task when the acting global model's performance drops below the threshold value. It will compare the performance of the deployed global model on a certain number of client devices to determine if the degradation is a global event. When the global model performance is lower than the preset threshold value for more than a fixed number of consecutive times, given that performance degradation is a global event, a new model training task is triggered.

Consequences:

Benefits:

- *Updatability.* The consistent updatability of the global model helps to maintain system performance and reduces the non-IID data effect. It is especially effective for clients that generate highly personalised data that causes the effectiveness of the global model to reduce much faster as new data is generated.
- *Model quality.* The ongoing model performance monitoring is effective to maintain the high quality of the global model used by the clients.

Drawbacks:

- *Computation cost.* The client devices will need to perform model evaluation periodically that imposes extra computational costs.
- *Communication cost.* The sharing of the evaluation results among clients to know if performance degradation is a global event is communication costly.

Related patterns: *Client Registry, Client Selector, Model Co-versioning Registry*

Known uses:

- *Microsoft Azure Machine Learning Designer*[12] provides a platform for machine learning pipeline creation that enables models to be retrained on new data.
- *Amazon SageMaker*[13] provides model deployment and monitoring services to maintain the accuracy of the deployed models.
- *Alibaba Machine Learning Platform*[14] provides end-to-end machine learning services, including data processing, feature engineering, model training, model prediction, and model evaluation.

3.2.4. Pattern 7: Deployment selector

Summary: A deployment selector deploys the converged global model to the selected model users to improve the prediction quality for different applications and tasks. As shown in Fig. 10, different versions of converged models are deployed to different groups of clients after evaluation.

Context: Client devices train local models using multi-task federated learning settings (a model is trained using data from multiple applications to perform similar and related tasks). These models need to be deployed to suitable groups of client devices.

Problem: Due to the inherent diversity and non-IID distribution among local data, the globally trained model may not be accurate enough for all clients or tasks.

Forces: The problem requires to balance the following forces:

- *Identification of clients.* The central server needs to match and deploy the global models to the different groups of client devices.
- *Training and storage of different models.* The central server needs to train and store different global models for diverse clients or applications.

Solution: A deployment selector examines and selects clients (i.e., model users) to receive the trained global model specified for them based on their data characteristics or applications. The

12 https://azure.microsoft.com/en-au/services/machine-learning/designer/
13 https://aws.amazon.com/sagemaker/
14 https://www.alibabacloud.com/product/machine-learning

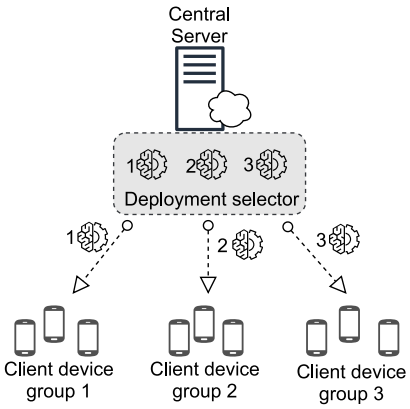

Fig. 10. Deployment selector.

deployment selector deploys the model to the client devices once the global model is completely trained.

Consequences:

Benefits:

- *Model performance.* Deploying converged global models to suitable groups of client devices enhances the model performance to the specific groups of clients or applications.

Drawbacks:

- *Cost.* There are extra costs for training of multiple personalised global models, deployment selection, storage of multiple global models.
- *Model performance.* The statistical heterogeneity of model trainers produces personalised local models, which is then generalised through *FedAvg* aggregation. We need to consider the performance tradeoff of the generalised global model deployed for different model users and applications.
- *Data privacy.* Data privacy challenges exist when the central server collects the client information to identify suitable models for different clients. Moreover, the global model might be deployed to model users that have never joined the model training process.

Related patterns: *Client Registry, Client Selector*

Known uses:

- *Azure Machine Learning*[15] supports mass deployment with a step of compute target selection.
- *Amazon SageMaker*[16] can host multiple models with multi-model endpoints.
- *Google Cloud*[17] uses model resources to manage different versions of models.

3.3. Model training patterns

Patterns about the model training and data preprocessing are group together as model training patterns, including *multi-task model trainer* that tackles non-IID data characteristics, *heterogeneous data handler* that deals with data heterogeneity in training datasets, and *incentive registry* that increases the client's motivatability through rewards.

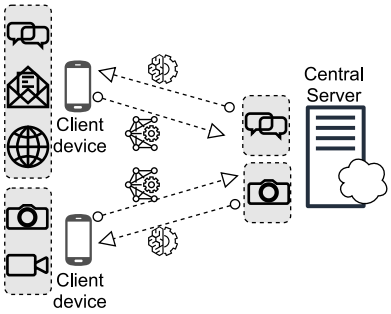

Fig. 11. Multi-task model trainer.

3.3.1. Pattern 8: Multi-task model trainer

Summary: In federated learning, a multi-task model trainer trains different-but-related models on local client devices to improve learning efficiency and model performance. As shown in Fig. 11, there are two groups of applications: (i) text-related applications (e.g., messaging, email, etc.), and (ii) image-related applications (camera, video, etc.). The related models are trained on client devices using data of related tasks.

Context: Local data has statistical heterogeneity property where the data distribution among different clients is skewed and a global model cannot capture the data pattern of each client.

Problem: Federated learning models trained with non-IID data suffer from low accuracy and are less generalised to the entire dataset. Furthermore, the local data that is highly personalised to the device users' usage pattern creates local models that diverge in different directions. Hence, the global model may have relatively low averaged accuracy.

Forces: The problem requires to balance the following forces:

- *Computation cost.* The complex model that solves the non-IID issue consumes more computation and energy resources every round compared to general federated model training. It also takes longer to compute all the training results from the different tasks before submitting them to the central server.
- *Data privacy.* To address the non-IID issue, more information from the local data needs to be explored to understand the data distribution pattern. This ultimately exposes client devices to local data privacy threats.

Solution: The multi-task model trainer performs different-but-related machine learning tasks on client devices. This enables the local model to learn from more local data that fit naturally to the related local models for different tasks. For instance, a multi-task model for the next-word prediction task is trained using the on-device text messages, web browser search strings, and emails with similar mobile keyboard usage patterns.

Consequences:

Benefits:

- *Model quality.* Multi-task learning improves the model performance by considering local data and loss in optimisation and obtaining a local weight matrix through this process. The local model fits for non-IID data in each node better than a global model.

15 https://docs.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment

16 https://docs.aws.amazon.com/sagemaker/latest/dg/multi-model-endpoints.html

17 https://cloud.google.com/ai-platform/prediction/docs/deploying-models

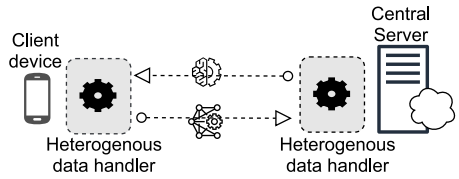

Fig. 12. Heterogeneous data handler.

- *System efficiency*. Multi-task training can train a model using more than one related task which increases the speed of model training to reach convergence.

Drawbacks:

- *Model quality*. Multi-task training often works only with convex loss functions and performs weak on non-convex loss functions.
- *Model portability*. As each client has a different model, the model's portability is a problem that makes it hard to apply multi-task training on cross-device FL.

Related patterns: *Client Registry, Model Co-versioning Registry, Client Cluster, Deployment Selector*

Known uses:

- *MultiModel*[18] is a neural network architecture by Google that simultaneously solves several problems spanning multiple domains, including image recognition, translation, and speech recognition.
- *MT-DNN*[19] is an open-source natural language understanding toolkit by Microsoft to train customised deep learning models.
- *Yahoo Multi-Task Learning for Web Ranking* is a multi-task learning framework developed by Yahoo! Labs to rank in web search.
- *MOCHA* (Smith et al., 2017) is a state-of-the-art federated multi-task learning algorithm that realises distributed multi-task learning on federated settings.
- *VIRTUAL* (Corinzia and Buhmann, 2019) is an algorithm for federated multi-task learning with non-convex models. The server and devices are treated as a star-shaped bayesian network, and model learning is performed on the network using approximated variational inference.

3.3.2. Pattern 9: Heterogeneous data handler

Summary: Heterogeneous data handler solves the non-IID and skewed data distribution issues through data volume and data class addition (e.g., data augmentation or generative adversarial network) while maintaining the local data privacy. The pattern is illustrated in Fig. 12, where the heterogeneous data handler operates at both ends of the system.

Context: Client devices possess heterogeneous data characteristics due to the highly personalised data generation pattern. Furthermore, the raw local data cannot be shared so the data balancing task becomes extremely challenging.

Problem: The imbalanced and skewed data distribution of client devices produces local models that are not generalised to the entire client network. The aggregation of these local models reduces global model accuracy.

Forces: The problem requires the following forces to be balanced:

- *Data efficiency*. It is challenging to articulate the suitable data volume and classes to be augmented to solve data heterogeneity on local client devices.
- *Data accessibility*. The heterogeneous data issue that exists within the client device can be solved by collecting all the data under a centralised location. However, this violates the data privacy of client devices.

Solution: A heterogeneous data handler balances the data distribution and solves the data heterogeneity issue in the client devices through data augmentation and federated distillation. Data augmentation solves data heterogeneity by generating augmented data locally until the data volume is the same across all client devices. Furthermore, the classes in the datasets are also populated equally across all client devices. Federated distillation enables the client devices to obtain knowledge from other devices periodically without directly accessing the data of other client devices. Other methods includes taking the quantified data heterogeneity weightage (e.g, Pearson's correlation, centroid averaging-distance, etc.) into account for model aggregation.

Consequences:

Benefits:

- *Model quality*. By solving the non-IID issue of local datasets, the performance and generality of the global model are improved.
- *Training efficiency*. The training process is also faster and more efficiency under IID data.

Drawbacks:

- *Computation cost*. It is computationally costly to deal with data heterogeneity together with the local model training.

Related patterns: *Client Registry, Client Selector, Client Cluster*

Known uses:

- *Astreae* (Duan et al., 2019) is a self-balancing federated learning framework that alleviates the imbalances by performing global data distribution-based data augmentation.
- Federated Augmentation (*FAug*) (Jeong et al., 2018) is a data augmentation scheme that utilises a generative adversarial network (GAN) which is collectively trained under the tradeoff between privacy leakage and communication overhead.
- Federated Distillation (*FD*) (Ahn et al., 2019) is a method that adopted knowledge distillation approaches to tackle the non-IID issue by obtaining the knowledge from other devices during the distributed training process, without accessing the raw data.

3.3.3. Pattern 10: Incentive registry

Summary: An incentive registry maintains the list of participating clients and their rewards that correspond to clients' contributions (e.g., data volume, model performance, computation resources, etc.) to motivate clients' participation. Fig. 13 illustrates a blockchain & smart contract-based incentive mechanism.

Context: The model training participation rate of client devices is low while the high participation rate is crucial for the global model performance.

Problem: Although the system relies greatly on the participation of clients to produce high-quality global models, clients are not mandatory to join the model training and contribute their data/resources.

Forces: The problem requires to balance the following forces:

[18] https://ai.googleblog.com/2017/06/multimodel-multi-task-machine-learning.html

[19] https://github.com/microsoft/MT-DNN

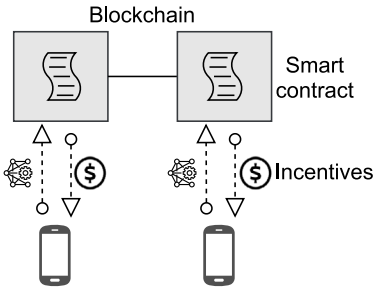

Fig. 13. Incentive registry.

- *Incentive scheme.* It is challenging to formulate the form of rewards to attract different clients with different participation motives. Furthermore, the incentive scheme needs to be agreed upon by both the learning coordinator and the clients, e.g., performance-based, data-contribution-based, resource-contribution-based, and provision-frequency-based.
- *Data privacy.* To identify the contribution of each client device, the local data and client information is required to be studied and analysed by the central server. This exposes the client devices' local data to privacy threats.

Solution: An incentive registry records all client's contributions and provide incentives based on the rate agreed. There are various ways to formulate the incentive scheme, e.g., deep reinforcement learning, blockchain/smart contracts, and the Stackelberg game model.

Consequences:

Benefits:

- *Client motivatability.* The incentive mechanism is effective in attracting clients to contribute data and resources to the training process.

Drawbacks:

- *System security.* There might be dishonest clients that submit fraudulent results to earn rewards illegally and harm the training process.

Related patterns: *Client Registry, Client Selector*

Known uses:

- *FLChain* (Bao et al., 2019) is a federated learning blockchain providing an incentive scheme for collaborative training and a market place for model trading.
- *DeepChain* (Weng et al., 2019) is a blockchain-based collaborative training framework with an incentive mechanism that encourages parties to participate in the deep learning model training and share the obtained local gradients.
- *FedCoin* (Liu et al., 2020) is a blockchain-based peer-to-peer payment system for federated learning to enable Shapley Value (SV) based reward distribution.

3.4. Model aggregation patterns

Model aggregation patterns are design solutions of model aggregation used for different purposes. *Asynchronous aggregator* aims to reduce aggregation latency and increase system efficiency, whereas *decentralised aggregator* targets to increase system reliability and accountability. *Hierarchical aggregator* is adopted to improve model quality and optimises resources. *Secure aggregator* is designed to protect the models' security.

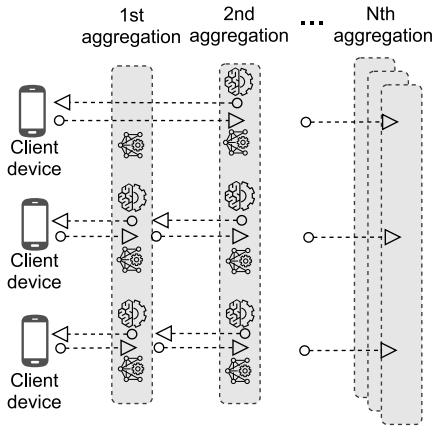

Fig. 14. Asynchronous aggregator.

3.4.1. Pattern 11: Asynchronous aggregator

Summary: To increase the global model aggregation speed every round, the central server can perform model aggregation asynchronously whenever a model update arrives without waiting for all the model updates every round. In Fig. 14, the asynchronous aggregator pattern is demonstrated as the first client device asynchronously uploads its local model during the second aggregation round while skipping the first aggregation round.

Context: In conventional federated learning, the central server receives all local model updates from the distributed client devices synchronously and performs model aggregation every round. The central server needs to wait for every model to arrive before performing the model aggregation for that round. Hence, the aggregation time depends on the last model update that reaches the central server.

Problem: Due to the difference in computation resources, the model training lead time is different per device. Furthermore, the difference in bandwidth availability, communication efficiency affects the model's transfer rate. Therefore, the delay in model training and transfer increases the latency in global model aggregation.

Forces: The problem requires the following forces to be balanced:

- *Model quality.* There will be possible bias in the global model if not all local model updates are aggregated in every iteration as important information or features might be left out.
- *Aggregation latency.* The aggregation of local models can only be performed when all the model updates are collected. Therefore, the latency of the aggregation process is affected by the slowest model update that arrives at the central server.

Solution: The global model aggregation is conducted whenever a model update is received, without being delayed by other clients. Then the server starts the next iteration and distributes the new central model to the clients that are ready for training. The delayed model updates that are not included in the current aggregation round will be added in the next round with some reduction in the weightage, proportioned to their respective delayed time.

Consequences:

Benefits:

- *Low aggregation latency.* Faster aggregation time per round is achievable as there is no need to wait for the model updates from other clients for the aggregation round. The bandwidth usage per iteration is reduced as fewer local model updates are transferred and received simultaneously every round.

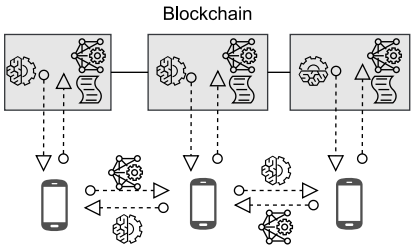

Fig. 15. Decentralised aggregator.

Drawbacks:

- *Communication cost.* The number of iteration to collect all local mode updates increases for the asynchronous approach. More iterations are required for the entire training process to train the model till convergence compares to synchronous global model aggregation.
- *Model bias.* The global model of each round does not contain all the features and information of every local model update. Hence the global model might have a certain level of bias in prediction.

Related patterns: *Client Registry, Client Selector, Model Co-versioning Registry, Client Update Scheduler*

Known uses:

- Asynchronous Online Federated Learning (*ASO-fed*) (Chen et al., 2020) is a framework for federated learning that adopted asynchronous aggregation. The central server update the global model whenever it receives a local update from one client device (or several client devices if the local updates are received simultaneously). On the client device side, online-learning is performed as data continue to arrive during the global iterations.
- Asynchronous federated SGD-Vertical Partitioned (*AFSGD-VP*) (Gu et al., 2020) algorithm uses a tree-structured communication scheme to perform asynchronous aggregation. The algorithm does not need to align the iteration number of the model aggregation from different client devices to compute the global model.
- Asynchronous Federated Optimisation (*FedAsync*) (Xie et al., 2020) is an approach that leverages asynchronous updating technique and avoids server-side timeouts and abandoned rounds while requires no synchronous model broadcast to all the selected client devices.

3.4.2. Pattern 12: Decentralised aggregator

Summary: A decentralised aggregator improves system reliability and accountability by removing the central server that is a possible single-point-of-failure. Fig. 15 illustrates the decentralised federated learning system built using blockchain and smart contract, while the model updates are performed through the exchange between neighbour devices.

Context: The model training and aggregation are coordinated by a central server and both the central server and the owner may not be trusted by all the client devices that join the training process.

Problem: In *FedAvg*, all the chosen devices have to submit the model parameters to one central server every round. This is extremely burdensome to the central server and network congestion may occur. Furthermore, centralised federated learning possesses a single-point-of-failure. Data privacy threats may also occur if the central server is compromised by any unauthorised entity. The mutual trust between the client devices and the central server may not be specifically established.

Forces: The problem requires to balance the following forces:

- *Decentralised model management.* The federated learning systems face challenges to collect, store, examine, and aggregate the local models due to the removal of the central server.
- *System ownership.* Currently, the central server is own by the learning coordinator that creates the federated learning jobs. The removal of the central server requires the re-definition of system ownership. It includes the authority and accessibility of learning coordinator in the federated learning systems.

Solution: A decentralised aggregator replaces the central server's role in a federated learning system. The aggregation and update of the models can be performed through peer-to-peer exchanges between client devices. First, a random client from the system can be an aggregator by requesting the model updates from the other clients that are close to it. Simultaneously, the client devices conduct local model training in parallel and send the trained local models to the aggregator. The aggregator then produces a new global model and sends it to the client network. Blockchain is the alternative to the central server for model storage that prevents single-point-of-failure. The ownership of the blockchain belongs to the learning coordinator that creates the new training tasks and maintains the blockchain. Furthermore, the record of models on a blockchain is immutable that increases the reliability of the system. It also increases the trustworthiness of the system as the record is transparent and accessible by all the client devices.

Consequences:

Benefits:

- *System reliability.* The removal of single-point-of-failure increases the system reliability by reducing the security risk of the central server from any adversarial attack or the failure of the entire training process due to the malfunction of the central server.
- *System accountability.* The adoption of blockchain promotes accountability as the records on a blockchain is immutable and transparent to all the stakeholders.

Drawbacks:

- *Latency.* The client device as a replacement of the central server for model aggregation is not ideal for direct communication with multiple devices. The blockchain consensus protocols may also cause latency in the model aggregation process.
- *Computation cost.* Client devices have limited computation power and resource to perform model training and aggregation parallel. Even if the training process and the aggregation is performed sequentially, the energy consumption to perform multiple rounds of aggregation is very high.
- *Storage cost.* High storage cost is required to store all the local and global models on storage-limited client devices or blockchain.
- *Data privacy.* Client devices can access the record of all the models under decentralised aggregation and blockchain settings. This might expose the privacy-sensitive information of the client devices to other parties.

Related patterns: *Model Co-versioning Registry, Incentive Registry*

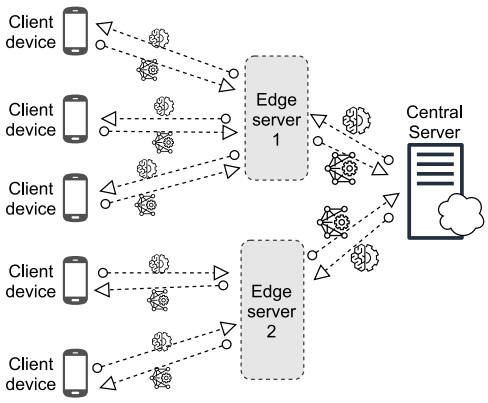

Fig. 16. Hierarchical aggregator.

Known uses:

- *Swarm Learning* (Warnat-Herresthal et al., 2021) is a decentralised approach that utilises edge computing, blockchain-based peer-to-peer networking and coordination while maintaining confidentiality without the need for a central coordinator.
- *BrainTorrent* (Roy et al., 2019) is a server-less, peer-to-peer approach to perform federated learning where clients communicate directly among themselves, specifically for federated learning in medical centres.
- *FedPGA* (Jiang and Hu, 2020) is a decentralised aggregation algorithm developed from *FedAvg*. The devices in *FedPGA* exchange partial gradients rather than full model weights. The partial gradient exchange pulls and merges the different slice of the updates from different devices and rebuild a mixed update for aggregation.
- A fully decentralised framework (Lalitha et al., 2018) is an algorithm in which users update their beliefs by aggregate information from their one-hop neighbours to learn a model that best fits the observations over the entire network.
- A Segmented gossip approach (Hu et al., 2019) splits a model into segmentation that contains the same number of non-overlapping model parameters. Then, the gossip protocol is adopted where each client stochastically selects a few other clients to exchange the model segmentation for each training iteration without the orchestration of a central server.

3.4.3. Pattern 13: Hierarchical aggregator

Summary: To reduce non-IID effects on the global model and increase system efficiency, a hierarchical aggregator adds an intermediate layer (e.g., edge server) to perform partial aggregations using the local model parameters from closely-related client devices before the global aggregation. In Fig. 16, edge servers are added as an intermediate layer between the central server and client devices to serve the client devices that are closer to them.

Context: The communication between the central server and the client devices is slowed down or frequently disrupted due to being physically distant from each other and are wirelessly connected.

Problem: The central server can access and store more data but requires high communication overhead and suffers from latency due to being physically distant from the client devices. Moreover, client devices possess non-IID characteristics that affect global model performance.

Forces: The problem requires the following forces to be balanced:

- *System efficiency.* The system efficiency of the server–client setting to perform federated learning is low, as the central server is burdensome to accommodate the communication and the model aggregations of the widely-distributed client devices.
- *Data heterogeneity.* In the server–client setting of a federated learning system, the data heterogeneity characteristics of client devices become influential and dominant to the global model production, as the central server deals with all the client devices that generate non-IID data.

Solution: A hierarchical aggregator adds edge servers between the central server and client devices. The combination of server-edge-client architecture can improve both computation and communication efficiency of the federated model training process. Edge servers collect local models from the nearest client devices, a subset of all the client devices. After every k1 round of local training on each client, each edge server aggregates its clients' models. After every k2 edge model aggregations, the cloud server aggregates all the edge servers' models, which means the communication with the central server happens every k1k2 local updates (Liu et al., 2020).

Consequences:

Benefits:

- *Communication efficiency.* The hierarchical aggregators speed up the global model aggregation and improve communication efficiency.
- *Scalability.* Adding an edge layer helps to scale the system by improving the system's ability to handling more client devices.
- *Data heterogeneity and non-IID reduction.* The partial aggregation in a hierarchical manner aggregates local models that have similar data heterogeneity and non-IIDness before the global aggregation on the central server. This greatly reduces the effect of data heterogeneity and non-IIDness on global models.
- *Computation and storage efficiency.* The edge devices are rich with computation and storage resources to perform partial model aggregation. Furthermore, edge devices are nearer to the client devices which increase the model aggregation and computation efficiency.

Drawbacks:

- *System reliability.* The failure of edge devices may cause the disconnection of all the client devices under those edge servers and affect the model training process, model performance, and system reliability.
- *System security.* Edge servers could become security breach points as they have lower security setups than the central server and the client devices. Hence, they are more prone to network security threats or becoming possible points-of-failure of the system.

Related patterns: *Client Registry, Client Cluster, Model Co-versioning Registry*

Known uses:

- *HierFAVG* is an algorithm that allows multiple edge servers to perform partial model aggregation incrementally from the collected updates from the client devices.
- Hierarchical Federated Learning (*HFL*) enables hierarchical model aggregation in large scale networks of client devices where communication latency is prohibitively large due to limited bandwidth. The *HFL* seeks a consensus on the model and uses edge servers to aggregate model updates from client devices that are geographically near.

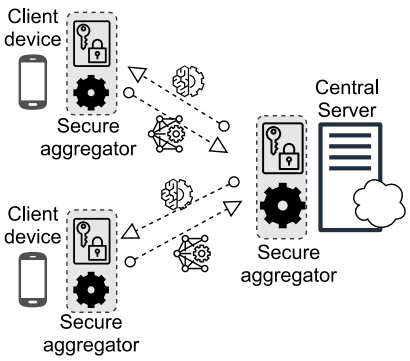

Fig. 17. Secure aggregator.

- Federated Learning + Hierarchical Clustering (*FL+HC*) is the addition of a hierarchical clustering algorithm to the federated learning system. The cluster is formed according to the data distributions similarity based on the following distance metrics: (1) Manhattan, (2) Euclidean, (3) Cosine distance metrics.
- *Astraea* is a federated learning framework that tackles non-IID characteristics of federated clients. The framework introduces a mediator to the central server and the client devices to balance the skewed client data distributions. The mediator performs the z-score-based data augmentation and downsampling to relieve the global imbalanced of training data.

3.4.4. Pattern 14: Secure aggregator

Summary: A secure aggregator manages the model exchange and aggregation security protocols to protect model security. Fig. 17 illustrates the security aggregator on each components with the security protocols embedded in them.

Context: The central server sends global models to any existing or unknown device every round with no data privacy and security protocols that protect the communication from unauthorised access. Furthermore, model parameters contain pieces of private user information that can be inferred by the data-hungry machine learning algorithms.

Problem: There are no security measures to tackle the honest-but-curious and active adversary security threats which exist in federated learning systems.

Forces: The problem requires to balance the following forces:

- *Client device security.* Client device security issues exist when dishonest and malicious client devices join the training process and poison the overall model performance by disrupting the training process or providing false updates to the central server.
- *Data security.* Data security of the client devices is challenged when the gradients or model parameters are inferred by unauthorised parties through the data-hungry machine learning algorithms.

Solution: A security aggregator handles the secure multiparty computation (SMC) protocols for model exchanges and aggregations. The protocols provide security proof to guarantee that each party knows only its input and output. For instance, homomorphic encryption is a method to encrypt the models and only allow authorised client devices and the central server to decrypt and access the models. Pairwise masking and differential privacy (DP) methods are applied to reduce the interpretability of the model by unauthorised party (Yang et al., 2019). The technique involves adding noise to the parameters or gradient or uses a generalised method.

Consequences:

Benefits:

- *Data security.* The secure aggregator protects the model from being access by adversarial and unauthorised parties through homomorphic encryptions and prevents information leakage due to the data-hungry property of machine learning models.

Drawbacks:

- *System efficiency.* The extra security processes affect the system efficiency if excessive security steps are required every round for every device. It also lowers the training and aggregation speed due to encryption and decryption time.
- *Model performance-privacy tradeoff.* The model performance is affected if the model privacy methods aggressively interfere with the model's interpretability due to being excessively obscure.
- *Compromised key.* For encryption and decryption functions, the possible compromise of the security keys increases the privacy threat.

Related patterns: *Client Registry, Model Co-versioning Registry*

Known uses:

- *SecAgg* (Bonawitz et al., 2017) is a practical protocol by Google for secure aggregation in the federated learning settings.
- *HybridAlpha* (Xu et al., 2019a) is a framework that manages the client devices that join the federated learning process. The security operation includes functional encryption, DP, and SMC.
- *TensorFlow Privacy Library*[20] provides an implementation of DP-SGD machine learning.
- *ZamaAI*[21] is an AI service platform that provides encryption technology that uses a homomorphic compiler to convert the model into an end-to-end encrypted parameters.
- Simple Encrypted Arithmetic Library (*SEAL*[22]) is a homomorphic encryption API introduced by Microsoft AI to allow computations to be performed directly on encrypted data.

3.4.5. Pattern 15: Training configurator

Summary: A training configurator provides a non-coding, platform-as-a-service for FL users to configure their FL model training hyperparameters and settings easily, without needing to code. Fig. 18 illustrates the training configurator.

Context: FL model training requires users to configure the training process which requires high proficiency in software and hardware engineering knowledge.

Problem: The usability of FL services is low as the technical skills required to implement a FL system is high and there is no easy-to-use platform-as-a-service for FL service users.

Forces: The problem requires to balance the following forces:

- *Reliability.* A user-friendly platform or framework should be reliable and available at all times for the user to configure the federated learning process or connectivity with multiple devices relentlessly.

[20] https://github.com/tensorflow/privacy/
[21] https://zama.ai/
[22] https://www.microsoft.com/en-us/research/project/microsoft-seal/

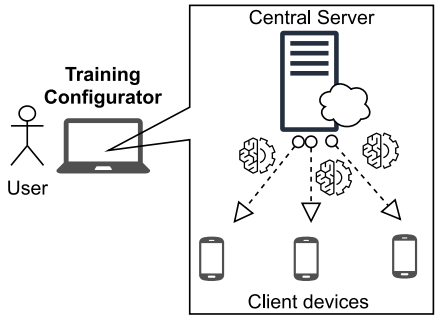

Fig. 18. Training configurator.

- *Robustness.* A user-friendly platform or framework can become a single-point-of-failure for the model training process it deploys.

Solution: A training configurator provides a platform-as-a-service to FL applications users to configure the FL training process through GUI, without having to code the system and configure the hardware by themselves. A series of FL model training configuration options and settings (training hyperparameters, secure aggregations, synchronous/asynchronous aggregation, multi-task training, incentives provision, etc.) can be selected by users with just a click of a button. This increases the usability of FL services or application as it lowers the knowledge boundary required to train and deploy a FL model. Many state-of-the-art approaches can be implemented by experts at the backend of the configurator to increase the efficiency and the performance of the FL training while maintaining the configurator's reliability.

Consequences:

Benefits:

- *Usability.* The training configurator enables users to configure the FL training processes without needing to code. It increases the usability of FL services or applications and simplifies the process to configure the central server and client devices' communication and aggregation processes.
- *System efficiency. A centrally maintained training configurator ensures high system efficiency and makes the adoption of state-of-the-art practices to implement the FL training easier.*

Drawbacks:

- *Flexibility.* Less customisability on the configuration options and model choices are available to users compared to FL systems that are built from scratch by the users.
- *Scalability.* The pattern may face scalability issues to support more users and devices associated with the expansion of the systems.

Related patterns: *Client Registry, Model Co-versioning Registry, Client Selector, Client Cluster, Secure Aggregator, Asynchronous Aggregator, Multi-task Model Trainer, Deployment Selector, Model Replacement Trigger, Incentive Registry*

Known uses:

- *AutoML*[23] by Google reduces the need for skilled data scientists to build machine learning and deep learning models. Users can simple provide labelled data as input and get the trained model as output.
- *Fritz AL*[24] is a ML platform that helps to bridge the gap between mobile developers and data scientists by providing quick model training and deployment or pre-trained SDK which provides out-of-the-box support.
- *MakeML*[25] is an AI service platform that creates object detection and segmentation ML models without code with datasets provided.

4. Discussion

To provide a clear view on how each pattern contributes or degrades different non-functional requirements, we have conducted a pattern-tradeoff analysis and presented it in Table 4. Note that our pattern framework may lack comprehensiveness. The pattern framework outlines the different strengths and tradeoffs of the patterns collected independently, without considering the tradeoffs dependency across different patterns. Hence, not all the quality attributes have tradeoffs caused by these patterns. Certain quality attributes are still short of the patterns to address them at the point of this work.

First, *client registry*, *client selector*, and *client cluster* are proposed for client device management to improve model performance, system, and training efficiency. However, these patterns introduce extra computation workload and security issues when more client information is required.

During the model training stage, the performance tradeoffs often occur due to the non-IID nature of the local data. The patterns proposed to address this issue are *heterogeneous data handler*, *client cluster*, and *hierarchical aggregator*. *Multi-task model trainer* adopts multi-task or transfer learning techniques to learn different models or personalise a global model on local data to optimise the model performance for clients with different local data characteristics, whereas the *deployment selector* effectively selects the user clients to receive the personalised models that fit their local data. These patterns however introduce extra computational cost and security threats. *Incentive registry* effectively motivates more clients to participant in the training process to improve the model performance but reliability and security issues may exist due to dishonest node attack and model bias issues.

For the model exchange and aggregation stages, communication and computation efficiency become a system bottleneck. To effectively tackle these issues, *client selector*, *client cluster*, *deployment selector*, *asynchronous aggregator*, and *hierarchical aggregator* are embedded in the system to optimise resource consumption. However, these patterns require extra client information (i.e., resource or performance) to perform the selection or scheduling of updates. Intuitively, the collection and analysis of the client information on the central server may lead to another form of data privacy violation. Furthermore, extra computation and communication resources are consumed to collect and analyse the client information, in addition to the model training task and the fundamental tasks of the client devices. Hence, *message compressor* and *hierarchical aggregator* are proposed to tackle these issues. The message compressor however degrades the quality of the local models depending on the degree of size reduction.

The system security issue is present due to the distributed ownership of federated learning system components. Unauthorised, adversarial clients may join the system and obtain model parameters from the system or harm the model or system performance by uploading dishonest updates. *Secure aggregator*, *model co-versioning registry*, and *client registry* aim to solve these challenges. Furthermore, *model co-versioning registry*, and *client registry* especially improve the accountability and traceability of the

23 https://cloud.google.com/automl

24 https://www.fritz.ai/

25 https://makeml.app/?from=github_potato_weigher

Table 4
Tradeoff anaylsis.

Pattern	Model performance	Scalability	Statistical heterogeneity	System heterogeneity	Accountability (Auditability, traceability)	Motivatability	Reliability	Communication efficiency	Computation efficiency	Security	Usability
Client registry	–	✓	–	–	✓	–	✓	–	✓	✗	–
Client selector	✓	✓	–	✓	✗	–	–	✓	✓	–	–
Client cluster	✓	✓	✓	✓	–	–	–	–	✗	–	–
Message compressor	✗	✓	–	–	–	–	–	✓	✗	–	–
Model co-versioning registry	–	–	–	–	✓	–	✓	–	✗	–	–
Model replacement trigger	✓	–	–	–	–	–	–	–	✓	–	–
Deployment selector	✓	✗	–	–	–	–	–	–	–	✗	–
Multi-task model trainer	✓	–	–	–	–	–	–	–	✗	✗	–
Heterogeneous data handler	✓	–	✓	–	–	–	–	–	✗	–	–
Incentive registry	–	–	–	–	✓	✓	–	–	✗	✗	–
Asynchronous aggregator	–	✓	–	✓	–	–	–	✗	✓	–	–
Decentralised aggregator	–	–	–	–	✓	–	✓	✗	✗	–	–
Hierarchical aggregator	✓	✓	✓	✓	–	–	✗	✓	✓	–	–
Secure aggregator	✗	✗	–	✗	–	–	–	–	✗	✓	–
Training configurator	–	–	–	–	–	–	✗	✓	✓	–	✓

training process. However, security threat exist as more client information may be collected. The trustworthiness between the client devices and the central server is also a challenge to gain the participation of clients. The central server may also be a single-point-of-failure that may affect the reliability of the system. Hence, *decentralised aggregator* is proposed to solve the issue. However, the decentralised setting suffers from communication and computation efficiency during model transfer and aggregation. Finally, the technical skills and knowledge to implement a FL system from scratch are high and challenging. This decreases the usability of FL applications or services. Hence, a *training configurator* as a user-friendly platform enables users to configure and build their FL system without needing to code or set up the hardware connections explicitly. However, the training configurator lacks customisability and suffers from reliability issues due to the single-point-of-failure.

5. Related work

In many real-world scenarios, machine learning applications are usually embedded as a software component to a larger software system at enterprise level. Hence, to promote enterprise level adoption of machine learning-based applications, many researchers view machine learning models as a component of a software system so that the challenges in building machine learning models can be tackled through systematic software engineering approaches.

Wan et al. (2019) studied how the adoption of machine learning changes software development practices. The work characterises the differences in various aspects of software engineering and task involved for machine learning system development and traditional software development. Lwakatare et al. (2019) propose a taxonomy that depicts maturity stages of use of machine learning components in the industrial software system and mapped the challenges to the machine learning pipeline stages. Building machine learning models is becoming an engineering discipline where practitioners take advantage of tried-and-proven methods to address recurring problems (Lakshmanan et al., 2020).

Washizaki et al. (2019) studies the machine learning design patterns and architectural patterns. The authors also proposed an architectural pattern for machine learning for improving operational stability (Yokoyama, 2019).

The research on federated learning system design was first done by Bonawitz et al. (2019), focusing on the high-level design of a basic federated learning system and its protocol definition. However, there is no study on the definition of architecture patterns or reusable solutions to address federated learning design challenges currently. Our work addresses this particular gap with respect to software architecture designs for federated learning as a distributed software system. To the best of our knowledge, this is the first comprehensive and systematic collection of federated learning architectural patterns. The outcomes are intended to provide architectural guidance for practitioners to better design and develop federated learning systems.

6. Conclusion

Federated learning systems are troubled by architectural challenges that need to be solved before they can be effectively adopted in the real-world. In this paper, we present 15 federated learning architectural patterns associated with the lifecycle of a model in federated learning. The pattern collection is provided as architectural guidance for architects to better design and develop federated learning systems. In our future work, we will explore the architectural designs that help improve the trust in federated learning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ahmad, A., Jamshidi, P., Pahl, C., Khaliq, F., 2014. A pattern language for the evolution of component-based software architectures. ECEASST 59.

Ahn, J., Simeone, O., Kang, J., 2019. Wireless federated distillation for distributed edge learning with heterogeneous data. In: PIMRC 2019. pp. 1–6.

Bai, J., Zhou, K., Xue, G., Zha, H., Sun, G., Tseng, B., Zheng, Z., Chang, Y., 2009. Multi-task learning for learning to rank in web search. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, pp. 1549–1552.

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D.A., Lynn, T., 2018. Microservices migration patterns. Softw. - Pract. Exp. 48 (11), 2019–2042.
Bao, X., Su, C., Xiong, Y., Huang, W., Hu, Y., 2019. Flchain: A blockchain for auditable federated learning with trust and incentive. In: BIGCOM '19. pp. 151–159.
Beck, K., Cunningham, W., 1987. Using pattern languages for object oriented programs. In: OOPSLA '12.
Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, B., Van Overveldt, T., Petrou, D., Ramage, D., Roselander, J., 2019. Towards federated learning at scale: system design. In: Talwalkar, A., Smith, V., Zaharia, M. (Eds.), Proceedings of Machine Learning and Systems. 1, pp. 374–388, https://proceedings.mlsys.org/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf.
Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K., 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. Association for Computing Machinery, New York, NY, USA.
Bowden, G.J., Maier, H.R., Dandy, G.C., 2012. Real-time deployment of artificial neural network forecasting models: Understanding the range of applicability. Water Resour. Res. 48 (10).
Briggs, C., Fan, Z., Andras, P., 2020. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–9. http://dx.doi.org/10.1109/IJCNN48605.2020.9207469.
Brinkkemper, S., 1996. Method engineering: engineering of information systems development methods and tools. Inf. Softw. Technol. 38 (4), 275–280, Method Engineering and Meta-Modelling.
Chai, Z., Ali, A., Zawad, S., Truex, S., Anwar, A., Baracaldo, N., Zhou, Y., Ludwig, H., Yan, F., Cheng, Y., 2020. Tifl: A tier-based federated learning system. In: ACM HPDC. In: HPDC '20, Association for Computing Machinery, New York, NY, USA, pp. 125–136.
Chen, Y., Ning, Y., Slawski, M., Rangwala, H., 2020. Asynchronous online federated learning for edge devices with non-iid data. In: 2020 IEEE International Conference on Big Data (Big Data). IEEE Computer Society, Los Alamitos, CA, USA, pp. 15–24. http://dx.doi.org/10.1109/BigData50022.2020.9378161, https://doi.ieeecomputersociety.org/10.1109/BigData50022.2020.9378161.
Corinzia, L., Buhmann, J.M., 2019. Variational federated multi-task learning. arXiv:1906.06268.
Duan, M., Liu, D., Chen, X., Tan, Y., Ren, J., Qiao, L., Liang, L., 2019. Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications. In: ICCD 2019. pp. 246–254.
EU, 2020. General data protection regulation (GDPR). URL https://eur-lex.europa.eu/eli/reg/2016/679/oj.
Gu, B., Xu, A., Huo, Z., Deng, C., Huang, H., 2020. Privacy-preserving asynchronous federated learning algorithms for multi-party vertically collaborative learning. arXiv:2008.06233.
Haddadpour, F., Kamani, M.M., Mokhtari, A., Mahdavi, M., 2021. Federated learning with compression: unified analysis and sharp guarantees. In: International Conference on Artificial Intelligence and Statistics. PMLR, pp. 2350–2358.
Hu, C., Jiang, J., Wang, Z., 2019. Decentralized federated learning: A segmented gossip approach. arXiv:1908.07782.
Huang, L., Shea, A.L., Qian, H., Masurkar, A., Deng, H., Liu, D., 2019. Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. J. Biomed. Inform. 99, 103291.
Jamshidi, P., Pahl, C., Mendonça, N.C., 2017. Pattern-based multi-cloud architecture migration. Softw. - Pract. Exp. 47 (9), 1159–1184.
Jeong, E., Oh, S., Kim, H., Park, J., Bennis, M., Kim, S.-L., 2018. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-IID private data. arXiv:1811.11479.
Jiang, J., Hu, L., 2020. Decentralised federated learning with adaptive partial gradient aggregation. CAAI Trans. Intell. Technol. 5 (3), 230–236.
Jiang, Y., Wang, S., Valls, V., Ko, B.J., Lee, W.-H., Leung, K.K., Tassiulas, L., 2022. Model pruning enables efficient federated learning on edge devices. IEEE Trans. Neural Netw. Learn. Syst. 1–13. http://dx.doi.org/10.1109/TNNLS.2022.3166101.
Jobin, A., Ienca, M., Vayena, E., 2019. The global landscape of AI ethics guidelines. Nat. Mach. Intell. 1 (9), 389–399.
Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering.
Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D., 2017. Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492.
Lakshmanan, L.V., Munn, M., Robinson, S., 2020. Machine Learning Design Patterns.
Lalitha, A., Shekhar, S., Javidi, T., Koushanfar, F., 2018. Fully decentralized federated learning. In: Third Workshop on Bayesian Deep Learning (NeurIPS).
Li, T., Sahu, A.K., Talwalkar, A., Smith, V., 2020. Federated learning: Challenges, methods, and future directions. IEEE Signal Process. Mag. 37 (3), 50–60.
Liu, Y., Ai, Z., Sun, S., Zhang, S., Liu, Z., Yu, H., 2020. In: Yang, Q., Fan, L., Yu, H. (Eds.), FedCoin: A Peer-to-Peer Payment System for Federated Learning. Springer International Publishing, Cham, pp. 125–138.
Liu, L., Zhang, J., Song, S.H., Letaief, K.B., 2020. Client-edge-cloud hierarchical federated learning. In: ICC 2020. pp. 1–6.
Lo, S.K., Liew, C.S., Tey, K.S., Mekhilef, S., 2019. An interoperable component-based architecture for data-driven IoT system. Sensors 19 (20).
Lo, S.K., Lu, Q., Paik, H.-Y., Zhu, L., 2021a. FLRA: A reference architecture for federated learning systems. In: Software Architecture. Springer International Publishing, pp. 83–98.
Lo, S.K., Lu, Q., Wang, C., Paik, H.-Y., Zhu, L., 2021b. A systematic literature review on federated machine learning: From a software engineering perspective. ACM Comput. Surv. 54 (5).
Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H., Crnkovic, I., 2019. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Kruchten, P., Fraser, S., Coallier, F. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, Cham, pp. 227–243.
McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A., 2017. Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. PMLR, pp. 1273–1282.
Meszaros, G., Doble, J., 1997. A pattern language for pattern writing. In: Pattern Languages of Program Design 3. Addison-Wesley Longman Publishing Co., Inc., USA, pp. 529–574.
Mills, J., Hu, J., Min, G., 2020. Communication-efficient federated learning for wireless edge intelligence in IoT. IEEE Internet Things J. 7 (7), 5986–5994.
Mirbel, I., Ralyté, J., 2006. Situational method engineering: combining assembly-based and roadmap-driven approaches. Requir. Eng. 11 (1), 58–78.
Roy, A.G., Siddiqui, S., Pölsterl, S., Navab, N., Wachinger, C., 2019. Braintorrent: A peer-to-peer environment for decentralized federated learning. arXiv:1905.06731.
Sattler, F., Wiedemann, S., Müller, K.R., Samek, W., 2020. Robust and communication-efficient federated learning from non-i.i.d. data. IEEE Trans. Neural Netw. Learn. Syst. 31 (9), 3400–3413.
Smith, V., Chiang, C.-K., Sanjabi, M., Talwalkar, A., 2017. Federated multi-task learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 4427–4437.
Wan, Z., Xia, X., Lo, D., Murphy, G.C., 2019. How does machine learning change software development practices? IEEE Trans. Softw. Eng. 1.
Wang, L., Wang, W., Li, B., 2019. CMFL: Mitigating communication overhead for federated learning. In: ICDCS 2019. pp. 954–964.
Warnat-Herresthal, S., Schultze, H., Shastry, K.L., Manamohan, S., Mukherjee, S., Garg, V., Sarveswara, R., Händler, K., Pickkers, P., Aziz, N.A., et al., 2021. Swarm learning for decentralized and confidential clinical machine learning. Nature 594 (7862), 265–270.
Washizaki, H., Uchida, H., Khomh, F., Guéhéneuc, Y., 2019. Studying software engineering patterns for designing machine learning systems. In: 2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 49–495.
Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y., Luo, W., 2019. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. IEEE Trans. Dependable Secure Comput. 1.
Xie, C., Koyejo, S., Gupta, I., 2020. Asynchronous federated optimization. arXiv:1903.03934.
Xu, R., Baracaldo, N., Zhou, Y., Anwar, A., Ludwig, H., 2019a. Hybridalpha: An efficient approach for privacy-preserving federated learning. In: AISec'19. Association for Computing Machinery, New York, NY, USA, pp. 13–23.
Xu, Z., Yu, F., Xiong, J., Chen, X., 2021. Helios: heterogeneity-aware federated learning with dynamically balanced collaboration. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 997–1002. http://dx.doi.org/10.1109/DAC18074.2021.9586241.
Yang, Q., Liu, Y., Chen, T., Tong, Y., 2019. Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. 10 (2).
Yokoyama, H., 2019. Machine learning system architectural pattern for improving operational stability. In: ICSA-C 2019. pp. 267–274.
Zdun, U., 2007. Systematic pattern selection using pattern language grammars and design space analysis. Softw. - Pract. Exp. 37 (9), 983–1016.
Zhang, W., Lu, Q., Yu, Q., Li, Z., Liu, Y., Lo, S.K., Chen, S., Xu, X., Zhu, L., 2021. Blockchain-based federated learning for device failure detection in industrial IoT. IEEE Internet Things J. 8 (7), 5926–5937. http://dx.doi.org/10.1109/JIOT.2020.3032544.