Article

Software Engineering Aspects of Federated Learning Libraries: A Comparative Survey

Hiba Alsghaier[1] and Tian Zhao[2]*

[1] University of Wisconsin – Milwaukee; alsghai2@uwm.edu
[2] University of Wisconsin – Milwaukee; tzhao@uwm.edu
* Correspondence: tzhao@uwm.edu

Abstract: Federated Learning (FL) has emerged as a pivotal paradigm for privacy-preserving machine learning. While numerous FL libraries have been developed to operationalize this paradigm, their rapid proliferation has created a significant challenge for practitioners and researchers: selecting the right tool requires a deep understanding of their often undocumented *software architectures* and *extensibility*, aspects that are largely overlooked by existing algorithm-focused surveys. This paper addresses this gap by conducting the first comprehensive survey of FL libraries from a *software engineering perspective*. We systematically analyze ten popular open-source FL libraries, dissecting their architectural designs, support for core and advanced FL features, and most importantly, their extension mechanisms for customization. Our analysis produces a novel taxonomy of FL concepts grounded in software implementation, a practical decision framework for library selection, and an in-depth discussion of architectural limitations and pathways for future development. The findings provide developers with actionable guidance for selecting and extending FL tools and offer researchers a clear roadmap for advancing FL infrastructure.

Keywords: Machine Learning, Federated Learning, Computer Security, Data Privacy, Software Library, Workflow Management

1. Introduction

Federated learning (FL) provided a new avenue for machine learning (ML) as a type of distributed learning technique to train ML models without compromising data privacy. FL was designed to promote greater access to private data without transferring raw data to central locations. This aligns well with stringent privacy regulations, such as the EU's General Data Protection Regulation and California's Consumer Privacy Act, which require effective privacy and security measures [1]. The decentralized nature of FL brings with it considerable privacy challenges, particularly related to inference attacks. These attacks can occur in two primary forms: inference over outputs, where sensitive information may be leaked through intermediate results, and inference during the learning process, where participants may infer private data from others during model exchanges [2]. These privacy vulnerabilities must be addressed to ensure the secure operation of FL applications, which necessitates the use of robust privacy-preserving techniques.

FL libraries provide system-level support for implementing FL applications taking into account data partitioning, model training, model aggregation, privacy protection, and secure communication. Although dozens of FL libraries have been developed, there are not many FL applications in the real world. It is imperative to understand the obstacles that prevent the adoption of FL in order to realize its potential. A FL system includes data owners, FL clients, and FL servers, where the data owners perform training locally with FL clients, while the FL servers aggregate the local updates, such as models or gradients, before sending the global updates to the clients.

The basic communication steps between a FL server and its clients are shown in Figure 1. Within this system, data owners care about data privacy and also need incentives to participate. The FL system provides privacy protection, but the concern for privacy must be balanced with the utility of the trained

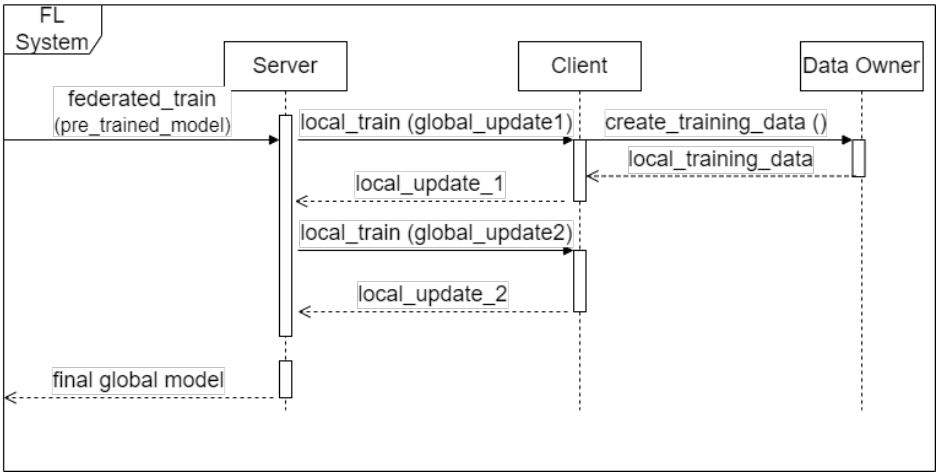

Figure 1. The sequence diagram of a FL system that shows the exchange between FL server and clients.

models and the cost of training. Stronger privacy protection reduces the utility of the trained model (e.g. lower accuracy) and increases the cost of training due to overhead of computation and communication. To manage these trade-offs, secure FL frameworks (e.g. UAV-assisted crowdsensing [3]) can jointly consider privacy, model utility, and incentive mechanisms. This kind of systems can employ local differential privacy and blockchain-based reward distribution to motivate high-quality participation while mitigating privacy risks. In many cases, incentive design and privacy enforcement must be co-optimized to maintain model performance and participation efficiency.

To balance the needs of privacy, model utility, and learning objectives, FL applications must use suitable ML models, privacy protection schemes, and security mechanisms. To reduce development costs, dozens of FL libraries have been developed. The main components of these libraries are illustrated in Figure 2. Most FL libraries include components for managing server and client communication, local training within clients, aggregation within servers, and some degree of privacy protection. Libraries that support real-world development may have modules for security provisioning to authenticate and authorize users, as well as auditing and anomaly detection for quality assurance.

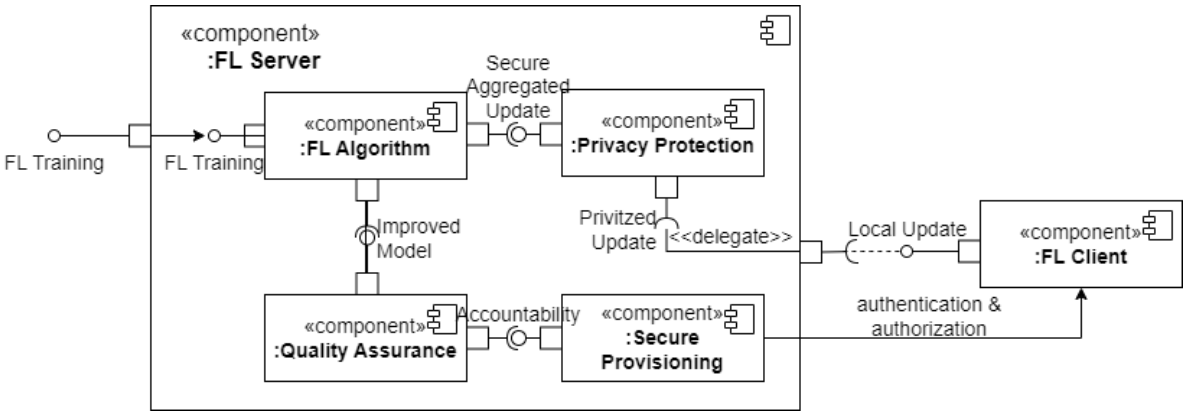

Figure 2. The system diagram of a FL system that shows its major components and their relationships.

Despite rapid advancements in the FL field, the practical adoption of FL technologies remains constrained by several limitations. These include privacy and security concerns that discourage data owners from participating, as well as software engineering challenges, particularly regarding the extensibility and configuration of FL libraries. While many surveys have been written to summarize the rapid accumulation of knowledge in this domain [4–7], very few reviews have focused on FL libraries to understand how core and advanced features are supported in software implementations and how these libraries can be extended to suit the requirements of specific FL applications. This survey addresses that gap by conducting a systematic analysis of how the software components of

major FL libraries support FL features through both native implementation and extension. We selected ten FL libraries based on their number of GitHub stars, which serves as a proxy for their popularity within the research and development community. Specifically, we examine how core aspects of FL, such as data partitioning, model training, aggregation, privacy protection, and communication, are supported in each library's workflow, and whether advanced features such as auditing, privacy verification, anomaly detection, and personalization are supported directly or through extension and customization.

In summary, this survey makes the following contributions, which distinguish it from prior algorithm- and theory-focused reviews:

A Software-Grounded Taxonomy We summarize a taxonomy of FL concepts (Section 3) explicitly framed around their implementation in software libraries, contrasting commonly supported core aspects with advanced features that require customization.

An Architectural Analysis of FL Libraries We provide a systematic overview of the top-ten FL libraries (Section 4) that goes beyond feature checklists. We analyze their *software architecture*, such as extension points, class hierarchies, and design patterns (e.g. strategy, pipeline), providing a blueprint for developers needing to understand, customize, or extend these frameworks.

A Practical Selection and Application Framework We synthesize our analysis into a multi-faceted guide for practitioners, featuring a comprehensive comparison table (Table 5), a decision tree (Figure 10), and concrete application scenarios (Section 5). This framework directly links library *capabilities* to application *requirements*.

A Critical Analysis of Limitations and Future Pathways We identify key *software engineering limitations* in current FL libraries (Section 6), such as fragmented support for auditing and verification. We then propose concrete, architecture-level modifications to address these gaps, providing a research agenda focused on building more trustworthy and extensible FL systems.

The roadmap to the main contribution of the paper is highlighted in Figure 3.

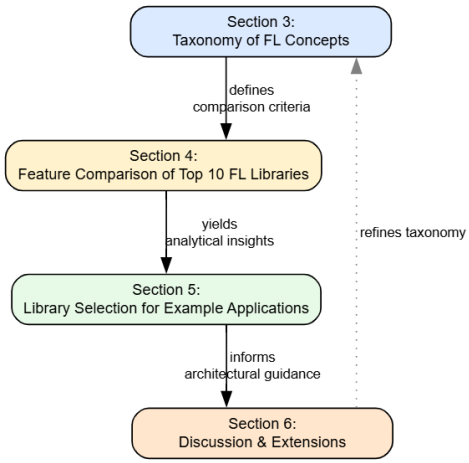

Figure 3. A roadmap for the main contribution of this survey.

2. Related Surveys

FL is a heavily surveyed topic. Table 1 presents a partial list of recent surveys, which can be broadly categorized according to their primary focus, including comprehensive surveys [4,8,11,19,24,31], applications [9,14,16,32], security & privacy [10,13,21], evaluation metrics [12,25], incentives [20], efficacy & effectiveness [27–29], and FL systems[23,26,30,33]. Comprehensive surveys [4,8] summarize the overall progress in the field, while other surveys concentrate on specific aspects of FL research such as privacy & security [10,34] and applications in IoT [16]. Although comprehensive surveys often address certain software-related aspects, the works most closely related to this study are [23,26,30]. Li

Table 1. A summary of some recent surveys on federated learning.

Year	Title	Topic
2020	Federated learning: a survey on enabling technologies, protocols, and applications [8]	comprehensive survey
2020	Federated learning in mobile edge networks: a comprehensive survey [9]	applications
2020	Threats to federated learning: a survey [10]	security and privacy
2021	Advances and open problems in federated learning [11]	comprehensive survey
2021	A survey for federated learning evaluations: goals and measures [12]	evaluation metric
2021	A comprehensive survey of privacy-preserving federated learning: a taxonomy, review, and future directions [13]	security and privacy
2021	A survey on federated learning: the journey from centralized to distributed on-site learning and beyond [4]	comprehensive survey
2021	A survey of federated learning for edge computing: Research problems and solutions [14]	applications
2021	Challenges, applications and design aspects of federated learning: a survey [15]	comprehensive survey
2021	Federated learning for Internet of Things: a comprehensive survey [16]	applications
2021	Federated learning on non-IID Data: a survey [17]	efficiency and effectiveness
2022	Federated learning review: Fundamentals, enabling technologies, and future applications [18]	applications
2022	From distributed machine learning to federated learning: a survey [19]	comprehensive survey
2022	A survey of incentive mechanism design for federated learning [20]	incentives
2022	Federated learning for smart healthcare: a survey [21]	applications
2022	Survey on Federated Learning Threats: concepts, taxonomy on attacks and defences, experimental study and challenges [21]	security and privacy
2022	A state-of-the-art survey on solving non-IID data in Federated Learning [22]	efficiency and effectiveness
2023	A survey on federated learning systems: vision, hype and reality for data privacy and protection [23]	FL systems
2023	A survey on federated learning: challenges and applications [24]	comprehensive survey
2024	FederatedTrust: a solution for trustworthy federated learning [25]	evaluation metric
2024	A tutorial on federated learning from theory to practice: foundations, software frameworks, exemplary use cases, and selected trends [26]	FL systems
2024	Knowledge distillation in federated edge learning: a survey [27]	efficiency and effectiveness
2024	A survey of what to share in federated learning: perspectives on model utility, privacy leakage, and communication efficiency [28]	efficiency and effectiveness
2024	Emerging trends in federated learning: from model fusion to federated X learning [29]	efficiency and effectiveness
2024	Comparative analysis of open-source federated learning frameworks-a literature-based survey and review [30]	FL systems
2025	Vertical Federated Learning for Effectiveness, Security, Applicability: A Survey [31]	comprehensive survey
2025	Federated learning for cyber physical systems: a comprehensive survey [32]	applications
2025	A Review on Federated Learning Architectures for Privacy-Preserving AI: Lightweight and Secure Cloud–Edge–End Collaboration [33]	FL systems

et al. [23] proposed a taxonomy that classifies recent advances in FL according to data partitioning, learning models, privacy mechanisms, communication methods, federation scale, and motivation. They also summarized several open-source FL libraries, including FATE, TFF, PySyft, PaddleFL, and FedML, and evaluated them with respect to effectiveness, efficiency, privacy, and autonomy. Luzon *et al.* [26] provided a detailed account of FL libraries by comparing their support for data

partitioning, aggregation methods, data distribution, adversarial robustness, differential privacy, explainability, personalization, and documentation. They further demonstrated six representative use cases combining different types of data partitioning, data distributions, and ML models. Riedel *et al.* [30] conducted a comparative analysis of fifteen open-source FL libraries using weighted evaluation criteria encompassing functionality, interoperability, and user-friendliness, although the selected weights reflect the authors' own preference. More recent surveys continue to refine the landscape of FL research. Ye *et al.* [31] presented a focused and systematic survey of vertical federated learning (VFL), classifying approaches along dimensions of effectiveness, security, and applicability. Quan *et al.* [32] provided a comprehensive review of FL in cyber-physical systems, emphasizing architectural considerations, communication constraints, and domain-specific challenges. Zhan *et al.* [33] examined lightweight and privacy-preserving FL architectures for cloud-edge-end collaboration, focusing on system-level design principles and deployment trade-offs.

While the surveys above provide valuable insights, they primarily analyze FL libraries through the lens of provided features, algorithmic support, or performance benchmarks. In contrast, our work adopts a distinct *software engineering* lens. We focus not only on *what* features a library offers but on *how* these features are implemented and integrated into a software architecture. Our analysis delves into extension points, inheritance hierarchies, and design patterns to reveal the underlying structure that dictates a library's flexibility, maintainability, and suitability for advanced requirements like formal verification and robust auditing. This architectural perspective is crucial for developers who need to customize these frameworks and has been largely absent from prior comparative studies.

In Table 2, we compare this work with selected surveys that share similar objectives, particularly those addressing FL libraries and their software characteristics.

Table 2. Comparison between selected surveys and this work based on the covered aspects.

Aspect	[8]	[16]	[4]	[23]	[34]	[26]	[30]	Ours
Data partitioning	✓	✓		✓			✓	✓
Aggregation & fusion		✓	✓			✓	✓	✓
Privacy protection	✓	✓	✓	✓	✓	✓	✓	✓
Communication	✓	✓	✓			✓		✓
Customization								✓
Auditing								✓
Anomaly detection	✓					✓		✓
Personalization	✓					✓		✓
Formal verification	✓				✓			✓
Extension points					✓			✓
Intrusion detection						✓		✓

3. Taxonomy

In this section, we briefly summarize the taxonomy of concepts that are important to FL libraries. The main concepts include data partitioning, model training, aggregation & fusion, privacy protection, communication.[1] Advanced concepts include auditing, anomaly detection, verification of privacy, and personalization. The relationships of the concepts are illustrated in Figure 4.

3.1. Data partitioning

Data partitioning refers to how training data is distributed among data owners. which includes horizontal/vertical partitioning and transfer learning [35]. In horizontal FL (HFL), training datasets have the same set of features, but different samples are split among data owners. In vertical FL (VFL), training datasets have the same set of sample space, but have different features. For example, demographic information and survey responses of the same individuals are owned by different entities.

[1] We omit model training since it is well covered in the literature.

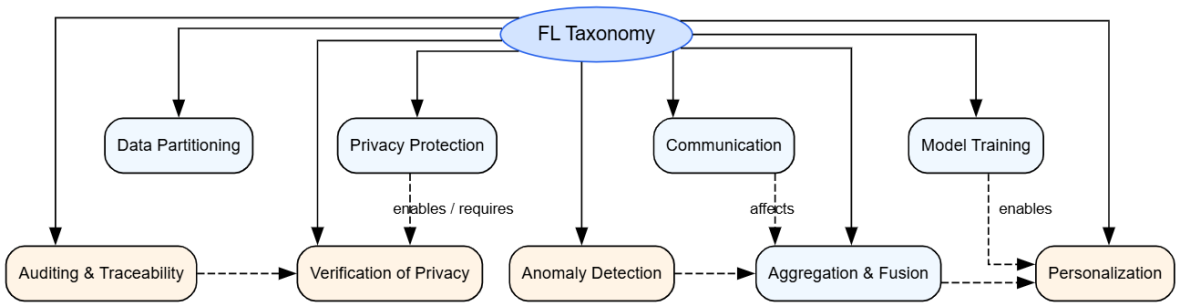

Figure 4. The relationship of the concepts in FL taxonomy, where blue color indicates main concepts and brown color indicates advanced concepts.

Transfer FL leverages a pre-trained model to improve the performance of a new model on a different task. Transfer FL can be applied to HFL or VFL by training on a larger data set and fine-tuning on smaller data sets to improve the accuracy of FL with limited data [36]. Training methods for VFL differ significantly from those of HFL and libraries often define separate modules for them. Most FL libraries place greater emphasis on HFL since VFL is more difficult to implement and has limited use-case scenarios.

3.2. Aggregation & fusion

Various methods such as FedAvg [37], FedProx [38], and FedNova [39] can be used to aggregate client output [40]. In secure aggregation, each client holds a private vector, and the server ensures that only the combined result is revealed, protecting individual contributions. For example, clients can add to their data with random perturbations (masking), which are designed to cancel out in the sum, guaranteeing correctness. Aggregation needs to consider high-dimensional data, dynamic user participation, and device limitations. It can also include key agreements and secret sharing to ensure robustness, allowing aggregation to proceed even when some users drop out or fail to complete the protocol. Simple approaches, such as arithmetic mean aggregation [41], are vulnerable to corrupted updates, since even a single adversarial or faulty update can significantly degrade the global model's performance. More robust methods, such as geometric median aggregation, offer resilience against such disruptions.

3.3. Privacy protection

Threats to data privacy include inference over the outputs or during the learning process. The former refers to the leakage of a client's data from intermediate output while the later refers to that a client infers information about another one's data when data is exchanged during training. For inference over the outputs, insider attacks are launched by users in the FL system, including both third parties and data holders, while outsider attacks are launched by eavesdroppers during the communications with the end users when the service is deployed [42]. Privacy preservation methods address the risk of inference attacks during the learning process by using SMPC, which allows parties to obtain the output of a function over input while protecting other data [42,43]. Many privacy-preserving methods assume honest participants and/or trusted aggregators. This assumption is removed in HybridAlpha [44], which uses functional encryption and a trusted third party, where each participant encrypts its result and the aggregator decrypts the aggregated results. The aggregator can only obtain the function output, such as the average weights of the local models. Information leakage can be prevented by applying Gaussian noise to the output of clients [45]. Secure aggregation can be combined with randomized response techniques Pseudo-random generators and Diffie-Hellman key exchange to provide rigorous differential privacy guarantees for both clients and the server [46].

3.4. Communication

FL typically has three communication actions: distribute the global model to devices, local training, and upload updated models for aggregation [47]. Key challenges include limited bandwidth, device dropouts, and non-uniform data distribution across devices. To address these, researchers have explored methods such as reducing update frequency, selective participation of devices, compressing data, and decentralized or peer-to-peer training approaches [47]. Iterative data exchange between clients and servers created high communication overhead, especially with large-scale implementations. This overhead impacts both the training speed and the convergence rate, as FL involves numerous devices exchanging substantial amounts of data. Overcoming this requires designing communication-efficient frameworks that optimize device selection, parameter transmission, and resource allocation, ultimately reducing delays and improving the training process in a network-constrained environment [48].

3.5. Anomaly detection

FL systems are susceptible to attacks to the central server, the communication process, or the clients [49]. Client vulnerabilities include data poisoning, model poisoning, backdoor attacks, and inference attacks. These threats exploit weaknesses in client devices or the data they contribute to manipulate or compromise the global FL model. The central server faces risks such as malicious servers, non-robust aggregation methods, and inference attacks, which can undermine the integrity and security of the aggregated model updates. Typical solution is to implement targeted defense measures to detect and mitigate anomalies [49]. Taxonomies of adversarial attacks and defenses were introduced [49–51] to categorize various attacks and countermeasures, which include guidelines for selecting appropriate defense strategies against specific adversarial threats. The signature-based intrusion detection systems are insufficient for addressing new threats such as zero-day attacks and advanced persistent threats [52,53]. Anomaly-based detection is a viable alternative, which leverages ML-based techniques including unsupervised learning for anomaly detection and supervised classification using labeled data to detect abnormal behaviors.

3.6. Auditing

In a FL system, regular audits [54] can ensure that privacy and security measures are followed throughout its life cycle. The audits can verify that the client updates are legitimate and do not introduce vulnerabilities or errors. Without audits, the model's operations can become opaque, leading to untraceable changes and potentially compromising the accuracy and reliability of the model [54,55]. To maintain transparency, it is essential to establish a mechanism that trace the changes in the global model to the updates of specific clients that influenced the performance of the global model [56–58]. Trace logs can provide such mechanism by maintaining detailed logs of all operations and parameter updates [59,60]. Advanced mechanisms include FedTracker [61], which uses a tamper-proof embedded watermark and extracted fingerprints to establish traceability, and PPTFL [62], which integrates traceability methods to detect outliers within FL systems. Also, blockchain-based FL platform [60] can record encrypted gradient updates on a behavior chain and evaluate their quality on an audit chain to identify malicious contributors without decrypting individual gradients.

3.7. Personalization

Most FL systems train global models for all participants, but a global model is not optimal when there is significant data heterogeneity. Personalization refers to methods that customize the training process for each client to obtain local models suitable for the clients while incurring higher communication costs. There are three algorithms [63] for personalization that also minimize communication costs.

1. User clustering: grouping clients and training a model for each group.
2. Data interpolation: combining local and global data to train a model.

3. Model interpolation: combining local and global models.

Data interpolation has higher communication costs, while model interpolation and user clustering offer privacy benefits with manageable communication overhead [63]. The adaptive personalized federated learning (APFL) algorithm [64] creates personalized models for each user by blending local and global models. Its generalization ability depends on the mixing parameter, the distribution divergence, and the training data size. A communication-efficient optimization algorithm progressively adapts the model, improving generalization compared to global models like FedAvg and SCAFFOLD. Tan *et al.* [65] provides a detailed classification of personalization methods that focuses on global model personalization, where data-based approaches (such as data augmentation) balance heterogeneity through synthetic data while client selection optimizes client participation for efficient representation. Model-based methods include regularization to align local and global models, meta-learning to enable rapid adaptation for diverse tasks, and transfer learning, which leverages the pre-trained layers to facilitate local customization.

3.8. Verification of privacy

Correctness verification in FL ensures that model training and updates are performed as intended, without deviation from the desired design and objectives. FedGRU [66] uses a FL-based neural network to verify the correctness of updates through encrypted parameter exchanges. VerifyNet [67] is a model designed to ensure the accuracy of server-provided results by returning the aggregated FL model along with a proof. Clients can verify this proof using a homomorphic hash function combined with a pseudorandom function to evaluate whether the aggregation is trustworthy. A trusted third party is responsible for the initial key generation before FL process starts. This entity is considered reliable and non-colluding with any FL participants. If the central server cannot be trusted, then zero-knowledge proof can be used to demonstrate to the clients that the aggregation is executed correctly [68]. Zero-knowledge proofs [7] allows a party to prove the validity of a statement without revealing additional information. This technique is useful for secure model updates and verifying data integrity without compromising sensitive details. Moreover, zero-knowledge clustering (ZeKoC) [69] was introduced to address the unsupervised weight clustering problem within FL. ZeKoC allows servers to manage clusters for weight selection and aggregation without direct access to the underlying training data. Secure aggregation can be combined with zero knowledge proof to prevent poisoning attacks [70], where zero-knowledge proof is used shift the burden of detecting attacks in the local models from server to clients.

4. Federated Learning Libraries

In this section, we compare top-ten FL libraries (as determined by the number of GitHub stars) by focusing on five main aspects including data partitioning, model training, aggregation & fusion, privacy protection, and communication[2]. We also examine their support for advanced features including auditing, anomaly detection, personalization, and verification of privacy. Lastly, we briefly discuss the extension points of each library. Table 3 compares the available FL libraries based on the developer types, maintenance status, and popularity on GitHub. All libraries are open-source but some are not actively maintained.

The objective of this section is to help readers determine the suitability of a FL libraries for their projects. For example, for FL project that requires vertical data partitioning, libraries like FATE, NVFlare, and PaddleFL might be more suitable choice since they provide built-in implementation of vertical FL (VFL) algorithms though NVFlare uses specialized VFL workflow while PaddleFL is not actively maintained and is not compatible to common libraries like Tensorflow or Pytorch. The popularity in Github also indicates relative user preference for FATE. Table 4 shows the hierarchical classification of the FL libraries analyzed in this section.

[2] The discussion of PySyft is less comprehensive since it is no longer a conventional FL library.

Table 3. Developer, status, and popularity of FL libraries

Library	Developer	State of Maintenance	Number of Github Stars
PySyft [71]	Academia	Active	9.7k
Flower [72]	Academia	Active	6.1k
FedML [73]	Academia	Active	3.9k
FATE [74]	Industry	Active	5.9k
TFF [75]	Industry	Active	2.4k
OpenFL [76]	Industry/Academia	Active	797
FedLab [77]	Academia	Active	780
NVFlare [78]	Industry	Active	768
IBM FL [79]	Industry	Active	510
PaddleFL [80]	Industry	Not active	507
Backdoors 101	Academia	Not active	342
FedJax [81]	Academia	Active	254
Xaynet	Academia	Not active	200
APPFL [82]	Academia	Active	89
Scaffold [83]	Academia	Not active	68
Substra [84]	Industry	Active	58
TorchFL [85]	Academia	Not active	41

4.1. FATE

Data partitioning

WeBank's FATE library[3] has native support for both horizontal FL (same features but different users) and vertical FL (same users but different features), as well as transfer learning scenarios. Its workflows use components such as `Reader`, `DataTransform`, and `Intersect`. In vertical FL, `PSI` (private set intersection) aligns sample IDs across parties. Users can add custom feature engineering via modular pipeline components (e.g. `FederatedFeatureBinning`, `FeatureScale`).

Model training

FATE has built-in algorithms that cover logistic regression, decision trees (`SecureBoost`), neural networks (`HomoNN`, `HeteroNN`), and LLM tuning via FATE-LLM. Its configurations are defined in the pipeline DSL or JSON, specifying roles (guest, host, arbiter) and algorithm parameters (e.g. `HomoNN`, `HeteroSecureBoost`). Users can implement custom trainers or loss functions thanks to decoupled component design.

Aggregation and fusion

FATE uses Secure Multi-Party Computation (SMPC) and Homomorphic Encryption (HE) for secure aggregation in vertical FL (e.g. `HeteroSecureBoost`). Aggregation is driven by the pipeline components, combining partial models or gradients securely via algorithm components (e.g. `HomoSecureBoost` and `HeteroLR`) that encapsulate aggregation/fusion logic transparently.

Privacy protection

FATE provides strong native privacy mechanisms with secure protocols such as HE/SMPC (e.g. Secret Sharing HE (SSHE) and Heterogeneous SecureBoost) and differential privacy that is integrated in certain algorithms (e.g. SecureBoost with DP).

Communication

The FATE-Flow orchestration framework coordinates multi-party workflows, with pluggable communication engines and native support for HTTP, gRPC, RabbitMQ/Pulsar, and OSX intercon-

[3] The latest version of FATE (https://github.com/FederatedAI/FATE) is 2.2.0 released in 2024.

Table 4. High-Level Categorization of Federated Learning Libraries

Primary Focus	Academia-Focused (Research & Simulation)	Industry-Focused (Production & Systems)
General Purpose & Flexible	*Flower*: Flexible framework with a strategy-based API for diverse FL algorithms. *FedML*: Scalable library supporting cross-silo and cross-device scenarios. *FedLab*: Lightweight, fine-grained control for prototyping and simulation.	*FATE*: The most comprehensive platform for VFL and secure multi-party computation. *IBM FL*: Enterprise-focused framework with a configuration-driven approach. *NVFlare*: Enterprise-grade with built-in auditing, specialized workflows, and robust deployment.
Specialized & Integrated	*PySyft*: Evolved into a privacy-preserving data governance framework, not a conventional FL library.	*TFF*: Tightly integrated with TensorFlow for simulating FL algorithms. *OpenFL*: Features support for Trusted Execution Environments (TEEs) like Intel SGX. *PaddleFL*: Integrated with the PaddlePaddle ecosystem, supports VFL via cryptographic protocols.

nection. The communication modes, encryption, and routing are set in cluster/pipeline config files, allowing deployment via Docker Compose, Kubernetes (KubeFATE), or standalone.

Extension and customization

FATE has partial support for auditing, anomaly detection, personalization, and formal privacy verification through extension. It provides logging and job tracking through FATE-Flow but it does not provide tamper-evident audit logs, cryptographic signatures, or compliance-level traceability. However, developers can log hash values, timestamps, or role actions within pipeline components and use external audit systems for secure logs. FATE does not include built-in anomaly detection (e.g. detecting poisoned models, drift, or faulty clients). Custom components can be added to the pipeline to monitor gradient updates for suspicious patterns or add hooks for outlier detection in training metrics. Personalization is supported for horizontal FL via fine-tuning. Horizontal FL personalization is achievable through `HomoNN` and `HomoLR` fine-tuning after global training and local adaptation steps using transfer learning or personalized layers. Vertical FL personalization is more complex due to feature-split modeling and limited direct access to global updates. FATE has no built-in federated meta-learning or personalization layers, but the pipeline system allows adding them. FATE also has no built-in framework such as zero-knowledge proofs and symbolic verification to validate privacy levels.

The extension points of FATE are Python decorators such as `@cpn.component(roles=[...])`. Each FL learning function such as `homo_nn` and `sshe_lr` is converted by this decorator into a component in the FATE-Flow pipeline. The supported roles of the component such as guest, host, and arbiter must be specified. While this design appears to be modular for extension, the actual implementation of the learning algorithms is tightly integrated, which may present challenges for adding new algorithms.

4.2. FedLab

Data partitioning

FedLab[4] is a simulation environment that can partition datasets horizontally across clients in both IID and non-IID fashions. Developers can use the `FederatedDataset` and `DataPartitioner` classes to define data splitting strategy and simulate realistic data distribution (e.g. label-skew and/or quantity-skew).

[4] The latest version of FedLab (https://github.com/SMILELab-FL/FedLab) is 1.3.0 released in 2022.

Model training

Local training logic is encapsulated in the subclasses of `ClientTrainer` based on PyTorch models. These trainers define data loading, forward/backward passes, and local updates. They can be paired with `ServerHandler` on the server side to coordinate training rounds.

Aggregation and fusion

The server-side aggregation logic is in the `ServerHandler` class, where developers define how to collect and combine model updates. Built-in options include synchronous `FedAvg`, asynchronous aggregation, and communication compression. Developers can customize fusion strategies, client sampling, or robust aggregation techniques.

Privacy protection

FedLab has no built-in cryptographic privacy mechanisms. However, its simulation environment is designed for experimenting with privacy protocols. Developers can insert modules for secure aggregation, add gradient clipping, differential privacy noise, or encryption/decryption steps in their `ClientTrainer` and `ServerHandler` code.

Communication

FedLab supports communication with TCP, RPC, and RPC with compression, and simulation of network constraints. Developers can configure channel settings when building `CommHandler`, enabling simulation of bandwidth limitations, asynchronous updates, and scalable client-server communication.

Extension and customization

FedLab does not support auditing, security verification, or privacy assurance though features such as anomaly detection, personalization, and auditing can be added through extension to the classes `ClientTrainer` and `ServerHandler`. For example, `FedAvgServerHandler` extends the synchronous server handler; `FedAvgClientTrainer` extends the SGD client trainer class. Many client classes have a serial version that trains multiple clients in a single process.

4.3. FedML

Data partitioning

FedML[5] supports horizontal FL, where data is partitioned via configuration parameters in scripts or YAML files. The data may be partitioned as IID, non-IID with label imbalance and controlled heterogeneity, or customized splits with Python programs.

Model training

FedML uses a worker-oriented API based on `ClientTrainer` and `ServerAggregator`. Each client uses a subclass of `ClientTrainer`, which handles model training, evaluation, loss tracking, and communication of updates. The server uses a subclass of `ServerAggregator` to handle aggregation logic. These are passed into `FedMLRunner`, which orchestrates the FL workflow, coordinating clients and the server

Aggregation and fusion

FedML supports a variety of aggregation strategies such as FedAvg, FedOpt, and FedProx. Aggregation logic lives in the `ServerAggregator` class, where the global model is updated based on client updates. A developer can override the aggregation method or plug in new algorithms via the modular server-side logic.

[5] The latest version of FedML (https://github.com/FedML-AI/FedML) is 0.8.9 released in 2023.

Privacy protection

Privacy mechanisms are not provided by default, but FedML allows customization such as adding the extension FedML-HE [86] for homomorphic encryption. Developers can also insert noise, clip gradient, or add secure aggregation in `ClientTrainer` or `ServerAggregator` classes.

Communication

FedML has communication methods such as single-process simulation, MPI, and NVIDIA collective communication library for high-performance environments. Deployment modes include cross-device, cross-silo settings using MQTT or gRPC. Communication options can be configured via CLI or YAML, which are processed by the `FedMLRunner` class.

Extension and customization

FedML has no builtin support for auditing and privacy verification. Developers can manually implement anomaly detection in the `ServerAggregator` by checking for outlier model updates (e.g. using distance metrics or cosine similarity). FedML includes some experimental support for FedProx, FedPer [87], or fine-tuning-style methods that are implemented through customizing the train or test logic in `ClientTrainer` to retain or adapt local components. Developers can explore personalization by changing model heads, adjusting local loss functions, or applying meta-learning techniques per client.

The extension points of FedML include `ClientTrainer` and `ServerAggregator`, the latter of which supports DF and HE. `FAServerAggregator` is a separate base class for federated averaging algorithms. FedML supports clients and servers in cross-silo, cross-device, and cross-cloud settings with their own aggregator classes.

4.4. Flower

Data partitioning

Flower[6] mainly support horizontal data partitioning methods. Data distribution is handled in simulation scripts that pass dataset splits into each `flwr.client.Client` (or `NumPyClient`) class.

Model training

Training logic is implemented in subclasses of `Client` by defining `get_parameters`, `fit`, and `evaluate` methods for local training and evaluation. The server orchestrates rounds via `flwr.ServerApp` or custom Strategy (e.g. FedAvg), which controls client sampling, rounds, and global parameter updates.

Aggregation and fusion

Aggregation logic is in server Strategy classes with common built-ins such as FedAvg, FedProx, and advanced strategies like FedStar. The strategy defines how the server collects, averages, and updates global model parameters. Developers can override strategy behavior or implement custom aggregation.

Privacy protection

Flower has privacy-preservation modules that support DP strategies with fixed or adaptive clipping, local DP, or server DP, and built-in secure aggregation via SecAgg protocols. Developers can compose DP and secure aggregation using a Flower Mods, which are callables that wrap around `ClientApps` to manipulate or inspect the incoming and the resulting outgoing messages.

[6] The latest version of Flower (https://github.com/adap/flower) is 1.21 released in 2025

Communication

Flower supports communication and deployment in either simulation mode or in real clients (`ClientApp` and `ServerApp`) through gRPC/HTTPS. It provides backend abstraction for cloud, edge, or fleet deployment.

Extension and customization

Flower has no built-in support for auditing or formal privacy verification though auditing is possible through custom logging in the `Strategy`, `ClientApp`, or `Client` implementations. Developers can enable anomaly detection by checking client updates in the `aggregate_fit` or `evaluate` methods by rejecting updates based on distance from global model, training loss spikes, or update norms. Flower Mods can also intercept client messages for analysis before aggregation. Personalization is supported through fine-tuning locally after global rounds, FedPer [87], FedAvgM [88], or meta-learning adaptations by customizing the `fit` or `evaluate` methods of `Client`.

The class `flwr.server.strategy.Strategy` is the main extension point of Flower library. It initializes the global model parameters, configures the next round of training, aggregates the current round of results, configures the next round of evaluation, aggregates the current round of evaluate results, and evaluates the current model parameters. Figure 5 illustrates how FL algorithms extend from the base class `Strategy`.

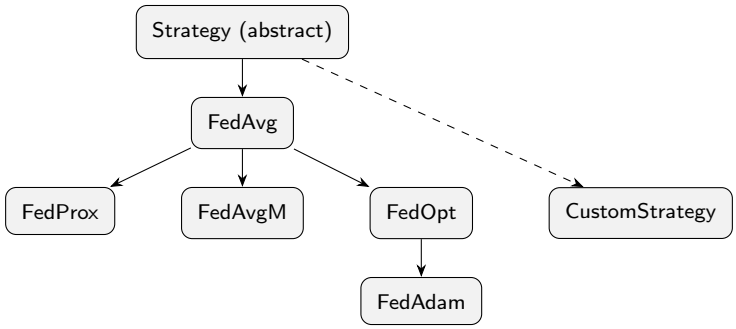

Figure 5. This diagram illustrates the inheritance hierarchy of the Strategy class in the Flower library.

4.5. IBM FL

Data partitioning

IBM's *Federated Learning Lib* (IBM FL)[7] is designed for horizontal FL, where each party holds data with the same features but different samples. Vertical FL is not supported out of the box and would require customizing data alignment and split modeling via new handlers. Parties define their own `DataHandler` in the party configuration YAML, which loads and pre-processes local data.

Model training

The FL training model (e.g. `KerasFLModel`, `PytorchFLModel`, and `SklearnFLModel`) is specified in the configuration file party.yml. The model architecture and logic (training, evaluation, saving) are defined in code, while the workflow picks them up via the configuration.

Aggregation and fusion

Fusion algorithms are defined at the aggregator side in aggregator.yml, selected by name (e.g. simple_avg, weighted_avg, and FedProx). A `FusionHandler` class implements the aggregation logic. The aggregator orchestrates training rounds, delegates fusion to the handler, and coordinates parties. This structure cleanly separates control (aggregator) from computation (fusion handler) events.

[7] The latest version of IBM FL (https://github.com/IBM/federated-learning-lib) is 2.0.3 released in 2023.

Privacy protection

IBM FL offers built-in fusion options that include privacy mechanisms, such as federated algorithms with differential privacy (DP) (e.g. Naive Bayes with DP) and secure aggregation methods. Privacy logic is embedded in the `FusionHandler` class, while secure communication (e.g. TLS) is configured in the protocol_handler. This decoupled design lets users plug in or extend privacy layers without major code changes.

Communication

Communication between aggregator and parties is managed via protocol_handler and connection sections in both party and aggregator configs, typically using Flask-based HTTP servers. Other protocols (e.g. gRPC) are also supported, enabling flexible deployment (on-prem, cloud, Docker/K8s). The modular protocol layer enables customization or replacement if needed.

Extension and customization

IBM FL does not have native support for auditing, anomaly detection, personalization, or formal privacy verification, but these can be implemented by extending handlers (e.g. logging) or integrating with external systems. IBM FL supports customization through the extension of the `FusionHandler` class and the `FLModel` class as illustrated in Figure 6.

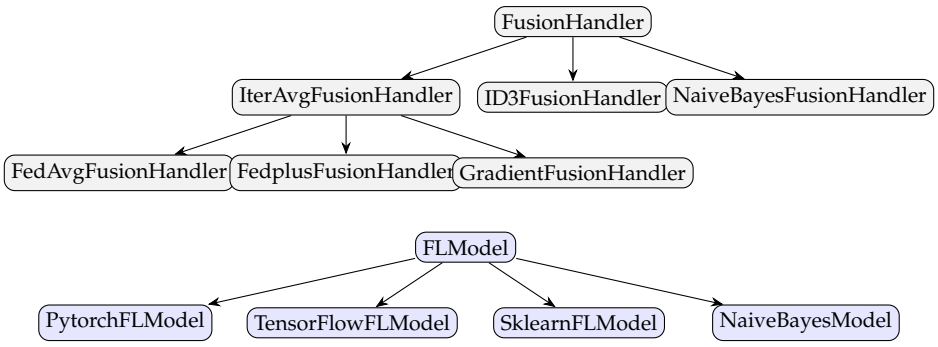

Figure 6. This diagram illustrates the inheritance hierarchies of the fusion handler and FL model classes in the IBM FL library.

4.6. NVFlare

Data partitioning

Nvidia FLARE (NVFlare)[8] supports both horizontal FL, vertical FL, and split learning. Users can define the dataset distributions in job configurations (YAML) via data loader code. Built-in workflows support horizontal FL and specialized workflows for vertical FL and split learning are also available.

Model training

Training logic is encapsulated in `Executor` (client-side) and `Controller` (server-side) components. Executors handle local training using frameworks such as PyTorch, TensorFlow, and NumPy. Users can configure workflows such as scatter & gather, cyclic weight transfer, or client-controlled validation in JSON or Python via the job API.

Aggregation and fusion

NvFlare includes implementations for algorithms such as FedAvg, FedProx, FedOpt, and SCAFFOLD, These are configured via the chosen workflow and controlled by the `Controller` component, which aggregates `Shareables` (model updates) from clients.

[8] The latest version of NVFlare (https://github.com/NVIDIA/NVFlare) is 2.6.2 released in 2025.

Privacy protection

NvFlare has filter-based privacy modules in its job and controller configuration. User can configure DP, HE, Private Set Intersection (PSI), and message quantization via filters running on the server and client sides to enforce privacy before data exchange.

Communication

NvFlare supports multiple deployment modes, which include simulation and deployment CLI and dashboard. Communication is handled securely using gRPC over TLS, optional peer-to-peer connections for split learning, and message streaming with quantization for efficient transfers.

Extension and customization

NvFlare has a built-in auditing framework, where all jobs, events, component executions, and message exchanges are logged via the `AuditService` class. Logs can be centrally collected and analyzed, and include traceable records of who participated, what was trained, when, and with which configuration. Developers can also plug in custom auditors through `AuditService` to send audit logs to a third-party system.

NvFlare supports personalization via the `Executor` class, where each client can implement local fine-tuning or split learning, which allows per-client model heads (different final layers for each client). It also supports strategies like Ditto [89], which trains a global and a local model simultaneously for personalization.

NvFlare does not support formal privacy verification such as DP guarantee with privacy bounds, cryptographic proof of update obfuscation, or formal proof checkers (e.g. CertiDP and ProfVerif). It also has no built-in support for anomaly detection, but developers can define custom filters on server or clients to inspect suspicious behavior such as unexpected update magnitudes, gradient divergence, or update timing anomalies.

`FLComponent` is the base class of all NvFlare components, including executors, controllers, responders, filters, aggregators, and widgets. FL components can fire and handle events, including federation events sent to different sites. NvFlare also defines `Learnable` abstraction for FL models and `Shareable` abstraction for data exchanged between clients and servers. Figure 7 illustrates the inheritance relation of the executor, controller, and aggregator components.

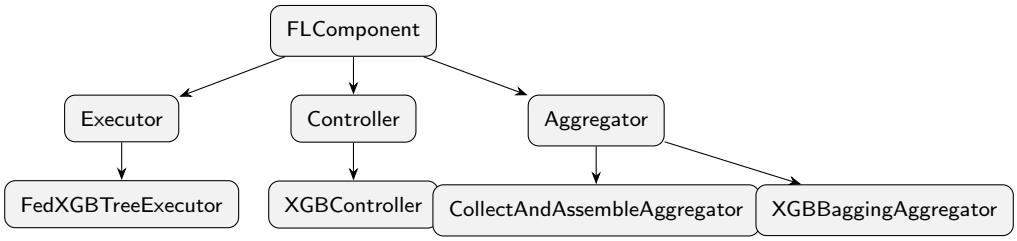

Figure 7. This diagram illustrates the inheritance hierarchy of the FL component class in the NvFlare library.

4.7. OpenFL

Data partitioning

OpenFL[9] supports horizontal FL configured in YAML files. Each collaborator extends `DataInterface` and uses its local shard descriptor to build data loaders.

Model training

Training tasks and evaluation tasks are defined by decorators in the `TaskRunner` code via `TaskInterface` and `ModelInterface`. Collaborators load the global model, train locally per round,

[9] The latest version of OpenFL (https://github.com/securefederatedai/openfederatedlearning) is 1.9 released in 2025.

and return model updates. The FL Plan contains task groups assigned to collaborators per round. The model architecture and optimizer registration is done via `ModelInterface`, enabling plug-and-play PyTorch or TensorFlow backend.

Aggregation and fusion

The `Aggregator` node (model owner) runs the federation rounds, which collects model updates from collaborators (local data owner), performs federated averaging (weighted by data size), and distributes the new model. Aggregation behavior is defined by the `TaskRunner`, FL plan, and aggregator logic.

Privacy protection

Data exchange between collaborator and aggregator node is secured over gPRC with TLS. OpenFL supports running inside Intel SGX enclaves (Trusted Execution Environments) and uses the Intel Trust Authority for runtime attestation of participants to ensure fidelity and trust.

Communication

An FL workflow is defined and distributed via the FL plan, which all participants load before starting. The CLIs launch secure gRPC servers, where each collaborator queries the aggregator for task instructions. OpenFL supports both simulation on single machine and multi-node deployment, all using secure gRPC communication.

Extension and customization

OpenFL has partially support for auditing through logging FL events such as collaborator registration, task assignments, and round status. Users can configure per-collaborator logs and export them centrally for manual auditing. Personalization is supported via code customization Developers can implement client-specific heads (e.g. FedPer [87]), fine-tuning models locally at each collaborator, or multi-task FL, where clients use the same base model but optimize for different outputs. There is no built-in support for formal privacy verification or anomaly detection but custom `TaskRunner` can check model accuracy and inspect updates before sending them to aggregator. Hooks can be added to aggregator to inspect model updates.

`AggregationFunction` is the base class for various aggregation functions, which are used by `Aggregator` to compose the results of the FL collaborators. `TaskRunner` is another extension point for runners using different ML models. Figure 8 illustrates the relationship of these classes.

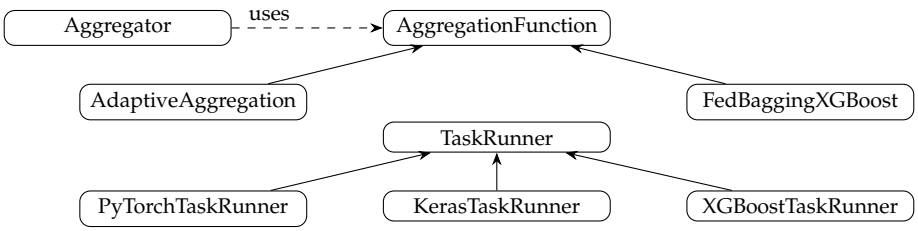

Figure 8. This diagram illustrates the inheritance hierarchy of the aggregation function and task runner classes in the OpenFL library.

4.8. PaddleFL

Data partitioning

PaddleFL[10] supports both horizontal and vertical FL (via SMPC). In vertical FL, data is aligned and encrypted using PrivC protocols. Partitioning and alignment is defined at compile time via

[10] The latest version of PaddleFL (https://github.com/PaddlePaddle/PaddleFL) is 1.2 released in 2021.

`FLStrategyFactory` settings and executed by `JobGenerator`, which builds FL job configurations in JSON/YAML.

Model training

Models are written in PaddlePaddle library. The `JobGenerator` uses the models plus the chosen strategy (e.g. FedAvg, DP-SGD, vertical neural net with PrivC) to produce runtime jobs for both server and clients `FLTrainer` instances on worker nodes run the training loops, feed the local data, and perform local updates. `FLServer` class coordinates the aggregation cycles

Aggregation and fusion

PaddleFL has built-in strategies via `FLStrategyFactory`, such as FedAvg, DP-SGD, and PrivC-based vertical learning strategies. Aggregation logic is embedded within the FL runtime, where the parameter server combines worker updates (weighted average or encrypted combination) each round.

Privacy protection

PaddleFL supports DP-SGD with gradient clipping and noise injection. It also implements PrivC [90] for secure two party computation and ABY3 [91] for secure three-party computation for vertical FL. These protocols preserve data confidentiality while aligning and jointly training models.

Communication

PaddleFL works in two phases. At compile-time, `JobGenerator` creates jobs for `FLServer`, `FLTrainer`, and `FLScheduler`. At runtime, the class `FLScheduler` orchestrates which trainers participate in each cycle. `FLServer` runs the parameter server logic. `FLTrainers` run local training, communicate via secure channels to `FLScheduler` and server nodes. Communication happens over HTTP/gRPC with optional secure channels. Jobs can be deployed across clusters via job submission tools

Extension and customization

PaddleFL does not support auditing, anomaly detection, or formal privacy verification. PaddleFL has limited support for personalized FL techniques like pFedMe [92], FedPer [87], or fine-tuning after global convergence. Developers can implement personalization manually by modifying the local training loop (inside `FLTrainer`) to include fine-tuning, meta-learning, or multi-task learning approaches post-aggregation.

`FLStrategyBase` is the base class or algorithms such as SGD with differential privacy, federated averaging, and secure aggregation. `FLTrainer` is the base for client trainers such as `FedAvgTrainer` and `SecAggTrainer`.

4.9. PySyft

The current version of PySyft[11] is no longer a conventional FL library like Flower, FedML, FedLab, or TFF. It has evolved into a privacy-preserving framework focused on data governance, remote data access, and secure data science workflows, not just model training across clients. Instead of defining the clients and server for each FL application, PySyft creates a data repository called datasite for each data owner. The datasites allow authorized users to submit code for machine learning purposes. A datasite holds private data and exposes the public results computed from the private data through the data owner API. A datasite can also host an aggregator that obtains the synchronized results from other datasites and performs aggregation to obtain the final result.

The computation model of PySyft is data-owner focused, where owners expose their data via data-site interface, and FL aggregators are clients of the data-sites. The typical extension points for other libraries, such as aggregation algorithms, FL collaborator selections, and model training, are

[11] The latest versino of PySyft (https://github.com/OpenMined/PySyft) is 0.9.5 released in 2025

external to the basic PySyft framework. PySyft offers privacy-preserving mechanisms such as SMPC, DP, remote data access with audit trails, and API for data scientists to work on private datasets.

4.10. TFF

Data partitioning

TensorFlow Federated (TFF)[12] is primarily designed for simulation of FL workflows, not for production deployment. It only supports horizontal FL in its public release.

Model training

Model is defined via a function that returns a `tff.learning.Model`. Users can create a training process with a builder function (e.g. `build_weighted_fed_avg`), passing in client and server optimizers and an aggregation factory. The resulting `LearningProcess` object defines `initialize` method and iterative `next` method to orchestrate training rounds.

Aggregation and fusion

Aggregation logic is configured via aggregation factories that return a base aggregator or aggregators wrapped or combined with components for clipping, secure aggregation, compression, or differential privacy. The aggregator is then used to build training process with composition of privacy and robustness strategies.

Privacy protection

TFF provides built-in support for differential privacy by wrapping aggregation algorithm with adaptive clipping and noise addition. Developers can also apply additional layers like zeroing, quantization, and secure aggregation by wrapping base aggregations. Formal privacy accounting (ϵ and δ estimates) is computed separately.

Communication

TFF uses a functional and compilation-based model, where users define federated computations (training and evaluation) via the FL API and they are compiled into internal `tff.Computation` objects representing execution graphs. TFF only supports simulation, where actual deployment requires custom runtime infrastructure.

Extension and customization

TFF does not have built-in support for auditing, anomaly detection, or formal privacy verification. Developers can simulate anomaly detection by modifying aggregation logic in the aggregation functions and simulating malicious clients in a controlled setting. Personalization can be simulated by fine-tuning client-side models after global rounds or by implementing algorithms like FedPer [87], FedRep [93], or pFedMe [92] manually.

`IterativeProcess` is the root for FL computations in TFF. `MeasuredProcess` is a subclass that defines a stateful process that produces metrics and is arbitrarily composable. The class `AggregationProcess` extends `MeasuredProcess` to aggregate client updates at the server. Figure 9 illustrates the relations of some of these classes.

5. Library Selection for Example Applications

While many FL libraries support core features such as model aggregation and client orchestration, their architectural designs and technical implementations differ substantially, leading to distinct performance characteristics and suitability for various application scenarios. This section provides practical guidance for selecting FL libraries. We first present a comprehensive comparative overview.

[12] The latest version of TFF (https://github.com/google-parfait/tensorflow-federated) is 0.88 released in 2024.

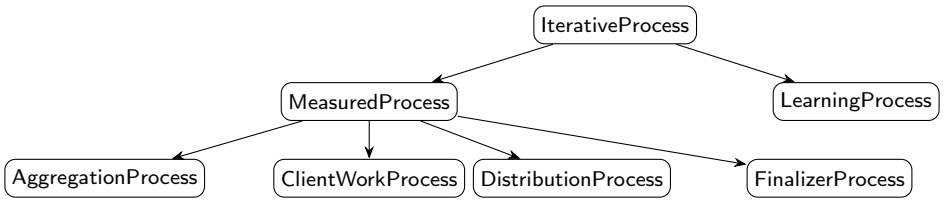

Figure 9. This diagram illustrates the inheritance hierarchy of the iterative process class in the TensorFlow Federated library.

Then we give an example decision tree for selecting libraries. Lastly, we describe three application examples to demonstrate how theoretical capabilities translate to practical implementations.

5.1. Comprehensive Library Comparison

Table 5 provides a systematic comparison of the ten surveyed FL libraries across four critical dimensions: application domains, infrastructure design, core technologies, and performance considerations. This synthesis reveals several important patterns in the FL ecosystem.

Application Specialization Libraries have evolved to serve distinct application niches. FATE and NVFlare excel in regulated vertical FL scenarios like finance and healthcare, while FedML and Flower demonstrate strengths in horizontal FL across edge devices and IoT. OpenFL occupies a unique position with its hardware-level security through Intel SGX, making it particularly suitable for medical imaging where data sensitivity is paramount.

Architectural Patterns Two dominant architectural paradigms emerge: the `pipeline` or `workflow` model (FATE, NVFlare) favored for complex, multi-party enterprise scenarios, and the `strategy/handler` pattern (Flower, FedML, FedLab) that offers greater flexibility for research and algorithm development. This fundamental design choice significantly impacts both extensibility and deployment complexity.

Technology Integration The libraries exhibit varying levels of integration with privacy-enhancing technologies (PETs). Production-focused frameworks like FATE and NVFlare provide built-in support for SMPC, HE, and PSI, while research-oriented libraries often treat PETs as extensible components, offering greater flexibility at the cost of implementation effort.

5.2. Decision Framework for Library Selection

Building on the comparative analysis in Table 5, Figure 10 provides a practical decision tree that maps key application requirements to suitable library choices. The framework considers three primary dimensions: data partitioning strategy, deployment environment constraints, and specific feature requirements.

The decision paths reveal important trade-offs. For instance, projects requiring vertical FL are naturally directed toward FATE or NVFlare, reflecting their mature support for cryptographic protocols and identity alignment. Conversely, edge computing scenarios with resource constraints lead to FedML or Flower, which offer lightweight communication protocols and efficient client management.

This decision tree serves as an initial screening tool. However, several additional factors warrant consideration:

Performance Trade-offs Libraries optimized for production deployment (FATE, NVFlare) typically incur higher initial setup costs but provide better long-term maintainability. Research-focused frameworks (FedLab, TFF) offer rapid prototyping capabilities but may require significant customization for production use.

Technology Stack Compatibility The choice may be constrained by existing infrastructure. TFF naturally integrates with TensorFlow ecosystems, while PaddleFL requires commitment to the

Table 5. Comprehensive Comparison of Federated Learning Libraries: Applications, Infrastructure, and Performance

Library	Primary Application Areas	Infrastructure Design	Key Technologies	Performance Considerations
FATE	Finance, Healthcare (VFL)	Microservices, KubeFATE, Pipeline-based	SMPC, HE, PSI, Federated ML Algorithms	High setup complexity; designed for cross-silo; robust but heavyweight.
FedML	IoT, Edge, General Research	Cross-device & Cross-silo, Runner-based Orchestration	MQTT, gRPC, MPI, High-performance NCCL	Scalable to many clients; supports simulation and deployment.
FedLab	Academic Research, Prototyping	Fine-grained Client/Handler, Simulation-focused	PyTorch, Custom TCP/RPC	Lightweight for prototyping; fine-grained control over data and network simulation.
Flower	IoT, Mobile, General Research, Cross-silo	Client-Server, Strategy-based, Framework-agnostic	gRPC/HTTPS, Pluggable Strategies (FedAvg, FedProx)	Flexible and lightweight; suitable for both simulation and production.
IBM FL	Enterprise, Cross-silo	Configuration-driven, Handler-based	Flask/gRPC, Fusion Handlers	Modular and extensible; designed for enterprise integration.
NVFlare	Healthcare, Finance, Regulated Industries	Event-driven, Service-oriented, Job-based	gRPC/TLS, Filters (DP, HE, PSI), AuditService	Production-ready with built-in auditing; secure and scalable deployment.
OpenFL	Medical Imaging, Trusted Environments	Aggregator-Collaborator, Plan-based	gRPC/TLS, Intel SGX (TEE)	Strong security via TEEs; suitable for sensitive data in healthcare.
PaddleFL	Vertical FL Scenarios, PaddlePaddle Ecosystem	Compile-time Job Generation, Scheduler-based	PrivC, ABY3, DP-SGD	Supports cryptographic VFL; not actively maintained; tied to PaddlePaddle.
PySyft	Data Governance, Privacy-Preserving Analytics	DataSite-centric, Remote Execution	SMPC, DP, Remote Data Access	Evolved beyond conventional FL; focuses on secure data science workflows.
TFF	Algorithm Simulation, Research	Functional, Compilation-based, Simulation-only	TensorFlow, Aggregation Factories	Excellent for research prototyping; not designed for production deployment.

PaddlePaddle platform. Framework-agnostic libraries like Flower provide greater flexibility but may involve additional integration effort.

Long-term Viability Maintenance status and community support critically impact library selection. While PaddleFL offers unique VFL capabilities, its inactive maintenance status poses significant adoption risks for long-term projects.

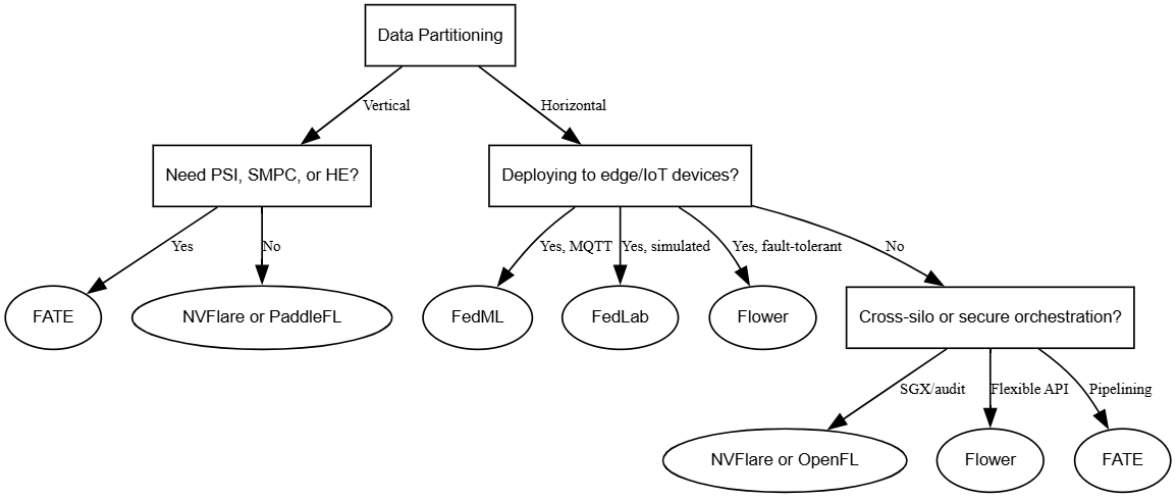

Figure 10. Decision tree for FL libraries based on data partitioning, deployment context, and features.

5.3. Example Applications

To illustrate how the abstract capabilities in Table 5 translate to concrete implementation choices, this section presents three representative application scenarios with distinct technical and regulatory requirements. Table 6 summarizes the key requirements for each scenario, while the following analysis demonstrates how library architectures align with specific application needs.

Table 6. Comparison of Application Requirements

Requirement	Medical Imaging	IoT Maintenance	Finance (VFL)
Data partition	Horizontal	Horizontal	Vertical
Data Sensitivity	Very High	Moderate	Extremely High
Heterogeneity	High	High	Low to Moderate
Client Resources	Moderate to High	Low (Edge)	High (Enterprise)
Comm. Constraints	Moderate	High	Low to Moderate
Privacy Req.	Regulatory	IP Protection	Regulatory, Legal
Scale	Dozens to 100s	1000s to Millions	Dozens
Recommended Libraries	NVFlare, OpenFL, Flower	FedML, FedLab, Flower	FATE, NVFlare, PaddleFL

Medical Image Analysis Medical image analysis involves processing large, sensitive datasets such as MRI or CT scans distributed across hospitals or clinics. These datasets are typically horizontally partitioned and non-IID due to differences in patient demographics, imaging protocols, and hardware. Strong privacy guarantees, regulatory compliance (e.g. HIPAA, GDPR), and robust orchestration are critical requirements.

Libraries such as NVFlare, OpenFL, and Flower are particularly well-suited for this domain. NVFlare (Section 4.6) provides built-in workflows for healthcare, native support for secure communication over gRPC, and a robust auditing subsystem via the `AuditService` class [78]. OpenFL (Section 4.7) supports deployment in trusted execution environments (Intel SGX) and provides `TaskRunner` and `AggregationFunction` suitable for integrating medical workflows [76]. Flower (Section 4.4) offers a flexible `Strategy` interface and secure aggregation with differential privacy via its modular `Mods` system [72], allowing adaptation to privacy policies and update inspection needs. These libraries are appropriate for institutions seeking trusted, extensible, and production-ready FL solutions.

Predictive Maintenance with IoT Data Predictive maintenance systems monitor edge devices, such as sensors embedded in industrial equipment, to forecast failures. These systems face practical

constraints: low compute power, intermittent network connectivity, and highly non-IID data from device heterogeneity. Communication efficiency, scalability to thousands of devices, and client failure tolerance are critical factors.

FedML (Section 4.3) is designed for scalable FL with cross-device and cross-silo deployments. It supports MQTT, gRPC, and various orchestration modes via `FedMLRunner` [73]. FedLab (Section 4.2) is suitable for prototyping in academic or industrial simulations due to its modular trainer/handler architecture and fine-grained control over data partitioning [77]. Both libraries allow developers to customize aggregation and training by extending `ClientTrainer` and `ServerAggregator`. `Flower` also serves well in this domain with built-in support for client sampling, compression, and fault-tolerant strategies via its `Strategy` interface. Lightweight deployment, simulation-friendly APIs, and extensibility make these libraries suitable for IoT-related application.

Consumer Financial Modeling Financial institutions such as banks, lenders, and credit agencies often hold complementary information (features) about overlapping customer sets, creating a need for vertical FL (VFL). The goal is to train models on feature-partitioned datasets while ensuring strict data privacy and regulatory compliance. VFL requires secure multi-party computation (SMPC), private set intersection (PSI), or homomorphic encryption (HE) to protect sensitive data during training.

Among the libraries compared, FATE (Section 4.1) offers the most mature and comprehensive support for VFL. It includes built-in PSI protocols for identity alignment, VFL algorithms such as `HeteroLR` and `HeteroSecureBoost`, and components orchestrated through `FATE-Flow` [74]. NVFlare (Section 4.6) supports both VFL and split learning via specialized workflows and filter-based privacy enforcement. It provides a scalable and auditable platform suitable for regulated industries. PaddleFL (Section 4.8) supports VFL using cryptographic protocols like PrivC and ABY3, although it is not actively maintained [80]. These libraries are best suited for use cases requiring secure, multi-party computation over distributed feature sets.

These examples demonstrate how application-specific constraints directly influence the selection of FL libraries. By considering the software architecture and native support for features such as vertical partitioning, privacy protocols, scalability, and extensibility (as analyzed in Section 4), developers can make informed design decisions that meet both performance and regulatory requirements. Nevertheless, many advanced requirements, such as auditing, anomaly detection, and formal privacy verification, still require custom development or third-party integration [26, 30], representing open challenges for FL infrastructure.

These examples demonstrate how application-specific constraints directly influence library selection, moving beyond feature checklists to consider architectural compatibility and ecosystem integration. The medical imaging case highlights the importance of built-in auditing and hardware security; IoT maintenance emphasizes communication efficiency and fault tolerance; financial modeling requires robust cryptographic foundations for vertical FL. In each case, successful implementation depends on aligning the library's architectural paradigm with the application's operational requirements.

Nevertheless, many advanced requirements, such as sophisticated anomaly detection and formal privacy verification, still require custom development or third-party integration [26,30], representing both a limitation of current libraries and an opportunity for future framework development.

6. Discussion, Limitations, and Future Pathways

Our comparative analysis, conducted through a software engineering lens, reveals several persistent *architectural* challenges and limitations in current FL libraries that are not apparent from a mere feature comparison. This section synthesizes these software-centric limitations, proposes concrete architectural solutions grounded in our analysis of the libraries' codebases, and outlines a pathway toward next-generation FL frameworks.

6.1. Limitations of Current FL Libraries

Despite the diversity and maturity of available FL libraries, our analysis identifies three fundamental limitations that hinder their broader adoption in production environments:

Fragmented Support for Advanced Features While core FL functionalities like federated averaging and basic privacy protection are well-supported, advanced capabilities remain inconsistently implemented. As detailed in Section 4, features such as formal privacy verification, robust anomaly detection, and comprehensive auditing are either absent, require significant custom development, or are supported by only one or two libraries (e.g., NVFlare's `AuditService`). This fragmentation forces developers to choose between functionality and library maturity, often resulting in costly integrations or security compromises.

High Configuration and Deployment Complexity Production-grade libraries like FATE and NVFlare, while powerful, exhibit high initial setup complexity. The microservices architecture of FATE and the event-driven model of NVFlare require substantial DevOps expertise, creating a steep learning curve that impedes rapid prototyping and adoption in research settings. This complexity often conflicts with the need for agile development and experimentation in machine learning workflows.

Architectural Gaps in Monitoring and Assurance A significant limitation across most libraries is the tight coupling of training logic with monitoring and verification functions. Libraries lack standardized interfaces for real-time model validation, privacy budget tracking, and performance monitoring. This architectural gap makes it difficult to implement continuous assurance mechanisms that are essential for regulated industries like healthcare and finance.

6.2. Comparison to Prior Works

Compared with the surveys summarized in Table 1, this work occupies a distinct position in the broader landscape of FL research. Earlier comprehensive surveys [1,4,8] primarily emphasize theoretical foundations, privacy mechanisms, and communication strategies. System- and application-oriented reviews [26,30–33] offer valuable overviews of FL frameworks and implementation challenges but focus mainly on algorithmic taxonomy, system functionality, or performance comparison.

In contrast, our study approaches FL from a *software engineering* perspective. It examines how current open-source libraries are architected, how their design decisions affect extensibility, configuration, and maintainability, and how these architectures can be extended to incorporate advanced mechanisms such as formal privacy verification, anomaly detection, and auditing. This perspective complements existing reviews by linking engineering design principles to the broader goals of secure, transparent, and trustworthy federated learning systems.

Moreover, while prior surveys often categorize FL frameworks by application domain or learning paradigm, our analysis highlights the *integration dimension*: how distinct functional concerns, verification, monitoring, and traceability, can be modularized and unified across libraries. This focus connects theoretical algorithmic advances with practical implementation, offering actionable guidance for library users facing the limitations described in Section 6.1.

6.3. Architectural Solutions for Current Limitations

The limitations identified in Section 6.1 stem primarily from architectural choices rather than algorithmic deficiencies. Below we propose concrete modifications to existing library architectures to address these gaps.

Formal Privacy Verification: automated privacy guarantee validation beyond basic differential privacy implementations.

Among the reviewed libraries, TensorFlow Federated (TFF) and FATE are the most amenable to integrating privacy verification modules. TFF's aggregation factories already encapsulate differential privacy and secure aggregation primitives, making them a natural location to insert verification hooks.

A formal verification layer could be introduced as a wrapper around the aggregation factory to perform symbolic checks or privacy-budget accounting before committing updates. FATE, which organizes computations through pipeline components and decorators, could expose a verification decorator that automatically registers each component's privacy parameters (e.g., noise scale, encryption scheme) into a global registry. This registry could support runtime verification and post-training auditing of privacy guarantees. Both approaches require minimal disruption to the existing pipeline architecture while enabling transparent verification.

Anomaly Detection: built-in mechanisms to identify and mitigate malicious updates and system anomalies.

Libraries such as FedML, Flower, and NVFlare are well suited for incorporating anomaly-detection mechanisms because their aggregation logic and message flow are mediated through extensible strategy or controller classes. In Flower, a practical modification would be to introduce an intermediate "inspection layer" between clients and the server's `Strategy` class. This layer could intercept and evaluate client updates using configurable detection modules, statistical distance checks, gradient clustering, or learned anomaly models, before aggregation. In FedML, anomaly detection could be integrated as an optional plugin to the `ServerAggregator`, leveraging its existing hooks for update validation. For NVFlare, which already implements event-based communication and filter chains, anomaly detectors could be realized as filters registered in the communication pipeline. This approach aligns with NVFlare's modular architecture and allows deployment-time configuration of detection policies.

Auditing and Traceability: comprehensive logging and provenance tracking for compliance and debugging.

Auditing capabilities are partially available in NVFlare and OpenFL, but their use could be generalized through a unified audit-service interface. In NVFlare, the existing `AuditService` could be extended to produce structured audit logs with model fingerprints, participant identifiers, and signed metadata that describe each training round. A schema-driven approach would allow these logs to be exported for compliance checking or visualization. OpenFL, built around task and aggregation interfaces, could embed auditing within its `TaskRunner` to capture the provenance of each model update. Extending this design with a plugin registry for external log sinks (e.g., databases or blockchain ledgers) would further enhance transparency and reproducibility.

6.4. Toward an Integrated FL Framework

Addressing the limitations in Section 6.1 requires a coherent architectural vision that transcends individual library improvements. Across frameworks, a consistent direction would be to decouple monitoring and verification functions from core training logic. This can be achieved by defining standard callback interfaces (e.g., `on_aggregate_start`, `on_update_validated`) that allow registration of verification or anomaly-detection modules, and adopting configuration-driven composition through YAML or JSON specifications. Such design patterns already appear in FedML's pipeline runner and NVFlare's job configuration model and could be generalized across libraries.

Ultimately, a next-generation FL framework should consolidate these capabilities, formal privacy verification, anomaly detection, and auditing, into a unified software stack. Among the existing libraries, NVFlare provides the most suitable foundation for such integration. Its event-driven architecture, service-oriented design, and modular component registry enable new functionality to be attached at multiple levels without disrupting the training pipeline. NVFlare's existing concepts of filters, event handlers, and audit services already map naturally to verification, detection, and traceability extensions. By generalizing these abstractions and standardizing their interfaces, NVFlare could evolve into a comprehensive, enterprise-ready FL platform that combines security assurance, transparency, and usability.

Other libraries such as FATE or Flower could adopt similar architectural paradigms, introducing event buses, audit schemas, and pluggable verification modules, to move toward the same goal of cohesive and trustworthy FL software ecosystems. The key insight is that overcoming current limitations requires not just new features, but fundamental architectural evolution toward modular, observable, and verifiable FL systems.

7. Conclusion

Federated Learning has emerged as a transformative paradigm for training machine learning models across distributed data sources while preserving data privacy. However, the practical adoption of FL technologies remains hindered by challenges related to privacy, security, and software engineering limitations. This survey provided a comprehensive comparative analysis of the top ten FL libraries, evaluated based on their support for core aspects (data partitioning, model training, aggregation, privacy protection, and communication) and advanced features (auditing, anomaly detection, personalization, and verification of privacy). By systematically examining these libraries, we aimed to guide developers in selecting the most suitable tools for their specific use cases and to identify gaps for future research and development.

Our findings demonstrate the diversity in functionality, extensibility, and maturity among FL libraries. Libraries like FATE, NVFlare, and PaddleFL excel in vertical FL and robust privacy mechanisms, making them ideal for regulated industries such as healthcare and finance. In contrast, FedML, FedLab, and Flower offer lightweight, flexible solutions for horizontal FL, particularly suited for IoT and edge computing scenarios. Despite these advancements, several challenges persist, including the need for better support for formal privacy verification, anomaly detection, and auditing in many libraries. Additionally, the rapid evolution of FL technologies necessitates continuous updates and community-driven improvements to these tools.

The three example applications, medical image analysis, predictive maintenance with IoT data, and consumer financial modeling, demonstrated how specific requirements influence library selection. These case studies underscored the importance of aligning technical capabilities with real-world constraints, such as regulatory compliance, resource limitations, and scalability needs.

7.1. Future Work

In future work, researchers should move beyond general-purpose FL platforms toward developing domain-specialized frameworks that address the unique requirements of cross-silo scenarios. In such settings, typical of educational or medical research, the number of participants is relatively small, but the level of required trust, auditability, and privacy assurance is high. A practical and actionable direction is to fork one or more existing libraries to create a streamlined and trustworthy variant tailored to this environment. Such a specialized framework could reduce the code base, remove features primarily designed for large-scale cross-device FL, and incorporate focused modules for privacy verification, anomaly detection, and auditing.

Combining the strengths of complementary libraries offers another promising path. For instance, NVFlare provides robust client provisioning, secure deployment, and lifecycle management suitable for regulated domains, while Flower offers a flexible and easily extensible architecture for pluggable aggregation strategies. A hybrid approach, integrating Flower's aggregation layer into NVFlare's orchestration and deployment stack, could yield an industrial-grade yet research-accessible framework for cross-silo federated learning. Further research should investigate the architectural and API-level harmonization required to enable such integration.

In addition, efforts should focus on defining common configuration schemas, standardized event interfaces, and reproducible benchmarks for evaluating privacy verification and auditing mechanisms. By pursuing these actionable goals, the FL community can transition from loosely coupled libraries toward interoperable, domain-focused infrastructures that support trustworthy and transparent federated learning in high-stakes applications.

Future research should also explore the convergence of federated learning with other cutting-edge AI paradigms. A particularly compelling direction is to leverage the privacy-preserving, collaborative training capabilities of FL frameworks for developing and scaling Large Model based Agents, facilitating their adaptation in decentralized and sensitive environments without centralizing data [94].

Abbreviation	Full Term
CNN	Convolutional Neural Network
DP	Differential Privacy
FL	Federated Learning
HE	Homomorphic Encryption
HFL	Horizontal Federated Learning
IDS	Intrusion Detection System
IoT	Internet of Things
KD	Knowledge Distillation
Non-IID	Non-Independent and Identically Distributed
PSI	Private Set Intersection
SecAgg	Secure Aggregation
SGD	Stochastic Gradient Descent
SMPC	Secure Multi-Party Computation
TEE	Trusted Execution Environment
TPA	Third-Party Auditor
VFL	Vertical Federated Learning

References

1. Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; Li, Y.; Liu, X.; He, B. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering* **2021**, *35*, 3347–3366.
2. Li, Z.; Lowy, A.; Liu, J.; Koike-Akino, T.; Parsons, K.; Malin, B.; Wang, Y. Analyzing Inference Privacy Risks Through Gradients in Machine Learning. *arXiv preprint arXiv:2408.16913* **2024**.
3. Wang, Y.; Su, Z.; Zhang, N.; Benslimane, A. Learning in the Air: Secure Federated Learning for UAV-Assisted Crowdsensing. *IEEE Transactions on Network Science and Engineering* **2021**, *8*, 1055–1069. https://doi.org/10.1109/TNSE.2020.3014385.
4. AbdulRahman, S.; Tout, H.; Ould-Slimane, H.; Mourad, A.; Talhi, C.; Guizani, M. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal* **2020**, *8*, 5476–5497.
5. Zhang, C.; Xie, Y.; Bai, H.; Yu, B.; Li, W.; Gao, Y. A survey on federated learning. *Knowledge-Based Systems* **2021**, *216*, 106775.
6. El Ouadrhiri, A.; Abdelhadi, A. Differential privacy for deep and federated learning: A survey. *IEEE access* **2022**, *10*, 22359–22380.
7. Balasubramanian, K.; Mala, K. Zero knowledge proofs: A survey. In *Algorithmic Strategies for Solving Complex Problems in Cryptography*; IGI Global, 2018; pp. 111–123.
8. Aledhari, M.; Razzak, R.; Parizi, R.M.; Saeed, F. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access* **2020**, *8*, 140699–140725.
9. Lim, W.Y.B.; Luong, N.C.; Hoang, D.T.; Jiao, Y.; Liang, Y.C.; Yang, Q.; Niyato, D.; Miao, C. Federated learning in mobile edge networks: A comprehensive survey. *IEEE communications surveys & tutorials* **2020**, *22*, 2031–2063.
10. Lyu, L.; Yu, H.; Yang, Q. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133* **2020**.
11. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning* **2021**, *14*, 1–210.
12. Chai, D.; Wang, L.; Yang, L.; Zhang, J.; Chen, K.; Yang, Q. A survey for federated learning evaluations: Goals and measures. *IEEE Transactions on Knowledge and Data Engineering* **2024**.
13. Yin, X.; Zhu, Y.; Hu, J. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–36.

14. Xia, Q.; Ye, W.; Tao, Z.; Wu, J.; Li, Q. A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Computing* **2021**, *1*, 100008.
15. Rahman, K.J.; Ahmed, F.; Akhter, N.; Hasan, M.; Amin, R.; Aziz, K.E.; Islam, A.M.; Mukta, M.S.H.; Islam, A.N. Challenges, applications and design aspects of federated learning: A survey. *IEEE Access* **2021**, *9*, 124682–124700.
16. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Poor, H.V. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys and Tutorials* **2021**, *23*, 1622–1658.
17. Zhu, H.; Xu, J.; Liu, S.; Jin, Y. Federated learning on non-IID data: A survey. *Neurocomputing* **2021**, *465*, 371–390.
18. Banabilah, S.; Aloqaily, M.; Alsayed, E.; Malik, N.; Jararweh, Y. Federated learning review: Fundamentals, enabling technologies, and future applications. *Information processing & management* **2022**, *59*, 103061.
19. Liu, J.; Huang, J.; Zhou, Y.; Li, X.; Ji, S.; Xiong, H.; Dou, D. From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems* **2022**, *64*, 885–917.
20. Zhan, Y.; Zhang, J.; Hong, Z.; Wu, L.; Li, P.; Guo, S. A survey of incentive mechanism design for federated learning. *IEEE Transactions on Emerging Topics in Computing* **2021**, *10*, 1035–1044.
21. Nguyen, D.C.; Pham, Q.V.; Pathirana, P.N.; Ding, M.; Seneviratne, A.; Lin, Z.; Dobre, O.; Hwang, W.J. Federated learning for smart healthcare: A survey. *ACM Computing Surveys (Csur)* **2022**, *55*, 1–37.
22. Ma, X.; Zhu, J.; Lin, Z.; Chen, S.; Qin, Y. A state-of-the-art survey on solving non-iid data in federated learning. *Future Generation Computer Systems* **2022**, *135*, 244–258.
23. Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; Li, Y.; Liu, X.; He, B. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering* **2023**, *35*, 3347–3366.
24. Wen, J.; Zhang, Z.; Lan, Y.; Cui, Z.; Cai, J.; Zhang, W. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics* **2023**, *14*, 513–535.
25. Sánchez, P.M.S.; Celdrán, A.H.; Xie, N.; Bovet, G.; Pérez, G.M.; Stiller, B. Federatedtrust: A solution for trustworthy federated learning. *Future Generation Computer Systems* **2024**, *152*, 83–98.
26. Luzón, M.V.; Rodríguez-Barroso, N.; Argente-Garrido, A.; Jiménez-López, D.; Moyano, J.M.; Del Ser, J.; Ding, W.; Herrera, F. A tutorial on federated learning from theory to practice: Foundations, software frameworks, exemplary use cases, and selected trends. *IEEE/CAA Journal of Automatica Sinica* **2024**, *11*, 824–850.
27. Wu, Z.; Sun, S.; Wang, Y.; Liu, M.; Jiang, X.; Li, R.; Gao, B. Knowledge Distillation in Federated Edge Learning: A Survey, 2024, [arXiv:cs.LG/2301.05849].
28. Shao, J.; Li, Z.; Sun, W.; Zhou, T.; Sun, Y.; Liu, L.; Lin, Z.; Mao, Y.; Zhang, J. A survey of what to share in federated learning: Perspectives on model utility, privacy leakage, and communication efficiency. *arXiv preprint arXiv:2307.10655* **2023**.
29. Ji, S.; Tan, Y.; Saravirta, T.; Yang, Z.; Liu, Y.; Vasankari, L.; Pan, S.; Long, G.; Walid, A. Emerging trends in federated learning: From model fusion to federated x learning. *International Journal of Machine Learning and Cybernetics* **2024**, pp. 1–22.
30. Riedel, P.; Schick, L.; von Schwerin, R.; Reichert, M.; Schaudt, D.; Hafner, A. Comparative analysis of open-source federated learning frameworks-a literature-based survey and review. *International Journal of Machine Learning and Cybernetics* **2024**, *15*, 5257–5278.
31. Ye, M.; Shen, W.; Du, B.; Snezhko, E.; Kovalev, V.; Yuen, P.C. Vertical federated learning for effectiveness, security, applicability: A survey. *ACM Computing Surveys* **2025**, *57*, 1–32.
32. Quan, M.K.; Pathirana, P.N.; Wijayasundara, M.; Setunge, S.; Nguyen, D.C.; Brinton, C.G.; Love, D.J.; Poor, H.V. Federated learning for cyber physical systems: a comprehensive survey. *IEEE Communications Surveys & Tutorials* **2025**.
33. Zhan, S.; Huang, L.; Luo, G.; Zheng, S.; Gao, Z.; Chao, H.C. A Review on Federated Learning Architectures for Privacy-Preserving AI: Lightweight and Secure Cloud–Edge–End Collaboration. *Electronics* **2025**, *14*, 2512.
34. Gosselin, R.; Vieu, L.; Loukil, F.; Benoit, A. Privacy and security in federated learning: A survey. *Applied Sciences* **2022**, *12*, 9901.
35. Su, L.; Lau, V.K. Hierarchical federated learning for hybrid data partitioning across multitype sensors. *IEEE Internet of Things Journal* **2021**, *8*, 10922–10939.
36. Ahmadzai, M.; Nguyen, G. Data Partitioning Effects in Federated Learning. *Proceedings http://ceur-ws. org ISSN* **2023**, *1613*, 0073.

37. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017). PMLR, Apr 2017, Vol. 54, pp. 1273–1282.
38. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated Optimization in Heterogeneous Networks. In Proceedings of the Proceedings of the 3rd International Conference on Machine Learning and Systems (MLSys 2020), 2020. FedProx was introduced as a generalization of FedAvg to address systems and statistical heterogeneity:contentReference[oaicite:1]index=1.
39. Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; Poor, H.V. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In Proceedings of the Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020. 34th Conference on Neural Information Processing Systems, Virtual, December 6–12, 2020.
40. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482* **2016**.
41. Pillutla, K.; Kakade, S.M.; Harchaoui, Z. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing* **2022**, *70*, 1142–1154.
42. Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H.H.; Farokhi, F.; Jin, S.; Quek, T.Q.; Poor, H.V. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE transactions on information forensics and security* **2020**, *15*, 3454–3469.
43. Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. *Science* **2015**, *349*, 255–260.
44. Mugunthan, V.; Peraire-Bueno, A.; Kagal, L. Privacyfl: A simulator for privacy-preserving and secure federated learning. In Proceedings of the Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 3085–3092.
45. Adnan, M.; Kalra, S.; Cresswell, J.C.; Taylor, G.W.; Tizhoosh, H.R. Federated learning and differential privacy for medical image analysis. *Scientific reports* **2022**, *12*, 1953.
46. Ngo, T.; Nguyen, D.C.; Pathirana, P.N.; Corben, L.A.; Delatycki, M.B.; Horne, M.; Szmulewicz, D.J.; Roberts, M. Federated deep learning for the diagnosis of cerebellar ataxia: Privacy preservation and auto-crafted feature extractor. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* **2022**, *30*, 803–811.
47. Shahid, O.; Pouriyeh, S.; Parizi, R.M.; Sheng, Q.Z.; Srivastava, G.; Zhao, L. Communication efficiency in federated learning: Achievements and challenges. *arXiv preprint arXiv:2107.10996* **2021**.
48. Chen, M.; Shlezinger, N.; Poor, H.V.; Eldar, Y.C.; Cui, S. Communication-efficient federated learning. *Proceedings of the National Academy of Sciences* **2021**, *118*, e2024789118.
49. Zhang, J.; Zhu, H.; Wang, F.; Zhao, J.; Xu, Q.; Li, H. Security and privacy threats to federated learning: Issues, methods, and challenges. *Security and Communication Networks* **2022**, *2022*, 2886795.
50. Shen, S.; Tople, S.; Saxena, P. Auror: Defending against poisoning attacks in collaborative deep learning systems. In Proceedings of the Proceedings of the 32nd annual conference on computer security applications, 2016, pp. 508–519.
51. Rodríguez-Barroso, N.; Jiménez-López, D.; Luzón, M.V.; Herrera, F.; Martínez-Cámara, E. Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. *Information Fusion* **2023**, *90*, 148–173.
52. Lavaur, L.; Pahl, M.O.; Busnel, Y.; Autrel, F. The evolution of federated learning-based intrusion detection and mitigation: a survey. *IEEE Transactions on Network and Service Management* **2022**, *19*, 2309–2332.
53. Huong, T.T.; Bac, T.P.; Long, D.M.; Luong, T.D.; Dan, N.M.; Thang, B.D.; Tran, K.P.; et al. Detecting cyberattacks using anomaly detection in industrial control systems: A federated learning approach. *Computers in Industry* **2021**, *132*, 103509.
54. Mothukuri, V.; Parizi, R.M.; Pouriyeh, S.; Huang, Y.; Dehghantanha, A.; Srivastava, G. A survey on security and privacy of federated learning. *Future Generation Computer Systems* **2021**, *115*, 619–640.
55. Zhang, C.; Yang, S.; Mao, L.; Ning, H. Anomaly detection and defense techniques in federated learning: a comprehensive review. *Artificial Intelligence Review* **2024**, *57*, 1–34.
56. Alazab, M.; RM, S.P.; Parimala, M.; Maddikunta, P.K.R.; Gadekallu, T.R.; Pham, Q.V. Federated learning for cybersecurity: Concepts, challenges, and future directions. *IEEE Transactions on Industrial Informatics* **2021**, *18*, 3501–3509.
57. Kavasidis, I.; Lallas, E.; Mountzouris, G.; Gerogiannis, V.C.; Karageorgos, A. A Federated Learning Framework for Enforcing Traceability in Manufacturing Processes. *IEEE Access* **2023**.
58. Cai, Z.; Tang, T.; Yu, S.; Xiao, Y.; Xia, F. Marking the Pace: A Blockchain-Enhanced Privacy-Traceable Strategy for Federated Recommender Systems. *IEEE Internet of Things Journal* **2023**.

59. Macak, M.; Vanát, I.; Merjavỳ, M.; Jevočin, T.; Buhnova, B. Towards process mining utilization in insider threat detection from audit logs. In Proceedings of the 2020 seventh international conference on social networks analysis, management and security (snams). IEEE, 2020, pp. 1–6.
60. Sun, Z.; Wan, J.; Yin, L.; Cao, Z.; Luo, T.; Wang, B. A blockchain-based audit approach for encrypted data in federated learning. *Digital Communications and Networks* **2022**, *8*, 614–624.
61. Shao, S.; Yang, W.; Gu, H.; Qin, Z.; Fan, L.; Yang, Q. Fedtracker: Furnishing ownership verification and traceability for federated learning model. *IEEE Transactions on Dependable and Secure Computing* **2024**.
62. Chen, J.; Xue, J.; Wang, Y.; Huang, L.; Baker, T.; Zhou, Z. Privacy-Preserving and Traceable Federated Learning for data sharing in industrial IoT applications. *Expert Systems with Applications* **2023**, *213*, 119036.
63. Mansour, Y.; Mohri, M.; Ro, J.; Suresh, A.T. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619* **2020**.
64. Deng, Y.; Kamani, M.M.; Mahdavi, M. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461* **2020**.
65. Tan, A.Z.; Yu, H.; Cui, L.; Yang, Q. Towards personalized federated learning. *IEEE transactions on neural networks and learning systems* **2022**, *34*, 9587–9603.
66. Liu, Y.; Zhang, S.; Zhang, C.; James, J. Fedgru: Privacy-preserving traffic flow prediction via federated learning. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2020, pp. 1–6.
67. Xu, G.; Li, H.; Liu, S.; Yang, K.; Lin, X. VerifyNet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security* **2019**, *15*, 911–926.
68. Wang, Z.; Dong, N.; Sun, J.; Knottenbelt, W. zkFL: Zero-Knowledge Proof-based Gradient Aggregation for Federated Learning. *arXiv preprint arXiv:2310.02554* **2023**.
69. Chen, Z.; Tian, P.; Liao, W.; Yu, W. Zero knowledge clustering based adversarial mitigation in heterogeneous federated learning. *IEEE Transactions on Network Science and Engineering* **2020**, *8*, 1070–1083.
70. Nguyen, T.; Thai, M.T. Preserving privacy and security in federated learning. *IEEE/ACM Transactions on Networking* **2023**.
71. Ziller, A.; Trask, A.; Lopardo, A.; Szymkow, B.; Wagner, B.; Bluemke, E.; Nounahon, J.M.; Passerat-Palmbach, J.; Prakash, K.; Rose, N.; et al. Pysyft: A library for easy federated learning. *Federated Learning Systems: Towards Next-Generation AI* **2021**, pp. 111–139.
72. Beutel, D.J.; Topal, T.; Mathur, A.; Qiu, X.; Fernandez-Marques, J.; Gao, Y.; Sani, L.; Li, K.H.; Parcollet, T.; de Gusmão, P.P.B.; et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390* **2020**.
73. He, C.; Li, S.; So, J.; Zeng, X.; Zhang, M.; Wang, H.; Wang, X.; Vepakomma, P.; Singh, A.; Qiu, H.; et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518* **2020**.
74. Liu, Y.; Fan, T.; Chen, T.; Xu, Q.; Yang, Q. Fate: An industrial grade platform for collaborative learning with data protection. *Journal of Machine Learning Research* **2021**, *22*, 1–6.
75. Patil, R.; Gupta, D.; Gururaj, H. Federated Learning using TensorFlow. In *Federated Learning Techniques and Its Application in the Healthcare Industry*; World Scientific, 2024; pp. 171–189.
76. Foley, P.; Sheller, M.J.; Edwards, B.; Pati, S.; Riviera, W.; Sharma, M.; Moorthy, P.N.; Wang, S.h.; Martin, J.; Mirhaji, P.; et al. OpenFL: the open federated learning library. *Physics in Medicine & Biology* **2022**, *67*, 214001.
77. Zeng, D.; Liang, S.; Hu, X.; Wang, H.; Xu, Z. Fedlab: A flexible federated learning framework. *Journal of Machine Learning Research* **2023**, *24*, 1–7.
78. Roth, H.R.; Cheng, Y.; Wen, Y.; Yang, I.; Xu, Z.; Hsieh, Y.T.; Kersten, K.; Harouni, A.; Zhao, C.; Lu, K.; et al. Nvidia flare: Federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291* **2022**.
79. Ludwig, H.; Baracaldo, N.; Thomas, G.; Zhou, Y.; Anwar, A.; Rajamoni, S.; Ong, Y.; Radhakrishnan, J.; Verma, A.; Sinn, M.; et al. Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987* **2020**.
80. Chai, X.; Zhang, M.; Tian, H. AI for Science: Practice from Baidu PaddlePaddle. In Proceedings of the 2024 Portland International Conference on Management of Engineering and Technology (PICMET). IEEE, 2024, pp. 1–12.
81. Ro, J.H.; Suresh, A.T.; Wu, K. FedJAX: Federated learning simulation with JAX. *arXiv preprint arXiv:2108.02117* **2021**.
82. Ryu, M.; Kim, Y.; Kim, K.; Madduri, R.K. APPFL: open-source software framework for privacy-preserving federated learning. In Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2022, pp. 1074–1083.

83. Karimireddy, S.P.; Kale, S.; Mohri, M.; Reddi, S.; Stich, S.; Suresh, A.T. Scaffold: Stochastic controlled averaging for federated learning. In Proceedings of the International conference on machine learning. PMLR, 2020, pp. 5132–5143.
84. Galtier, M.N.; Marini, C. Substra: a framework for privacy-preserving, traceable and collaborative machine learning. *arXiv preprint arXiv:1910.11567* **2019**.
85. Khimani, V.; Jabbari, S. TorchFL: A Performant Library for Bootstrapping Federated Learning Experiments. *arXiv preprint arXiv:2211.00735* **2022**.
86. Jin, W.; Yao, Y.; Han, S.; Gu, J.; Joe-Wong, C.; Ravi, S.; Avestimehr, S.; He, C. FedML-HE: An Efficient Homomorphic-Encryption-Based Privacy-Preserving Federated Learning System, 2024, [arXiv:cs.LG/2303.10837].
87. Arivazhagan, M.G.; Aggarwal, V.; Singh, A.K.; Choudhary, S. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818* **2019**.
88. Hsu, T.; Qi, H.; Brown, M. Measuring the Effects of Non-IID Data in Federated Learning. In Proceedings of the Proceedings of the 1st Workshop on Federated Learning for Data Privacy and Confidentiality, NeurIPS, 2019.
89. Li, T.; Hu, S.; Beirami, A.; Smith, V. Ditto: Fair and Robust Federated Learning Through Personalization, 2021, [arXiv:cs.LG/2012.04221].
90. He, K.; Yang, L.; Hong, J.; Jiang, J.; Wu, J.; Dong, X.; Liang, Z. PrivC – A Framework for Efficient Secure Two-Party Computation. In Proceedings of the Security and Privacy in Communication Networks (SecureComm 2019), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 305; Chen, S.; Choo, K.R.; Fu, X.; Lou, W.; Mohaisen, A., Eds., Orlando, FL, USA, 2019; pp. 394–407. https://doi.org/10.1007/978-3-030-37231-6_23.
91. Mohassel, P.; Rindal, P. ABY3: A Mixed Protocol Framework for Machine Learning. In Proceedings of the Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). ACM, 2018, pp. 35–52. https://doi.org/10.1145/3243734.3243760.
92. Dinh, C.T.; Tran, N.H.; Nguyen, T.D. Personalized Federated Learning with Moreau Envelopes. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Curran Associates, Inc., 2020, Vol. 33, pp. 21394–21405.
93. Collins, L.; Hassani, H.; Hassanpour, N. Exploiting Shared Representations for Personalized Federated Learning. In Proceedings of the International Conference on Machine Learning (ICML). PMLR, 2021, pp. 2089–2099.
94. Wang, Y.; Pan, Y.; Su, Z.; Deng, Y.; Zhao, Q.; Du, L.; Luan, T.H.; Kang, J.; Niyato, D. Large Model Based Agents: State-of-the-Art, Cooperation Paradigms, Security and Privacy, and Future Trends. *IEEE Communications Surveys & Tutorials* **2025**, pp. 1–1. https://doi.org/10.1109/COMST.2025.3576176.