MMT: ACHIEVING EXACT FEDERATED UNLEARNING WITH IMPROVED POST-UNLEARNING PERFORMANCE

Ze Yu Zhang[1,2*], Nhung Bui[1*], Arun Verma[3*], Bolin Ding[2], Bryan Kian Hsiang Low[1,3]
[1]Department of Computer Science, National University of Singapore, Singapore
[2]Alibaba Group, Singapore
[3]Singapore-MIT Alliance for Research and Technology Centre, Singapore
*Equal contribution

ABSTRACT

Federated learning (FL) enables multiple clients to collaboratively train a global model by exchanging model parameters or gradients without sharing their private data. However, these shared updates inherently embed the influence of clients' local datasets into the global model. In practical applications, it is often necessary to *quickly* remove or "unlearn" a specific client's influence from the global model to comply with data privacy regulations or to mitigate the impact of malicious ones while minimizing operational downtime and security vulnerabilities. To address these challenges, we propose two FL-agnostic algorithms called BMT and MMT, which ensure the complete removal of the client's influence with minimal delay. While BMT derives a new global model initialization by aggregating isolated pre-trained *local* models, MMT selectively aggregates *sub-FL* models trained across disjoint client subsets to better capture their cross-influence on the global model. We empirically show that both algorithms lead to improved post-unlearning performance across different data modalities and model architectures.

1 INTRODUCTION

It is well-established that the success of modern machine learning (ML) models is driven significantly by the availability of vast, heterogeneous datasets. For example, leading production systems, such as Google Search auto-completion (Shokouhi, 2013), Facebook ad ranking (He et al., 2014), and Instagram feed ordering (Vorotilov & Shugaepov, 2023), harness millions of user interactions to achieve state-of-the-art performance and robustness. However, centrally storing and managing these vast datasets entails substantial effort and, more importantly, rigorous stewardship of data privacy (GDPR, 2016; CCPA, 2018). *Federated learning* (FL) (McMahan et al., 2017; Li et al., 2019; Kairouz et al., 2021) addresses these challenges by enabling numerous clients, each acting as a data owner, to collaboratively train ML models without sharing their private data, bypassing the need for centralized datasets. Consequently, FL is particularly well-suited for privacy-sensitive applications, such as finance (Long et al., 2020; Byrd & Polychroniadou, 2020), healthcare (Brisimi et al., 2018; Rieke et al., 2020), and edge computing (Hard et al., 2018; Lim et al., 2020).

While clients do not share their private data with the global ML model in FL, they do share model parameters or gradients optimized locally on their own datasets (McMahan et al., 2017). These updates are aggregated into the global model over multiple rounds, embedding the influence of the clients' local datasets into the global model's parameters. Consequently, when a client requests the removal of their data, it is expected that their data's influence be removed from the global model to comply with "Right to be Forgotten" in modern data privacy regulations (GDPR, 2016; CCPA, 2018). Furthermore, there is often a need to remove the influence of malicious clients from the global model to mitigate potential security risks and preserve its integrity (Fang et al., 2020; Tolpegin et al., 2020). These challenges have spurred significant interest in *federated unlearning* (FU) (Liu et al., 2021; Wang et al., 2022; Liu et al., 2023), which focuses on enabling the global model to effectively remove or "unlearn" the data influence from a specific client.

Retraining from scratch (RFS) without the client-to-be-unlearned is a straightforward yet computationally prohibitive approach to achieve FU. Due to the extensive retraining time, RFS presents a

dilemma: awaiting unlearning may cause prolonged disruptions, particularly in time-sensitive applications like financial services (Liu & Ren, 2024) or supply chains (Wallace et al., 2003), leading to significant real-world losses. Conversely, continuing to use a model with residual influence risks data privacy violation, potential security vulnerabilities, or suboptimal performance if the client is malicious. This raises a question: ***How can we achieve exact FU (total removal of data influence) with minimal delay, i.e. the global model performance is quickly recovered post-unlearning?***

Previous works on exact FU often exhibit a trade-off between unlearning efficiency and global model performance, *e.g.,* by quantizing model parameters (Xiong et al., 2023) or clustering clients (Qiu et al., 2023). Additionally, some methods incur impractical storage overheads that scale linearly with the number of FL training rounds (Tao et al., 2024) (see Appendix A for a comprehensive review of related works). To address these issues, we propose simple yet effective algorithms called Bi-Model Training (BMT) and Multi-Model Training (MMT) (Section 3). Specifically, BMT retains isolated copies of *local* models and aggregates them to derive a new global model initialization. While this approach naturally leverages trained knowledge, it fails to capture the joint influence of multiple clients on the global model. Conversely, training multiple *sub-FL* models on the power set of clients is not only computationally expensive but also leads to substantial redundancy, as the influence of the client-to-be-unlearned is distributed across multiple models. To address these challenges, we propose MMT, which selectively aggregates sub-FL models trained across disjoint client subsets to better capture the cross-influence among clients on the global model. Our experiments show that BMT and MMT achieve exact FU with significantly lower delay than existing baselines across different data modalities and model architectures (Section 4). Notably, both algorithms can be seamlessly integrated with any standard FL protocol without modifying the underlying optimization.

2 Problem Setting

Federated Learning (FL). We consider a centralized **cross-silo** FL setting, where a central, trusted server coordinates training across distributed clients. We assume that a common hypothesis class $\mathcal{H}$ is shared by the server and the clients. The server owns a global model $h_\theta \in \mathcal{H}$, with parameters $\theta \in \Theta$. In each communication round (*round* for short), the server iteratively updates the global model using the shared model updates from a set of clients $\mathcal{C} := \{1, ..., N\}$, with N clients. Without a loss of generality, client c possesses a local, supervised dataset $\mathcal{D}_c \subset \mathcal{X} \times \mathcal{Y}$ defined over the input space $\mathcal{X}$ and the label space $\mathcal{Y}$, which is not shared with the central server or other clients $c' \neq c, c' \in \mathcal{C}$. We denote by $\mathcal{D} := \{\mathcal{D}_c\}_{c=1}^N$ a set of local datasets. A FL training algorithm $\mathcal{A} : \mathcal{C} \times \mathcal{D} \to \Theta$ generates global model parameters $\theta_t \in \Theta$. One of the popular goals for $\mathcal{A}$ is minimizing the empirical risk/loss across $\mathcal{C}$, i.e. $\arg\min_{\theta\in\Theta} \sum_{c=1}^N w_c f(\mathcal{D}_c; \theta)$, where $f(\mathcal{D}_c; \theta) := {}^1\!/_{|\mathcal{D}_c|} \sum_{(x,y)\in\mathcal{D}_c} \ell(h_\theta(x); y)$ is the empirical loss measured on the local dataset of client c, with the loss function ℓ, and weighted proportionally to its carnality $w_c := \frac{|\mathcal{D}_c|}{\sum_{c'=1}^N |\mathcal{D}_{c'}|}$.

Exact Federated Unlearning (Exact FU). Let $\mathcal{I}(h_\theta) \subseteq \mathcal{C}$ denote the subset of clients whose data have influenced the model h_θ through training or FL aggregation. In our framework, $\mathcal{I}(\cdot)$ is tracked exactly from model provenance: for each stored local model or sub-FL model trained using a recorded client subset S (see Section 3), we record $\mathcal{I}(h) = S$. We say that the global FL model h_θ has exactly unlearned a client c if $c \notin \mathcal{I}(h_\theta)$. Therefore, exact FU refers to the process of completely removing the influence of a client's training data from the global model.

3 Methodology

We now present our algorithm, Multi-Model Training (MMT), which can serve as a plug-and-play extension for any standard FL protocol. As we will explain later, MMT can be simplified to Bi-Model Training (BMT) under specific parameter configurations.

① **Initialization.** The central server starts the standard FL training process (see Appendix B) by randomly initializing the global model. This randomly initialized global model is then shared with all clients. Each client updates the global model using its local training data and then shares the model update (updated model or gradients) with the server. Compared to the standard initialization

in FL training process, each client makes a copy of the locally updated global model[1], i.e., *local model*. As the initial global model is randomly initialized, these local models are, by design, isolated from the influence of other clients' data. MMT then initializes the sub-FL models (see Definition 1 and Section D) using the model updates of clients corresponding to the sub-FL models.

② **FL training.** After receiving the model updates, the central server aggregates them to get the aggregated global and sub-FL models and then each client receives these updated global and its sub-FL models from the central server and then trains them using its training data. After updating these models, each client shares the global and its sub-FL model updates with the central server. Apart from this, each client also updates their local model. MMT maintains global, sub-FL, and local models, and their updates do not need to be synchronized with the training rounds of the global FL model. By eliminating overlapping influence (see Definition 2), MMT incurs at most a *logarithmic multiplicative overhead* in training cost relative to RFS. This is because sub-FL models at the same depth in the tree have disjoint datasets, so their combined training cost is roughly equivalent to training the global FL model (assuming same hyperparameters). Also, the depth of a balanced binary tree scales sub-linearly ($\lceil \log_2 n \rceil$ more generally $\lceil \log_b n \rceil$ for branching factor b) with the number of leaf nodes, provided the unlearning probability across clients remains approximately uniform. Unlike other exact FU algorithms (Xiong et al., 2023; Tao et al., 2024) with linear memory costs from storing model checkpoints, MMT does not require storing extra checkpoints during training.

③ **Unlearning.** To unlearn a client, the server first discards the current global and affected sub-FL models, including all associated optimizer/training state (e.g., momentum/Adam moments, scheduler state, batch-norm running statistics, and any server-side optimizer state), to ***ensure the complete removal of any residual influence from the unlearned client, thereby providing exact unlearning guarantees*** (see formal proof in supplementary material). The server then requests local model copies from clients not involved in any remaining sub-FL models. Once the server receives all requested local models, it aggregates them with the most influential unaffected sub-FL models (excluding affected descendants) to initialize the new global and sub-FL models, following the FL algorithm. After removing the affected sub-FL models, the remaining influence tree may no longer minimize Influence Degradation Score (see Definition 3) for the remaining clients. This leaves two options: (i) build a new influence tree while reusing existing sub-FL models as much as possible, or (ii) continue using the current tree, which may be suboptimal but retains previously trained sub-FL models. The server then restarts FL training with newly initialized global and sub-FL (if any) models, with freshly initialized optimizer/training state, fully free from the unlearned client's influence.

④ **New client.** Adding a new client during ongoing FL can degrade the current influence tree compared to rebuilding it with the new client from the start. When a client requests to join, the server waits for the current round to finish. Once complete, the server can either build a new influence tree reusing existing sub-FL models or continue with the current tree, preserving previously trained sub-FL models and adding new ones as needed. The central server begins FL training with the new client by sharing the current global and sub-FL models, which the client updates using its data and returns them to the central server. It also provides a randomly initialized global model that the client updates and adopts as its local model for future rounds.

BMT is a special case of MMT without sub-FL models. It is more resource-efficient but has lower post-unlearning performance, though still better than RFS. More details and illustrations related to MMT and BMT are provided in supplementary material.

4 EXPERIMENTS

Datasets & Models. We conduct experiments across four vision tasks: MNIST (LeCun et al., 1998) and FashionMNIST (FMNIST) (Xiao et al., 2017) using MLPs with 30 and 80 hidden units, respectively; CIFAR-10 (Krizhevsky et al., 2009) using a 2-layer CNN network with 5×5 convolutional layers, 2×2 max pooling, two fully connected layers (32 hidden units) and ReLU activation; and CIFAR-100 (Krizhevsky et al., 2009) using VGG-16 model (Simonyan, 2014). We also finetune a pretrained GPT-2 (Radford et al., 2019) and Llama-3.2-3B model (Dubey et al., 2024) for next-token prediction on two text datasets: language identification (LI) (Papluca, 2021) and multilingual sentiment analysis (MSA) (Qiang, 2023). More experimental details are given in Appendix G.

[1]The locally updated global model in the first communication rounds is the equivalent to the initial global model trained on only a single client's training data. The different FL stages of MMT are illustrated in Fig. 3.

Baselines. We benchmark BMT and MMT against the following exact FU baselines: (1) **Retraining from Scratch** (RFS): FL training is restarted using only the remaining clients; (2) **FedCIO** (Qiu et al., 2023); (3) **Exact-Fun** (Xiong et al., 2023); and (4) **FATS** (Tao et al., 2024).

Main Results: Global Model Performance in Sequential Unlearning. Fig. 1 shows the test accuracy of the global model after unlearning $1-2$ clients sequentially, with each unlearning request marked by ▲. Both methods consistently outperform other baselines and even surpass RFS by a large margin on CIFAR-10. Among the two methods, MMT often achieves faster convergence than BMT post-unlearning. Additionally, Tab. 1 shows that both methods significantly reduce the number of communication rounds to reach the predefined accuracy threshold after unlearning compared to RFS across all tasks. These results highlight the scalability and effectiveness of our approaches across different modalities, model architectures, and model sizes.

We provide results for continual learning-unlearning simulation and ablation studies (*e.g.*, effect of branching factors, clients' unlearning probabilities, label imbalance ratio) in Appendix G.

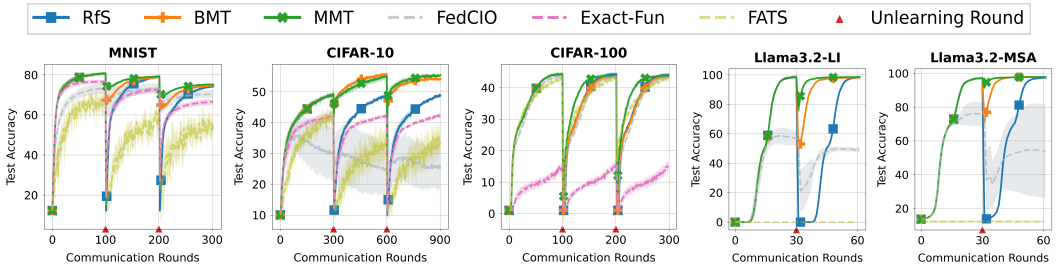


Figure 1: Test accuracy of the global model in sequential unlearning. Results are averaged over 5 random seeds, with standard deviations shown as shaded regions. Due to space constraints, full results with evaluation for FMNIST, GPT-2 LI, and GPT-2 MSA are provided in Appendix G.1.

Table 1: Number of communication rounds to reach predefined accuracy thresholds post-unlearning. (Δ) shows the difference relative to RFS. We note that **FedCIO, FATS, and Exact-Fun are omitted here as they fail to meet the accuracy thresholds across all settings**.

Dataset	Acc. (%)	RFS	BMT (Δ)	MMT (Δ)
MNIST	75	51	34 (-17)	**16** (-35)
FMNIST	65	26	16 (-10)	**3** (-23)
CIFAR-10	45	162	**3** (-159)	**3** (-159)
CIFAR-100	40	54	54 (+0)	**37** (-17)
GPT-2 LI	70	16	11 (-5)	**6** (-10)
GPT-2 MSA	65	49	22 (-27)	**7** (-42)
Llama-3.2 LI	85	21	6 (-15)	**2** (-19)
Llama-3.2 MSA	85	19	4 (-15)	**1** (-18)

Table 2: MIA AUC on the global model. BMT and MMT achieve comparable results to RFS across all settings, indicating the influence of unlearned clients has been effectively removed from the global model.

Dataset	No Unl.	RFS	BMT	MMT
MNIST	0.609	0.520	0.522	0.532
FMNIST	0.508	0.492	0.490	0.490
CIFAR-10	0.489	0.496	0.485	0.489
CIFAR-100	0.911	0.521	0.542	0.517
GPT-2 LI	0.724	0.632	0.618	0.632
GPT-2 MSA	0.715	0.480	0.523	0.527
Llama-3.2 LI	0.553	0.612	0.612	0.612
Llama-3.2 MSA	0.639	0.485	0.485	0.485

MIA results. Tab. 2 reports the MIA AUC on the global model *before* (No Unlearning) and *after* unlearning. Both methods often successfully reduce the MIA success rate post-unlearning and closely match the gold-standard RFS performance. These results highlight that our approaches have effectively removed the influence of unlearned clients from the global model.

5 Conclusion

This work proposes two algorithms, BMT and MMT, for exact federated unlearning. Our algorithms ensure the complete removal of an unlearned client's data while having better performance post-unlearning with the remaining clients than retraining from scratch. Our algorithms are particularly useful in practical scenarios where model updation in collaborative environments cannot afford long delays, with minimal tolerance for interruptions. Our extensive experimental results demonstrate the effectiveness of the proposed algorithms. A few promising directions for future work include developing principled methods for designing influence trees under resource constraints (e.g., a limited number of trainable sub-FL models), and exploring how to adapt the tree, after unlearning or client addition, to minimize IDS while maximizing reuse of existing sub-FL models.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. The Mens, Manus, and Machina (M3S) is an interdisciplinary research group (IRG) of the Singapore MIT Alliance for Research and Technology (SMART) centre.

REFERENCES

Alireza Aghabagherloo, Aydin Abadi, Sumanta Sarkar, Vishnu Asutosh Dasu, and Bart Preneel. Impact of data duplication on deep neural network-based image classifiers: Robust vs. standard models. In *Proc. IEEE SPW*, pp. 177–183, 2025.

Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv:1511.06481*, 2015.

Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. Coded machine unlearning. *IEEE Access*, pp. 88137–88150, 2021.

Miltiadis Allamanis. The adverse effects of code duplication in machine learning models of code. In *Proc. ACM SIGPLAN Onward!*, pp. 143–153, 2019.

Sara Babakniya, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy, and Salman Avestimehr. SLoRA: Federated parameter efficient fine-tuning of language models. In *Proc. NIPS Workshop on Federated Learning in the Age of Foundation Models*, 2023.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *Proc. IEEE S&P*, pp. 141–159, 2021.

Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *Int. J. Med. Inform.*, pp. 59–67, 2018.

Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *Proc. ICML*, pp. 1092–1104, 2021.

Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

Nhung Bui, Xinyang Lu, Rachael Hwee Ling Sim, See-Kiong Ng, and Bryan Kian Hsiang Low. On newton's method to unlearn neural networks. *arXiv:2406.14507*, 2024.

David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. In *Proc. ICAIF*, pp. 1–9, 2020.

Jonathon Byrd and Zachary Lipton. What is the effect of importance weighting in deep learning? In *Proc. ICML*, pp. 872–881, 2019.

Xiaoyu Cao, Jinyuan Jia, Zaixi Zhang, and Neil Zhenqiang Gong. FedRecover: Recovering from poisoning attacks in federated learning using historical information. In *Proc. IEEE S&P*, pp. 1366–1383, 2023.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *Proc. IEEE S&P*, pp. 463–480, 2015.

Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *Proc. NeurIPS*, pp. 409–415, 2000.

CCPA. California consumer privacy act of 2018, 2018. California Civil Code Title 1.81.5.

Vikram S. Chundawat, Ayush K. Tarun, Murari Mandal, and Mohan Kankanhalli. Can bad teaching induce forgetting? Unlearning in deep networks using an incompetent teacher. In *Proc. AAAI*, pp. 7210–7217, 2023.

Corinna Cortes, Yishay Mansour, and Mehryar Mohri. Learning bounds for importance weighting. In *Proc. NIPS*, pp. 442–450, 2010.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, USA, 2nd edition, 2006. ISBN: 978-0-471-24195-9.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Proc. NIPS*, pp. 10088–10115, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. Technical report, Meta, 2024.

Chongyu Fan, Jiancheng Liu, Yihua Zhang, Eric Wong, Dennis Wei, and Sijia Liu. SalUn: Empowering machine unlearning via gradient-based weight saliency in both image classification and generation. In *Proc. ICLR*, 2024.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *Proc. USENIX Security*, pp. 1605–1622, 2020.

Jack Foster, Stefan Schoepf, and Alexandra Brintrup. Fast machine unlearning without retraining through selective synaptic dampening. In *Proc. AAAI*, pp. 12043–12051, 2024.

Xiangshan Gao, Xingjun Ma, Jingyi Wang, Youcheng Sun, Bo Li, Shouling Ji, Peng Cheng, and Jiming Chen. VeriFi: Towards verifiable federated unlearning. *IEEE TDSC*, 2024.

GDPR. General Data Protection Regulation, Article 17: Right to erasure ('right to be forgotten'). *OJEU*, 2016. Regulation (EU) 2016/679.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proc. CVPR*, pp. 9304–9312, 2020.

Jinu Gong, Osvaldo Simeone, and Joonhyuk Kang. Bayesian variational federated learning and unlearning in decentralized networks. In *Proc. SPAWC*, pp. 216–220, 2021.

Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. In *Proc. AAAI*, pp. 11516–11524, 2021.

Hanlin Gu, Gongxi Zhu, Jie Zhang, Xinyuan Zhao, Yuxing Han, Lixin Fan, and Qiang Yang. Unlearning during learning: An efficient federated machine unlearning method. In *Proc. IJCAI*, 2024.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In *Proc. ICML*, pp. 3832–3842, 2020.

Anisa Halimi, Swanand Kadhe, Ambrish Rawat, and Nathalie Baracaldo. Federated unlearning: How to efficiently erase a client in FL? *arXiv:2207.05521*, 2022.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv:1811.03604*, 2018.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proc. ACM AdKDD*, pp. 1–9, 2014.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *Proc. ICLR*, 2022.

David A Huffman. A method for the construction of minimum-redundancy codes. *IRE*, pp. 1098–1101, 1952.

Jinghan Jia, Yihua Zhang, Yimeng Zhang, Jiancheng Liu, Bharat Runwal, James Diffenderfer, Bhavya Kailkhura, and Sijia Liu. SOUL: Unlocking the power of second-order optimization for LLM unlearning. In *Proc. EMNLP*, pp. 4276–4292, 2024.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14:1–210, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. In *Proc. NeurIPS*, pp. 1957–1987, 2023.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, pp. 2278–2324, 1998.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *arXiv:1908.07873*, 2019.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proc. MLSys*, pp. 429–450, 2020.

Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE COMST*, 22:2031–2063, 2020.

Gaoyang Liu, Xiaoqiang Ma, Yang Yang, Chen Wang, and Jiangchuan Liu. FedEraser: Enabling efficient client-level data removal from federated learning models. In *Proc. IWQOS*, pp. 1–10, 2021.

Na Liu and Shuyun Ren. Production disruption in supply chain systems: impacts on consumers, supply chain agents and the society. *Ann. Oper. Res.*, pp. 1–24, 2024.

Ziyao Liu, Yu Jiang, Jiyuan Shen, Minyi Peng, Kwok-Yan Lam, Xingliang Yuan, and Xiaoning Liu. A survey on federated unlearning: Challenges, methods, and future directions. *ACM Computing Surveys*, 2023.

Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated learning: privacy and incentive*, pp. 240–254. 2020. ISBN: 978-3-030-63076-8.

Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schölkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership inference attacks against language models via neighbourhood comparison. In *ACL Findings*, pp. 11330–11343, 2023.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, pp. 1273–1282, 2017.

Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Variational bayesian unlearning. In *Proc. NeurIPS*, pp. 16025–16036, 2020.

Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv:2209.02299*, 2022.

Zibin Pan, Zhichao Wang, Chi Li, Kaiyan Zheng, Boqi Wang, Xiaoying Tang, and Junhua Zhao. Federated unlearning with gradient descent and conflict mitigation. In *Proc. AAAI*, pp. 19804–19812, 2025.

Papluca. Language identification dataset. `https://huggingface.co/datasets/papluca/language-identification`, 2021. Accessed 2 Aug 2025.

Martin Pawelczyk, Seth Neel, and Himabindu Lakkaraju. In-context unlearning: Language models as few shot unlearners. In *Proc. ICML*, pp. 40034–40050, 2024.

Tianyi Qiang. Multilingual sentiment analysis dataset. https://huggingface.co/datasets/tyqiangz/multilingual-sentiments, 2023. Accessed 2 Aug 2025.

Hongyu Qiu, Yongwei Wang, Yonghui Xu, Lizhen Cui, and Zhiqi Shen. FedCIO: Efficient exact federated unlearning with clustering, isolation, and one-shot aggregation. In *Proc. IEEE BigData*, pp. 5559–5568, 2023.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI Blog, 2019.

Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *npj Digital Medicine*, pp. 119, 2020.

Sebastian Schelter. Amnesia – Towards machine learning models that can forget user data very fast. In *The 1st International Workshop on Applied AI for Database Systems and Applications (AIDB19)*, 2019.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, New York, USA, 2014. ISBN: 978-1-107-05713-5.

Nir Shlezinger, Mingzhe Chen, Yonina C Eldar, H Vincent Poor, and Shuguang Cui. Uveqfed: Universal vector quantization for federated learning. *IEEE Trans. Signal Process*, pp. 500–514, 2020.

Milad Shokouhi. Learning to personalize query auto-completion. In *Proc. ACM SIGIR*, pp. 103–112, 2013.

Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Youming Tao, Cheng-Long Wang, Miao Pan, Dongxiao Yu, Xiuzhen Cheng, and Di Wang. Communication efficient and provable federated unlearning. *VLDB Endowment*, 17(5):1119–1131, 2024.

Ayush K. Tarun, Vikram S. Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *IEEE TNNLS*, pp. 13046–13055, 2023.

Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *Proc. ESORICS*, pp. 480–501, 2020.

Vladislav Vorotilov and Ilnur Shugaepov. Scaling the instagram explore recommendations system. Meta Blog, 2023.

William A Wallace, David Mendonca, EE Lee, John E Mitchell, and J Chow. Managing disruptions to critical interdependent infrastructures in the context of the 2001 world trade center attack. *Beyond September 11th: An account of post-disaster research*, pp. 165–198, 2003.

Junxiao Wang, Song Guo, Xin Xie, and Heng Qi. Federated unlearning via class-discriminative pruning. In *Proc. WWW*, pp. 622–632, 2022.

Ziyao Wang, Zheyu Shen, Yexiao He, Guoheng Sun, Hongyi Wang, Lingjuan Lyu, and Ang Li. FLoRA: Federated fine-tuning large language models with heterogeneous low-rank adaptations. In *Proc. NIPS*, pp. 22513–22533, 2024.

Leijie Wu, Song Guo, Junxiao Wang, Zicong Hong, Jie Zhang, and Jingren Zhou. On knowledge editing in federated learning: Perspectives, challenges, and future directions. *arXiv:2306.01431*, 2023.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

Zuobin Xiong, Wei Li, Yingshu Li, and Zhipeng Cai. Exact-fun: An exact and efficient federated unlearning approach. In *Proc. IEEE ICDM*, pp. 1439–1444, 2023.

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. ARCANE: An efficient architecture for exact machine unlearning. In *Proc. IJCAI*, pp. 4006–4013, 2022.

Liping Yi, Han Yu, Gang Wang, Xiaoguang Liu, and Xiaoxiao Li. pFedLoRA: Model-heterogeneous personalized federated learning with lora tuning. *arXiv:2310.13283*, 2023.

Wei Yuan, Hongzhi Yin, Fangzhao Wu, Shijie Zhang, Tieke He, and Hao Wang. Federated unlearning for on-device recommendation. In *Proc. ACM WSDM*, pp. 393–401, 2023.

Sajjad Zarifzadeh, Philippe Liu, and Reza Shokri. Low-cost high-power membership inference attacks. In *Proc. ICML*, pp. 58244–58282, 2024.

Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, pp. 106775, 2021.

A Related Works

Federated Unlearning. The subfield of *federated unlearning* (FU) (Liu et al., 2021; Wang et al., 2022; Liu et al., 2023) tackles unlearning given data decentralization and limited access to the unlearned client's data, which contrasts to common centralized data assumptions. FU algorithms are classified as *active* if the unlearned client is involved during unlearning, or *passive* otherwise (Liu et al., 2023). The unlearned client in active FU algorithms permits access to its data, which enables *partial data removal* (Gong et al., 2021; Wang et al., 2022; Halimi et al., 2022; Xiong et al., 2023; Tao et al., 2024). In contrast, passive unlearning algorithms are executed by the FL server, potentially aided by the remaining clients, and mostly target *client removal* (Liu et al., 2021; Qiu et al., 2023; Cao et al., 2023; Yuan et al., 2023; Gao et al., 2024; Tao et al., 2024). Since the unlearned client's data is inaccessible, passive unlearning algorithms often rely on stored information (e.g., gradients or global models) and exhibit inherent computational trade-offs. Additionally, FU algorithms can be classified as *approximate* or *exact* (Wu et al., 2023). Despite being prominent, approximate FU algorithms (e.g., Liu et al. (2021); Gong et al. (2021); Halimi et al. (2022); Cao et al. (2023); Yuan et al. (2023)) may retain residual influence from the unlearned (malicious) client and violate data privacy regulations.

Exact Federated Unlearning. FU algorithms with exact unlearning guarantees are underexplored (Xiong et al., 2023; Qiu et al., 2023; Tao et al., 2024). For example, Exact-Fun (Xiong et al., 2023) and FATS (Tao et al., 2024) use quantization and sampling, respectively, to efficiently restart FL training from a stored checkpoint where the influence of leaving the client is negligible. FedCIO (Qiu et al., 2023) adopts a client-clustering algorithm based on data distribution and restricts FL retraining to only a model subset trained on the unlearned client. However, these algorithms require significant computational resources to store intermediate checkpoints and degrade the global model performance with large quantization and the number of clusters. Moreover, in the worst case, these algorithms are slow in FL retraining, leading to undesirable delays in deploying the unlearned global model. Our proposed algorithms, BMT and MMT, address these limitations.

Machine Unlearning. The field of *machine unlearning* (Cao & Yang, 2015; Bourtoule et al., 2021; Nguyen et al., 2022) focuses on selectively removing the influence of specific data from a trained ML model without retraining from scratch. Existing machine unlearning methods are broadly classified into two categories: *exact unlearning* and *approximate unlearning*. Exact unlearning methods can produce a model identical to a retrained model and are often justifiable through its algorithmic designs. Unfortunately, efficient exact unlearning methods are only known for a few model classes (Cauwenberghs & Poggio, 2000; Schelter, 2019; Brophy & Lowd, 2021; Cao & Yang, 2015); while model-agnostic methods are mostly ensemble-based approaches (Bourtoule et al., 2021; Aldaghri et al., 2021; Yan et al., 2022), which can get computationally inefficient with large ensembles. By relaxing its goal, approximate unlearning methods seek a model with similar behavior as a retrained model and sometimes can be theoretically justifiable (Guo et al., 2020; Golatkar et al., 2020; Nguyen et al., 2020). Many empirical approximate unlearning methods are derived from gradient ascent and its variants (Graves et al., 2021; Tarun et al., 2023), knowledge distillation (Chundawat et al., 2023; Kurmanji et al., 2023), optimization methods (Jia et al., 2024; Bui et al., 2024), localization techniques (Fan et al., 2024; Foster et al., 2024), and in-context learning (Pawelczyk et al., 2024). Most machine unlearning methods assume centralized data availability (e.g., (Cao & Yang, 2015; Guo et al., 2020; Bourtoule et al., 2021)).

Comparison with Existing Exact FU Methods. Recent exact federated-unlearning algorithms such as Exact-Fun (Xiong et al., 2023), FATS (Tao et al., 2024), and FedCIO (Qiu et al., 2023) follow the same checkpoint reuse blueprint. Each method

- stores global model checkpoints during normal federated training;
- locates a checkpoint whose computation graph excludes the departing client; and
- resumes training on the remaining clients so that the final model matches retraining from scratch in distribution.

This shared strategy delivers provable exactness because the cached checkpoint acts as a clean initialization after the unlearning request. Despite this common foundation, the three methods face distinct practical limitations:

- **Checkpoint explosion.** Exact-Fun and FATS save a full global model every round, so the server's memory grows linearly with the number of rounds R ($\mathcal{O}(R \times |\theta|)$).
- **Quantization-induced accuracy loss.** Exact-Fun relies on weight quantization to satisfy its probabilistic guarantee; tighter guarantees require more aggressive quantization and thus lower accuracy.
- **Cluster sensitivity.** FedCIO first partitions clients by data similarity. If the departing client lies in a large or heterogeneous cluster, the method degenerates to full retraining.
- **Slow post-unlearning recovery.** All three baselines still need many additional rounds before the unlearned model regains pre-unlearning accuracy, an unacceptable delay for latency-critical services such as finance or medical triage, supply-chain monitoring, etc.).

B STANDARD FL PROTOCOL

We denote the global model parameters at the t-th round by $\theta^{(t)}$. The server broadcasts $\theta^{(t)}$ (with $\theta^{(0)}$ often being randomly initialized) to all clients. Then, client c updates the copy of $\theta^{(t)}$ via empirical risk minimization on their local dataset $\mathcal{D}_c$ and sends the model update $\Delta\theta_c^{(t)}$ back to the server. We refer to this client-level training process as *local training*. The server aggregates the updates from all clients in $\mathcal{C}$ and updates the global model, e.g., if $\Delta\theta_c^{(t)}$ is the parameter difference resp. to $\theta^{(t)}$, FedAvg uses a weighted average aggregation method, i.e. $\theta^{(t+1)} = \theta^{(t)} + \sum_{c=1}^{N} w_c \Delta\theta_c^{(t)}$. The server repeats the process until $h_{\theta^{(t+1)}}$ converges or a stopping criterion (e.g., T rounds) is met. The parameters of a fully trained global model at the end of the FL training process are denoted by θ^*.

C BI-MODEL TRAINING (BMT)

To have a better global model after unlearning, we must design a new FL training process that allows exact federated unlearning while having a better initialization than random initialization. One way to achieve better initialization is to design methods that can exploit the remaining clients' existing knowledge. To do this, we propose a method named Bi-Model Training (BMT) that can be incorporated into any existing federated learning framework. The main idea of BMT is to have an additional model for each client that is only trained on its data, making these models unaffected by other clients' training data. We refer to this model as *local model*. We use the term *global model* for referring to the global model, which is trained using all client's data and used for deployment. We now discuss how BMT can be incorporated into the different stages of any existing federated learning framework, (as depicted in Fig. 2), namely: Initialization, FL Training, Unlearning, and New Client joining the FL process, whose details are given as follows.

① **Initialization.** The central server starts the standard FL training process (described in Section 2) by randomly initializing the global global model. This randomly initialized global model is then shared with all clients. Each client updates the global model using its local training data and then shares the model update (updated model or gradients) with the central server. As compared to the standard initialization in any FL training process, each client makes a copy of the locally updated global model[2] (i.e., local model). As the initial global model is randomly initialized, these local models are, by design, isolated from the influence of other clients' training data.

② **FL training.** After receiving the first model updates, the central server aggregates them to get the aggregated global model as per the underlying FL algorithm (McMahan et al., 2017; Shlezinger et al., 2020; Zhang et al., 2021). In each subsequent communication round, each client receives the updated global model from the central server and then trains it using its training data. After updating the global model, each client shares the model update with the central server. Besides the standard FL training process, each client also updates their local model using their training data.

③ **Unlearning.** Let c be the client whose influence needs to be completely removed from the global model after a communication round t and $\mathcal{C}_{t,r}$ be the set of remaining clients, i.e., $\mathcal{C}_{t,r} = \mathcal{C}_t \setminus \{c\}$. The central server first discards the current global model and requests each client to share their current copy of local models. Once the central server receives the local models from

[2]The locally updated global model in the first communication rounds is the same as the model that is a copy of the initial global model and trained on client's training data.

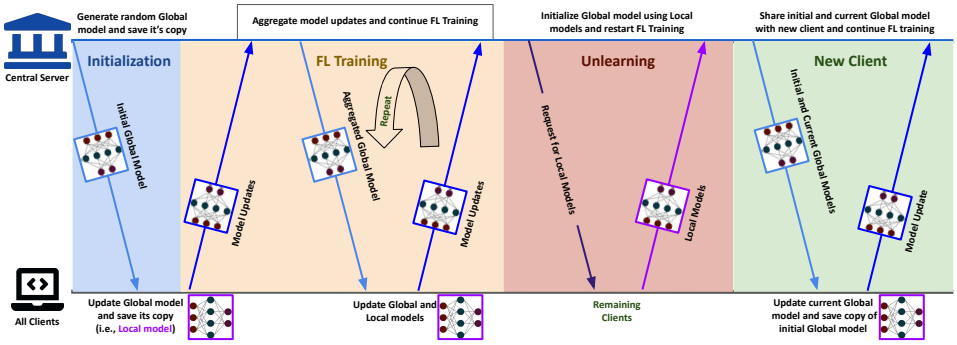


Figure 2: Bi-Models Training (BMT) in the different stages of any federated learning framework.

all remaining clients, the central server aggregates them to get the new initialization for the global model as per the underlying FL algorithm, e.g., for FedAvg, the central server performs weighted aggregation on the remaining client's local models, where each client's weight is proportional to their respective training data. Our experimental results (in Section 4) show that the resulting initialized global model performs better than random model initialization as done in RFS. Lastly, the central server restarts the FL training process with newly initialized global model, which is completely free from the influence of the unlearned client's data.

④ **New client.** When a new client wants to join the ongoing FL collaboration, the central server waits until the end of the ongoing communication round. Once it is over, the central server starts the FL training process with the new client by sharing the current global model with the new client, who then updates the current global model using its training data and shares the model update with the central server. Apart from this, the central client also shares the randomly initialized global model with the new client, who updates it, which then acts as the local model of the new client for subsequent rounds. Other clients do not influence this local model, as the initial global model is randomly initialized.

C.1 Pseudo-codes of BMT

BMT has two models for each client: global and local. All clients train their local model on their data in isolation, whereas the global model is trained using the underlying FL training protocol. To completely remove a client influence from the global model, the central server first discards the global model and then uses the local models of the remaining clients to re-initialize the global model, which is further updated via FL training. This process ensures that BMT, by design, guarantees the exact federated unlearning. Further, using the remaining clients' local models leads to an initialization of the global model that is already influenced by the remaining clients to some extent, leading to a better performance than RFS, as corroborated by our experiments in Section 4. The trade-off for improved post-unlearning performance is the cost of pre-training local models. This trade-off is worthwhile for applications that require exact unlearning and fast deployment of good model.

D Illustrations and Pseudo-codes of MMT

As illustrated in Fig. 3, MMT can be easily incorporated into any existing FL framework.

Illustration of changes in influence tree after unlearning. For unlearning a client, the central server first discards the current global model and related sub-FL models (as shown in Fig. 4b after unlearning client 2) and then requests all clients not in any of the remaining sub-FL models to share their current copy of local models. Once the central server receives all requested local models, it aggregates them with sub-FL models (choosing only the most influential unaffected sub-FL model over its descendants) to get the new initialization for the global model as per the underlying FL algorithm. After removing the sub-FL models related to the unlearned client, the remaining influence tree may no longer have the lowest IDS for the remaining clients. It leads to two options: create a new influence tree while using earlier sub-FL models as much as possible (as shown in Fig. 4c) or keep using the existing influence tree, which may not be the best but retains the sub-FL models trained over time. Lastly, the central server restarts the FL training process with the newly initial-

BMT Training Bi-Models Training - Training Phase

Input: FL algorithm $\mathfrak{A}$, number of communication rounds T

***** **Server** *****
Randomly initialize global model θ_1
Set communication round $t = 1$
while $t \leq T$ **do**
    Select clients C_t
    for each client i in C_t **do**
        Send global model θ_t to client i
    end for
    Wait for model updates $\{\Delta_i\}_{i \in C_t}$ from clients
    Aggregate updates to obtain new global model:
    $\theta_{t+1} = \mathfrak{A}(\theta_t, \{\Delta_i\}_{i \in C_t})$
    $t \leftarrow t + 1$
end while

***** **Clients** *****
Receive initial global model θ_1 from server
Initialize local model $\theta_i^{\text{local}} \leftarrow \theta_1$
for each communication round t **do**
    Receive global model θ_t from server
    Update global model using own data:
    $\theta_{i,t} \leftarrow \text{Update}(\theta_t, \text{data}_i)$
    Update local model using own data:
    $\theta_i^{\text{local}} \leftarrow \text{Update}(\theta_i^{\text{local}}, \text{data}_i)$
    Compute model update $\Delta_i = \theta_{i,t} - \theta_t$
    Send Δ_i to server
end for

BMT Unlearning Bi-Models Training - Unlearning Phase

Input: Unlearning client c, current communication round $t \leq T$

***** **Server** *****
Discard current global model θ_t
Set remaining clients $C_{t,r} = C_t \setminus \{c\}$
for each client i in $C_{t,r}$ **do**
    Request local model θ_i^{local}
end for
Receive local models $\{\theta_i^{\text{local}}\}_{i \in C_{t,r}}$
Aggregate local models to initialize new global model:
$\theta_{t+1} = \mathfrak{A}_{\text{init}}(\{\theta_i^{\text{local}}\}_{i \in C_{t,r}})$
Restart FL training from θ_{t+1} with $t \leftarrow t + 1$

***** **Clients** *****
if Client i receives request for local model **then**
    Send local model θ_i^{local} to server
end if
Restart FL training from θ_{t+1} with $t \leftarrow t + 1$

ized global and sub-FL (if any) models, (as shown in Fig. 4d), which are completely free from the influence of the unlearned client's data.

BMT New Client Bi-Models Training - Adding a New Client

Input: New client $n+1$

***** **Server** *****
if Training is ongoing **then**
    Wait until current round completes
    Send global model θ_t to new client $n+1$
    Include new client in client set C_{t+1}
else
    Start new FL process including new client
    Initialize global model θ with latest parameters θ_t
end if

***** **New Client** $n+1$ *****
Receive global model θ from server
Initialize local model $\theta_{n+1}^{\text{local}} \leftarrow$ Randomly initialized global model θ_1
for each subsequent communication round t **do**
    Receive global model θ_t from server
    Update global model using own data:
        $\theta_{n+1,t} \leftarrow \text{Update}(\theta_t, \text{data}_{n+1})$
    Update local model using own data:
        $\theta_{n+1}^{\text{local}} \leftarrow \text{Update}(\theta_{n+1}^{\text{local}}, \text{data}_{n+1})$
    Compute model update $\Delta_{n+1} = \theta_{n+1,t} - \theta_t$
    Send Δ_{n+1} to server
end for

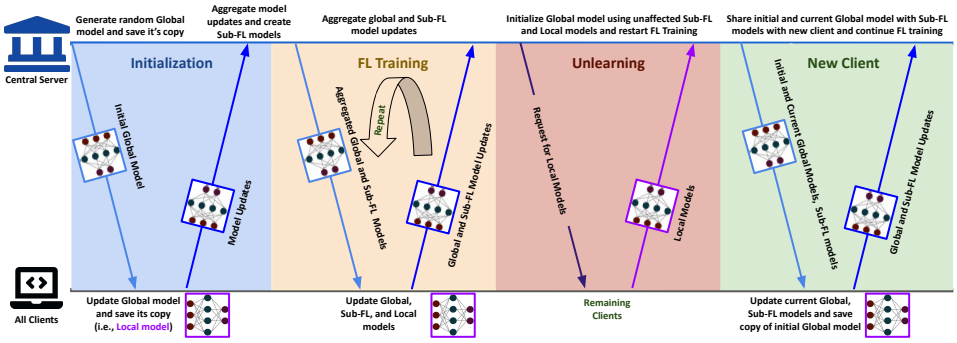
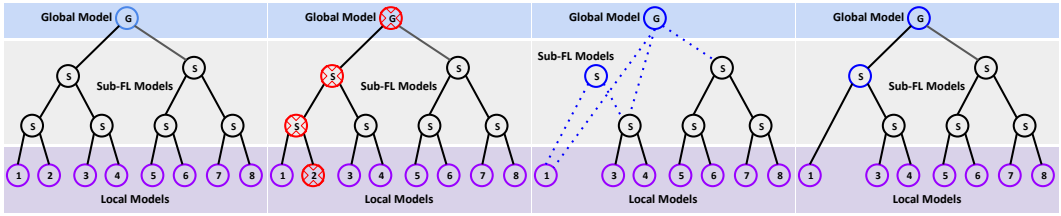


Figure 3: Multi-Model Training (MMT) in the different stages of any federated learning algorithm. Global and sub-FL training proceed in parallel but may not be synchronized across rounds.



(a) Influence tree (b) Affected nodes (c) Initialization (d) New influence tree

Figure 4: **4a:** Influence Tree for 8 clients having the same unlearning probability. **4b:** Showing the global and all sub-FL models affected after unlearning client 2 by the node's red cross and red outline. **4c:** Initialization of global and new sub-FL models, where a dotted blue line shows the models used to initialize them. **4d:** Final influence tree after unlearning the client.

D.1 PSEUDO-CODES OF MMT

MMT Training Multi-Models Training - Training Phase

Input: FL algorithm $\mathfrak{A}$, rounds T, influence tree $\mathcal{T}$

***** **Server** *****
Randomly initialize global model θ_1
Set round $t \leftarrow 1$
Initialization:
Send θ_1 to all clients
Receive updates $\{\Delta_i^{\text{global}}\}$ from clients
Update global model: $\theta_2 \leftarrow \mathfrak{A}(\theta_1, \{\Delta_i^{\text{global}}\})$
Initialize sub-FL models $\{\theta_s^{(1)}\}$ using $\{\Delta_i^{\text{global}}\}$
$t \leftarrow 2$
Training:
while $t \leq T$ **do**
    For each client i in C_t:
        Send θ_t, $\{\theta_s^{(t)}\}_{s \in \mathcal{S}_i}$
    Receive $\{\Delta_i^{\text{global}}, \{\Delta_i^{(s)}\}\}$ from clients
    Update global model: $\theta_{t+1} \leftarrow \mathfrak{A}(\theta_t, \{\Delta_i^{\text{global}}\})$
    for each sub-FL model s **do**
        Update sub-FL model: $\theta_s^{(t+1)} \leftarrow \mathfrak{A}(\theta_s^{(t)}, \{\Delta_i^{(s)}\}_{i \in S_s})$
    end for
    $t \leftarrow t + 1$
end while

***** **Clients** *****
Receive θ_1 from server
Update global model: $\theta_{i,1}^{\text{global}} \leftarrow \text{Update}(\theta_1, \text{data}_i)$
Initialize local model: $\theta_i^{\text{local}} \leftarrow \theta_{i,1}^{\text{global}}$
Compute update: $\Delta_i^{\text{global}} \leftarrow \theta_{i,1}^{\text{global}} - \theta_1$
Send Δ_i^{global} to server
for each round $t \geq 2$ **do**
    Receive θ_t, $\{\theta_s^{(t)}\}_{s \in \mathcal{S}_i}$ from server
    Update global model: $\theta_{i,t}^{\text{global}} \leftarrow \text{Update}(\theta_t, \text{data}_i)$
    Update local model: $\theta_i^{\text{local}} \leftarrow \text{Update}(\theta_i^{\text{local}}, \text{data}_i)$
    for each $s \in \mathcal{S}_i$ **do**
        Update sub-FL model: $\theta_{i,t}^{(s)} \leftarrow \text{Update}(\theta_s^{(t)}, \text{data}_i)$
    end for
    Compute updates:
        $\Delta_i^{\text{global}} \leftarrow \theta_{i,t}^{\text{global}} - \theta_t$
        $\Delta_i^{(s)} \leftarrow \theta_{i,t}^{(s)} - \theta_s^{(t)}$ for $s \in \mathcal{S}_i$
    Send Δ_i^{global}, $\{\Delta_i^{(s)}\}$ to server
end for

MMT Unlearning Multi-Models Training - Unlearning Phase

Input: Unlearning client c, current communication round t

***** **Server** *****
Discard current global model θ_t and sub-FL models associated with client c
Identify unaffected sub-FL models $\mathcal{S}_{\text{remain}}$ and their client sets S_s
Determine clients not in any remaining sub-FL models:
    $C_{t,r} = C_t \setminus \left(\{c\} \cup \bigcup_{s \in \mathcal{S}_{\text{remain}}} S_s\right)$
Request local models θ_i^{local} from clients in $C_{t,r}$
Receive local models $\{\theta_i^{\text{local}}\}_{i \in C_{t,r}}$
Aggregate local models and unaffected sub-FL models to initialize new global model:
    Choose the most influential unaffected sub-FL models over descendants
    $\theta_{t+1} = \mathfrak{A}_{\text{init}}\left(\{\theta_i^{\text{local}}\}_{i \in C_{t,r}}, \{\theta_s^{(t)}\}_{s \in \mathcal{S}_{\text{unaffected}}}\right)$
Decide to create a new influence tree or retain existing one
Restart FL training from θ_{t+1} with $t \leftarrow t+1$

***** **Clients** *****
if Client i receives request for local model **then**
    Send local model θ_i^{local} to server
end if
Restart FL training with updated models received from server

MMT New Client Multi-Models Training - Adding a New Client

Input: New client $n+1$

***** **Server** *****
if Training is ongoing **then**
    Wait until current round completes
    Decide whether to create new influence tree or update existing one
    Update sub-FL models to include new client if necessary
    Send current global model θ_t and corresponding sub-FL models to new client $n+1$
    Include new client in client set C_{t+1}
else
    Determine influence tree including new client
    Start new FL process including new client
    Initialize global model θ with latest parameters θ_t
end if

***** **New Client** $n+1$ *****
Receive global model θ and sub-FL models $\{\theta_s\}$ from server
Initialize local model:
    $\theta_{n+1}^{\text{local}} \leftarrow$ Randomly initialized global model θ_1
for each subsequent communication round t **do**
    Receive global model θ_t and sub-FL models $\{\theta_s^{(t)}\}_{s \in \mathcal{S}_{n+1}}$ from server
    Update global model using own data: $\theta_{n+1,t}^{\text{global}} \leftarrow \text{Update}(\theta_t, \text{data}_{n+1})$
    Update local model using own data: $\theta_{n+1}^{\text{local}} \leftarrow \text{Update}(\theta_{n+1}^{\text{local}}, \text{data}_{n+1})$
    For each sub-FL model $s \in \mathcal{S}_{n+1}$:
        Update sub-FL model using own data: $\theta_{n+1,t}^{(s)} \leftarrow \text{Update}(\theta_s^{(t)}, \text{data}_{n+1})$
    Compute model updates: $\Delta_{n+1}^{\text{global}} = \theta_{n+1,t}^{\text{global}} - \theta_t$
    For each $s \in \mathcal{S}_{n+1}$: $\Delta_{n+1}^{(s)} = \theta_{n+1,t}^{(s)} - \theta_s^{(t)}$
    Send $\Delta_{n+1}^{\text{global}}$ and $\{\Delta_{n+1}^{(s)}\}_{s \in \mathcal{S}_{n+1}}$ to server
end for

E EXACT FEDERATED UNLEARNING GUARANTEES

We use the term *global model* for referring to the global model, which is trained using all client's data and used for deployment. To have a better global model after unlearning, we must design a new FL training process that allows exact federated unlearning while having a better initialization than random initialization. One way to achieve better initialization is to design algorithms that can exploit the remaining clients' existing knowledge. To do this, we propose a algorithm named Multi-Models Training (MMT) that can be incorporated into any existing federated learning framework. The main idea of MMT is to have an additional model for each client that is only trained on its data, making these models unaffected by other clients' training data. We refer to this model as *local model*. However, the local model only accounts for an individual client's influence and does not capture the joint influence of multiple clients. Since all clients influence the global model, we should also capture the joint influence of different clients and then use it to better reinitialize the global model. We can train FL models to capture the joint influence using only a subset of clients. We refer to such FL models as ***sub-FL models*** and next define them formally.

Definition 1 (Sub-FL Model)**.** *Given a subset of clients $S \subset [N]$ with $|S| \geq 2$, a sub-FL model is trained from scratch by the central server using the same FL protocol as the global model to minimize the empirical risk over all clients in S.*

One can train all possible sub-FL models (power set of clients excluding the global model) to capture the influence of all possible interactions among clients. However, it is not computationally feasible as these sub-FL models increase exponentially with the number of clients (i.e., $2^n - n - 2$ for n clients). Another problem of training arbitrary sub-FL models leads to a situation of ***overlapping influence***, which is defined as follows:

Definition 2 (Overlapping Influence)**.** *Let h_{θ_i} and h_{θ_j} be two sub-FL models with influence sets $I(h_{\theta_i}), I(h_{\theta_j}) \subseteq \mathcal{C}$. They exhibit* overlapping influence *if $I(h_{\theta_i}) \cap I(h_{\theta_j}) \neq \emptyset$ and neither set contains the other, i.e., $I(h_{\theta_i}) \nsubseteq I(h_{\theta_j})$ and $I(h_{\theta_j}) \nsubseteq I(h_{\theta_i})$.*

When two sub-FL models share a client, that client's loss is effectively counted twice in the aggregation. Existing works shows that such uneven re-weighting shifts the solution away from the (unweighted) empirical risk minimizer and can strictly increase expected error (Cortes et al., 2010; Shalev-Shwartz & Ben-David, 2014; Alain et al., 2015; Bottou et al., 2018; Byrd & Lipton, 2019). Empirically, duplicating examples or clients is known to hurt generalization in both vision and code-model settings (Aghabagherloo et al., 2025; Buda et al., 2018; Allamanis, 2019). To avoid this, sub-FL models should be trained on disjoint client subsets or trained such that one sub-FL model's client set is a proper superset of another sub-FL model's client set.

One possible way to achieve this is to organize sub-FL models in a hierarchical tree structure. In the tree structure, the root node represents the global model while the leaf nodes correspond to the local models, and intermediate nodes represent sub-FL models, with each child node having disjoint sets of clients compared to its siblings. As we move from the root node to the leaf nodes, each sub-FL model branches into further subsets, maintaining either disjoint relationships or superset relations.

The ***tree structure ensures varying levels of influence among sub-FL models*** throughout the hierarchy. We refer to this hierarchical tree structure as an ***influence tree***. After unlearning a client, we should aggregate the sub-FL models with greater influence (those influenced by more remaining clients) and local models to get the initialization for the global model. If the number of models to aggregate is less, it implies that the initialization of the global model contains more joint influence of remaining clients. This relationship inspires our proposed metric ***influence degradation score*** that measures how good an influence tree is, which we formally define as follows.

Definition 3 (Influence Degradation Score (IDS))**.** *Let $\mathcal{T}$ be any influence tree structure. The influence degradation score for $\mathcal{T}$, denoted by $s(\mathcal{T})$, is the average number of sub-FL and local models that are aggregated to get the initial global model after unlearning any client.*

While the tree structure inherently prevents overlapping influence, it is unclear which structure minimizes IDS, as unlearning probabilities, i.e., the likelihood of clients requesting unlearning, may vary across clients. Since our goal is to minimize IDS, we show that a binary influence tree constructed via Huffman coding achieves the lowest IDS among all n-ary influence tree structures for $n > 2$.

Theorem 1. *Given an n-ary influence tree $\mathcal{T}$ with $n > 2$, there exists a binary influence tree $\mathcal{T}_2$ with a smaller IDS, i.e., $s(\mathcal{T}_2) < s(\mathcal{T})$. Let p_c be the unlearning probability for client c. Then, Huffman*

coding with n symbols representing clients and weights $\{p_c\}_{c=1}^n$ gives the optimal binary influence tree $\mathcal{T}_{Huffman}$ such that $s(\mathcal{T}_{Huffman}) \leq s(\mathcal{T}_2)$ for any influence tree $\mathcal{T}_2$ for the same group of clients.

Proof. We first define the k-split influence node, which is a node in an influence tree with $k > 2$ leaf nodes. We now consider the influence tree $\mathcal{T}$, where k-split influence node only has leaf nodes as children. We denote this node as d, and the set of its leaf nodes is denoted by $\mathcal{C}$. We now follow the following procedure. First, we remove the edge between the node d and any two of its leaf nodes (siblings), denoted by i and j. We create a sub-FL model with these two removed nodes and then add the node for this sub-FL model as a child to the node d. We denote the resulting tree as $\mathcal{T}'$. Let $f(\mathcal{T}, c)$ represent the number of sub-FL and local models that are aggregated to get the initial global modal after unlearning client c in the given influence tree $\mathcal{T}$.

Note that $f(\mathcal{T}', c) = f(\mathcal{T}, c) - 1$ for $c \in C \setminus \{i, j\}$ as one less leaf node to aggregate due to sub-FL model for $\{i, j\}$ leaf nodes, and $f(\mathcal{T}', c) = f(\mathcal{T}, c)$ for $c \in \{i, j\}$ as sub-FL model is no longer useful due to influence of leaf node i or j. With this, we have following IDS due to the node d:

$$\begin{aligned} s_d(\mathcal{T}) &= \sum_{c\in\mathcal{C}} p_c f(\mathcal{T}, c) = \sum_{c\in\mathcal{C}\setminus\{i,j\}} p_c(f(\mathcal{T}', c) + 1) + \sum_{c\in\{i,j\}} p_c(f(\mathcal{T}', c) \\ &= k - 2 + \sum_{c\in\mathcal{C}\setminus\{i,j\}} p_c f(\mathcal{T}', c) + \sum_{c\in\{i,j\}} p_c(f(\mathcal{T}', c) \qquad \text{(node } d \text{ had } k \text{ leaf nodes)} \\ &= k - 2 + \sum_{c\in\mathcal{C}} p_c f(\mathcal{T}', c) = s_d(\mathcal{T}') \\ \implies s_d(\mathcal{T}) &> s_d(\mathcal{T}'). \qquad \text{(as } k > 2) \end{aligned} \tag{1}$$

Iteratively apply the same procedure on the rest of the child nodes until every node only has two children. After this, we obtain a binary tree. Since each operation strictly reduces IDS, $s_d(\mathcal{T}_2) < s_d(\mathcal{T})$. When the original tree already has some child nodes L that already belong to a binary subtree $\mathcal{T}_L$, we treat this subtree as a single child node c'_k and apply the aforementioned operations on the child nodes that do not yet belong to a binary subtree. If all the child nodes belong to some binary subtree, we check from bottom-up to find the largest binary subtrees and treat them as a single child node to apply the aforementioned operations. Following this procedure, we can transform any arbitrary tree into a binary tree. In general,

$$f(\mathcal{T}', l) = f(\mathcal{T}_L, l) + f(\mathcal{T}', c') = f(\mathcal{T}_L, l) + f(\mathcal{T}, c') - 1 = f(\mathcal{T}, l) - 1$$

for $c' \in \mathcal{C} \setminus \{i, j\}$, $l \in L$ and $f(\mathcal{T}', l) = f(\mathcal{T}_L, l) + f(\mathcal{T}', c') = f(\mathcal{T}_L, l) + f(\mathcal{T}, c') = f(\mathcal{T}, l)$ for $c' \in \{i, j\}$, $l \in L$. Notice that $f(\mathcal{T}', l) = f(\mathcal{T}_L, l) + f(\mathcal{T}', c')$ always holds. Therefore, the inequality in Eq. (1) generalizes for any tree structure with the generalized operation. After applying this procedure on any arbitrary tree $\mathcal{T}$ with at least one k-split influence node, the resulting binary tree $\mathcal{T}_2$ always has a strictly smaller value of IDS, i.e., $s(\mathcal{T}_2) < s(\mathcal{T})$. Now we will proof the second part of theorem. Assume N is the number of non-root nodes, q_d is the probability of reaching a non-root node d starting from the root node, and N_d^s is the number of siblings of a non-root node i. Note that for a node d, $q_d = \sum_{c\in\mathcal{C}_d} p_c$ where $\mathcal{C}_d$ is the set of all the client nodes (i.e., leaf nodes) that are descendants of node d and p_c is the probability of unlearning of the c-th descendant. Given an influence binary tree $\mathcal{T}_2$ having n clients with known unlearning probability of each client, the IDS is given as follows:

$$s(\mathcal{T}) = \sum_{c=1}^{n} p_c f(S, c) = \sum_{d=1}^{N} q_d * N_d^s. \tag{2}$$

For $\sum_{c=1}^n p_c f(S, c)$, we can group all leaf nodes that share some common ancestor node d into a collection, with $\mathcal{C}_d$ denoting this set. Since the same node has the same N_c^s, we can sum p_c of all such leaf nodes and rewrite $\sum_{c=1}^n p_c f(S, c)$ as $\sum_{d=1}^N \sum_{c\in\mathcal{C}_d} p_c * N_d^s = \sum_{d=1}^N q_d * N_d^s$. Since each node of the binary tree has only one sibling, we have

$$\sum_{d=1}^{N} q_d * N_d^s = \sum_{d=1}^{N} q_d * 1 = \sum_{d=1}^{N} \sum_{c\in\mathcal{C}_d} p_c = \sum_{c=1}^{n} p_c * l_c$$

where n is the number of leaf nodes, l_c is the depth of the c-th leaf node, and p_c is the probability of reaching a non-root node c starting from the root node. As splitting the q_d of each non-leaf node

into $\mathcal{C}_d = \{p_1, p_2, ..., p_\tau\}$, where $q_d = \sum_{c \in \mathcal{C}_d} p_c$ and each element in $\mathcal{C}_d$ corresponds to a p_c of a leaf node, which is a descendant of that non-leaf node. Therefore, we can write $\sum_{d=1}^{N} q_d * 1$ as the sum of $p_c * l_c$ of all possible branches that reach each leaf node, as all the ancestor nodes of all leaf nodes have exactly one sibling.

Finally, notice that $\sum_{c=1}^{n} p_c * l_c$ is the expected code word length, if the same binary tree is used to represent a binary prefix-free code encoding scheme. Since Huffman coding is optimal for minimizing $\sum_{c=1}^{n} p_c * l_c$, $s(\mathcal{T}_{\text{Huffman}}) \leq s(\mathcal{T}_2)$ where $\mathcal{T}_{\text{Huffman}}$ is an influence tree constructed following Huffman coding and $\mathcal{T}_2$ is any binary influence trees for the same set of p_c. □

With this result, we can use Huffman coding (Huffman, 1952) to construct an influence tree that has the lowest IDS. In some real-life applications, the client's unlearning probability can be unknown. In such cases, we can assume that each client is equally likely to be unlearned, hence having the same unlearning probability. A client (on the leaf node) influences a sub-FL model if there is a path from the model to the leaf node representing that client.

Our work bridges the aforementioned gap in exact federated unlearning by providing zero delay exactness, bounded resource use, and rapid accuracy recovery. Unlike checkpoint heavy or quantization based approaches, our BMT and MMT strategies store at most one extra model per client plus a small, fixed collection of sub-FL models on the server, so memory grows only with the number of participating clients rather than with the number of training rounds. This scaling matches the typical cross silo scenario, where a limited set of organizations collaborate and storage is plentiful. After an unlearning request, the global model is re-initialized from already trained sub models instead of random weights and therefore regains its pre unlearning accuracy within a few rounds, eliminating the extended downtime seen with retraining from scratch or cluster based methods such as FedCIO. In short, our design removes the checkpoint explosion, avoids the accuracy loss introduced by weight quantization, and prevents the slow performance recovery that has so far impeded practical, time-critical deployments of exact federated unlearning.

E.1 Proof of Exact Unlearning Guarantees

This section presents the formal proof MMT and BMT achieving exact unlearning. We first introduce the definitions and results needed for these proofs.

Training model. Let $D = (D_1, \ldots, D_N)$ denote client datasets and let r denote all explicit randomness, independent of D. A training *state* is $m = (\theta, \xi)$, consisting of model parameters and optimizer/training state. Let M denote server-visible metadata and protocol choices; after removing client u, the metadata is denoted M_{-u}.

Definition 4 (Training primitives)**.** *A* state *is $m = (\theta, \xi)$ with parameters θ and training state ξ. Fix server-visible protocol choices M and explicit randomness r, and let ξ_0 denote a canonical fresh training state. Training is modeled as compositions of deterministic primitives:*

$\textsc{Init}(r) = (\theta_0(r), \xi_0)$ *: initialize the random model parameters and fresh training states.*

$\textsc{Copy}(m) = m$ *: make an identical copy of the current model state.*

$\textsc{ResetOpt}((\theta, \xi), r) = (\theta, \xi_0)$ *: keep model parameters, clean up optimizer history*

$\textsc{TrainOn}(m, D_i, r)$ *: the M-specified local training procedure on D_i starting from m, and*

$\textsc{Agg}(m_1, \ldots, m_k; M, r)$ *: the M-specified server aggregation rule.*

Only TrainOn *needs private client data, and it reads only the dataset passed to it.*

Conditioned on (r, M), the sequence of primitive calls and all non-data arguments (including scheduling, aggregation rules, and hyperparameters) do not depend on any D_i except through the explicit appearance of D_i as an argument to TrainOn. After unlearning client u, the orchestration uses only M_{-u}.

Definition 5 (Influence set)**.** *For any state m produced by the training computation, define its* influence set $\mathcal{I}(m) \subseteq \mathcal{C}$ *as the set of clients whose datasets may influence m. Formally, $\mathcal{I}(m)$ is the smallest set satisfying:*
(i) $\mathcal{I}(\textsc{Init}(r)) = \emptyset$;
*(ii) data-free primitives (*Copy, ResetOpt*) preserve influence sets;*

(iii) $\mathcal{I}(\textsc{Agg}(m_1, \ldots, m_k)) = \bigcup_{j=1}^{k} \mathcal{I}(m_j)$;
(iv) $\mathcal{I}(\textsc{TrainOn}(m, D_i, r)) = \mathcal{I}(m) \cup \{i\}$.
For a deployed model h_θ *extracted from* $m = (\theta, \xi)$, *define* $\mathcal{I}(h_\theta) \doteq I(m)$.

Lemma 1 (Dependence only on influence set)**.** *Fix* (r, M) *and assume data-independent orchestration. For any state* m, *the value of* m *depends on the datasets only through* $\{D_i\}_{i \in I(m)}$. *Equivalently, if* $u \notin I(m)$, *then for any replacement* D'_u *that leaves* M *unchanged,*

$$m(D_1, \ldots, D_u, \ldots, D_N; r, M) = m(D_1, \ldots, D'_u, \ldots, D_N; r, M).$$

Proof. Conditioned on (r, M), the training expression generating m is fixed. We proceed by structural induction over this expression. INIT depends only on r.

Data-free primitives are deterministic functions of their inputs and introduce no new data dependence. AGG depends only on its input states and thus on the union of their influence sets. TRAINON accesses only (m, D_i, r) and introduces dependence on D_i. In all cases, m depends only on datasets indexed by $\mathcal{I}(m)$. □

Definition 6 (Exact federated unlearning)**.** *An unlearning procedure* U *for client* u *is* exact *if, for all replacements* D'_u *that keep metadata unchanged,*

$$U(D_1, \ldots, D_u, \ldots, D_N; r, M_{-u}) = U(D_1, \ldots, D'_u, \ldots, D_N; r, M_{-u}).$$

Theorem 2. BMT *and* MMT *achieve exact unlearning.*

Proof. Let r and M_{-u} be fixed. Consider two models differing only in the removed client's dataset, D_u versus D'_u. BMT and MMT discard all stored states m with $u \in \mathcal{I}(m)$ and retain only states $\{m_1, \ldots, m_k\}$ satisfying $u \notin \mathcal{I}(m_j)$. By Lemma 1, each retained state m_j is invariant to replacing D_u by D'_u, so the retained collection is identical in both models. The post-unlearning initialization

$$m^{(0)}_{\text{post}} = \textsc{ResetOpt}(\textsc{Agg}(m_1, \ldots, m_k; M_{-u}, r), r)$$

is a deterministic function of $(m_1, \ldots, m_k, M_{-u}, r)$ and therefore identical in both models. Moreover, since Agg unions influence sets and ResetOpt preserves them,

$$I\left(m^{(0)}_{\text{post}}\right) = \bigcup_{j=1}^{k} I(m_j),$$

so it excludes u because every retained m_j excludes u.

Subsequent training invokes TRAINON only on datasets D_i with $i \neq u$. Since TRAINON is the only primitive that can add a client to the influence set, u is never reintroduced. Thus, the final deployed state m_{post} satisfies $u \notin I(m_{\text{post}})$. Applying Lemma 1, m_{post} is invariant to replacing D_u by D'_u, which is exactly the definition of exact unlearning. □

Remark 1 (Exactness vs. retraining-from-scratch)**.** *RfS is also exact under Definition 6 since it never invokes* $\mathrm{TrainOn}(\cdot, D_u, \cdot)$ *after removing* u; BMT *and* MMT *need not match* RFS *parameters, they satisfy the same invariance notion with a better initialization.*

F STORAGE AND COMPUTATIONAL COMPLEXITY

In this section, we analyze the storage and computational complexity for RFS, BMT, and MMT, where M denotes the storage required for a single model, C is the baseline training cost of RFS, and n is the number of participating clients. Storage complexity is as follows: RFS requires M storage on the server and M per client; BMT doubles the per-client storage to $2M$ for an additional local model, while server still requires M; and MMT requires server to store $(n-1)M$ model for the internal nodes of a balanced influence tree with n leave nodes. Each client in MMT, participating in maximum $\lceil \log_2 n \rceil$ models and maintaining a local model, uses maximum $(\lceil \log_2 n \rceil + 1)M$. While for MMT the server stores $O(n)$ sub-FL models, cross-silo settings usually contain less than a hundred parties, so the memory cost is typically on the range of Gigabytes (Table 6), which is small relative to modern server hardware. Moreover, applying memory-efficient model training techniques such as LoRA/QLoRA can further reduce the requirement by only storing model adapters (Hu et al., 2022; Dettmers et al., 2023; Babakniya et al., 2023; Yi et al., 2023; Wang et al., 2024).

Table 3: Storage and computational resource requirements.

Algorithms	Storage (#models)		Total compute
	Server	Per-client	
RfS	M	M	C
BMT	M	$2M$	$2C$
MMT (balanced)	$(n-1)M$	$(\lceil \log_2 n \rceil + 1)M$	$(\lceil \log_2 n \rceil + 1)C$

In terms of computational complexity (assuming zero communication overhead), RFS has the baseline cost of C, corresponding to one full round of training across all clients; BMT approximately doubles this cost to $2C$, as each client must update both the global model and an additional isolated model; and MMT organizes clients into a balanced influence tree, the computational cost increases with the tree depth. Specifically, each of the $\lceil \log_2 n \rceil$ internal levels processes all n clients once, and each client also trains its isolated model. This results in a maximum total computational cost of $(\lceil \log_2 n \rceil + 1)C$. Table 3 summarizes storage and computational complexity. The empirical results in Section G.2 closely align as storage usage matches exactly, and compute usage is within 8%.

G EXPERIMENTS

FL Setup & Training. Our experiments focus on non-IID data silos. For simplicity, we assume the number of clients equates the number of classes in our datasets. Due to high training cost, we only consider top-8 classes with the most training samples on LI and MSA. To simulate non-IID data, we assign client i with most samples from class i and a few from each of the remaining classes. We denote ρ as the ratio between the major class and the minor class. Clients' dataset have the same sizes of $\mathcal{D}_c^{train}$ for training and $\mathcal{D}_c^{test}$ for testing. While this setting only amplifies label skewness while fixing clients' dataset sizes, we also provide results for a Dirichlet-based non-IID setting where both label proportions and the dataset sizes differ across clients in App. G.3. The dataset configurations and their corresponding model architectures are detailed in Tab. 4.

Table 4: Summary of the datasets & models. † means each client only has samples from 1 class.

Dataset	#classes	ρ	$\mathcal{D}_c^{train}$ size	$\mathcal{D}_c^{test}$ size	Model
MNIST	10	0.02	200	200	MLP
FMNIST	10	0.02	200	200	MLP
CIFAR-10	10	0.2	1000	300	2-layer CNN
CIFAR-100	100	0.1	400	100	VGG-16
LI	top-8	$\sim 0^\dagger$	200	100	GPT-2 & Llama-3.2+LoRA
MSA	top-8	$\sim 0^\dagger$	200	100	GPT-2 & Llama-3.2+LoRA

All experiments are conducted on NVIDIA L40 (46GB) and H100 (80GB) GPUs. For FL training, we adopt a popular FL algorithm FedAvg (McMahan et al., 2017). The training hyperparameters (e.g., number of communication rounds, local epochs, batch size, etc.) are detailed in Tab. 5. Additionally, we finetune the pretrained LLaMA-3.2 model using LoRA adapters with a rank of $r = 16$, scaling factor $\alpha = 32$, and a dropout rate of 0.05.

Table 5: Hyperparameters used for local training. 'BS' is batch size and 'LR' is learning rate.

Datasets	#rounds	#epochs	BS	Optimizer	LR	Decay	Clipping
MNIST	100	1	20	SGD	0.01	0.1	10
FashionMNIST	100	1	20	SGD	0.01	0.1	10
CIFAR-10	300	1	64	AdamW	0.01	0.1	10
CIFAR-100	100	10	64	SGD	0.005	10^{-5}	-
GPT-2 LI	50	1	8	Adam	10^{-6}	0.01	10
GPT-2 MSA	80	1	8	Adam	10^{-6}	0.01	10
Llama-3.2 LI	30	1	8	Adam	10^{-7}	0.01	1
Llama-3.2 MSA	30	1	8	Adam	10^{-7}	0.01	1

Unlearning Settings. In sequential unlearning, we use the $\{0, 2\}$ as the default order of clients to be unlearned. This order is deliberately chosen to MMT's disadvantage as the global model cannot be replaced by any of the sub-FL models and must be aggregated from our sub-FL models.

Metrics. We report accuracy on an aggregated test set comprising all clients' test sets, including those would join/leave the collaboration at later rounds. Additionally, we conduct membership inference attacks (MIA) (Zarifzadeh et al., 2024; Mattern et al., 2023) to evaluate whether the membership status (i.e., being included during FL training) of unlearned clients can be inferred from the global model's output post-unlearning. An exact FU algorithm should have an MIA AUC close to 0.5, showing that the global model retains no distinguishing information about the unlearned clients' data beyond random guessing.

G.1 Full Main Results

Sequential Unlearning. Fig. 5 presents the full results, including additional evaluation on FMNIST, GPT-2 LI, and GPT-2 MSA, in sequential unlearning. Similarly, we observe that BMT and MMT consistently achieves the fastest convergence among the tested baselines thanks to better post-unlearning initialization.

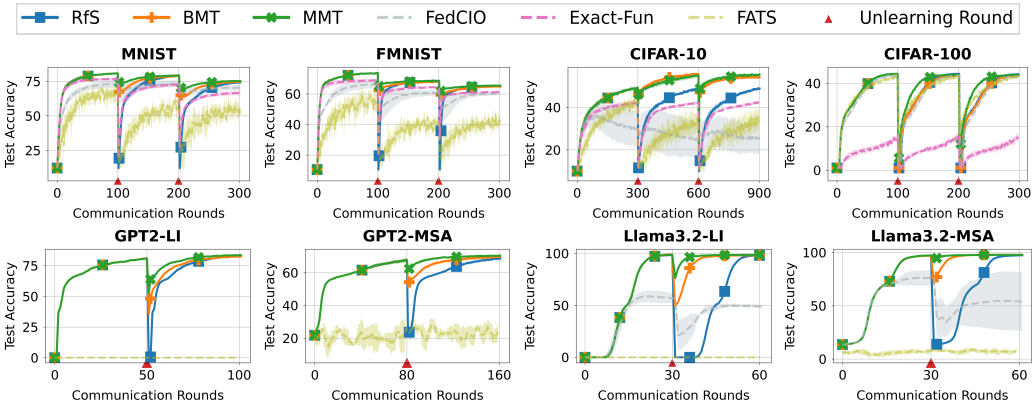


Figure 5: Test accuracy of the global model in the sequential unlearning. Results are averaged over 5 random seeds, with standard deviations shown as shaded regions.

Continual Learning & Unlearning. Fig. 6 shows the test accuracy of the global model in a dynamic environment where clients join (marked by ▲) and leave (marked by ▲) the collaboration over time. We show experimental results on the design of different sub-FL trees for client expansion in Fig. 9a. Overall, these results highlight the practical utility of our approaches in supporting simultaneous learning and unlearning within real-world environments.

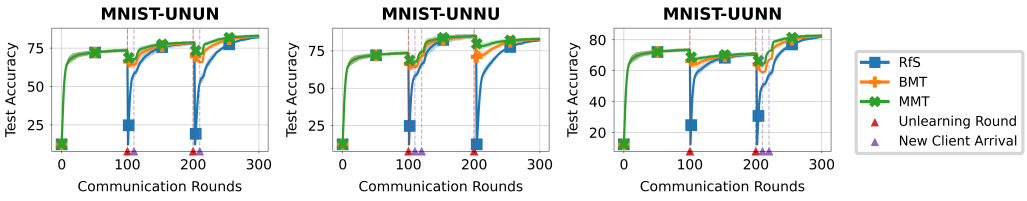


Figure 6: Test accuracy on MNIST under varying continual learning and unlearning orderings (**N**: New Client Arrival, **U**: Unlearning). Results are averaged across 3 random seeds.

G.2 Efficiency

Table 6 presents the peak GPU memory usage (in MB) and total compute cost (in GFLOPs) of our algorithms across different settings, which corroborates our analysis in Section F. We note that although both algorithms add storage and compute overheads, *storage is no longer the primary*

bottleneck when training large models as disk space is relatively inexpensive. Meanwhile, retraining time directly translates to service downtime, which can be minimized in BMT and MMT.

Table 6: Peak GPU memory (MB) and total compute (GFLOPs) required for different algorithms. Memory is reported separately for the server and each client.

Dataset	Algorithm	Peak GPU Memory (MB)		GFLOPs
		Server	Per-client	
MNIST	RfS	1559.1	1559.1	36.43
	BMT	1559.1	3118.2	72.86
	MMT	14031.9	7795.5	161.07
FMNIST	RfS	1561.3	1561.3	97.15
	BMT	1561.3	3122.6	194.31
	MMT	14051.7	7806.5	429.54
CIFAR-100	RfS	2746.7	2746.7	5.59×10^7
	BMT	2746.7	5493.4	1.12×10^8
	MMT	271900.0	21973.6	4.13×10^8

Comparison to Exact-Fun and FATS. Compared to our algorithms, Exact-Fun and FATS require storing model checkpoints at every training round. Thus, their memory usage grows *linearly* with training duration. In contrast, BMT and MMT scale with the number of clients, which is typically small in cross-silo settings. In time-critical applications involving sensitive, private data, the improved post-unlearning accuracy and faster convergence of BMT and MMT offer a favorable trade-off for their modest overhead.

G.2.1 Deployment feasibility beyond FLOPs

In large-model deployments, practical bottlenecks often include communication volume and peak memory usage, not only total compute. We therefore evaluate a deployment-oriented instantiation of BMT and MMT using parameter-efficient fine-tuning (QLoRA), where a shared quantized backbone is maintained and only lightweight per-node adapters are trained and communicated. We report end-to-end overheads (communication, storage, peak client/server memory, and total FLOPs) for GPT-2 and Llama-3.2-3B in supplementary material, showing that adapters reduce per-update payload and total communication by orders of magnitude and substantially reduce peak memory, improving practicality in the cross-silo setting. A feasibility analysis for exact federated unlearning should consider multiple system bottlenecks, not only total FLOPs. For large models, practical constraints are often dominated by (i) communication volume, (ii) server-side storage footprint, and (iii) peak client/server accelerator memory. While Section F analyzes storage and compute scaling for BMT/MMT, we further quantify these practical bottlenecks under a deployment-oriented instantiation based on parameter-efficient fine-tuning.

QLoRA instantiation. We pair BMT/MMT with QLoRA by maintaining a shared quantized backbone and training only lightweight adapters for each model instance (local and sub-FL nodes). Under this design, the additional auxiliary models correspond to small adapters rather than full model copies. Communication also corresponds to adapter exchange rather than full-parameter exchange.

Metrics. We report: (1) total communication across all clients and all trained model instances; (2) stored footprint, separated into backbone ("Model") and adapter size ("Adapt"); (3) peak CPU/GPU memory on clients and on the server; and (4) total FLOPs aggregated across all clients. Tables 7 and 8 summarize the results.

Key takeaways. Across GPT-2 and Llama-3.2-3B tasks, QLoRA reduces the communicated payload per update from a full model to an adapter (e.g., GPT-2: 237.37 MB $\rightarrow$ 2.26 MB; Llama-3.2-3B: 6127.88 MB $\rightarrow$ 17.52 MB), which yields orders-of-magnitude reductions in total communication (e.g., GPT-2 LI, MMT: 11.59 GB $\rightarrow$ 0.11 GB; Llama-3.2-3B MSA, MMT: 299.21 GB $\rightarrow$ 0.85 GB). It also significantly lowers peak memory usage on both clients and server (Table 8) and reduces total FLOPs (Table 7). The remaining MMT pre-training overhead is governed by the branching factor b (see Fig. 9a), with $b = n$ recovering the compute-efficient BMT endpoint.

Table 7: Communication, storage, and compute overheads for Full fine-tuning versus QLoRA. "Model (MB)" is the stored backbone footprint under each mode. "Adapt (MB)" is the adapter size (also the communicated payload under QLoRA). "LI" denotes language identification and "MSA" denotes multilingual sentiment analysis.

Method	Task	Model	Mode	Comm (GB)	Model (MB)	Adapt (MB)	FLOPs (10^{14})
BMT	LI	GPT-2	Full	6.95	237.37	0.00	2.03
			QLoRA	0.07	119.49	2.26	1.36
BMT	MSA	GPT-2	Full	6.95	237.37	0.00	1.69
			QLoRA	0.07	119.49	2.26	1.13
BMT	LI	Llama-3.2-3B	Full	179.53	6127.88	0.00	18.22
			QLoRA	0.51	2156.39	17.52	12.14
BMT	MSA	Llama-3.2-3B	Full	179.53	6127.88	0.00	19.93
			QLoRA	0.51	2156.39	17.52	13.28
MMT	LI	GPT-2	Full	11.59	237.37	0.00	3.48
			QLoRA	0.11	119.49	2.26	2.32
MMT	MSA	GPT-2	Full	11.59	237.37	0.00	2.59
			QLoRA	0.11	119.49	2.26	1.73
MMT	LI	Llama-3.2-3B	Full	299.21	6127.88	0.00	30.17
			QLoRA	0.85	2156.39	17.52	20.10
MMT	MSA	Llama-3.2-3B	Full	299.21	6127.88	0.00	33.60
			QLoRA	0.85	2156.39	17.52	22.40

Table 8: Peak memory usage (GB) for Full fine-tuning versus QLoRA, reported separately for CPU and GPU on clients and the server.

Method	Task	Model	Mode	Client Peak (GB)		Server Peak (GB)	
				CPU	GPU	CPU	GPU
BMT	LI	GPT-2	Full	4.44	2.10	4.68	1.70
			QLoRA	3.64	1.02	4.10	1.35
BMT	MSA	GPT-2	Full	4.32	2.10	4.61	1.57
			QLoRA	3.44	1.02	3.98	1.29
BMT	LI	Llama-3.2-3B	Full	45.34	42.04	51.32	24.85
			QLoRA	18.61	7.91	20.66	8.92
BMT	MSA	Llama-3.2-3B	Full	45.19	42.06	51.18	24.86
			QLoRA	18.37	10.20	20.44	8.92
MMT	LI	GPT-2	Full	6.15	2.10	8.48	9.31
			QLoRA	4.38	1.06	5.57	5.44
MMT	MSA	GPT-2	Full	5.82	2.10	8.28	9.91
			QLoRA	4.17	1.03	5.64	6.11
MMT	LI	Llama-3.2-3B	Full	82.47	48.02	144.47	175.01
			QLoRA	32.61	9.82	52.05	34.92
MMT	MSA	Llama-3.2-3B	Full	81.19	48.05	143.19	156.81
			QLoRA	30.98	10.62	50.11	27.54

G.3 Ablation Studies

Label Imbalance Ratio. Fig. 7 presents the model performance on MNIST across varying label imbalance ratios ρ, where $\rho = 1.0$ indicates uniform label distribution and $\rho \approx 0$ indicates extremely imbalance between the majority and minority classes. We observe that the gaps among the tested methods become more pronounced with lower ρ (higher imbalance), with MMT converging the fastest, followed by BMT and RFS. We attribute this improved post-unlearning performance to the binary structure in MMT, which preserves the majority of cross-client interactions. These results further demonstrate that our approaches are well-suited for real-world deployments where non-IID data distributions across clients are often the norm.

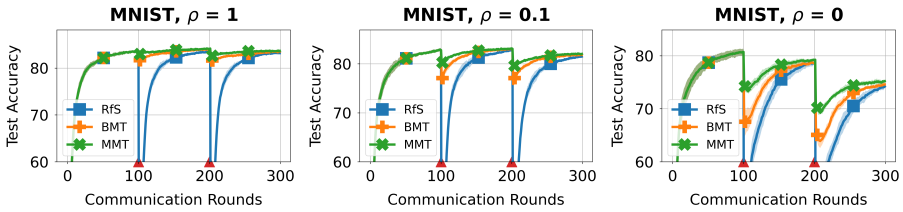


Figure 7: **Effects of label imbalance ratio.** While BMT and MMT show comparable performance under uniform label distributions ($\rho = 1$), MMT shows the fastest convergence rate as label distributions become increasingly imbalanced, followed by BMT and RFS.

Dirichlet-based Non-IID Setting. Fig. 8 shows the global model performance when the sample sizes for each class per client follow a Dirichlet distribution, with $\alpha = 0.5$ for vision tasks and $\alpha = 0.1$ for language tasks. Our results show that MMT and BMT continue to achieve better post-unlearning performance than RFS and other baselines, especially on more complex tasks like CIFAR-10 and CIFAR-100.

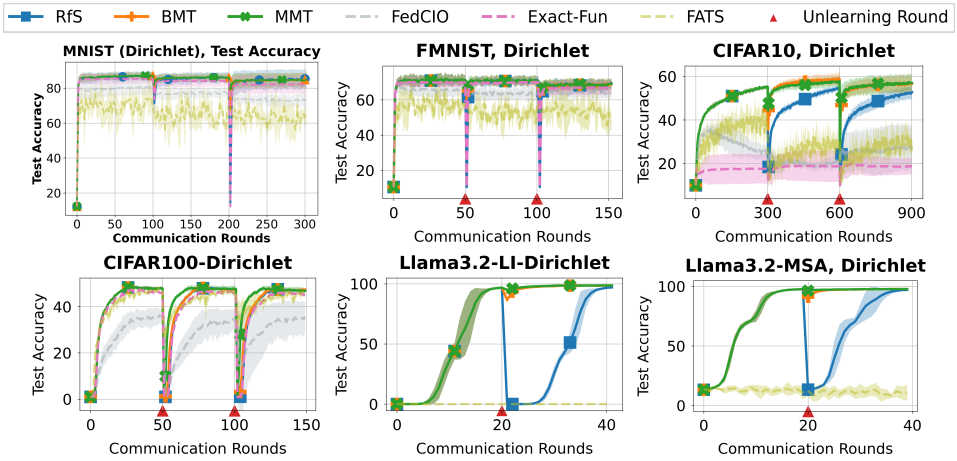
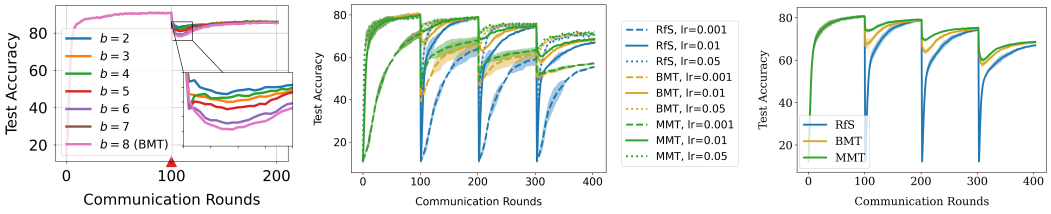


Figure 8: Global model performance in Dirichlet-based non-IID setting (3 random seeds).



(a) MMT's branching factor b (b) Learning rate in FL training (c) FedProx

Figure 9: Global model performance on MNIST with (a) varying branching factor b, (b) varying learning rate in FL training, and (c) FedProx as an alternative FL algorithm. Results are reported across 5 random seeds.

Varying Branching Factor. Fig. 9a shows how the branching factor $b \in [1, n]$ of the influence tree affects performance, where $b = 1$ corresponds to RFS (hence omitted) and $b = n$ recovers BMT. Varying b provides an explicit *compute-performance trade-off*: larger b yields a shallower tree (smaller depth) and therefore fewer sub-FL models per client to update, reducing the additional training overhead. Conversely, a smaller b yields deeper trees and higher pre-training cost. On the performance side, a smaller b tends to yield higher test accuracy because each sub-FL model is trained on a larger subset of clients (roughly $\frac{n}{b}$ clients at the first level in a balanced tree), which

better captures clients' cross-influence on the global model and provides a stronger post-unlearning initialization. In contrast, the endpoint $b = n$ (BMT) is the most compute-efficient special case but does not maintain intermediate sub-FL models that capture joint influence, which can lead to weaker post-unlearning initialization than a smaller b. Overall, the default binary structure ($b = 2$) offers the best trade-off in our settings.

Varying Learning Rate. We show the global model performance on MNIST across a range of learning rates $\{0.001, 0.01, 0.05\}$ in Fig. 9b. MMT consistently achieves the fastest convergence after unlearning, followed by BMT and RFS, regardless of the learning rate. These results demonstrate the robustness and effectiveness of BMT and MMT across different learning rates.

Varying FL Algorithms. Fig. 9c shows global model performance on MNIST using FedProx (Li et al., 2020), an alternative FL algorithm designed for non-IID settings, with $\mu = 0.1$. We observe the same trend as our experiments with FedAvg (Fig. 1): MMT converges fastest after unlearning, followed by BMT and RFS. This demonstrates that our proposed approaches are generic and effective across different FL frameworks.

G.4 Comparison with Approximate Federated Unlearning Algorithms

In this experiment, we compare the MIA AUC of our proposed algorithms with two recent approximate FU methods, FedOSD (Pan et al., 2025) and FedAU (Gu et al., 2024), on the MNIST and CIFAR-100 datasets. Table 9 shows that FedOSD and FedAU have high MIA AUC scores, which deviate significantly from a perfect exact FU algorithms RFS, particularly on CIFAR-100. These results suggest that the residual influence of unlearned clients remains in the global model after unlearning with approximate FU algorithms.

Table 9: MIA AUC comparison with approximate FU methods† (averaged across 3 seeds). Unlike BMT and MMT, MIA AUC for FedOSD and FedAU deviate significantly from RFS, particularly on CIFAR-100.

Tasks	**No Unl.**	**RFS**	**BMT**	**MMT**	**FedOSD†**	**FedAU†**
MNIST	0.6094	0.5199	0.5224	0.5318	0.5439	0.5355
CIFAR-100	0.9319	0.5208	0.5418	0.5169	0.9445	0.9016

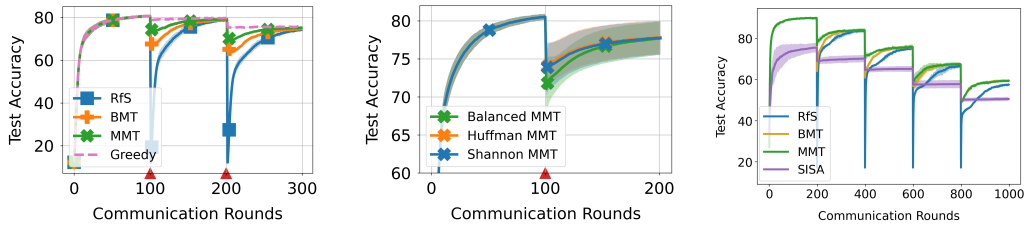
Generally, these results highlight a fundamental trade-off in machine unlearning: while approximate algorithms such as FedOSD and FedAU are appealing due to their lower computational and memory overhead, they often struggle to guarantee perfect unlearning and compromise the unlearned clients' privacy. In contrast, BMT and MMT require additional resources to maintain multiple model instances, but this overhead is **justified by their theoretical guarantees and empirically demonstrated effectiveness** via improved post-unlearning performance and strong defense against MIA attack. Therefore, we believe that our exact FU algorithms remain essential in scenarios where strong assurances of unlearning are needed.

G.5 Supplementary Experiments

Known Unlearning Order. If the unlearning order is known in advance, e.g., clients can subscribe to the service for a fixed duration and then leave upon expiration, an optimal MMT structure can be constructed by placing soon-to-expire clients near the root and greedily building the tree. Fig. 10a shows that greedy MMT outperforms the balanced MMT, which makes it preferable when the unlearning order is known.

Non-uniform Unlearning Probabilities. Fig. 10b shows the model performance when the unlearned client is sampled 100 times with pre-defined non-uniform unlearning probabilities under different tree construction algorithms. MMT structures based on Shannon-Fano coding and Huffman coding visibly outperform the default balanced MMT, with Huffman-MMT achieving slightly better results than the Shannon-Fano counterpart. These results align with the classical information theory results that Huffman coding is more optimal than Shannon-Fano coding for prefix-free code (Cover & Thomas, 2006).

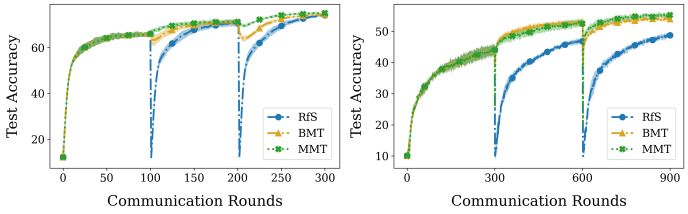
Benchmarking against SISA. Adapting the model-agnostic unlearning method SISA (Bourtoule et al., 2021) to the FL setting, we train an isolated model for each client and remove only the affected model when a client leaves to achieve exact FU. Unfortunately, we observe that SISA often performs suboptimally in the presence of non-IID clients in Fig. 10c, which suggests that it is a less competitive benchmark for exact FU.



(a) Known unlearning order (b) Non-uniform unlearning prob. (c) Benchmarking against SISA

Figure 10: Performance on MNIST for (a) non-uniform unlearning probabilities, (b) known unlearning order, and (c) benchmarking against SISA unlearning method. Results are averaged across 3 random seeds.

Unlearning Adversarial Clients. In this experiment, we consider a scenario where the server aims to remove adversarial clients from the federated learning collaboration and undo their impact on the global model. To simulate adversarial behavior, we corrupt a client's local data by replacing its labels with random values. As shown in Fig. 11, all methods benefit from the removal of adversarial clients, but BMT and MMT consistently outperform RFS, demonstrating greater robustness and recovery capability in the presence of noisy or malicious data.



(a) MNIST (b) CIFAR-10

Figure 11: Test accuracy of the global model when unlearning adversarial clients. Results are averaged across 5 random trials.